Microsoft

Collect, Combine, and Transform Data Using Power Query in Excel and Power BI

Gil Raviv

Sample files on the web

FREE SAMPLE CHAPTER

SHARE WITH OTHERS

Microsoft

Collect, Combine, and Transform Data Using Power Query in Excel and Power BI

Gil Raviv

COLLECT, COMBINE, AND TRANSFORM DATA USING POWER QUERY IN EXCEL AND POWER BI

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

ISBN-13: 978-1-5093-0795-1
ISBN-10: 1-5093-0795-8

Library of Congress Control Number: 2018954693

01 18

Trademarks

Warning and Disclaimer

PUBLISHER
Mark Taub

ACQUISITIONS EDITOR
Trina MacDonald

DEVELOPMENT EDITOR
Ellie Bru

MANAGING EDITOR
Sandra Schroeder

SENIOR PROJECT EDITOR
Tonya Simpson

COPY EDITOR
Kitty Wilson

INDEXER
Erika Millen

PROOFREADER
Abigail Manheim

TECHNICAL EDITOR
Justin DeVault

COVER DESIGNER
Twist Creative, Seattle

COMPOSITOR
codemantra

COVER IMAGE
Malosee Dolo/ShutterStock

Contents at a Glance

Contents

Figure Credits

Chapter 1, "Introduction to Power Query," Figures 1-1 through 1-9 courtesy of Microsoft Corporation.

Chapter 2, "Basic Data Preparation Challenges," Figures 2-1 through 2-16 courtesy of Microsoft Corporation.

Chapter 3, "Combining Data from Multiple Sources," Figures 3-1 through 3-8 courtesy of Microsoft Corporation.

Chapter 4, "Combining Mismatched Tables," Figures 4-1 through 4-11 courtesy of Microsoft Corporation.

Chapter 5, "Preserving Context," Figures 5-1 through 5-11 courtesy of Microsoft Corporation.

Chapter 6, "Unpivoting Tables," Figures 6-1 through 6-9 courtesy of Microsoft Corporation.

Chapter 7, "Advanced Unpivoting and Pivoting of Tables," Figures 7-1 through 7-5 courtesy of Microsoft Corporation.

Chapter 8, "Addressing Collaboration Challenges," Figures 8-1 through 8-10 courtesy of Microsoft Corporation.

Chapter 9, "Introduction to the Power Query M Formula Language," Figures 9-2 through 9-10 courtesy of Microsoft Corporation.

Chapter 10, "From Pitfalls to Robust Queries," Figures 10-2 through 10-9 courtesy of Microsoft Corporation.

Chapter 11, "Basic Text Analytics," Figures 11-1 through 11-14, Figures 11-16 and 11-17 courtesy of Microsoft Corporation.

Chapter 12, "Advanced Text Analytics: Extracting Meaning," Figures 12-3 through 12-8, Figures 12-10 through 12-14 courtesy of Microsoft Corporation.

Chapter 13, "Social Network Analytics," Figures 13-1 through 13-11 courtesy of Microsoft Corporation.

Chapter 14, "Final Project: Combining It All Together," Figures 14-1 through 14-3 courtesy of Microsoft Corporation.

Foreword

When we set out to build the original Power Query add-in for Excel, we had a simple yet ambitious mission: connecting to and transforming the world's data. Five years later, we've moved beyond the original Excel add-in with native integration into Excel, Power BI, Power Apps, and a growing set of products that need to extract and transform data. But our original mission remains largely unchanged. With the ever-increasing heterogeneity of data, in many ways, our mission feels even more ambitious and challenging than ever. Much of today's computing landscape is centered around data, but data isn't always where or how you need it—we continue to advance Power Query with the goal of bridging that gap between the raw and desired states of data.

Throughout the Power Query journey, the user community has played a critical role in shaping the product through suggestions and feedback. The community has also played a central role in developing valuable educational content. As one of the key drivers of Power Query's native integration into Excel 2016, Gil is well placed to provide valuable insights and tips for a variety of scenarios. Even after his tenure at Microsoft, Gil has remained an active and influential member of the Power Query community. Happy querying!

—Sid Jayadevan, Engineering Manager for Power Query, Microsoft Corporation

For readers not familiar with Power Query, it is an incredibly powerful and extensible engine that is the core of Microsoft BI tools. It enhances self-service business intelligence (BI) with an intuitive and consistent experience for discovering, combining, and refining data across a wide variety of sources. With data preparation typically touted as 80% of any BI solution, having a firm grasp of Power Query should be your first step in any sort of reporting or data discovery initiative. In addition to the core Power Query functionalities, Gil covers more advanced topics, such as how to use Power Query to automate data preparation and cleansing, how to connect to social networks to capture what your customers are saying about your business, how to use services like machine learning to do sentiment analysis, and how to use the M language to make practically any type of raw data a source of insights you glean value from. This book stands out in that it provides additional companion content with completed samples, data sources, and step-by-step tutorials.

Gil is a former member of the Excel team and the Microsoft Data Team. He directly contributed to the features and design of Power Query and has an amazing wealth of knowledge using Power Query and showing how it can make difficult data integration problems easy. That said, despite Power Query's inherently extensible and easy-to-use design, mastering it for enterprise scenarios can still be difficult. Luckily for the reader, as an avid community member, forum contributor, conference presenter, peer mentor, and Power BI MVP, Gil Raviv is a master at taking complex concepts and decomposing them into very easy-to-follow steps, setting the reader up for success and making this book a must have for any BI specialist, data systems owner, or businessperson who wants to get value out of the data around him.

—Charles Sterling, Senior Program Manager,
Microsoft Corporation

About the Author

Gil Raviv is a Microsoft MVP and a Power BI blogger at https://DataChant.com. As a Senior Program Manager on the Microsoft Excel Product team, Gil led the design and integration of Power Query as the next-generation Get Data and data-wrangling technology in Excel 2016, and he has been a devoted M practitioner ever since.

With 20 years of software development experience, and four U.S. patents in the fields of social networks, cyber security, and analytics, Gil has held a variety of innovative roles in cyber security and data analytics, and he has delivered a wide range of software products, from advanced threat detection enterprise systems to protection of kids on Facebook.

In his blog, DataChant.com, Gil has been chanting about Power BI and Power Query since he moved to his new home in the Chicago area in early 2016. As a Group Manager in Avanade's Analytics Practice, Gil is helping Fortune 500 clients create modern self-service analytics capability and solutions by leveraging Power BI and Azure.

You can contact Gil at gilra@datachant.com.

Acknowledgments

Writing this book is one of the scariest things I have willingly chosen to do, knowing I was going to journey into an uncharted land where only a few people have gone before and approach an ever-evolving technology that is relatively unfamiliar yet can drastically improve the professional lives of many users. How can I share the knowledge of this technology in a way that will enable you to harness its true essence and empower you to make a real impact on your business?

The writing of this book would not have been possible without the help and inspiration I received from many people.

First, I would like to thank my readers at DataChant.com. Your feedback and support made this endeavor possible. You have taught me the power of sharing.

Thank you to my wife and children, for being stranded at home with me for many days in late 2017 and the colder parts of 2018 to support my work. Thank you for your support. I hope you can also blame the winter in Chicago for staying with me so many weekends.

Special thanks to Trina MacDonald, my senior editor at Pearson. You reached out to me one day with an idea to write a book and have been supporting me all the way in publishing a completely different one. Good luck in your new journey.

Thank you to Justin DeVault, my first Six Sigma Master Black Belt client. As a technical editor, you combined your business savvy and technical prowess to review 14 chapters, 71 exercises, and 211 exercise files to ensure that the book can deliver on its promise. Without your insights, we could not have made it. You were the best person for this job.

To Microsoft Press, Pearson, Loretta Yates and the whole publishing team that contributed to the project, thank you! Thank you, Songlin Qiu, Ellie Bru, and Kitty Wilson for editing and proofreading and Tonya Simpson for orchestrating the production efforts; you have all magically transformed 14 chapters of Word documents into this book.

To my dear friend Yohai Nir, thank you for the rapport and guidance through the initial stages of the book.

Thank you to Luis Cabrera-Cordon, for reviewing Chapter 12. I hope that this chapter will help more business analysts use Microsoft Cognitive Services and gain new insights without the help of developers or data scientists.

To the amazing Program Managers Guy Hunkin, Miguel Llopis, Matt Masson, and Chuck Sterling: Thank you for the ongoing support and technical advice. Your work is truly inspirational.

Sid Jayadevan, Eli Schwarz, Vladik Branevich, and the brilliant people on the Redmond and Israeli development teams: It was a real pleasure working with you to deliver Power Query in Excel 2016.

To Yigal Edery, special thanks for accepting me into the ranks of the Microsoft Excel team and for challenging me to do more. I will never forget the night you pulled me over on the side of the road to share feedback and thank me.

Rob Collie, I wouldn't be here without you. You had welcomed me to PowerPivotPro.com as a guest blogger and a principal consultant, and you helped me make the leap into a brave new world.

Marco Russo, Ken Puls, Chris Webb, Matt Allington, and Reza Rad—My fellow Microsoft MVPs and Power BI bloggers—you are my role models, and I thank you for the inspiration and vast knowledge.

Since I joined the Avanade Analytics team in early 2017, I have learned so much from all of you at Avanade. Special thanks to Neelesh Raheja for your mentorship and leadership. You have truly expanded my horizons in the sea of analytics.

Finally, to my parents. Although I now live 6,208 miles away, I want to thank you. Dad, you had taught me how to crunch numbers and use formulas in Excel many years ago. And, Mom, your artistic talent is influencing my Power BI visuals every day.

—Gil Raviv

Introduction

Did you know that there is a data transformation technology inside Microsoft Excel, Power BI, and other products that allows you to work miracles on your data, avoid repetitive manual work, and save up to 80% of your time?

- Every time you copy/paste similar data to your workbook and manually clean it, you are wasting precious time, possibly unaware of the alternative way to do it better and faster.
- Every time you rely on others to get your data in the right shape and condition, you should know that there is an easier way to reshape your data once and enjoy an automation that works for you.
- Every time you need to make quick informed decisions but confront massive data cleansing challenges, know you can now easily address these challenges and gain unprecedented potential to reduce the time to insight.

Are you ready for the change? You are about to replace the maddening frustration of the repetitive manual data cleansing effort with sheer excitement and fun, and throughout this process, you may even improve your data quality and tap in to new insights.

Excel, Power BI, Analysis Services, and PowerApps share a game-changing data connectivity and transformation technology, Power Query, that empowers any person with basic Excel skills to perform and automate data importing, reshaping, and cleansing. With simple UI clicks and a unified user experience across wide variety of data sources and formats, you can resolve any data preparation challenge and become a master data wrangler.

In this book, you will tackle real data challenges and learn how to resolve them with Power Query. With more than 70 challenges and 200 exercise files in the companion content, you will import messy and disjointed tables and work your way through the creation of automated and well-structured datasets that are ready for analysis. Most of the techniques are simple to follow and can be easily reused in your own business context.

Who this book is for

This book was written to empower business users and report authors in Microsoft Excel and Power BI. The book is also relevant for SQL Server or Azure Analysis Services developers who wish to speed up their ETL development. Users who create apps using Microsoft PowerApps can also take advantage of this book to integrate complex datasets into their business logic.

Whether you are in charge of repetitive data preparation tasks in Excel or you develop Power BI reports for your corporation, this book is for you. Analysts, business intelligence specialists, and ETL developers can boost their productivity by learning the techniques in this book. As Power Query technology has become the primary data stack in Excel, and as Power BI adoption has been tremendously accelerating, this book will help you pave the way in your company and make a bigger impact.

The book was written to empower all Power Query users. Whether you are a new, moderate, or advanced user, you will find useful techniques that will help you move to the next level.

Assumptions

Prior knowledge of Excel or Power BI is expected. While any Excel user can benefit from this book, you would gain much more from it if you meet one of the following criteria. (Note that meeting a single criterion is sufficient.)

- You frequently copy and paste data into Excel from the same sources and often need to clean that data
- You build reports in Excel or Power BI that are connected to external sources, and wish to improve them
- You are familiar with PivotTables in Excel
- You are familiar with Power Pivot in Excel and wish to simplify your data models
- You are familiar with Power Query and want to move to the next level
- You develop business applications using PowerApps and need to connect to data sources with messy datasets
- You are a developer in Analysis Services and wish to speed up your ETL development

How this book is organized

The book is organized into 14 chapters that start from generic and simpler data challenges and move on to advanced and specific scenarios to master. It is packed with hands-on exercises and step-by-step solutions that provide the necessary techniques for mastering real-life data preparation challenges and serve as a long-term learning resource, no matter how many new features will be released in Power Query in the future.

In Chapter 1, "Introduction to Power Query," you will be introduced to Power Query and gain the baseline knowledge to start the exercises that follow.

In Chapter 2, "Basic Data Preparation Challenges," you will learn how to tackle relatively basic common data challenges. If you carry out frequent data cleansing tasks at work, you will find this chapter extremely helpful. You will be introduced to the simplest techniques to automate your data cleansing duties, with simple mouse clicks and no software development skills. If you are new to Power Query, you will already start saving time by following the techniques in this chapter.

In Chapter 3, "Combining Data from Multiple Sources," you will learn how to combine disjointed datasets and append multiple tables in the Power Query Editor. You will learn how to append together multiple workbooks from a folder and combine multiple worksheets in a robust manner—so when new worksheets are added, a single refresh of the report will suffice to append the new data into your report.

In Chapter 4, "Combining Mismatched Tables," you will move to the next level and learn how to combine mismatched tables. In real-life scenarios your data is segmented and siloed, and often is not consistent in its format and structure. Learning how to normalize mismatched tables will enable you to gain new insights in strategic business scenarios.

In Chapter 5, "Preserving Context," you will learn how to extract and preserve external context in your tables and combine titles and other meta information, such as filenames and worksheet names, to enrich your appended tables.

In Chapter 6, "Unpivoting Tables," you will learn how to improve your table structure to utilize a better representation of the entities that the data represents. You will learn how the Unpivot transformation is a cornerstone in addressing badly designed tables, and harness the power of Unpivot to restructure your tables for better analysis. You will also learn how to address nested tables and why and how to ignore totals and subtotals from your source data.

In Chapter 7, "Advanced Unpivoting and Pivoting of Tables," you will continue the journey in Unpivot transformations and generalize a solution that will help you unpivot any summarized table, no matter how many levels of hierarchies you might have as rows and columns. Then, you will learn how to apply Pivot to handle multiline records. The techniques you learn in this chapter will enable you to perform a wide range of transformations and reshape overly structured datasets into a powerful and agile analytics platform.

As a report author, you will often share your reports with other authors in your team or company. In Chapter 8, "Addressing Collaboration Challenges," you will learn about basic collaboration challenges and how to resolve them using parameters and templates.

In Chapter 9, "Introduction to the Power Query M Formula Language," you will embark in a deep dive into M, the query language that can be used to customize your queries to achieve more, and reuse your transformation on a larger scale of challenges. In this chapter, you will learn the main building blocks of M—its syntax, operators, types, and a wide variety of built-in functions. If you are not an advanced user, you can skip this chapter and return later in your journey. Mastering M is not a prerequisite to becoming a master data wrangler, but the ability to modify the M formulas when needed can boost your powers significantly.

The user experience of the Power Query Editor in Excel and Power BI is extremely rewarding because it can turn your mundane, yet crucial, data preparation tasks into an automated refresh flow. Unfortunately, as you progress on your journey to master data wrangling, there are common mistakes you might be prone to making in the Power Query Editor, which will lead to the creation of vulnerable queries that will fail to refresh, or lead to incorrect results when the data changes. In Chapter 10, "From Pitfalls to Robust Queries," you will learn the common mistakes, or pitfalls, and how to avoid them by building robust queries that will not fail to refresh and will not lead to incorrect results.

In Chapter 11, "Basic Text Analytics," you will harness Power Query to gain fundamental insights into textual feeds. Many tables in your reports may already contain abundant textual columns that are often ignored in the analysis. You will learn how to apply common transformations to extract meaning from words, detect keywords, ignore common words (also known as stop words), and use Cartesian Product to apply complex text searches.

In Chapter 12, "Advanced Text Analytics: Extracting Meaning," you will progress from basic to advanced text analytics and learn how to apply language translation, sentiment analysis, and key phrase detection using Microsoft Cognitive Services. Using Power Query Web connector and a few basic M functions, you will be able to truly extract meaning from text and harness the power of artificial intelligence, without the help of data scientists or software developers.

In Chapter 13, "Social Network Analytics," you will learn how to analyze social network data and find how easy it is to connect to Facebook and gain insights into social activity and audience engagement on any brand, company, or product on Facebook. This exercise will also enable you to work on unstructured JSON datasets and practice Power Query on public datasets.

Finally, in Chapter 14, "Final Project: Combining It All Together," you will face the final challenge of the book and put all your knowledge to the test applying your new data-wrangling powers on a large-scale challenge. Apply the techniques from this book to combine dozens of worksheets from multiple workbooks, unpivot and pivot the data, and save Wide World Importers from a large-scale cyber-attack!

About the companion content

We have included this companion content to enrich your learning experience. You can download this book's companion content by following these instructions:

1. Register your book by going to www.microsoftpressstore.com and logging in or creating a new account.
2. On the Register a Product page, enter this book's ISBN (9781509307951), and click Submit.

3. Answer the challenge question as proof of book ownership.
4. On the Registered Products tab of your account page, click on the Access Bonus Content link to go to the page where your downloadable content is available.

The companion content includes the following:

- Excel workbooks and CSV files that will be used as messy and badly formatted data sources for all the exercises in the book. No need to install any external database to complete the exercises.
- Solution workbooks and Power BI reports that include the necessary queries to resolve each of the data challenges.

The following table lists the practice files that are required to perform the exercises in this book.

Chapter	File(s)
Chapter 1: Introduction to Power Query	C01E01.xlsx C01E01 - Solution.xlsx C01E01 - Solution.pbix
Chapter 2: Basic Data Preparation Challenges	C02E01.xlsx C02E01 - Solution.xlsx C02E02.xlsx C02E02 - Solution - Part 1.xlsx C02E02 - Solution - Part 2.xlsx C02E02 - Solution - Part 3.xlsx C02E02 - Solution - Part 1.pbix C02E02 - Solution - Part 2.pbix C02E02 - Solution - Part 3.pbix C02E03 - Solution.xlsx C02E03 - Solution - Part 2.xlsx C02E03 - Solution.pbix C02E03 - Solution - Part 2.pbix C02E04.xlsx C02E04 - Solution.xlsx C02E04 - Solution.pbix C02E05.xlsx C02E05 - Solution.xlsx C02E05 - Solution.pbix C02E06.xlsx C02E06 - Solution.xlsx C02E06 - Solution.pbix

Chapter	File(s)
	C02E07.xlsx C02E07 - Solution.xlsx C02E07 - Solution.pbix C02E08.xlsx C02E08 - Solution.xlsx C02E08 - Solution.pbix
Chapter 3: Combining Data from Multiple Sources	C03E01 - Accessories.xlsx C03E01 - Bikes.xlsx C03E01 - Clothing.xlsx C03E01 - Components.xlsx C03E03 - Products.zip C03E03 - Solution.xlsx C03E03 - Solution.pbix C03E04 - Year per Worksheet.xlsx C03E04 - Solution 01.xlsx C03E04 - Solution 02.xlsx C03E04 - Solution 01.pbix C03E04 - Solution 02.pbix
Chapter 4: Combining Mismatched Tables	C04E01 - Accessories.xlsx C04E01 - Bikes.xlsx C04E02 - Products.zip C04E02 - Solution.xlsx C04E02 - Solution.pbix C04E03 - Products.zip C04E03 - Solution.xlsx C04E03 - Solution.pbix C04E04 - Products.zip C04E04 - Conversion Table.xlsx C04E04 - Solution - Transpose.xlsx C04E04 - Solution - Transpose.pbix C04E05 - Solution - Unpivot.xlsx C04E05 - Solution - Unpivot.pbix C04E06 - Solution - Transpose Headers.xlsx C04E06 - Solution - Transpose Headers.pbix C04E07 - Solution - M.xlsx C04E07 - Solution - M.pbix
Chapter 5: Preserving Context	C05E01 - Accessories.xlsx C05E01 - Bikes & Accessories.xlsx C05E01 - Bikes.xlsx C05E01 - Solution.xlsx C05E01 - Solution 2.xlsx C05E01 - Solution.pbix C05E01 - Solution 2.pbix

Chapter	File(s)
	C05E02 - Bikes.xlsx C05E02 - Solution.xlsx C05E02 - Solution.pbix C05E03 - Products.zip C05E03 - Solution.xlsx C05E03 - Solution.pbix C05E04 - Products.xlsx C05E04 - Solution.xlsx C05E04 - Solution.pbix C05E05 - Products.xlsx C05E05 - Solution.xlsx C05E05 - Solution.pbix C05E06 - Products.xlsx C05E06 - Jump Start.xlsx C05E06 - Jump Start.pbix C05E06 - Solution.xlsx C05E06 - Solution.pbix
Chapter 6: Unpivoting Tables	C06E01.xlsx C06E02.xlsx C06E03.xlsx C06E03 - Wrong Solution.pbix C06E03 - Solution.xlsx C06E03 - Solution.pbix C06E04.xlsx C06E04 - Solution.xlsx C06E04 - Solution.pbix C06E05.xlsx C06E05 - Solution.xlsx C06E05 - Solution.pbix C06E06.xlsx C06E06 - Solution.xlsx C06E06 - Solution.pbix
Chapter 7: Advanced Unpivoting and Pivoting of Tables	C07E01.xlsx C07E01 - Solution.xlsx C07E01 - Solution.pbix C07E02.xlsx C07E02.pbix C07E03 - Solution.xlsx C07E03 - Solution.pbix C07E04.xlsx C07E04 - Solution.xlsx C07E04 - Solution.pbix C07E05 - Solution.xlsx C07E05 - Solution.pbix

Chapter	File(s)
Chapter 8: Addressing Collaboration Challenges	C08E01.xlsx C08E01 - Alice.xlsx C08E01 - Alice.pbix C08E01 - Solution.xlsx C08E01 - Solution.pbix C08E02 - Solution.pbix C08E02 - Solution.pbit C08E02 - Solution 2.pbit C08E03 - Solution.xlsx C08E03 - Solution 2.xlsx C08E04 - Solution.xlsx C08E04 - Solution.pbix C08E05.xlsx C08E05.pbix C08E05 - Folder.zip C08E05 - Solution.xlsx C08E05 - Solution.pbix
Chapter 9: Introduction to the Power Query M Formula Language	C09E01 – Solution.xlsx C09E01 – Solution.pbix
Chapter 10: From Pitfalls to Robust Queries	C10E01.xlsx C10E01 - Solution.xlsx C10E02 - Solution.xlsx C10E02 - Solution.pbix C10E03 - Solution.xlsx C10E03 - Solution.pbix C10E04 - Solution.xlsx C10E04 - Solution.pbix C10E05.xlsx C10E05 - Solution.xlsx C10E05 - Solution.pbix C10E06.xlsx C10E06 - Solution.xlsx C10E06 - Solution.pbix C10E06-v2.xlsx
Chapter 11: Basic Text Analytics	Keywords.txt Stop Words.txt C11E01.xlsx C11E01 - Solution.xlsx C11E01 - Solution.pbix C11E02 - Solution.xlsx C11E02 - Refresh Comparison.xlsx C11E02 - Solution.pbix

Chapter	File(s)
	C11E03 - Solution.xlsx C11E04 - Solution.xlsx C11E04 - Solution.pbix C11E05 - Solution.xlsx C11E05 - Solution.pbix C11E06 - Solution.xlsx C11E06 - Solution.pbix C11E07 - Solution.pbix
Chapter 12: Advanced Text Analytics: Extracting Meaning	C12E01 - Solution.xlsx C12E01 - Solution.pbix C12E02.xlsx C12E02 - Solution.xlsx C12E02 - Solution.pbix C12E02 - Solution.pbit C12E03 - Solution.xlsx C12E03 - Solution.pbix C12E04.xlsx C12E04.pbix C12E04 - Solution.xlsx C12E04 - Solution.pbix C12E05 - Solution.pbix C12E06 - Solution.xlsx C12E06 - Solution.pbix
Chapter 13: Social Network Analytics	C13E01 - Solution.xlsx C13E01 - Solution.pbit C13E02 - Solution.xlsx C13E02 - Solution.pbit C13E03 - Solution.xltx C13E03 - Solution.pbit C13E04 - Solution.xlsx C13E04 - Solution.pbix C13E05 - Solution.xlsx C13E05 - Solution.pbix C13E06 - Solution.xlsx C13E06 - Solution.pbix
Chapter 14: Final Project: Combining It All Together	C14E01 - Goal.xlsx C14E01.zip C14E01 - Solution.xlsx C14E01 - Solution.pbix C14E02 - Compromised.xlsx C14E02 - Solution.xlsx C14E02 - Solution.pbix

System requirements

You need the following software and hardware to build and run the code samples for this book:

- Operating System: Windows 10, Windows 8, Windows 7, Windows Server 2008 R2, or Windows Server 2012
- Software: Office 365, Excel 2016 or later versions of Excel, Power BI Desktop, Excel 2013 with Power Query Add-In, or Excel 2010 with Power Query Add-In

How to get support & provide feedback

The following sections provide information on errata, book support, feedback, and contact information.

This page intentionally left blank

CHAPTER 10

From Pitfalls to Robust Queries

Data is a precious thing and will last longer than the systems themselves.

—Tim Berners-Lee

IN THIS CHAPTER, YOU WILL

- Learn the main factors in creating weak queries and common pitfalls that lead to refresh failures and incorrect data
- Learn how awareness, best practices, and M modifications can help you prevent the pitfalls
- Learn how to avoid refresh failures due to the automatic detection and change of column types
- Learn how to avoid dangerous filtering that can lead to partial data in your reports
- Learn when reordering the position of columns is effective and how to reorder a subset of columns
- Learn how to remove and select columns and avoid future refresh errors
- Rename columns by their location in the table or by specific values
- Learn the dangers of splitting a column into columns instead of rows
- Improve the M formula when you merge multiple columns

The user experience of the Power Query Editor in Excel and Power BI is extremely rewarding, as it can turn your mundane yet crucial data preparation tasks into an automated-refresh flow. Unfortunately, as you progress on your journey to master data wrangling, you will face some common mistakes that many people make in the Power Query Editor. These mistakes can lead to the creation of vulnerable queries that will fail to refresh when the data changes. Even worse, these mistakes can lead to incorrect data in your reports. In this chapter, you will learn about these common pitfalls and how to build robust queries to avoid them.

The techniques in this chapter will help you create robust queries and think a step ahead in the never-ending battle to maintain a reporting system that will last longer than the data.

See Also I discuss the 10 pitfalls on my blog, at https://datachant.com/tag/pitfalls/. While you can explore all of the pitfalls in more details in my blog, this chapter encapsulates the main points and provides new examples, exercises, and solution sample files. The tenth pitfall, which focuses on the removal of duplicates, has already been discussed in Chapter 2, "Basic Data Preparation Challenges."

The Causes and Effects of the Pitfalls

The Power Query Editor operates on two core principles:

- It provides an easy-to-use user interface that translates your steps into programmatic transformation instructions.
- It loads a snapshot or a preview of the data to enable you to build the transformation logic.

While these principles are crucial to your success in resolving many data preparations challenges, they are also the culprits for key mistakes you may unintentionally make, which can lead to refresh failures or unexpected situations of missing or incorrect data, as illustrated in Figure 10-1.

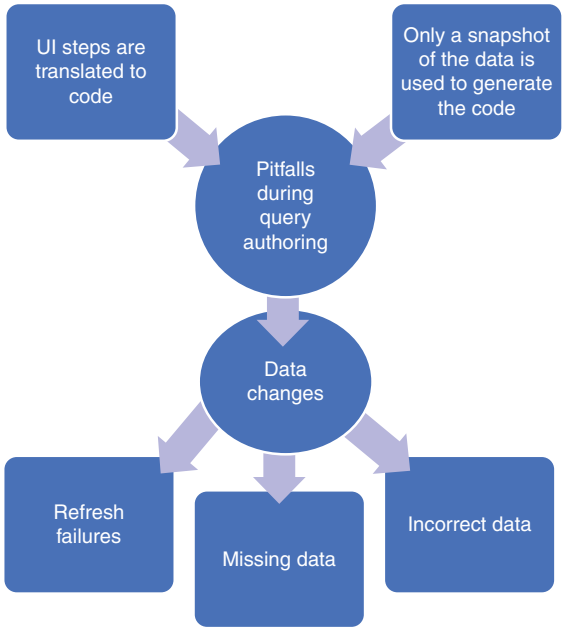


FIGURE 10-1 Several factors during the query authoring lead to refresh failures and data inconsistencies when the data changes.

Each step that you perform using the Power Query Editor user interface is added in Applied Steps, with an auto-generated formula line that plays a role in the bigger sequence of the transformation steps. As you build the transformation steps in the Power Query Editor, Power Query needs to make constant "decisions" about how to translate your UI selections into code. For example, the Unpivot Columns step described in Chapter 6, "Unpivoting Tables," is generated as an M expression to reference the unselected columns in your table instead of the selected ones. As another example of a translation decision, when you import a worksheet in an Excel file, Power Query "decides" in the M expression to reference the worksheet by name and not by its position in the workbook.

To be clear, the translation decisions that the Power Query Editor make are not random. They were designed by the Power Query team to help you create the necessary queries to reach your goals. But, while these translations might make sense during the creation of a query, they may fail you in the future. When you create a query, some of the decisions the Power Query Editor makes to generate the code rely on a preview of your data. When the data changes in a future refresh, some of the queries may no longer be robust enough to endure the changes.

In a typical case, when your data changes, a refresh error occurs, and it is an easy task for you to fix the query. But often, refresh failures keep returning, as your data may zigzag between multiple formats. Fixing the query to resist certain changes in your data today may not be able to handle other changes tomorrow.

The most common scenario for refresh failures originates in column name changes in your source tables. Most of the pitfalls in this chapter are related to incorrect handling of column names, which leads to refresh errors. You may think at this stage that it's not a big deal: Modifying your queries to reference the new column names is an easy task, and it can take only few minutes of your time to do so. You are right. It's easy to edit a query and adapt to the column changes.

But by now you have become very proficient with the Power Query Editor, and your queries may be quite complex. So, when your queries fail due to nonexistent column names, you may end up spending precious time fixing multiple steps that reference these column names. Do you think it makes sense to be locked in to a never-ending task of fixing the queries whenever a column name is missing in your data source? By now, your reports may be so important that hundreds of people in your organization could be dependent on them. A failure to refresh a report may hit you at an inconvenient time.

Finally, some of the pitfalls may not lead to refresh errors when the data changes. Instead, queries may simply ignore a portion of the data, which will lead to incorrect calculations, skewed charts, and incorrect KPIs in your report. Detecting these failures is not easy after the fact; avoiding them altogether is easier and is the focus of this chapter.

Before you look more closely at the pitfalls and how to avoid them, let's examine several common principles that you should be aware of first. These principles help minimize the pitfalls and are divided into three themes: awareness, best practices, and M modifications.

Awareness

To increase your awareness and reduce the chances of refresh failures or incorrect results in your reports, make sure you follow these recommendations after each transformation step you make in the Power Query Editor:

- Review the transformed results in the Preview pane. Make sure the data looks right. Scroll down to review more rows. Scroll right to review all columns. If you find anything odd, determine whether your step was created correctly or if it is time to consider an alternative transformation step.
- Always keep the formula bar open, and tell your colleagues to do so as well. (By default, the formula bar is hidden.) To enable the formula bar, go to the View tab and select the formula bar check box. Enabling the formula bar is fundamental to avoiding the pitfalls.
- Review the code in the formula bar. You don't need to understand M to effectively review the code. Pay close attention to the hardcoded elements in a formula, such as column names and values—especially if you provided these hardcoded elements in a dialog box or through a selection of a UI control in the relevant step. For example, if you delete Column1 in your query, you may see in the formula bar that the column was actually named Column1 with a trailing space. There is a good chance that this column name will be fixed in the future by the owner of the data source, and your step will fail. Frequent inspection of the formula bar will increase your awareness of such situations and allow you to prevent future failures.

Best Practices

The pitfalls described in this chapter can often be avoided by adhering to several best practices:

- Maintain a data contract with the colleagues who own the external data source. Communicate often and share the assumptions on your data. Work together to build change controls that help you stay prepared for planned changes. Make sure colleagues are aware of your reports, and engage with them so that they feel that they are part of your reporting project and share in the success of the project.
- Keep unrefreshed revisions of your report to track breaking changes. Your refreshed report today may fail or show unexpected results. By maintaining the old revisions of the report, you can compare the reports to identify the root causes for failures.
- As you will learn in this chapter, some pitfalls can be entirely avoided by making the right choices (for example, column removal) or by avoiding performing some unnecessary transformations (for example, changed type, column reordering).

M Modifications

There are a number of modifications you can perform at the formula level to create robust queries. While avoiding each pitfall may involve its own unique M modifications, the most common modification that you will rely on is based on the function *Table.ColumnNames*. This function allows you to modify a reference to a hardcoded column name into a dynamic reference that will not fail. You already encountered such a scenario in Exercise 3-4 in Chapter 3, "Combining Data from Multiple Sources." In that example, you appended multiple worksheets into a single table and promoted the contextual year data, 2015, as a column name for the first column. Then you renamed the column from 2015 to *Release Year*. The function *Table.ColumnNames* allowed you to rename the first column instead of renaming the 2015 column, which strengthened the query and helped you avoid a potential future refresh failure.

In this chapter, you will see various M modifications that include the *Table.ColumnNames* function to avoid the pitfalls. You will learn many other M modifications that will help you create robust queries and minimize the failures as the data changes.

Pitfall 1: Ignoring the Formula Bar

Early on in this book, as you followed along with the exercises, you many times needed to rely on information in the formula bar to better understand the transformation at hand or lightly modify the formula to achieve your goals. In Chapter 9, "Introduction to the Power Query M Formula Language," you learned the main principles of the M language, and by now you have formed some solid preferences about how much you like to use M. While many power users are eager to harness M to achieve their goals, others see M as a strange programming language that is not easy to understand.

If you are in the first group of users, the first pitfall will be easy to avoid. But if you belong to the second group, and M seems too alien for you to master, you will find this message very comforting: You do not need to learn M to resolve 90% of the data challenges you will face. Lightweight manipulations of the M formula will suffice to achieve most, if not all, of your goals. But if you use the Power Query Editor to create reports that have an impact on your organization, you will find the formula bar a strategic ally for your success.

Turn on the formula bar. Review it often. Learn how to identify the static elements, such as column names and values. These static values will be encapsulated in brackets or double quotes. See if these values make sense, and adhere to the best practices listed earlier in this chapter to prevent most of the pitfalls.

In case you jumped straight to this chapter and missed all the preceding chapters, here is how you turn on the formula bar: In the Power Query Editor, on the View tab, select the Formula Bar check box, as shown in Figure 10-2.

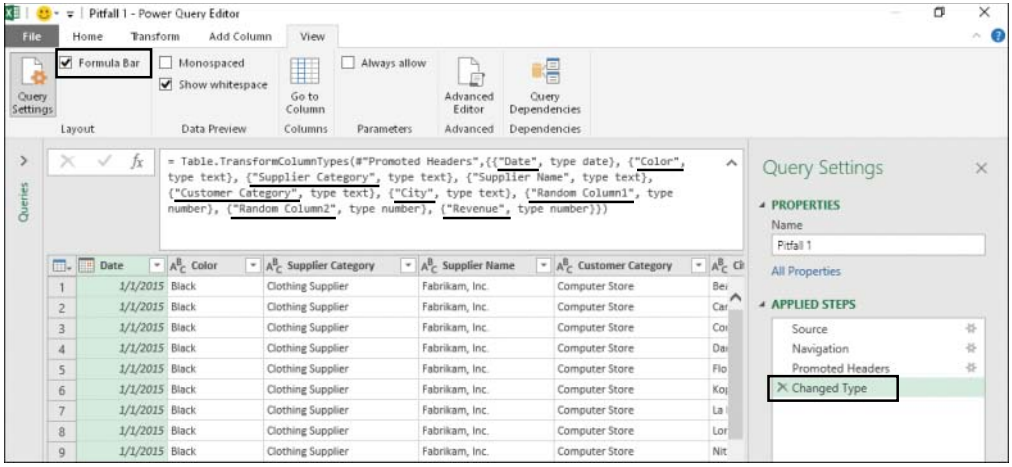


FIGURE 10-2 By enabling the formula bar, you can see how steps like the Changed Type refer to column names.

Exercise 10-1: Using the Formula Bar to Detect Static References to Column Names

This quick exercise demonstrates how to detect static column names in the formula bar and sets the stage for dealing with the second pitfall. You will use the sample workbook C10E01.xlsx, which can be downloaded from https://aka.ms/DataPwrBIPivot/downloads. This sample workbook summarizes the revenues from orders of the fictional company Wide World Importers.

1. Download the workbook C10E01.xslx and save it in the folder *C:\Data\C10*.

2. Open a blank new workbook or a new Power BI report.

   In Excel: On the Data tab, select Get Data, From File, From Workbook.

   In Power BI Desktop: On the Home tab, select Get Data, and in the Get Data dialog box, select File and then select Excel.

3. Select the file C10E01.xlsx and select Import.

4. In the Navigator dialog box, select Sheet1 and click Edit.

5. If the formula bar is not visible, go to the View tab and select the Formula Bar check box.

6. Notice that the last step in Applied Steps is Changed Type, and the following formula appears in the formula bar (refer to Figure 10-2):

   ```
   = Table.TransformColumnTypes(#"Promoted Headers",{{"Date", type date}, {"Color",
   type text}, {"Supplier Category", type text}, {"Supplier Name", type text},
   {"Customer Category", type text}, {"City", type text}, {"Random Column1", type
   number}, {"Random Column2", type number}, {"Revenue", type number}})
   ```

   Without focusing on the actual meaning of the step, you can see that there are many values here in double quotes. These values are the kind of things you should look for in the formula

bar as you create your own queries if you want to avoid many of the pitfalls. In many cases these double-quoted values represent column names or values that you entered in dialog boxes.

In this case, you can see that there are two columns, named Random Column1 and Random Column2. Let's assume that, for the sake of this exercise, these columns contain meaningless random numbers, and they are not needed in your report. In real-life scenarios, you will encounter column names that may have some arbitrary or temporary context to them. While you may keep them in your report or want to remove them, you can see in the Changed Type step that your code is already hardcoded with a reference to these columns. If these columns are not important, there is a good chance that you will no longer find them in the source table in the future, as the owner of the external data source may choose to remove them.

The following steps demonstrate the refresh error at this stage.

7. Close the Power Query Editor and load your query to your Excel worksheet or Power BI report.

8. In this step, let's pretend you are a different user now, and the owner of the data source. Open the workbook C10E01.xlsx and remove Random Column1 and Random Column2 from the worksheet. Save the workbook. You can go back to your previous role to discover the impact of this step.

9. Go back to your report and refresh it. You see the obvious refresh error:

   ```
   Expression.Error: The column 'Random Column1' of the table wasn't found.
   ```

   It is important to note that you have not explicitly specified the random columns in your report. Instead, the Power Query Editor "decided" to reference these columns. You will learn more about this in Pitfall 2, but for now, let's just look at how you can tackle this scenario through the three main principles mentioned earlier: awareness, best practices, and M modifications.

   By having the formula bar open and looking for the double-quoted values in step 6, you discovered the risk of having your formula hardcoded with unimportant column names. You can now approach the data owners and bring their awareness to the existence of such columns. You can then define a data contract that allows you to be notified if they intend in the future to remove or rename these columns, and if these columns have any hidden business value, you may want to include them in your reports.

10. Finally, by modifying the M formula, you can remove the references to the random columns by removing the following part from the formula:

    ```
    {"Random Column1", type number}, {"Random Column2", type number},
    ```

    Here is the robust revision of the formula, which ignores the random columns:

    ```
    = Table.TransformColumnTypes(#"Promoted Headers",{{"Date", type date}, {"Color",
    type text}, {"Supplier Category", type text}, {"Supplier Name", type text},
    {"Customer Category", type text}, {"City", type text}, {"Revenue", type number}})
    ```

    There are still many columns in this formula that may be removed or renamed in the future. Do you really need to reference all the column names in this line? This is the topic of the second pitfall.

Pitfall 2: Changed Types

In Exercise 10-1, you encountered the most common pitfall in the Power Query Editor and the number-one factor for refresh failures due to column name changes: the Changed Type step. In the Applied Steps pane in Figure 10-2, the last step, Changed Type, was automatically added when you loaded the worksheet to the Power Query Editor.

To avoid the second pitfall, check the Applied Steps pane for Changed Type steps that you didn't explicitly create. The main scenario in which Changed Type is prone to refresh errors is when you load text files and spreadsheets to the Power Query Editor or when you promote the first row as headers. Any minor changes to column names or column removal on the data source will lead to refresh errors.

The simplest and most common solution to avoid this pitfall is to delete the Changed Type step in Applied Steps, as shown in Figure 10-3.

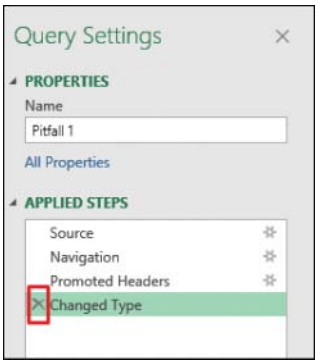


FIGURE 10-3 Delete the Changed Type step to avoid the second pitfall.

To clarify, you *do* need correct column types for some columns in your tables. Without the definition of types in some columns, you will not be able to perform arithmetic operations such as Sum and Average or apply time intelligence on these columns in Excel PivotTables or Power BI visualizations. This is why the Power Query Editor was designed to implicitly detect the types of all the columns in the table and change them for you. Without this automatic step, if you forget to explicitly change the type, you will not be able to calculate the sum of a numeric column when used in the Values pane of PivotTables or Power BI visuals.

Instead of allowing the automatic Changed Type step, delete it, and change the types manually—and consider doing it as late in the process as possible. Here are the reasons why.

Tip "The better the later" is a useful motto for changing types in Power Query. It is a good practice to change the column types in the Power Query Editor rather than rely on the automatic Changed Type—and the later in the process you explicitly change the types, the better.

If you perform type changes as the latest necessary step, you gain the following advantages:

- **Performance/refresh time:** Type changes (for example, transforming text into datetime values) require some computation effort of the M engine. If you intend to significantly narrow down the number of rows by applying filters throughout the transformation sequence, you can decrease the refresh time of the query by manually applying the change type after the last filter step.
- **Error handling:** Changing column types may lead to cell-based errors for values that cannot be converted to the new type. When these errors appear earlier in the chain of transformation steps, it is sometimes more difficult to troubleshoot the issue.

> **See Also** For an example of an error that is difficult to detect due to an early Changed Type step, read the article https://datachant.com/2017/01/11/10-common-mistakes-powerbi-powerquery-pitfall-2/.

- **Persistence:** Early changes of types may not be persistent in some scenarios. For example, when you append multiple workbooks from a folder, the type changes made on the sample query level do not propagate to the appended results.

By now, you may be wondering if it is possible to configure the Power Query Editor to stop the auto-detection and change of types so you can avoid manually deleting the steps, as shown in Figure 10-3. The answer is yes. But you need to configure it for each workbook or Power BI file, and you cannot set it for all your future reports.

To configure the Power Query Editor to stop the auto-detection and change of types, launch the Query Options dialog box in Excel (on the Data tab, Get Data, Query Options), or the Options dialog box in Power BI Desktop (on the File tab, Option and Settings, Options). Go to Current Workbook in the Query Options dialog box in Excel or the Current File in the Options dialog box in Power BI Desktop and deselect the box Automatically Detect Column Types and Headers for Unstructured Sources, as shown in Figure 10-4.

Unfortunately, there is no such a check box under Global, Data Load, so you need to repeat the process of deselecting this box for each new report.

Finally, after you delete or prevent the creation of the default Changed Type step, you should explicitly change the types for the numeric and date/time columns that are needed for your analysis, or modify the original Changed Type step as explained in step 10 in Exercise 10-1. Note that in the formula that was described in Exercise 10-1 step 10, you have several text columns that are better removed from the Changed Type step to prevent future refresh failures if these columns are renamed. These columns are marked in bold in the next formula:

```
= Table.TransformColumnTypes(#"Promoted Headers",{{"Date", type date}, {"Color",
type text}, {"Supplier Category", type text}, {"Supplier Name", type text},
{"Customer Category", type text}, {"City", type text}, {"Revenue", type number}})
```

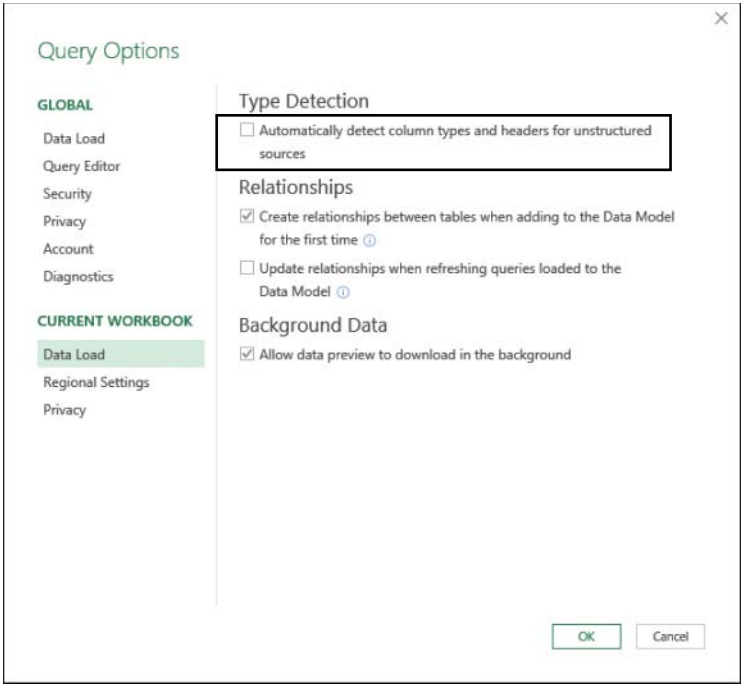


FIGURE 10-4 Disable the automatic detection and change of types.

To improve your query, you can either delete the Changed Type step in Applied Steps and manually change the types of Date and Revenue columns, or modify the M formula as follows:

```
= Table.TransformColumnTypes(#"Promoted Headers",{{"Date", type date},
{"Revenue", type number}})
```

Tip If you work with tables that contain too many columns to perform an explicit type conversion and are looking for an automatic way to detect column types without referencing all column names, you will find the M expression in the following article very useful: http://datachant.com/2018/05/14/automatic-detection-of-column-types-in-powerquery.

Pitfall 3: Dangerous Filtering

The third pitfall is one of the most dangerous pitfalls. It is almost invisible; you can easily miss it when you create the query, and you may ignore its impact when it is refreshed. The third pitfall may create a bias in your reports, and when you will detect it, it may be too late: Your biased report may lead to badly informed business decisions. It all starts with the filter control in the Power Query Editor and its derived filtering step. It is such a trivial and common step that most of our queries include it.

Exercise 10-2, Part 1: Filtering Out Black Products

Before we really examine the risk of the third pitfall, let's look at a basic scenario that demonstrates the filtering error and how to avoid it. You will use the same sample data as in Exercise 10-1. As the chief analyst of Wide World Importers, you have been asked to analyze the impact on business if you stop importing products whose color is black. You decide to filter all the black products in your queries.

1. Open a blank new workbook or a new Power BI report.

   In Excel: On the Data tab, select Get Data, From File, From Workbook.

   In Power BI Desktop: On the Home tab, select Get Data, and in the Get Data dialog box, select File and then select Excel.

2. Select the file C10E01.xlsx and select Import.

3. In the Navigator dialog box, select Sheet1 and click Edit.

4. Delete the Changed Type step in Applied Steps.

5. Ensure that the formula bar is visible.

6. Change the type of the Date column to *Date* and the type of the Revenue column to *Decimal Number*.

   (At this point, following steps 4–6 you have successfully passed Pitfalls 1 and 2.)

7. Because the data contains many more colors than Black and Blue, filter out all the black products by clicking the filter control of the Color column and deselecting Black, as shown in Figure 10-5.

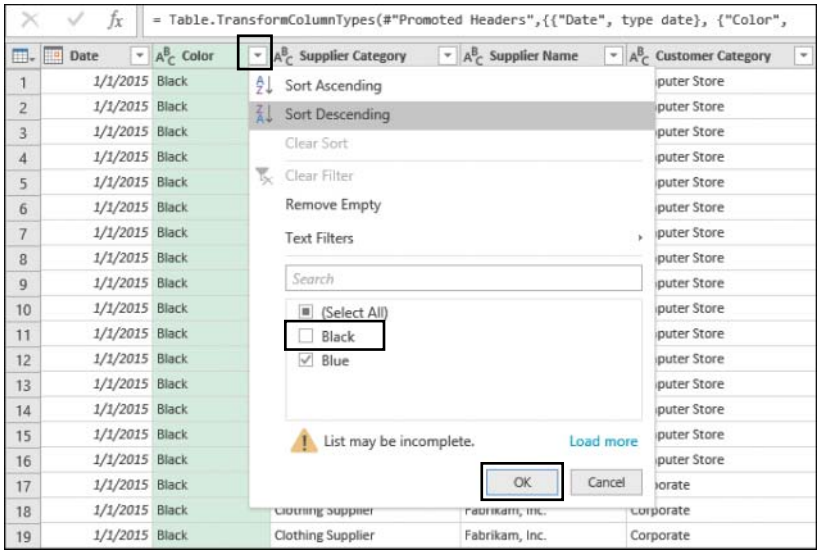


FIGURE 10-5 Deselecting the value Black leads to dangerous results.

Following this filtering step, you may expect to have as output products of all colors except Black.

8. Inspect the resulting code in the formula bar and especially notice the following code:

```
= Table.SelectRows(#"Changed Type", each ([Color] = "Blue"))
```

Fortunately, because you're aware of the first pitfall, you now inspect the formula bar more often, and you have learned to pay close attention to double-quoted values. So, the problem is obvious to you, and you easily spot it at this stage. Deselecting Black led the Power Query Editor to incorrectly assume that you selected Blue. As a result, only the blue products are filtered. If you use the output of this query without checking the formula bar, you may end up having only blue products in your analysis instead of having all the non-black products.

Fixing the formula at this stage is easy. You need to change the equal sign before Blue (= *"Blue"*) to a not-equal sign before Black (<> *"Black"*), as shown here:

```
= Table.SelectRows(#"Changed Type", each ([Color] <> "Black"))
```

You can also fix the incorrect condition without making this M modification by following steps 9 and 10.

9. Delete the Filtered Rows step from Applied Steps and select the filter control of the Color column. You will now apply the filter again in a better way.

10. In the Filter pane, select Text Filters and then select Does Not Equal. When the Filter Rows dialog box opens, enter *Black* next to Does Not Equal, and click OK to close the dialog box.

Tip If you are not comfortable with M conditions, as explained in step 8, you can use the Text Filters dialog box to define the filtering condition, as explained in steps 9 and 10, and avoid the third pitfall altogether. It is always better to use Text Filters than selecting the values in the text pane.

You can download the solution files C10E02 - Solution.xlsx and C10E02 - Solution.pbix from https://aka.ms/DataPwrBIPivot/downloads.

The Logic Behind the Filtering Condition

At this stage, you may stop trusting the filtering rationale of the Power Query Editor and might start being concerned that you have unintentionally entered the third pitfall in many of your existing queries. How often have you used the filter control as shown in the preceding example? Fortunately, in most cases, when you deselect specific values in the Filter pane, the condition is created as you expect it to be, and you do not make the mistake shown in Exercise 10-2.

Here are the two rules that the Power Query Editor follows to auto-generate the filter condition when you select values in the Filter pane:

- When you try to filter values in the Filter pane, and the number of unselected values in the Filter pane is equal to the number of selected values, the Power Query Editor generates a positive condition, using the equal sign on the selected values.
- When the number of selected values in the Filter pane is higher than the number of unselected values, the Power Query Editor always favors the negative condition, using the not-equal sign on the items that are not selected.

Let's look at these rules on a simple column with the values 1 to 6:

1. Open a new Excel file or Power BI report and launch the Power Query Editor with a blank query.
2. In the formula bar, enter the following line:

   ```
   = {1..6}
   ```

3. In List Tools, on the Transform tab, click To Table and click OK to close the To Table dialog box.
4. Select the filter control of Column1. The Filter pane opens.
5. Deselect value 1 and close the Filter pane. Notice that 1 has been correctly filtered out from the table. Also notice that the formula bar includes the following line:

   ```
   = Table.SelectRows(#"Converted to Table", each ([Column1] <> 1))
   ```

6. Click the Filter pane again, and this time deselect both values 1 and 2. The results are as you expect. Notice the following line in the formula bar:

   ```
   = Table.SelectRows(#"Converted to Table", each ([Column1] <> 1 and [Column1] <> 2))
   ```

7. Click the Filter pane again. This time deselect values 1, 2, and 3. The results look good. Only the values 4, 5, and 6 are shown in the Preview pane. But the formula bar contains the positive condition (positive in this context is a condition that uses the equal sign on the values you didn't deselect in the Filter pane):

   ```
   = Table.SelectRows(#"Converted to Table", each ([Column1] = 4 or [Column1] = 5 or
   [Column1] = 6))
   ```

   However, the correct condition is the negative condition (negative condition in this context is a condition that uses the not-equal sign on the values you deselected in the Filter pane), which should look like this:

   ```
   = Table.SelectRows(#"Converted to Table", each ([Column1] <> 1 and [Column1] <> 2
   and [Column1] <> 3))
   ```

Why is the positive condition dangerous when you deselected the values 1, 2, and 3? Imagine that in the future you also have the values 7, 8, 9, and 10 in your data, and not just 1 to 6. In step 7 you wanted to filter out values 1, 2, and 3 and keep all the rest. When you refresh the report in the future, values 7, 8, 9, and 10 will be filtered out from your report if you use the positive condition.

This example is simple, but it gives you a taste of what can happen when you work with large datasets and encounter the third pitfall: Only a subset of the values will be loaded to the pane. In Exercise 10-2, for example, you had more than two colors in the dataset, but only the black and blue values were loaded in the Filter pane. Deselecting the Black value led to an equal number of selected and not selected values, which then led to the positive condition.

One of the most dangerous and common factors that will lead you to the third pitfall is using the search box in the Filter pane. You usually use the search box when you have a long list of values in the Filter pane, and you want to zoom in on the desired values instead of scrolling down to the results. Deselecting values from the found values will *always* lead to a positive condition, and may get you in trouble.

Exercise 10-2, Part 2: Searching Values in the Filter Pane

You will use the sample workbook C10E01.xlsx, which you used in Exercise 10-1 and 10-2 part 1. The workbook can be downloaded from https://aka.ms/DataPwrBIPivot/downloads.

1. If you skipped the former exercises in this chapter, download the workbook C10E01.xslx and save it in the folder *C:\Data\C10*.

2. Open a blank new workbook or a new Power BI report.

   In Excel: On the Data tab, select Get Data, From File, From Workbook.

   In Power BI Desktop: On the Home tab, select Get Data, and in the Get Data dialog box, select File and then select Excel.

3. Select the file C10E01.xlsx and select Import.

4. In the Navigator dialog box, select Sheet1 and click Edit.

5. To avoid the second pitfall, delete the Changed Type step in Applied Steps and change the type of the Date column to *Date* and the type of the Revenue column to *Decimal Number*.

   Imagine that you want to filter out all rows whose city is *Baldwin City*.

6. Select the filter control of City column. The Filter pane opens. Since the list of cities that starts with the letter *A* is long, and you would need to scroll down to find the cities that start with the letter *B*, you may prefer using the search box in the Filter pane. Let's zoom in to all the cities that start with *Ba* to find the value Baldwin City.

   Enter the prefix *Ba* in the search box. The results include multiple cities in the pane, and luckily *Baldwin City* is the first result, so you can easily deselect it. Unfortunately, when you look at the formula bar you can see that the resulting expression has an incorrect positive condition:

   ```
   = Table.SelectRows(#"Changed Type", each ([City] = "Baraboo" or [City] = "Bayou
   Cane" or [City] = "Bazemore" or [City] = "Beaver Bay" or [City] = "Bombay Beach" or
   [City] = "Greenback" or [City] = "Wilkes-Barre"))
   ```

As a result, you will fall prey to the third pitfall, as your report will include all the cities that contains the substring *Ba* except for *Baldwin City*, and all the other cities (and there are many of them that don't start with *Ba*) will be filtered out.

7. To fix this issue, you would need to fix the formula:

```
= Table.SelectRows(#"Changed Type", each ([City] <> "Baldwin City"))
```

As you can see, the steps you made in this exercise are common, so you may experience the third pitfall quite often. The best defenses are to use the Text Filters dialog box instead of the search box in the Filter pane. But if you insist on using the search box, keep verifying the contents of the formula bar, and correct your formulas when needed.

Pitfall 4: Reordering Columns

The fourth pitfall happens when you reorder the position of columns in the Power Query Editor. In many cases, you don't really care about the exact order of your columns but would like to apply a specific order on a small subset of the columns. For example, you might want to flip between two columns or move a newly added custom column from the right end of the table to a certain position.

When you perform the reordering step, a *Table.ReorderColumns* function is generated with a reference to all the column names in your table. By referencing all columns, you weaken your query and increase the chance of refresh failures in the future, when columns are renamed or removed from the source data.

In many cases, the reordering step may be removed from Applied Steps altogether. For example, when you create a custom column that applies some calculations on another column in the table, you might want to place the new column next to its source column to verify the correctness of the code. In such cases, after you have confirmed that your code is correct, it is recommended that you delete the reorder step to reduce the chance of refresh failures.

In some cases the reordering of a table is important. If you want to control the order of the fields that will be shown in a PivotTable in Excel, or if you need to load a query to a worksheet, you can control the order of the columns. There are also some advanced scenarios that require a certain order of columns in your query. For example, in Exercise 4-4 in Chapter 4, "Combining Mismatched Tables," you relied on the reordering step to move a calculated column to the beginning of the table to transpose it and then use it as headers.

If your reorder step is crucial, try to apply it after you keep only the columns that are really needed in your report. (You'll learn more about the selection of columns when we get to the fifth pitfall.) It would not make sense to load a table with hundreds of columns, reorder them, and then keep the columns you need. Keeping only the dozen or so columns you need and reordering them will ensure that you create a much more robust query. In Exercise 10-3, you will learn how to reorder a subset of columns without referencing the complete column names, which will further improve the robustness of your query.

Exercise 10-3, Part 1: Reordering a Subset of Columns

You start this exercise with the Wide World Importers revenue dataset from Exercise 10-1 and perform a basic reordering of the columns City and Revenue.

1. Open a blank new workbook or a new Power BI report.

   In Excel: On the Data tab, select Get Data, From File, From Workbook.

   In Power BI Desktop: On the Home tab, select Get Data, and in the Get Data dialog box, select File and then select Excel.

2. Select the file C10E01.xlsx and select Import.

3. In the Navigator dialog box, select Sheet1 and click Edit.

4. Delete the Changed Type step from the Applied Steps pane.

5. Move the City column to be the second column and Revenue to be the third column, as shown in Figure 10-6.

Before:

	Date	Color	Supplier...	Supplier...	Customer Cat...	City	Random...	Random C...	Revenue
1	1/1/2015	Black	Clothing Supplier	Fabrikam, Inc.	Computer Store	Beaver Bay	0.669669664	0.716165645	4471.2
2	1/1/2015	Black	Clothing Supplier	Fabrikam, Inc.	Computer Store	Caruthers	0.791225426	0.172431372	616.4
3	1/1/2015	Black	Clothing Supplier	Fabrikam, Inc.	Computer Store	Conesus Lake	0.027167715	0.486586884	2235.6
4	1/1/2015	Black	Clothing Supplier	Fabrikam, Inc.	Computer Store	Darling	0.905738004	0.1090538	201.25
5	1/1/2015	Black	Clothing Supplier	Fabrikam, Inc.	Computer Store	Flowella	0.308817947	0.137578493	1490.4
6	1/1/2015	Black	Clothing Supplier	Fabrikam, Inc.	Computer Store	Kopperl	0.814972919	0.272847164	1107.45
7	1/1/2015	Black	Clothing Supplier	Fabrikam, Inc.	Computer Store	La Bolt	0.404748404	0.16436377	2235.6

After:

```
= Table.ReorderColumns(#"Promoted Headers",{"Date", "City", "Revenue", "Color", "Supplier Category", "Supplier
Name", "Customer Category", "Random Column1", "Random Column2"})
```

	Date	City	Revenue	Color	Supplier...	Supplier...	Customer Cat...	Random...	Random C...
1	1/1/2015	Beaver Bay	4471.2	Black	Clothing Supplier	Fabrikam, Inc.	Computer Store	0.669669664	0.716165645
2	1/1/2015	Caruthers	616.4	Black	Clothing Supplier	Fabrikam, Inc.	Computer Store	0.791225426	0.172431372
3	1/1/2015	Conesus Lake	2235.6	Black	Clothing Supplier	Fabrikam, Inc.	Computer Store	0.027167715	0.486586884
4	1/1/2015	Darling	201.25	Black	Clothing Supplier	Fabrikam, Inc.	Computer Store	0.905738004	0.1090538
5	1/1/2015	Flowella	1490.4	Black	Clothing Supplier	Fabrikam, Inc.	Computer Store	0.308817947	0.137578493
6	1/1/2015	Kopperl	1107.45	Black	Clothing Supplier	Fabrikam, Inc.	Computer Store	0.814972919	0.272847164
7	1/1/2015	La Bolt	2235.6	Black	Clothing Supplier	Fabrikam, Inc.	Computer Store	0.404748404	0.16436377
8	1/1/2015	Lompoc	281.75	Black	Clothing Supplier	Fabrikam, Inc.	Computer Store	0.534107271	0.927508108

FIGURE 10-6 Reordering the City and Revenue columns in the Power Query Editor.

6. Inspect the formula bar, which contains the following line:

```
= Table.ReorderColumns(#"Promoted Headers",{"Date", "City", "Revenue", "Color",
"Supplier Category", "Supplier Name", "Customer Category", "Random Column1",
"Random Column2"})
```

   Obviously, this function will fail to refresh if column names such as Random Column1 and Random Column2 change in the source table. Can you rewrite this formula to refer to the list of

column names above for the reordering step, but somehow only mention *"City"* and *"Revenue"* in the code? Yes, you can.

7. Remove this part from the preceding formula:

```
{"Date", "City", "Revenue", "Color", "Supplier Category", "Supplier Name",
"Customer Category", "Random Column1", "Random Column2"}
```

Replace it with the following code, which returns the same list of column names:

```
List.InsertRange(List.Difference(Table.ColumnNames(#"Promoted Headers"),
{"City", "Revenue"}), 1, {"City", "Revenue"})
```

The final formula is as follows:

```
= Table.ReorderColumns(#"Promoted Headers",List.InsertRange(List.Difference(
Table.ColumnNames(#"Promoted Headers"), {"City", "Revenue"}), 1, {"City", "Revenue"}))
```

Let's look at how, with a combination of *List.InsertRange*, *List.Difference*, and *Table.ColumnNames*, you achieved your goal and generated the same list of column names by referencing only *City* and *Revenue*.

List.InsertRange receives a list as an input and inserts another list in a certain zero-based offset. So, if you have a list A, to add *City* and *Revenue* as the second and third items in list A, you can write the following formula:

```
List.InsertRange(A, 1, {"City", "Revenue"}
```

As you can see at the preceding *Table.ReorderColumns* formula, *List.InsertRange* is used as explained above. Now, if list A can contain all the column names except City and Revenue, you will be able to apply *List.InsertRange* on A and get the desired order. This is where *List.Difference* has a role. This function accepts a list as its first argument and another list as its second, and it returns a new list with all the items in the first list that are not in the second list (the left-anti join of the two lists). So, if you apply *List.Difference* on a list and use a subset of the list as the second argument, you get as a result a new list that contains all the items from the original list except for all the items in the subset.

Therefore, if B is the list of column names, then the following function will return all the column names except City and Revenue:

```
A = List.Difference(B, {"City", "Revenue"})
```

Now when you use *Table.ColumnNames* in the following formula, instead of *B*, you can build all the pieces for the full expression:

```
B = Table.ColumnNames(#"Promoted Headers")
```

Finally, when you combine all the elements together, you reach the final formula (provided here in multiline format for better readability):

```
= Table.ReorderColumns(
    #"Promoted Headers",
    List.InsertRange(
        List.Difference(
            Table.ColumnNames(#"Promoted Headers"),
            {"City", "Revenue"}
        ),
        1,
        {"City", "Revenue"}
    )
)
```

Exercise 10-3, Part 2: The Custom Function *FnReorderSubsetOfColumns*

The technique you have been exploring in part 1 would be useful in many reports, and you might want to reuse it. You can simplify your experience by using a custom function that implements the logic you used in step 7. To see how to do this, continue the exercise with the following steps:

8. Create a blank query and paste the following code into the Advanced Editor:

```
(tbl as table, reorderedColumns as list, offset as number) as table =>
    Table.ReorderColumns(
        tbl,
        List.InsertRange(
            List.Difference(
                Table.ColumnNames(tbl),
                reorderedColumns
            ),
            offset,
            reorderedColumns
        )
    )
```

9. Rename the custom function *FnReorderSubsetOfColumns*.
10. Remove the Reordered Columns step in Applied Steps and click the fx button in the formula bar to create a new step. Then apply the following formula in the formula bar:

```
= FnReorderSubsetOfColumns(#"Promoted Headers", {"City", "Revenue"}, 1)
```

The results are the same as you obtained after step 5. You can see that invoking the function *FnReorderSubsetOfColumns* is easy. The function receives the table, the subset of the reordered column names as a list, and the zero-based index; it then performs the reordering in a robust manner, without referencing any other column names.

> **See Also** There are other ways to implement Table.ReorderColumns and avoid refresh errors. You can use a third argument to ignore missing fields or add null values for missing fields. While these techniques prevent refresh failures, they may give you unexpected results. Read more at https://datachant.com/2017/01/18/power-bi-pitfall-4/.

You can download the solution files C10E03 - Solution.xlsx and C10E03 - Solution.pbix from https://aka.ms/DataPwrBIPivot/downloads.

Pitfall 5: Removing and Selecting Columns

The fifth pitfall is related to a very common operation in the Power Query Editor: removing columns. While the removal of unnecessary columns is a crucial part of building efficient reports (fewer columns means a smaller memory footprint and smaller file size), each time you delete a column, you weaken your query and expose it to the possibility of future refresh failures.

Each time you remove a column in the Power Query Editor, you take the risk that a future refresh may fail when the removed column is missing in the external data source. Do you have a good data contract with the owner of the source table? Consider that, for the same reasons you decide to remove certain columns, the owner of the source table might do the same in the future, and determine that these columns are unimportant.

To reduce the risk of refresh failures, you can follow a simple best practice: Focus on the columns you want to keep rather than on the ones you need to remove. The Power Query Editor enables you to remove or keep columns. While it is a more direct user experience to press the Delete button on the selected columns you wish to remove, it is recommended in many cases that you select the columns you wish to keep. Exercise 10-4 demonstrates this process on Random Column1 and Random Column2 in the Wide World Importers dataset from Exercise 10-1.

Exercise 10-4: Handling the Random Columns in the Wide World Importers Table

1. Open a blank new workbook or a new Power BI report.

   In Excel: On the Data tab, select Get Data, From File, From Workbook.

   In Power BI Desktop: On the Home tab, select Get Data, and in the Get Data dialog box, select File and then select Excel.

2. Select the file C10E01.xlsx and select Import.

3. In the Navigator dialog box, select Sheet1 and click Edit.

4. Delete the Changed Type step in Applied Steps.

5. Remove the two random columns from Exercise 10-1 by selecting Random Column1 and Random Column2 and pressing Delete.

6. Notice that the formula bar includes the following line:

   ```
   = Table.RemoveColumns(#"Promoted Headers",{"Random Column1", "Random Column2"})
   ```

   Now, if the source table will no longer include one of these columns, the refresh will fail. In the next step you will remove the random columns in a different way, and improve the robustness of the query.

7. Remove the last step in Applied Steps, and then select Choose Columns on the Home tab and deselect Random Column1 and Random Column2. Close the dialog box and notice the following line in the formula bar:

```
= Table.SelectColumns(#"Promoted Headers",{"Date", "Color", "Supplier Category",
"Supplier Name", "Customer Category", "City", "Revenue"})
```

Ignoring the Missing Column

In some situations, the number of columns you need to keep may be too high, and the risk of removing a few columns may be lower than the problems associated with specifying large number of columns.

Often, it doesn't matter if you remove or select columns. You will eventually deal with external data sources that are likely to change. To help prevent refresh errors, there is an optional argument that you can use in *Table.RemoveColumns* and *Table.SelectColumns* that allows you to ignore errors instead of failing to refresh. The third argument is *MissingField.Ignore* or *MissingField.UseNull*.

MissingField.Ignore ignores the missing column, while *MissingField.UseNull* keeps the column name on errors but fills it with nulls. *MissingField.UseNull* is more practical than its sibling *MissingField.Ignore* in conjunction with *Table.SelectColumns,* as it enables you to ensure that your selected column names will be included in the end results. However, both options may expose you to errors that are difficult to detect. Thus, a refresh failure may be preferable to the unexpected results that you may incur with these arguments.

Selecting or Removing Columns Based on Their Position

In many scenarios, removing or selecting columns based on their position is more certain than referencing them by name. Using the function *Table.ColumnNames* to get the list of all column names and *List.Range* to retrieve a subset of the columns enables you to select any subset of columns based on their position.

Each one of the following formulas removes the first column in a table:

```
= Table.RemoveColumns(Source, List.First(Table.ColumnNames(Source)))
= Table.RemoveColumns(Source, Table.ColumnNames(Source){0})
```

Using *List.FirstN*, this formula removes the first two columns in the table:

```
= Table.RemoveColumns(Source, List.FirstN(Table.ColumnNames(Source), 2))
```

This formula removes the last column in the table:

```
= Table.RemoveColumns(Source, List.Last(Table.ColumnNames(Source), 1))
```

And this formula keeps the second and third column names in the table:

```
= Table.SelectColumns(Source, List.Range(Table.ColumnNames(Source), 1, 2))
```

List.Range receives a list as the first argument, a zero-based offset, and the count of items to return. You can apply the code *List.Range(Table.ColumnNames(Source), 1, 2)* to return the two column names in the Source table in offset 1, which is the second position in the list.

You can also select individual columns. The following formula is equivalent to the one above:

```
= Table.SelectColumns(Source, {Table.ColumnNames(Source){0}, Table.ColumnNames(Source){1}})
```

This code is practical when you need to select nonadjacent columns. In the case of the Wide World Importers table, this formula removes the random columns, assuming that they are always the seventh and eighth columns (offsets 6 and 7):

```
= Table.RemoveColumns(#"Promoted Headers", {Table.ColumnNames(#"Promoted Headers"){6},
Table.ColumnNames(#"Promoted Headers"){7}})
```

Selecting or Removing Columns Based on Their Names

There are countless possibilities for selecting or removing columns in M. The following two examples involve removing the random columns in a generic way. The following formula applies *List.Select* on the column names to remove columns that contain the substring *Random*:

```
= Table.RemoveColumns(#"Promoted Headers", List.Select(Table.ColumnNames(
#"Promoted Headers"), each Text.Contains(_, "Random")))
```

And here are the same results, using *Table.SelectColumns* and the negative logic (that is, you can select all the columns that don't contain *"Random"*):

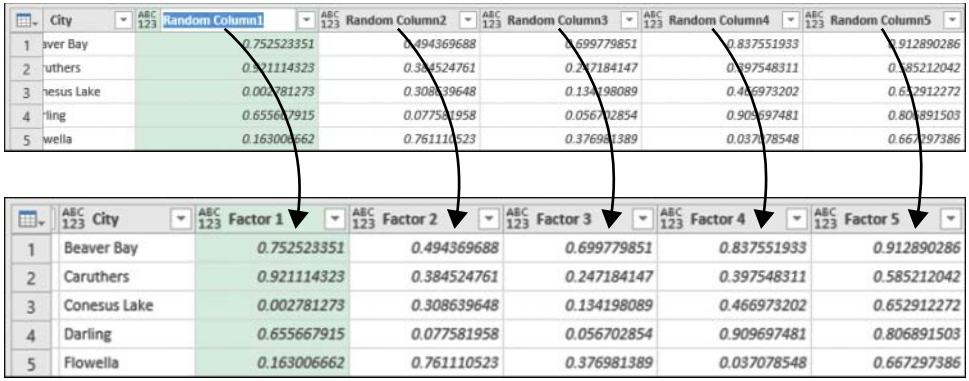
```
= Table.SelectColumns(#"Promoted Headers", List.Select(Table.ColumnNames(
#"Promoted Headers"), each not Text.Contains(_, "Random")))
```

You can download the solution files C10E04 - Solution.xlsx and C10E04 - Solution.pbix from https://aka.ms/DataPwrBIPivot/downloads.

Pitfall 6: Renaming Columns

Renaming columns is another common data preparation step. It is common to rename columns quite often to improve the user experience and expose report consumers to user-friendly names. But each time you rename a column in the Power Query Editor, you expose the query to higher chances of refresh failures in the future. Following the techniques described in the section "Pitfall 5: Removing and Selecting Columns," earlier in this chapter, you can increase the robustness of the query by modifying the formula and avoiding referencing the current column names.

Let's look at this issue on the Wide World Importers dataset and examine the different ways to rename columns. For the sake of this example, assume that you expect that all the column names that start with the prefix *Random Column* will in the future be renamed in the source table. Say that in your analysis, you were asked to rename these columns *Factor 1, Factor 2,* and so forth, as shown in Figure 10-7.



	City	Random Column1	Random Column2	Random Column3	Random Column4	Random Column5
1	ver Bay	0.752523351	0.494369688	0.699779851	0.837551933	0.912890286
2	uthers	0.921114323	0.384524761	0.247184147	0.397548311	0.585212042
3	nesus Lake	0.002781273	0.308639648	0.134198089	0.466973202	0.652912272
4	ling	0.655667915	0.077581958	0.056702854	0.909697481	0.806891503
5	wella	0.163006662	0.761110523	0.376981389	0.037078548	0.667297386

	City	Factor 1	Factor 2	Factor 3	Factor 4	Factor 5
1	Beaver Bay	0.752523351	0.494369688	0.699779851	0.837551933	0.912890286
2	Caruthers	0.921114323	0.384524761	0.247184147	0.397548311	0.585212042
3	Conesus Lake	0.002781273	0.308639648	0.134198089	0.466973202	0.652912272
4	Darling	0.655667915	0.077581958	0.056702854	0.909697481	0.806891503
5	Flowella	0.163006662	0.761110523	0.376981389	0.037078548	0.667297386

FIGURE 10-7 A column renaming challenge is to rename the *Random* columns without referencing their names.

Exercise 10-5: Renaming the *Random* Columns in the Wide World Importers Table

For this exercise, you should use the sample workbook C10E05.xlsx, which can be downloaded from https://aka.ms/DataPwrBIPivot/downloads. The sample workbook summarizes the fictional revenues of Wide World Importers.

1. Download the workbook C10E05.xslx and save it in the folder *C:\Data\C10*.

2. Open a blank new workbook or a new Power BI report.

   In Excel: On the Data tab, select Get Data, From File, From Workbook.

   In Power BI Desktop: On the Home tab, select Get Data, and in the Get Data dialog box, select File and then select Excel.

3. Select the file C10E05.xlsx and select Import.

4. In the Navigator dialog box, select Sheet1 and click Edit.

5. Delete the Changed Type step from the Applied Steps pane. You can see in the Preview pane of the Power Query Editor that you now have seven *Random* columns. In this exercise, you will learn multiple ways to rename these columns.

   Manually rename the columns Random Column1 and Random Column2 to *Factor 1* and *Factor 2*, respectively. The formula bar now includes the following line:

   ```
   = Table.RenameColumns(#"Promoted Headers",{{"Random Column1", "Factor 1"},
   {"Random Column2", "Factor 2"}})
   ```

When the source table no longer includes one of these columns, the refresh will fail.

Imagine that the data owner of the source table notifies you that he plans in the future to rename these columns, but he guarantees that the columns will always be positioned in the same order in the table. Under this assumption, you can reference these columns by their position. Here, for example, is how you can rename the first two random columns:

```
= Table.RenameColumns(#"Promoted Headers",{{Table.ColumnNames(#"Promoted Headers"){6},
"Factor 1"}, {Table.ColumnNames(#"Promoted Headers"){7}, "Factor 2"}})
```

The Custom Function *FnRenameColumnsByIndices*

To rename the columns based on their position in the table, you can write a custom function that enables you to rename a large subset of column names without the need to separately write each pair of old and new column names. In this section you will learn how to create such a function.

For example, once the function is ready, and named *FnRenameColumnsByIndices*, you can invoke it to rename the random columns in step 5 to *Factor 1* and *Factor 2*:

```
= FnRenameColumnsByIndices(#"Promoted Headers", {"Factor 1", "Factor 2"}, {6, 7})
```

The custom function approach is scalable, allowing you to rename a large number of columns at once and even import the old and new column names from an external list. The function also enables you to use a range of indices. For example, to rename all columns starting from the fourth column and ending at the tenth column to Factor 1, Factor 2,...Factor 7, without using the custom function, you could apply smart transformations.

Let's start with an ineffective method. You could write a lot of code that specifies each pair of old and new column names, as follows:

```
= Table.RenameColumns(
    #"Promoted Headers",
        {
            {Table.ColumnNames(#"Promoted Headers"){6}, "Factor 1"},
            {Table.ColumnNames(#"Promoted Headers"){7}, "Factor 2"},
            {Table.ColumnNames(#"Promoted Headers"){8}, "Factor 3"},
            {Table.ColumnNames(#"Promoted Headers"){9}, "Factor 4"},
            {Table.ColumnNames(#"Promoted Headers"){10}, "Factor 5"},
            {Table.ColumnNames(#"Promoted Headers"){11}, "Factor 6"},
            {Table.ColumnNames(#"Promoted Headers"){12}, "Factor 7"}
        }
    )
```

To avoid this complication, you can create smart renaming as follows:

```
= FnRenameColumnsByIndices(
    #"Promoted Headers",
    List.Transform(
        {1..7},
        each "Factor " & Text.From(_)
    ),
    {6..12}
)
```

In this formula, the second argument is a dynamic list that is created using *List.Transform*. Its first argument is a list of indices from 1 to 7. Its output is a transformed list of text concatenating the prefix *Factor* and the relevant index. The third argument of *FnRenameColumnsByIndices* is a list between 6 and 12 for the indices of the columns to rename.

Now, let's review the function *FnRenameColumnsByIndices*:

```
(Source as table, ColumnNamesNew as list, Indices as list) =>
let
    ColumnNamesOld = List.Transform( Indices, each Table.ColumnNames(Source){_} ),
    ZippedList = List.Zip( { ColumnNamesOld, ColumnNamesNew } ),
    #"Renamed Columns" = Table.RenameColumns( Source, ZippedList )
in
    #"Renamed Columns"
```

Let's look at this custom function, step by step. The arguments are *Source* for the table to rename, *ColumnNamesNew* for the list of new column names, and *Indices* for the list of indices in the source table. The first line inside the *let* expression receives the indices and returns the relevant column names in the Source table:

```
ColumnNamesOld = List.Transform(Indices, each Table.ColumnNames(Source){_}),
```

The next line uses *List.Zip* to create a list of nested lists. Each nested list contains two members—the old and new column names from the same index of the different lists:

```
ZippedList = List.Zip( { ColumnNamesOld, ColumnNamesNew } ),
```

For example, this *List.Zip* formula:

```
List.Zip({"a","b","c"}, {"A", "B", "C"}}
```

returns the following list of nested lists:

```
{{"a", "A"}, {"b", "B"}, {"c", "C"}}
```

The latter format is the required format for the second argument of *Table.RenameColumns*—a list of nested lists, which are pairs of old and new column names, and is used in the third line inside the *let* expression:

```
#"Renamed Columns" = Table.RenameColumns(Source, ZippedList)
```

The *Table.TransformColumnNames* Function

There is yet another way to rename the columns. Recall that in Chapter 4, you used the function *Table.TransformColumnNames* to rename all columns. You applied it to replace underscores with spaces or capitalize the headers. This function can also be used in this scenario to replace the column names using a generic rule:

```
= Table.TransformColumnNames(#"Promoted Headers", each Text.Replace(_, "Random Column",
"Factor "))
```

The advantage of this method is that it successfully renames the columns, even if the random columns will be reordered in the source table. Still, you need to be careful about the renaming logic you use to avoid renaming columns that you were not supposed to rename.

Tip In most cases, a simple rename, as you have always done, will suffice. Don't overthink it.

You can download the solution files C10E05 - Solution.xlsx and C10E05 - Solution.pbix from https://aka.ms/DataPwrBIPivot/downloads.

Pitfall 7: Splitting a Column into Columns

The seventh pitfall can be as dangerous as the filtering pitfall (Pitfall 4), as it may lead to missed data. It can happen when you apply Split Column By Delimiter into columns on delimiter-separated values with a varying range of values.

Split Column By Delimiter is typically used to achieve two types of operations:

- The basic operation enables you to split a column into multiple columns. It is often used to split date and time columns into separate date and time columns or to divide client names into first and last names. The Split Column By Delimiter dialog box enables you by default to split a column into columns.
- The advanced operation of Split Column By Delimiter enables you to split multiple comma or other delimiter-separated values into rows. This operation is extremely useful. It allows you to create a new table that associates each split value with its entity. As shown in Figure 10-8, you can create a reference table to the source table that pairs between product codes and colors to find out how many products you have by colors.

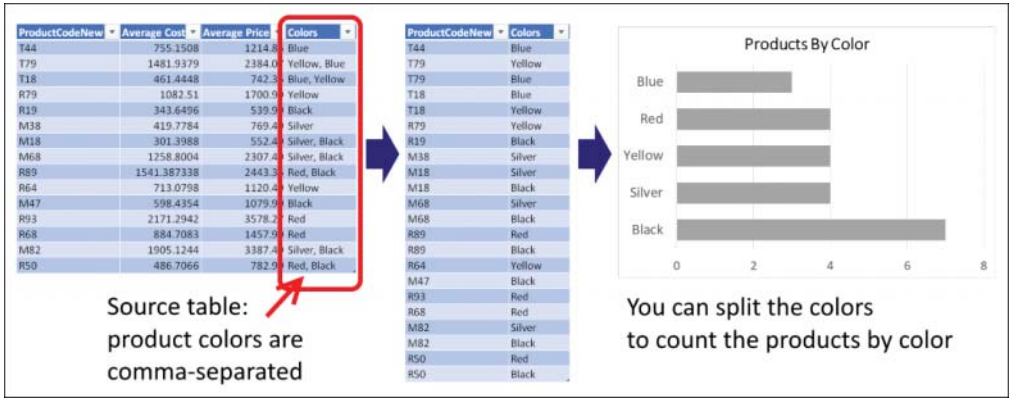


FIGURE 10-8 Split the comma-separated AdventureWorks Colors column to find how many products are released by color.

In Exercise 2-8 in Chapter 2, you worked with the AdventureWorks product table and split the Colors column to find the associations between products and colors in order to determine how many products are released by color. Recall that you split the Colors column into rows to solve that challenge. The next exercise will show you what can happen when you split a column into columns instead of into rows.

The seventh pitfall, as you will see in Exercise 10-6, happens when you split delimited-separated values and ignore the advanced (and relatively hidden) option to split the column into rows. If you don't select that option, you end up with the default option of splitting the column into columns, which exposes your queries to missing crucial information when new data is loaded to your report.

Exercise 10-6: Making an Incorrect Split

In this exercise you will repeat Exercise 2-8, but this time, you will see what happens when you use the incorrect split option, Split into Columns. This exercise demonstrates the risks and shows you how easy it is to fall prey to this pitfall.

You will use the sample workbook C10E06.xlsx, which can be downloaded from https://aka.ms/DataPwrBIPivot/downloads. The sample workbook summarizes the AdventureWorks product codes (a new variation of codes) by average cost, average price, and comma-separated colors. As the head of data science in AdventureWorks, you would like to create a report that shows how many products you have for each color (refer to Figure 10-8).

1. Download the workbook C10E06.xslx and save it in the folder *C:\Data\C10*.
2. Open a blank new workbook or a new Power BI report.

   In Excel: On the Data tab, select Get Data, From File, From Workbook.

   In Power BI Desktop: On the Home tab, select Get Data, and in the Get Data dialog box, select File and then select Excel.
3. Select the file C10E06.xlsx and select Import.
4. In the Navigator dialog box, select Products and click Edit.
5. Delete the Changed Type step from the Applied Steps pane.
6. In the Queries pane, right-click Products and select Reference. Your goal is to create a new table with a mapping between the product codes and the colors.
7. Rename the new query *Products and Colors* and keep the new query selected.
8. On the Home tab, select Choose Columns, and in the dialog box that opens, select ProductCodeNew and Colors. Click OK to close the dialog box, and all the unselected columns are removed from your Products and Colors query.
9. Select the Colors column, and on the Transform tab, select Split Column and then select By Delimiter. In the Split Column by Delimiter dialog box that opens, notice that the comma delimiter is selected by default and the option Each Occurrence of the Delimiter is selected, as shown in Figure 10-9. Unfortunately, by default, when you close the dialog box, the split is made into columns rather than rows. To split by rows, you need to expand the Advanced Options section and switch the Split Into option from Columns to Rows, as you did in Exercise 2-8.

   At this stage, keep the default split by columns option, so you can learn the implications of going through the default experience. Click OK to close the dialog box.

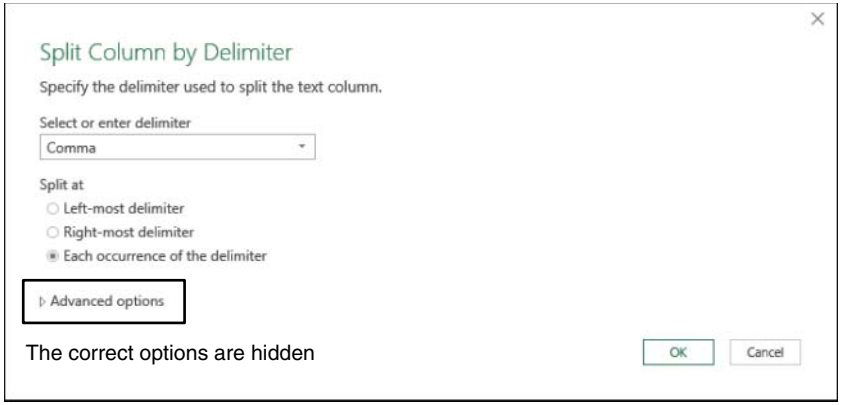


FIGURE 10-9 The Split Column by Delimiter dialog box hides the Split into Rows option under Advanced Options.

10. Notice in the Power Query Editor that the Colors column has been split into Colors.1 and Colors.2. In Applied Steps, select the Split Column By Delimiter step and review the line in the formula bar:

```
= Table.SplitColumn(Products_Table, "Colors", Splitter.SplitTextByDelimiter(",",
QuoteStyle.Csv), {"Colors.1", "Colors.2"})
```

This formula instructs the M engine to split Colors into two columns—only two. You will put it to the test soon.

11. Because your goal is to map between product codes and colors, you need to use the Unpivot transformation to have a table of code and color pairs, so select the ProductCodeNew column.

12. On the Transform tab, expand the Unpivot Columns drop-down and select Unpivot Other Columns. Colors.1 and Colors.2 are transformed into Attribute and Value columns. The latter includes the colors, now paired correctly to their corresponding product codes. At this stage, notice that some colors start with a space character. To fix this, you can apply a Trim step on the Value column or go to Applied Steps and in Split Column By Delimiter, click the settings icon and change the delimiter to Custom, and then enter a comma and a space (,). Here is the corrected formula for the Split Column By Delimiter step:

```
= Table.SplitColumn(Products_Table, "Colors", Splitter.SplitTextByDelimiter(", ",
QuoteStyle.Csv), {"Colors.1", "Colors.2"})
```

13. Remove the Attribute column and load the queries to your report to start the analysis. If you like, build a bar chart by using a PivotChart like the one shown in Figure 10-8.

14. To test what happens when a certain product code has more than two colors, save your report and open C10E06.xslx.

15. In cell D3, add two more colors to the existing two and change the Colors value from Yellow, Blue to Yellow, Blue, Black, Red. Save the workbook C10E06.xslx and close it. Go back to your exercise report and refresh it. The new colors are not included in the report.

To solve this problem, you should follow Exercise 2-4 and split the Colors column into rows. However, if for some reason you must split the column into columns and not into rows, you can scale up your

solution by increasing the number of columns to split into. Say that you know in advance that the number of colors cannot exceed 20. You can use this formula at the Split Column By Delimiter step:

```
= Table.SplitColumn(Products_Table, "Colors", Splitter.SplitTextByDelimiter(", ",
QuoteStyle.Csv), 20)
```

Replacing the hardcoded part, *{"Colors.1", "Colors.2"}*, with *20*, which is the maximum number of columns you expect, will strengthen your query and ensure that you will not miss data.

You can download the solution files C10E06 - Solution.xlsx and C10E06 - Solution.pbix from https://aka.ms/DataPwrBIPivot/downloads.

Make sure you also download the workbook C10E06-v2.xlsx, which contains the second version of the workbook C10E06.xlsx, with the modified color values. The queries in the solution files assume that you have the workbook C10E06-v2.xlsx in the folder *C:\Data\C10*.

Pitfall 8: Merging Columns

In the preceding section, "Splitting a Column into Columns," you learned how to avoid the seventh pitfall by splitting columns into rows instead of into columns. You also saw that if you simply must split a comma-separated column into columns, you can switch the second argument and call for the number of columns to generate. In this section, you will learn how to improve the query when you handle the reverse transformation: merging columns.

When you merge multiple columns into a single column, the generated formula first transforms all the numeric columns into text and then combines all the columns together. The following code is auto generated (and indented and formatted here with multiple lines for better readability) when you merge three columns in the Source table, the first of which are numeric (Numeric Column1 and Numeric Column 2) and the third of which is textual (Textual Column3):

```
#"Merged Columns" = Table.CombineColumns(
    Table.TransformColumnTypes(
        Source, {
            {"Numeric Column1", type text},
            {"Numeric Column2", type text}
        },
        "en-US"
    ),
    {"Numeric Column1", "Numeric Column2", "Textual Column3"},
    Combiner.CombineTextByDelimiter(":", QuoteStyle.None),
    "Merged"
)
```

In this formula, you can see that *Table.TransformColumnTypes* enforces the type conversion of all the numeric columns, and then merges the relevant columns. This formula exposes the query to unnecessary refresh failures. You can modify the code and scale it to merge a given list of columns, without referencing any hard-coded column names.

Say that you have the column names to merge in the ColumnsToMerge list. Here is the modified formula you can use instead of the preceding one:

```
#"Merged Columns" = Table.CombineColumns(
    Table.TransformColumnTypes(
        Source,
        List.Transform(
            ColumnsToMerge,
            each {_, type text}
        ),
        "en-US"
    ),
    ColumnsToMerge,
    Combiner.CombineTextByDelimiter(":", QuoteStyle.None),
    "Merged")
```

The main change between the two formulas is that this part of the code:

```
{
   {“Numeric Column1”, type text},
   {“Numeric Column2”, type text}
}
```

is replaced with a *List.Transform* function that generates the same code, but without referencing the column names:

```
List.Transform(
    ColumnsToMerge,
    each {_, type text}
)
```

This code iterates over each column name in ColumnsToMerge and transforms it into a list of column names and a text type: *{_, type text}*.

While you may not commonly need such a function, this example is an important demonstration of the use of list functions to write scalable and robust versions of your auto-generated formulas.

More Pitfalls and Techniques for Robust Queries

I mentioned earlier that I talk about 10 pitfalls in my blog series at DataChant, and you can see that this chapter does not cover the ninth and tenth pitfalls. This section mentions them briefly, and for more information, you can visit my blog.

The ninth pitfall has to do with expanding table columns. Expand Table Columns is a common transformation that you apply when you combine multiple files from a folder, merge between two tables, or work with unstructured datasets such as JSON.

When you expand table columns, you are required to select the columns to expand. As a result, the Power Query Editor auto-generates a formula with hardcoded column names, and new column names can be missed. To learn more about the Expand Table Columns transformation and how to avoid missing new columns, go to https://datachant.com/2017/02/07/power-bi-pitfall-9/.

The tenth pitfall focuses on the removal of duplicates, and it is covered earlier in this book, in Exercise 2-7, Chapter 2.

Summary

In this chapter you have improved your data-wrangling skills by focusing on long-term thinking. It looks at how you can make your queries last longer, without needing to constantly edit them as the data changes. You have learned about the most common pitfalls that lead to refresh failures and incorrect data—as well as three important themes for avoiding them: awareness, best practices, and M modifications. Having the formula bar active and reviewing it frequently can help you detect unexpected hardcoded double-quoted values, or incorrect conditions.

In this chapter you have also learned some best practices that can reduce failures. Having a formal or informal data contract with your colleagues who own the data sources can help you build the correct assumptions on the data and make good decisions about resolving your data preparation challenges.

You have also learned how to apply lightweight modifications to M formulas to strengthen your queries. To conclude this chapter, Table 10-1 lists the main pitfalls, their impacts, and how to avoid them.

TABLE 10-1 The Main Pitfalls, Their Impacts, and How to Avoid Them

Pitfall Number and Main Feature	Impact	How to Avoid the Pitfall
Pitfall 1: Ignoring the formula bar	Lack of awareness of the potential failures of the auto-generated M code	Activate the formula bar and keep reviewing its code. Look for double-quoted values to verify that they are consistent with your user interface selections.
Pitfall 2: Changed types	High probability for future refresh errors	Delete the Changed Type step. Turn off automatic detection of types in the Query Options dialog box for each workbook or Power BI report. Change the types manually on numeric/date/time columns. The later you change the types, the better.
Pitfall 3: Dangerous filtering	High probability for incorrect data	Avoid using the values in the Filter pane. Use the Filter Rows dialog box instead or audit the formula bar to ensure that your filtering logic was created as expected.
Pitfall 4: Reordering columns	High probability for future refresh errors	Avoid reordering columns if possible. Use M function to reorder only a subset of the columns.
Pitfall 5: Removing and selecting columns	Medium probability for future refresh errors	If you have a small number of columns to keep, use Choose Columns instead of Remove Columns. In many cases the columns you choose to keep are more likely to stay in your source data. Use M to remove or select columns by their position in the table.
Pitfall 6: Renaming columns	Medium probability for future refresh errors	Use the function *Table.TransformColumnNames* in M to rename columns by their values, or use *Table.ColumnNames* with zero-based indices to rename by position.
Pitfall 7: Splitting a column into columns	High probability for missing data	Avoid using Split into Columns. Use the advanced option and Split into Rows instead. If Split into Columns is required, use the maximum number of columns as the second argument.
Pitfall 8: Merging columns	Medium probability for refresh errors	All the columns will be referenced in the code. Modify the M code if needed to merge a large number of columns or if the list of column names to merge is dynamic.
Pitfall 9: Expanding table column	Medium probability for missing data	See https://datachant.com/2017/02/07/power-bi-pitfall-9/.
Pitfall 10: Merging duplicates	High probability for refresh errors and failures to create relationship between lookup and fact tables	See Chapter 2, Exercise 2-7, in the section "When a Relationship Fails," and make sure you lowercase/uppercase and trim the values before you remove duplicates in the column that is in the relationship in lookup tables.

Index

Numerals and Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X-Y-Z