

Mastering Power Query in Power BI and Excel

Learning real-world Power Query and M Techniques for a better data analysis



Author: Reza Rad, Leila Etaati

August 2021

Edition one

Agenda

PUBLISHED BY

RADACAD Systems Limited

[web address\[https://radacad.com/\]](https://radacad.com/)



24 Riverhaven Drive,

Wade Heads,

Whangaparaoa 0932

New Zealand

Copyright © 2021 by RADACAD, Reza Rad, Leila Etaati

All rights reserved. No part of the contents of this book may be reproduced or transmitted in any form or by any means without the publisher's written permission.

Agenda

| | |
|--|------------|
| AGENDA | 3 |
| FOREWORD | 6 |
| ABOUT THE AUTHORS | 7 |
| INTRODUCTION: FROM THE AUTHORS | 11 |
| PART 1: COMBINING DATA TABLES | 14 |
| CHAPTER 1: APPEND VS. MERGE IN POWER BI AND POWER QUERY | 15 |
| CHAPTER 2: REFERENCE VS. DUPLICATE IN POWER BI; POWER QUERY BACK TO BASICS | 23 |
| CHAPTER 3: CHOOSE THE RIGHT MERGE JOIN TYPE IN POWER BI | 33 |
| CHAPTER 4: RELATIONSHIP IN POWER BI WITH MULTIPLE COLUMNS | 44 |
| CHAPTER 5: COMBINING DIMENSION TABLES IN POWER BI USING POWER QUERY | 49 |
| CHAPTER 6: CREATING A SHARED DIMENSION IN POWER BI USING POWER QUERY | 64 |
| CHAPTER 7: CARTESIAN PRODUCT IN POWER QUERY: MULTIPLY ALL SETS OF ALL PAIRS IN POWER BI | 79 |
| CHAPTER 8: FIND MISMATCH ROWS WITH POWER QUERY IN POWER BI | 84 |
| CHAPTER 9: BE CAREFUL WHEN MERGING ON TEXT FIELDS IN POWER BI USING POWER QUERY | 89 |
| CHAPTER 10: DATES BETWEEN MERGE JOIN IN POWER QUERY | 95 |
| CHAPTER 11: AGE BANDING IN POWER BI USING POWER QUERY – MERGE QUERIES BASED ON BETWEEN | 102 |
| PART 2: GROUPING AND AGGREGATION | 111 |
| CHAPTER 12: GROUPING IN POWER QUERY; GETTING THE LAST ITEM IN EACH GROUP | 112 |
| CHAPTER 13: COUNT OF UNIQUE VALUES (DISTINCTCOUNT) IN POWER BI THROUGH POWER QUERY GROUP BY TRANSFORMATION | 122 |
| CHAPTER 14: CREATE ROW NUMBER FOR EACH GROUP IN POWER BI USING POWER QUERY | 127 |
| PART 3: FUZZY OPERATIONS | 133 |

Agenda

| | |
|---|-------------------|
| CHAPTER 15: FUZZY MATCHING IN POWER BI AND POWER QUERY; MATCH BASED ON SIMILARITY THRESHOLD | 134 |
| CHAPTER 16: FUZZY GROUPING IN POWER BI USING POWER QUERY | 141 |
| CHAPTER 17: FUZZY CLUSTERING IN POWER BI USING POWER QUERY: FINDING SIMILAR VALUES | 148 |
| <u>PART 4: PARAMETERS AND CUSTOM FUNCTIONS</u> | <u>155</u> |
| CHAPTER 18: POWER QUERY PARAMETERS VS. WHAT IF PARAMETERS: POWER BI IMPLEMENTATION USE CASES | 156 |
| CHAPTER 19: CUSTOM FUNCTIONS MADE EASY IN POWER BI DESKTOP | 160 |
| CHAPTER 20: CHANGE THE SOURCE OF POWER BI DATASETS DYNAMICALLY USING POWER QUERY PARAMETERS | 178 |
| <u>PART 5: PERFORMANCE TUNING</u> | <u>186</u> |
| CHAPTER 21: WATCH YOUR STEPS! POWER QUERY PERFORMANCE CAUTION FOR POWER BI | 187 |
| CHAPTER 22: NOT FOLDING; THE BLACK HOLE OF POWER QUERY PERFORMANCE | 191 |
| CHAPTER 23: PERFORMANCE TIP FOR POWER BI; ENABLE LOAD SUCKS MEMORY UP | 203 |
| CHAPTER 24: ALL YOU NEED TO KNOW ABOUT THE INCREMENTAL REFRESH IN POWER BI: LOAD CHANGES ONLY | 216 |
| <u>PART 6: ERROR HANDLING, EXCEPTION REPORTING, AND DATA PROFILING</u> | <u>225</u> |
| CHAPTER 25: EXCEPTION REPORTING IN POWER BI: CATCH THE ERROR ROWS IN POWER QUERY | 226 |
| CHAPTER 26: EXCEPTION REPORTING IN POWER QUERY AND POWER BI; PART 2 CATCHING ERROR ROWS FOR ALL COLUMNS IN THE TABLE | 238 |
| CHAPTER 27: KNOW YOUR DATA BETTER BY COLUMN PROFILING IN POWER BI USING POWER QUERY | 253 |
| CHAPTER 28: CREATE A PROFILING REPORT IN POWER BI: GIVE THE END USER INFORMATION ABOUT THE DATA | 259 |
| CHAPTER 29: GET THE ERROR COUNT WITH THE PROFILING DATA OF POWER BI DATA TABLE USING POWER QUERY | 265 |
| <u>PART 7: ADVANCED ANALYTICS USING POWER QUERY</u> | <u>272</u> |
| CHAPTER 30: TIME SERIES ANOMALY DETECTION IN POWER BI USING COGNITIVE SERVICE AND POWER QUERY | 273 |
| CHAPTER 31: POWER BI AND AZURE ML MAKE THEM WORK WITH POWER QUERY | 281 |

| | |
|--|------------|
| CHAPTER 32: TEXT ENTITY EXTRACTION IN POWER BI, TEXT ANALYTICS SERVICE | 294 |
| PART 8: POWER QUERY FORMULA LANGUAGE: M | 301 |
| CHAPTER 33: POWER BI QUICK TIP: THE FORMULA BAR IN POWER QUERY | 302 |
| CHAPTER 34: BASICS OF M: POWER QUERY FORMULA LANGUAGE | 304 |
| CHAPTER 35: M OR DAX? THAT IS THE QUESTION! | 313 |
| CHAPTER 36: BASICS OF VALUE STRUCTURES IN M – POWER QUERY FORMULA LANGUAGE | 319 |
| CHAPTER 37: WRITING CUSTOM FUNCTIONS IN POWER QUERY M | 331 |
| CHAPTER 38: DAY NUMBER OF YEAR, POWER QUERY CUSTOM FUNCTION | 340 |
| CHAPTER 39: POWER QUERY FUNCTION THAT RETURNS MULTIPLE VALUES | 349 |
| CHAPTER 40: FIBONACCI SEQUENCE: UNDERSTANDING THE POWER QUERY RECURSIVE FUNCTION FOR POWER BI | 354 |
| CHAPTER 41: POWER QUERY LIBRARY OF FUNCTIONS; SHARED KEYWORD | 362 |
| CHAPTER 42: GET A LIST OF QUERIES IN POWER BI | 369 |
| PART 9: TABLE AND LIST FUNCTIONS | 374 |
| CHAPTER 43: PRE CONCATENATE LIST OF VALUES IN POWER BI USING POWER QUERY | 375 |
| CHAPTER 44: SEARCH FOR A COLUMN IN THE ENTIRE DATABASE WITH TABLE.COLUMNNAMES IN POWER QUERY AND POWER BI | 382 |
| CHAPTER 45: LIST.ACCUMULATE HIDDEN GEM OF POWER QUERY LIST FUNCTIONS IN POWER BI | 390 |
| CHAPTER 46: REMOVE COLUMNS WITH SPECIFIC PATTERNS NAME IN POWER BI USING POWER QUERY | 399 |
| CHAPTER 47: EXPORT DATA FROM POWER QUERY TO LOCAL MACHINE OR SQL SERVER USING R SCRIPTS | 404 |
| CHAPTER 48: GENERATE RANDOM LIST OF NUMBERS IN POWER BI DATASET USING POWER QUERY | 409 |
| BOOK WRAP UP | 415 |
| OTHER BOOKS FROM REZA RAD AND LEILA ETAATI | 416 |

Foreword

In this era of Digital Transformation, empowering business users to make decisions informed by data trends quickly has become critical for every organization. However, data is never in the right shape to derive value, and it often requires significant effort to make it such, to the point where roughly 80% of the time spent in this end-to-end process is focused on Data Preparation tasks.

Power Query provides a Low-Code Self-Service Data Preparation experience natively integrated within several Microsoft products, empowering business users to get data into the right shape to enable decision-making quickly.

Leila and Reza are world-class Power Query experts with deep knowledge about the product. They're also very active across the Community, making them extremely knowledgeable about key customer scenarios, best practices, and anti-patterns. Not only they are experts in Power Query, but they also master the art of teaching, being capable of turning the most complex concepts into simple explanations that are easy to follow and understand for users, ranging from novice to advanced.

Whether you are new to Power Query or consider yourself a Power Query ninja, this book has something for you. It will help you with your journey of mastering the Data Preparation techniques and functionality exposed in the product – Definitely a must-read (and fun!) book for everyone aspiring to get the most value out of their data.

Miguel Llopis is the Product Manager Lead for Power Query, M & Dataflows at Microsoft. Having worked on Power Query for the past ten years, Miguel is not only responsible for many of the quirks in the product but also extremely committed and passionate about turning those around into delightful and magical Low-Code Data Preparation experiences that can make the World a better place (to prep data).



About the Authors

Reza Rad

Reza Rad is a [Microsoft Regional Director](https://rd.microsoft.com/en-us/reza-rad), an Author, Trainer, Speaker, and Consultant. He has a BSc in Computer engineering; he has more than 20 years of experience in data analysis, BI, databases, programming, and development,



mainly on Microsoft technologies. He is a [Microsoft Data Platform MVP](https://mvp.microsoft.com/en-us/PublicProfile/4030647?fullName=Reza%20%20Rad) for 11 straight years (from 2011 till now) for his dedication to Microsoft BI. Reza is an active blogger and co-founder of [RADACAD](https://radacad.com/). Reza is also co-founder and co-organizer of the [Difinity](http://difinity.co.nz/) conference in New Zealand, and the [Power BI Summit](https://www.globalpowerbisummit.com/) (the biggest Power BI conference)

- CEO at RADACAD
- Chairman and Director at Power BI Summit
- Chairman and Director at Difinity conference (Since 2017)
- Microsoft Regional Director (Since 2018)
- Microsoft Data Platform MVP (Since 2011)
- Leader of Data, Insights, and Power BI user group in Auckland, New Zealand (Since Microsoft Fast Track Recognized Solution Architect – Power Platform)
- Power BI All-Star award winner
- Microsoft Power Platform Fast Track Solution Architect – Power BI
- Dynamic Communities Emerald award winner
- Author of more than ten books on BI, analytics, and Power BI
- Speaker at many conferences such as Ignite, Microsoft Business Applications Summit, PASS, etc.
- Blogger and content creator
- Consultant and trainer
- Microsoft Certified trainer
- Microsoft Certified professional

About the Authors

His articles on different aspects of technologies, mainly on BI, can be found on his blog: <https://radacad.com/blog>[\[https://radacad.com/blog\]](https://radacad.com/blog).

He wrote some books on Microsoft BI and also is writing some others. He was also an active member on online technical forums such as MSDN and Experts-Exchange, was a moderator of MSDN SQL Server forums, and is an MCP, MCSE, and MCITP of BI. He is the leader of [the Power BI and Analytics users group](https://www.meetup.com/New-Zealand-Business-Intelligence-User-Group/)[\[https://www.meetup.com/New-Zealand-Business-Intelligence-User-Group/\]](https://www.meetup.com/New-Zealand-Business-Intelligence-User-Group/). He is also the author of the very popular book [Power BI from Rookie to Rock Star](https://radacad.com/online-book-power-bi-from-rookie-to-rockstar)[\[https://radacad.com/online-book-power-bi-from-rookie-to-rockstar\]](https://radacad.com/online-book-power-bi-from-rookie-to-rockstar), which is free with more than 1700 pages of content and the [Power BI Pro Architecture](https://www.apress.com/gp/book/9781484240144)[\[https://www.apress.com/gp/book/9781484240144\]](https://www.apress.com/gp/book/9781484240144). He also wrote a book on [Row-Level Security in Power BI](https://www.amazon.com/gp/product/B082SFR2J4)[\[https://www.amazon.com/gp/product/B082SFR2J4\]](https://www.amazon.com/gp/product/B082SFR2J4), [Basics of Power BI modeling](https://www.amazon.com/gp/product/B08HWNZ7GC)[\[https://www.amazon.com/gp/product/B08HWNZ7GC\]](https://www.amazon.com/gp/product/B08HWNZ7GC) and [Power BI DAX Simplified](https://www.amazon.com/gp/product/B099SBN1XP)[\[https://www.amazon.com/gp/product/B099SBN1XP\]](https://www.amazon.com/gp/product/B099SBN1XP), which are valuable assets for Power BI users.

He is an International Speaker in Microsoft Ignite, Microsoft Business Applications Summit, Data Insight Summit, PASS Summit, SQL Saturday, and user groups. And He is a Microsoft Certified Trainer.

LinkedIn: [Reza Rad](https://www.linkedin.com/in/rezarad/)[\[https://www.linkedin.com/in/rezarad/\]](https://www.linkedin.com/in/rezarad/)

Twitter: [Reza Rad](https://twitter.com/Rad_Reza)[\[https://twitter.com/Rad_Reza\]](https://twitter.com/Rad_Reza)



Dr. Leila Etaati



Dr. Leila Etaati is the first [Microsoft AI MVP](https://mvp.microsoft.com/en-us/PublicProfile/5002206?fullName=Leila%20Etaati) in New Zealand and Australia. She has a Ph.D. in Information Systems from the University Of Auckland. And she has been working with information systems since 20 years ago. She is the Co-director and data scientist in RADACAD. She is the co-organizer of the Power BI and Analytics user group (meetup) in Auckland with more than 2500 members. She is the co-director of conferences such as the [Difinity Conference](http://difinity.co.nz/) (2017 till now) and the [Power BI Summit](https://www.globalpowerbisummit.com/) (the biggest Power BI conference). She is a Data Scientist, BI Consultant, Trainer, and international Speaker. She is one of the very few AI and Data Platform Dual Microsoft MVPs in the world. Leila is an active Technical [Microsoft AI blogger](http://radacad.com/author/leila) for RADACAD.

- CEO at RADACAD
- Chairman and Director at Power BI Summit
- Chairman and Director at Difinity conference (Since 2017)
- Microsoft Data Platform MVP (Since 2016)
- Microsoft AI MVP (Since 2017)
- Leader of Data, Insights, and Power BI user group in Auckland, New Zealand (Since 2016)
- Regional Leader of Global AI Bootcamp
- Leader of Artificial Intelligence user group in Auckland (Since 2018)
- Power BI All-Star award winner
- Dynamic Communities Emerald award winner
- Author of more books on Data Science, AI, analytics, and Power BI
- Speaker at many conferences such as Ignite, Microsoft Business Applications Summit, PASS, etc.

About the Authors

- Blogger and content creator
- Consultant and trainer
- Microsoft Certified trainer
- Microsoft Certified professional

She worked in Industries including banking, financial, power and utility, manufacturing. She is an invited lecturer at the University of Auckland. And she is a trainer in Data Analytics, AI, and Data Science courses in RADACAD.

Leila speaks in international SQL Server and BI conferences such as Microsoft Ignite, PASS Summit, PASS Business Analytics, PASS Rally, and many SQL Saturdays in the USA, Europe, Australia, and New Zealand on Machine Learning and Analytics topics.

LinkedIn: [Leila Etaati](https://www.linkedin.com/in/leila-etaati/)

Twitter: [Leila Etaati](https://twitter.com/leila_etaati)



Introduction: from the authors

Any data analytics solution requires data population and preparation. With the rise of data analytics solutions these years, the need for this data preparation becomes even more essential. Power BI is a helpful data analytics tool that is used worldwide by many users.

As a Power BI (or Microsoft BI) developer, it is essential to learn how to prepare the data in the right shape and format needed. You need to learn how to clean the data and build it in a structure that can be modeled easily and used high performant for visualization.

Data preparation and transformation is the backend work. If you consider building a BI system as going to a restaurant and ordering food. The visualization is the food you see on the table nicely presented. The quality, the taste, and everything else come from the hard work in the kitchen. The part that you don't see or the backend in the world of Power BI is Power Query.

You may already be familiar with other data preparation and transformation technologies, such as T-SQL, SSIS, Azure Data Factory, Informatica, etc. Power Query is a data transformation engine capable of preparing the data in the format you need. The good news is that to learn Power Query; you don't need to know programming. Power Query is for citizen data engineers. However, this doesn't mean that Power Query is not capable of performing advanced transformation.

Power Query exists in many Microsoft tools and services such as Power BI, Excel, Dataflows, Power Automate, Azure Data Factory, etc. Through the years, this engine became more powerful. These days, we can say this is essential learning for anyone who wants to do data analysis with Microsoft technology to learn Power Query and master it.

Unfortunately, because Power Query and data preparation is the kitchen work of the BI system, many Power BI users skip the learning of it. They become aware of it somewhere along their BI project. Once they get familiar with it, they realize there are tons of things they could have implemented easier, faster, and in a much more maintainable way using Power Query. In other words, they learn mastering Power Query is the key skill toward mastering Power BI.

We have been working with Power Query since the very early release of that in 2013, named Data Explorer, and wrote blog articles and published videos about it. The

number of articles we published under this subject easily exceeds hundreds. Through those articles, some of the fundamentals and key learnings of Power Query are explained. We thought it is good to compile some of them in a book series.

A good analytics solution combines a good data model, good data preparation, and good analytics and calculations. Reza has written another book about the Basics of modeling in Power BI and a book on Power BI DAX Simplified. This book is covering the data preparation and transformations aspects of it.

This book series is for you if you are building a Power BI solution. Even if you are just visualizing the data, preparation and transformations are an essential part of analytics. You do need to have the cleaned and prepared data ready before visualizing it.

This book is compiled into a series of two books, which will be followed by a third book later;

- Getting started with Power Query in Power BI and Excel (already available to be purchased separately)
- Mastering Power Query in Power BI and Excel (This book)
- Power Query dataflows (will be published later)

This book deeps dive into real-world challenges of data transformation. It starts with combining data sources and continues with aggregations and fuzzy operations. The book covers advanced usage of Power Query in scenarios such as error handling and exception reports, custom functions and parameters, advanced analytics, and some helpful table and list functions. The book continues with some performance tuning tips, and it also explains the Power Query formula language (M) and the structure of it, and how to use it in practical solutions.

We do recommend you to read the book Getting Started with Power Query in Power BI and Excel before reading this book. This book assumes that you know the basics and fundamentals of Power Query (As it is explained in book 1 of this series)

This is not a book to explain every single function or aspect of Power Query. The approach in this book is to have practical examples. Every chapter is based on real-world examples of using a combination of functions to solve a challenge. You can start from any chapter and finish at any chapter. The order of chapters suggested in this book is just a guideline to help you have a smooth flow of topics. Each chapter can be read without needing other chapters. Examples of this book are designed in a way that you can use the learning straight away in your Power BI file.

Although this book is written for Power BI and all the examples are presented using the Power BI. However, the examples can be easily applied to Excel, Dataflows, and other tools and services using Power Query.

Most of the chapters of this book come from our blog articles and videos, and countless comments from my readers are applied to it. If you ever feel you have a question that you can't get through this book, please contact us directly.

Download the files and codes for this book from [here](https://radacad.com/books/files/Mastering Power Query in Power BI and Excel.zip)
[\[https://radacad.com/books/files/Mastering Power Query in Power BI and Excel.zip\]](https://radacad.com/books/files/Mastering Power Query in Power BI and Excel.zip)

Part 1: Combining data tables

Chapter 1: Append vs. Merge in Power BI and Power Query



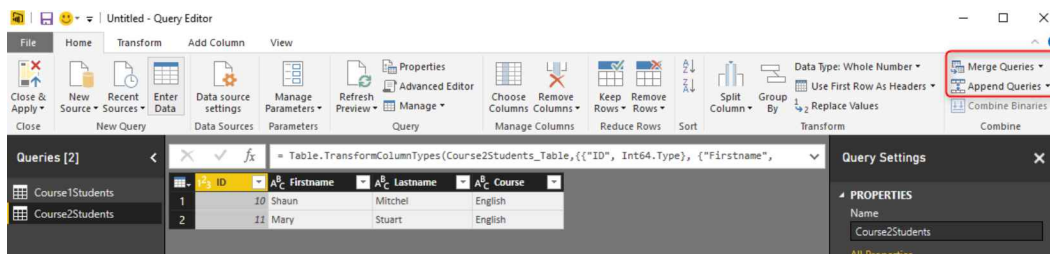
Combining two queries in Power Query or Power BI is one of the most basic and essential tasks you would need to do in most data preparation scenarios. There are two types of combining queries; Merge, and Append. Database developers easily understand the difference, but the majority of Power BI users are not developers. This chapter will explain the difference between Merge and Append and situations that you should use each.

Why Combine Queries?

This might be the first question that comes into your mind; Why should I combine queries? The answer is that; You can do most of the things you want in a single query. However, it will be very complicated with hundreds of steps very quickly. On the other hand, your queries might be used in different places. For example, one of them might be used as a table in the Power BI model and also play data preparation for another query. Combining queries is a big help in writing better and simpler queries. I'll show you some examples of combining queries.

The result of a combined operation on one or more queries will be only one query. You can find Append or Merge in the Combine Queries section of the Query Editor in Power BI or Excel.

Part 1: Combining data tables



Append

Append means results of two (or more) queries (which are tables themselves) will be combined into one query in this way:

- Rows will be appended after each other. (for example, appending a query with 50 rows with another query with 100 rows will return a result set of 150 rows)
- Columns will be the same number of columns for each query*. (for example, col1, col2,..., col10 in the first query, after appending with same columns in the second query will result in one query with a single set of col1,col2, ..., col10)

There is an exception for the number of columns which I'll talk about it later. Let's first look at what Append looks like in action;

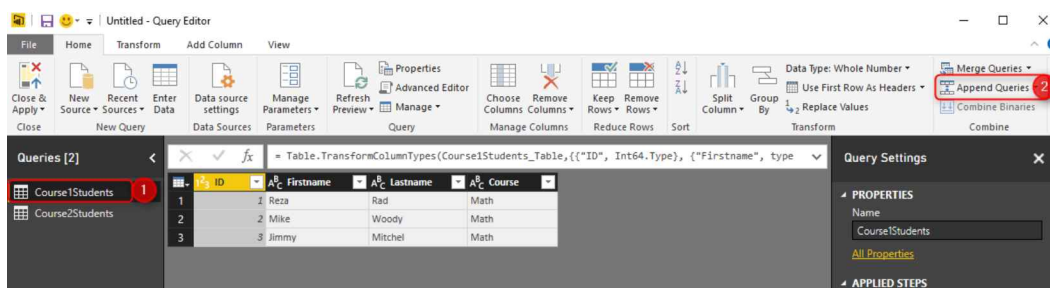
Consider two sample data sets; one for students of each course, Students of course 1:

| ID | Firstname | Lastname | Course |
|----|-----------|----------|--------|
| 1 | Reza | Rad | Math |
| 2 | Mike | Woody | Math |
| 3 | Jimmy | Mitchel | Math |

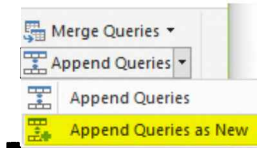
and Students of course 2:

| ID | Firstname | Lastname | Course |
|----|-----------|----------|---------|
| 10 | Shaun | Mitchel | English |
| 11 | Mary | Stuart | English |

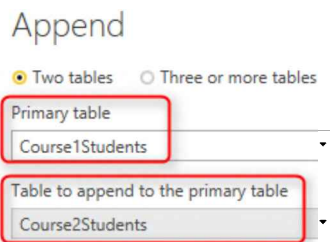
To append these queries, Click on one of them and select Append Queries from the Combine section of the Home tab in Query Editor



If you want to keep the existing query result and create a new query with the appended result, choose Append Queries as New; otherwise, just select Append Queries. In this example, I'll do Append Queries as New because I want to keep existing queries intact.



You can choose the primary table (normally, this is the query that you have selected before clicking on Append Queries) and the table to append.



You can also choose to append Three or more tables and add tables to the list as you wish. For this example, I have only two tables, so I'll continue with the above configuration. Append Queries simply append rows after each other, and because column names are exactly similar in both queries, the result set will have the same columns.

| ID | Firstname | Lastname | Course |
|----|-----------|----------|---------|
| 1 | Reza | Rad | Math |
| 2 | Mike | Woody | Math |
| 3 | Jimmy | Mitchel | Math |
| 4 | Shaun | Mitchel | English |
| 5 | Mary | Stuart | English |

The result of Append as simple as that

Part 1: Combining data tables

| ID | Firstname | Lastname | Course |
|----|-----------|----------|--------|
| 1 | Reza | Rad | Math |
| 2 | Mike | Woody | Math |
| 3 | Jimmy | Mitchel | Math |

| ID | Firstname | Lastname | Course |
|----|-----------|----------|---------|
| 10 | Shaun | Mitchel | English |
| 11 | Mary | Stuart | English |

| ID | Firstname | Lastname | Course |
|----|-----------|----------|---------|
| 1 | Reza | Rad | Math |
| 2 | Mike | Woody | Math |
| 3 | Jimmy | Mitchel | Math |
| 10 | Shaun | Mitchel | English |
| 11 | Mary | Stuart | English |

Append is similar to UNION ALL in T-SQL.

What about Duplicates?

Append Queries will NOT remove duplicates. You have to use Group By or Remove Duplicate Rows to get rid of duplicates.

What if columns in source queries are not exactly matched?

Append requires columns to be exactly similar to work in the best condition. If columns in source queries are different, append still works but create one column in the output per each new column. If one of the sources doesn't have that column, the cell value of that column for those rows will be null.

Merge

Merge is another type of combining queries that are based on matching rows rather than columns. The output of Merge will be a single query with;

- There should be joining or matching criteria between two queries. (for example, StudentID column of both queries to be matched with each other)
- The number of rows will be dependent on matching criteria between queries
- The number of Columns will be dependent on what columns are selected in the result set. (Merge will create a structured column as a result).

Chapter 1: Append vs. Merge in Power BI and Power Query

Understanding how Merge works might look more complicated, but it will be very easy with an example. Let's have a look at that in action;

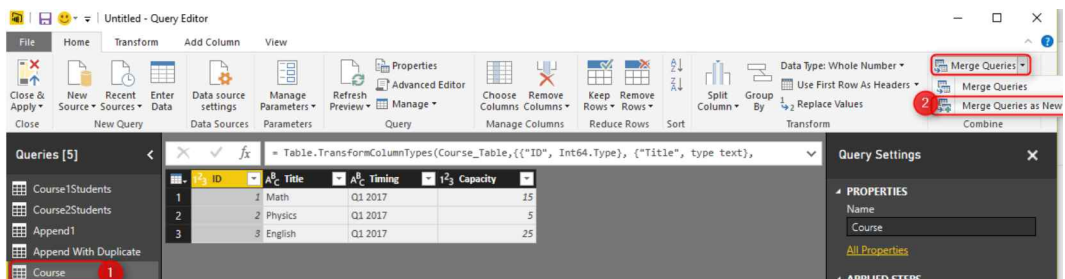
In addition to the tables in the first example, consider that there is another table for Course's details as below:

| i ² ₃ ID | A ^B _C Title | A ^B _C Timing | i ² ₃ Capacity |
|--------------------------------|-----------------------------------|------------------------------------|--------------------------------------|
| 1 | Math | Q1 2017 | 15 |
| 2 | Physics | Q1 2017 | 5 |
| 3 | English | Q1 2017 | 25 |

Now, if I want to combine the Course query with the Appended result of courseXstudents to see which students are part of which course with all details in each row, I need to use Merge Queries. Here is the appended result again;

| i ² ₃ ID | A ^B _C Firstname | A ^B _C Lastname | A ^B _C Course |
|--------------------------------|---------------------------------------|--------------------------------------|------------------------------------|
| 1 | Reza | Rad | Math |
| 2 | Mike | Woody | Math |
| 3 | Jimmy | Mitchel | Math |
| 10 | Shaun | Mitchel | English |
| 11 | Mary | Stuart | English |

Select Course Query first, and then Select Merge Queries (as New)



Merging Queries require joining criteria. Joining criteria is a field(s) in each source query that should be matched with each other to build the resulting query. In this example, I want to Merge the Course query with Append1, based on the Title of the course.

Part 1: Combining data tables

Merge

Select tables and matching columns to create a merged table.

Course 1

| ID | Title | Timing | Capacity |
|----|---------|---------|----------|
| 1 | Math | Q1 2017 | 15 |
| 2 | Physics | Q1 2017 | 5 |
| 3 | English | Q1 2017 | 25 |

Append1 2

| ID | Firstname | Lastname | Course |
|----|-----------|----------|---------|
| 1 | Reza | Rad | Math |
| 2 | Mike | Woody | Math |
| 3 | Jimmy | Mitchel | Math |
| 10 | Shaun | Mitchel | English |
| 11 | Mary | Stuart | English |

Join Kind
Left Outer (all from first, matching from second)

i The selection has matched 2 out of the first 3 rows.

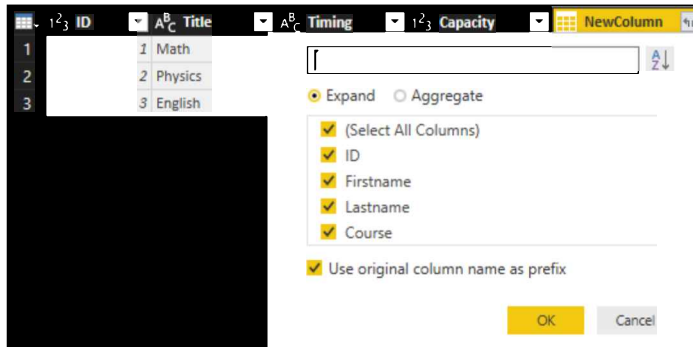
I'll talk about types of join later. For now, continue the selection, and you will see these two queries match each other based on the Course title. The resulting query will be the same as the first query (Course in this example), plus one additional column named NewColumn with a table in each cell. This is a structured column that can be expanded into underlying tables. If you click on an empty cell area containing one of these tables, you will see the sub-table underneath.

| ID | Title | Timing | Capacity | NewColumn |
|----|---------|---------|----------|-----------|
| 1 | Math | Q1 2017 | 15 | Table |
| 2 | Physics | Q1 2017 | 5 | Table |
| 3 | English | Q1 2017 | 25 | Table |

| ID | Firstname | Lastname | Course |
|----|-----------|----------|--------|
| 1 | Reza | Rad | Math |
| 2 | Mike | Woody | Math |
| 3 | Jimmy | Mitchel | Math |

Chapter 1: Append vs. Merge in Power BI and Power Query

Now click on Expand column icon, and expand the New Column to all underneath the table structure



The result will be a table including columns from both tables and rows matching each other.

| 123 ID | A6C Title | A6C Timing | 123 Capacity | 123 NewColumn.ID | A6C NewColumn.Firstname | A6C NewColumn.Lastname | A6C NewColumn.Course |
|--------|-----------|------------|--------------|------------------|-------------------------|------------------------|----------------------|
| 1 | 1 Math | Q1 2017 | 15 | 1 Reza | Rad | Math | |
| 2 | 1 Math | Q1 2017 | 15 | 2 Mike | Woody | Math | |
| 3 | 1 Math | Q1 2017 | 15 | 3 Jimmy | Mitchel | Math | |
| 4 | 3 English | Q1 2017 | 25 | 10 Shaun | Mitchel | English | |
| 5 | 3 English | Q1 2017 | 25 | 11 Mary | Stuart | English | |
| 6 | 2 Physics | Q1 2017 | 5 | null | null | null | |

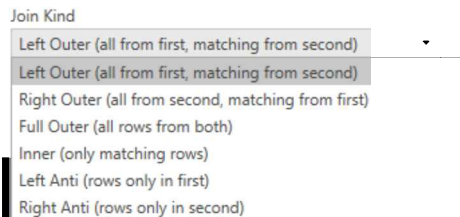
Columns on the left-hand side are coming from the Course table, columns on the right-hand side are coming from the Students' table. Values in the rows only appear in matching criteria. The first three rows are students of the Math course, then two students for the English course, and because there is no student for the Physics course, you will see null values for students' columns.



Merge is similar to JOIN in T-SQL

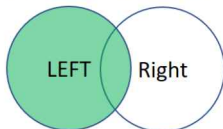
Join Types

There are six types of joins supported in Power BI as below. It depends on the effect on the result set based on matching rows, each of these types works differently.



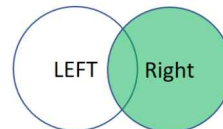
Explaining what each join type will do is a different chapter in this book. For now, this picture explains it very well:

LEFT Outer



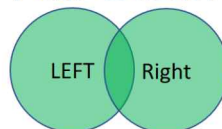
All rows from left and matching from right

RIGHT Outer



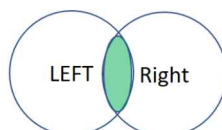
All rows from right and matching from left

Full Outer



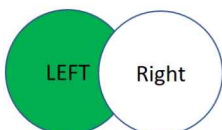
All rows from both: matching and not matching

Inner



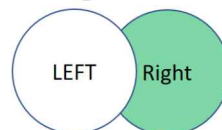
Only matching rows

Left Anti



Not matching rows from left

Right Anti



Not matching rows from right

Chapter 2: Reference vs. Duplicate in Power BI; Power Query Back to Basics

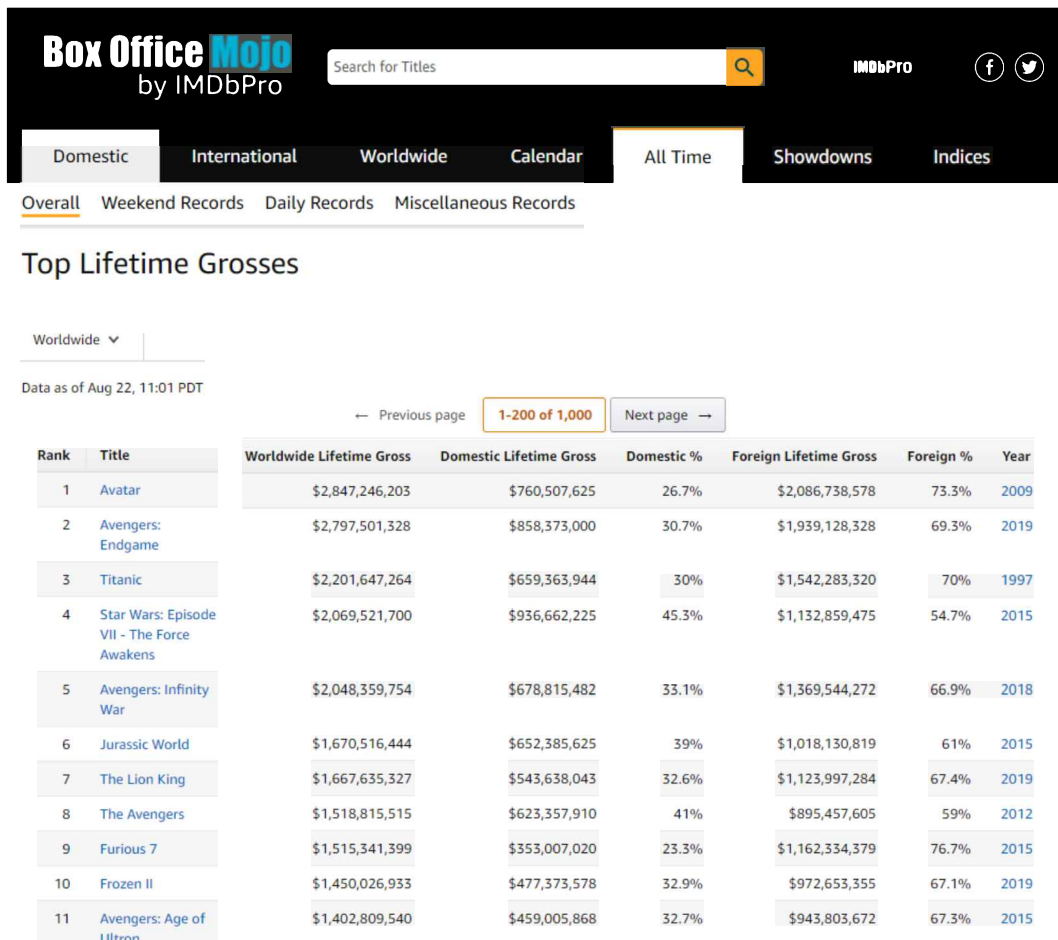


When you work with tables and queries in Power Query and Power BI, you get the option to copy them through these actions: Duplicate or Reference. In my sessions and courses, it has always been a question: What is the actual difference between these two actions. The explanation is simple but very important to understand. Because when you know the difference, you will use it properly. In this short chapter, I'll explain the reference and the difference between that with Duplicate.

Duplicate

If you want to copy an entire query with all of its steps, then Duplicate is your friend. Let's see this in action. For example, Let's assume that we got the data from a web page that shows us the best seller's movies information. If you have done the example of the movie in this book previously, This information is available on the [Box Office Mojo website](https://www.boxofficemojo.com/chart/ww_top_lifetime_gross/?area=XWW)[https://www.boxofficemojo.com/chart/ww_top_lifetime_gross/?area=XWW], as below:

Part 1: Combining data tables



Box Office Mojo by IMDbPro

Search for Titles

Domestic International Worldwide Calendar All Time Showdowns Indices

Overall Weekend Records Daily Records Miscellaneous Records

Top Lifetime Grosses

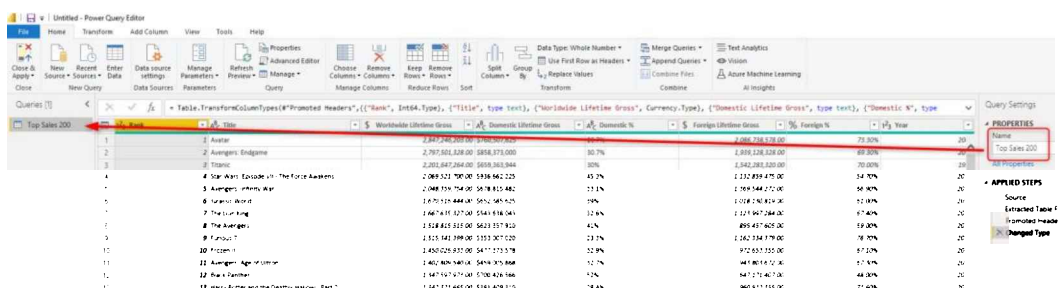
Worldwide ▾

Data as of Aug 22, 11:01 PDT

← Previous page 1-200 of 1,000 Next page →

| Rank | Title | Worldwide Lifetime Gross | Domestic Lifetime Gross | Domestic % | Foreign Lifetime Gross | Foreign % | Year |
|------|--|--------------------------|-------------------------|------------|------------------------|-----------|------|
| 1 | Avatar | \$2,847,246,203 | \$760,507,625 | 26.7% | \$2,086,738,578 | 73.3% | 2009 |
| 2 | Avengers: Endgame | \$2,797,501,328 | \$858,373,000 | 30.7% | \$1,939,128,328 | 69.3% | 2019 |
| 3 | Titanic | \$2,201,647,264 | \$659,363,944 | 30% | \$1,542,283,320 | 70% | 1997 |
| 4 | Star Wars: Episode VII - The Force Awakens | \$2,069,521,700 | \$936,662,225 | 45.3% | \$1,132,859,475 | 54.7% | 2015 |
| 5 | Avengers: Infinity War | \$2,048,359,754 | \$678,815,482 | 33.1% | \$1,369,544,272 | 66.9% | 2018 |
| 6 | Jurassic World | \$1,670,516,444 | \$652,385,625 | 39% | \$1,018,130,819 | 61% | 2015 |
| 7 | The Lion King | \$1,667,635,327 | \$543,638,043 | 32.6% | \$1,123,997,284 | 67.4% | 2019 |
| 8 | The Avengers | \$1,518,815,515 | \$623,357,910 | 41% | \$895,457,605 | 59% | 2012 |
| 9 | Furious 7 | \$1,515,341,399 | \$353,007,020 | 23.3% | \$1,162,334,379 | 76.7% | 2015 |
| 10 | Frozen II | \$1,450,026,933 | \$477,373,578 | 32.9% | \$972,653,355 | 67.1% | 2019 |
| 11 | Avengers: Age of Ultron | \$1,402,809,540 | \$459,005,868 | 32.7% | \$943,803,672 | 67.3% | 2015 |

In Power Query, we got data From the Web and selected this source;



File Home Transform Add Column View Tools Help

Queries: 1 Top Sales 200

Table.TransformColumnTypes(*Promoted Headers*, {"Rank", Int64.Type, ("Title", type text), ("Worldwide Lifetime Gross", Currency.Type), ("Domestic Lifetime Gross", Currency.Type), ("Domestic %", type text), ("Foreign Lifetime Gross", Currency.Type), ("Foreign %", type text), ("Year", type text)})

| Rank | Title | Worldwide Lifetime Gross | Domestic Lifetime Gross | Domestic % | Foreign Lifetime Gross | Foreign % | Year |
|------|--|--------------------------|-------------------------|------------|------------------------|-----------|------|
| 1 | Avatar | \$2,847,246,203.00 | \$760,507,625.00 | 26.7% | \$2,086,738,578.00 | 73.3% | 2009 |
| 2 | Avengers: Endgame | \$2,797,501,328.00 | \$858,373,000.00 | 30.7% | \$1,939,128,328.00 | 69.3% | 2019 |
| 3 | Titanic | \$2,201,647,264.00 | \$659,363,944.00 | 30% | \$1,542,283,320.00 | 70.0% | 1997 |
| 4 | Star Wars: Episode VII - The Force Awakens | \$2,069,521,700.00 | \$936,662,225.00 | 45.3% | \$1,132,859,475.00 | 54.7% | 2015 |
| 5 | Avengers: Infinity War | \$2,048,359,754.00 | \$678,815,482.00 | 33.1% | \$1,369,544,272.00 | 66.9% | 2018 |
| 6 | Jurassic World | \$1,670,516,444.00 | \$652,385,625.00 | 39% | \$1,018,130,819.00 | 61% | 2015 |
| 7 | The Lion King | \$1,667,635,327.00 | \$543,638,043.00 | 32.6% | \$1,123,997,284.00 | 67.4% | 2019 |
| 8 | The Avengers | \$1,518,815,515.00 | \$623,357,910.00 | 41% | \$895,457,605.00 | 59% | 2012 |
| 9 | Furious 7 | \$1,515,341,399.00 | \$353,007,020.00 | 23.3% | \$1,162,334,379.00 | 76.7% | 2015 |
| 10 | Frozen II | \$1,450,026,933.00 | \$477,373,578.00 | 32.9% | \$972,653,355.00 | 67.1% | 2019 |
| 11 | Avengers: Age of Ultron | \$1,402,809,540.00 | \$459,005,868.00 | 32.7% | \$943,803,672.00 | 67.3% | 2015 |
| 12 | New Year's Eve | \$1,402,809,540.00 | \$459,005,868.00 | 32.7% | \$943,803,672.00 | 67.3% | 2015 |
| 13 | New Year's Eve | \$1,402,809,540.00 | \$459,005,868.00 | 32.7% | \$943,803,672.00 | 67.3% | 2015 |
| 14 | New Year's Eve | \$1,402,809,540.00 | \$459,005,868.00 | 32.7% | \$943,803,672.00 | 67.3% | 2015 |

PROPERTIES: Name, Top Sales 200, Webpage

APPLIED STEPS: Source, Extracted Table, Promoted Headers, Changed Type

After doing all some transformations, you realize that this data is only for the first couple of hundreds of best seller movies because that web page doesn't have the remaining movies. To get the remaining, you need to navigate page 2, which has a different URL, but the same data structure.

Chapter 2: Reference vs. Duplicate in Power BI; Power Query Back to Basics

boxoffice Mojo.com/chart/vw_top_lifetime_gross/?area=XWW&offset=200

Bookmarks Submit MVP Activity Puzzle Book Template Dataflow Docs LMS Full-function mortg... TAX Power BI resources Presentation Dev

Box Office Mojo
by IMDbPro

Search for Titles

IMDbPro

Domestic International Worldwide Calendar All Time Showdowns Indices

Overall Weekend Records Daily Records Miscellaneous Records

Top Lifetime Grosses

Worldwide

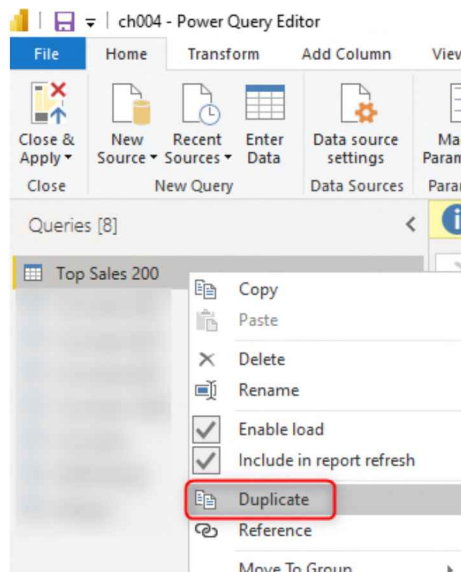
Data as of Aug 23, 16:44 PDT

Previous page 201-400 of 1,000 Next page

| Rank | Title | Worldwide Lifetime Gross | Domestic Lifetime Gross | Domestic % | Foreign Lifetime Gross | Foreign % | Year |
|------|--|--------------------------|-------------------------|------------|------------------------|-----------|------|
| 201 | Ralph Breaks the Internet | \$529,323,962 | \$201,091,711 | 38% | \$328,232,251 | 62% | 2018 |
| 202 | Hotel Transylvania 3: Summer Vacation | \$528,583,774 | \$167,510,016 | 31.7% | \$361,073,758 | 68.3% | 2018 |
| 203 | The Boss Baby | \$527,965,936 | \$175,003,033 | 33.2% | \$352,962,903 | 66.8% | 2017 |
| 204 | Dunkirk | \$527,016,307 | \$189,740,665 | 36% | \$337,275,642 | 64% | 2017 |
| 205 | How to Train Your Dragon: The Hidden World | \$525,687,703 | \$160,799,505 | 30.6% | \$364,888,198 | 69.4% | 2019 |

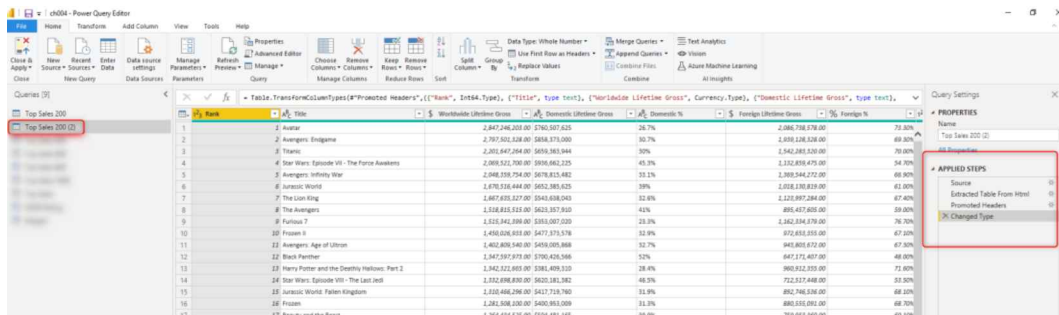
Well, what do you need to do? You have to do all those steps on page 2 as well. Let's keep this example static and basic (Because in complex scenarios when you have many pages, you may use functions and parameters to loop through all pages and combine them all. If you are interested to learn about that, read the chapter about it later in this book). Let's say you want to do all the steps you have done for page one, now for page two. To do that, you can leverage Duplicate.

Create a duplicate of Box Office Mojo (I called it; Tops Sales 200)



When you create the Duplicate query, it will be an exact copy of the first query, with all steps of it. These two queries are exactly like each other. No difference!

Part 1: Combining data tables



Duplicate copies a query with all the applied steps as a new query, an exact copy.

After creating the copy, then you can go to the source step to change the URL:



Using Duplicate, you managed to copy a query with all steps and then make changes in your new query. Your original query is intact.

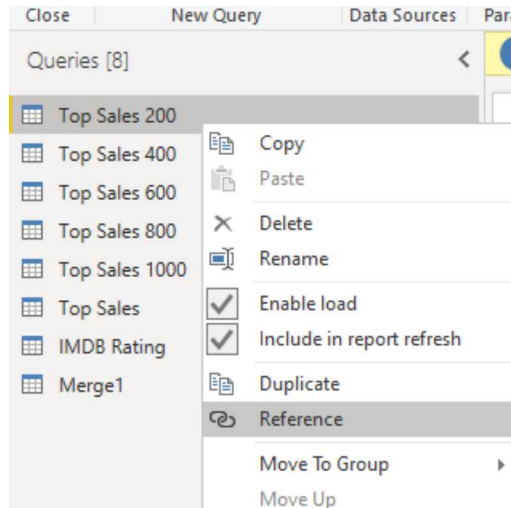
Duplicate is the option to choose when you want to copy a query but do a different step configuration.

Reference

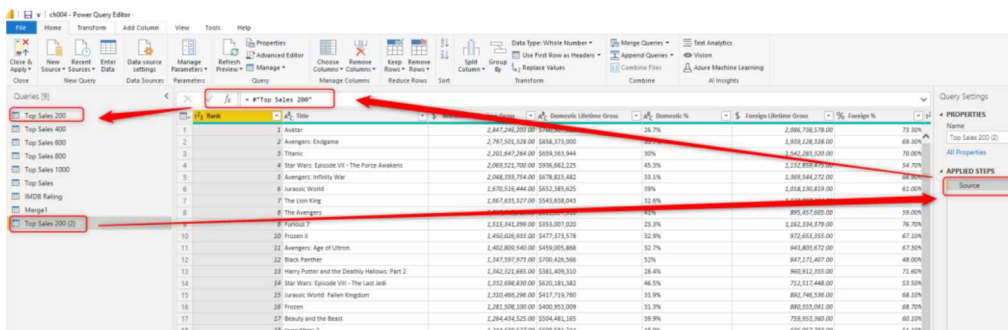
Reference is another way of copying a query. However, the big difference is that; when you reference a query, the new query will have only one step: sourcing from the original query. A referenced query will not have the applied steps of the original query. Let's see this option in action. Continuing the example above, we want to create a new query that

combines the page 1 and page 2 results. However, we do NOT want to change any existing queries because we want to use those as the source for other operations.

With a right-click on Top Sales 200, I can create a Reference.



Reference will create a new query which is a copy of the Top Sales 200, but only contains one single step:



The only step in the new query is sourcing the data from the original query. What does it mean? It means if you make changes in the original query, then this new query will be impacted.

Reference will create a new query that has only one step: Getting data from the original query.

Now we can use this query to append to the Top Sales 400 and other queries;

Part 1: Combining data tables

Append

Concatenate rows from three or more tables into a single table.

☐ Two tables

☒ Three or more tables

1

Available tables

| |
|----------------|
| Top Sales 200 |
| Top Sales 400 |
| Top Sales 600 |
| Top Sales 800 |
| Top Sales 1000 |

2

Add >>

3

Tables to append

Top Sales 200

OK

Cancel

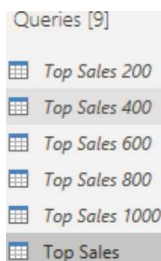
The result would be a query that contains all pages;

| Rank | Title | Domestic Lifetime Gross | Domestic % of Total | Foreign Lifetime Gross |
|------|---|-------------------------|---------------------|------------------------|
| 1 | Avatar | 2,847,246,000.00 | 26.7% | 2,086,716,578.00 |
| 2 | Avengers: Endgame | 2,797,501,538.00 | 30.7% | 2,039,128,328.00 |
| 3 | Titanic | 2,201,642,064.00 | 20% | 2,542,283,320.00 |
| 4 | Star Wars: Episode IX - The Force Awakens | 2,069,511,700.00 | 45.3% | 2,152,898,470.00 |
| 5 | Avengers: Infinity War | 2,048,335,794.00 | 33.1% | 2,809,544,272.00 |
| 6 | Jurassic World | 1,670,536,444.00 | 39% | 2,018,130,810.00 |
| 7 | The Lion King | 1,667,615,127.00 | 32.6% | 2,125,907,284.00 |
| 8 | The Avengers | 1,518,815,131.00 | 41% | 895,451,505.00 |
| 9 | Pirates 7 | 1,513,941,899.00 | 23.3% | 2,162,474,379.00 |
| 10 | Frozen II | 1,403,026,933.00 | 32.9% | 872,853,853.00 |
| 11 | Avengers: Age of Ultron | 1,402,805,140.00 | 32.7% | 843,805,672.00 |
| 12 | Star Partner | 1,343,557,573.00 | 33% | 847,571,467.00 |
| 13 | Harry Potter and the Deathly Hallows - Part 2 | 1,342,521,865.00 | 38.4% | 860,922,555.00 |
| 14 | Star Wars: Episode VIII - The Last Jedi | 1,332,696,830.00 | 46.5% | 712,517,448.00 |

To learn more about append and the difference with Merge, read the chapter related to that later in this book. In this example, we used the Reference option to create a copy of the original query and then continue some extra steps. There are many other usages for Reference.

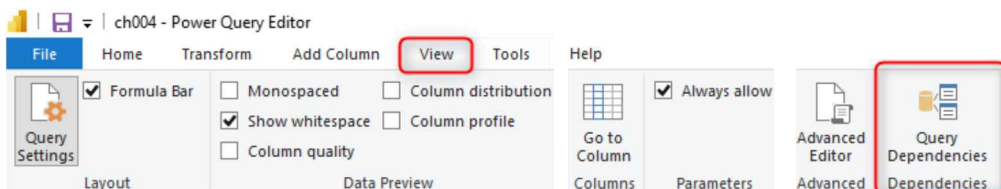
Reference is a good choice when you want to branch a query into different paths. One path follows several steps, and another follows different steps, and both share some steps in the original query.

After doing the append in this example, it is good to uncheck the enable load on Page 1 and Page 2 queries to save some memory in Power BI.



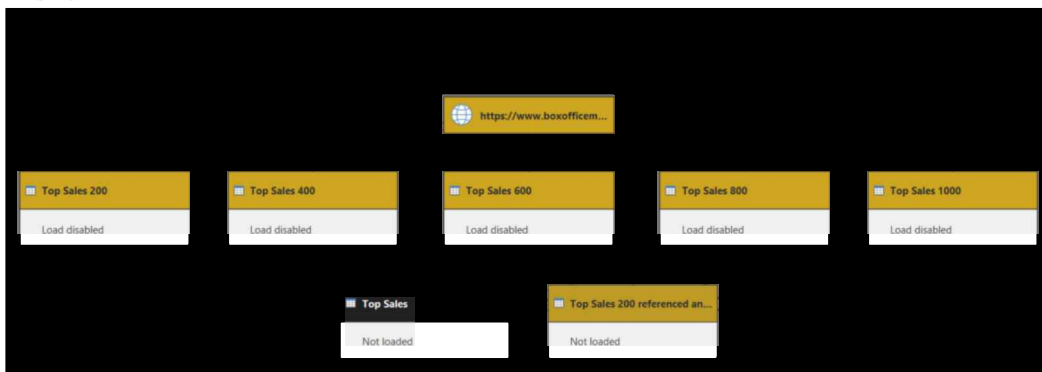
Query Dependency

Finding which query is dependent (or referenced from) on which query can be challenging when you have too many queries. That is why we have the Query Dependency menu option in the View tab of Power Query;



For our example above, this is the query dependency diagram;

Query Dependencies



Duplicate vs. Reference

Now that you know there are two options when you copy a query let's look at their difference.

Isolation from the Original or Dependency to the Original

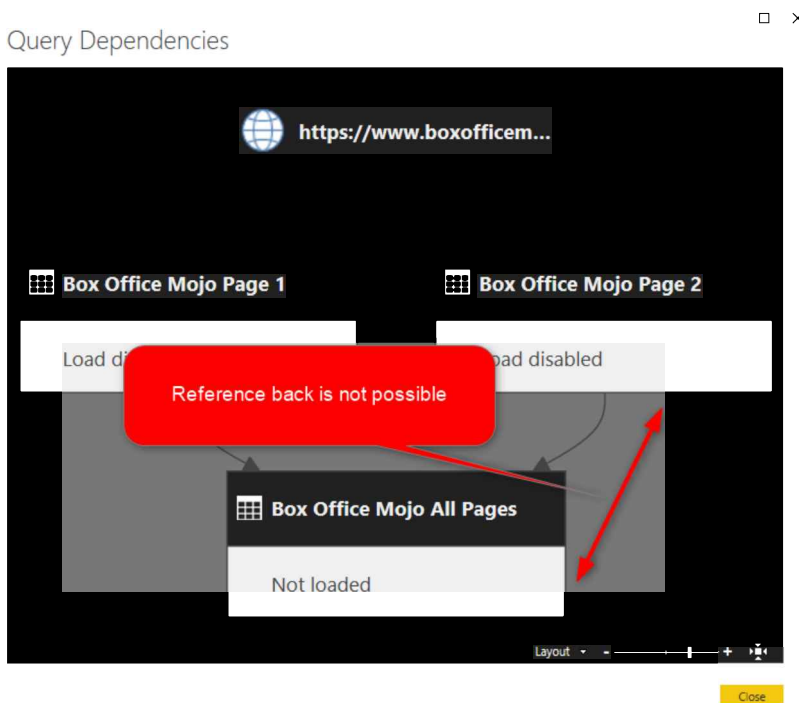
Duplicate creates a new copy with all the existing steps. The new copy will be isolated from the original query. You can make changes in the original or the new query, and they will NOT affect each other. On the other hand, reference is a new copy with only one single step: getting data from the original query. If you make a change in the

Part 1: Combining data tables

original query, the new query will be impacted. For example; If you remove a column from the original query, the new query will not have it if it used the Reference method for copying.

Limitation of the Reference

You can not use referenced queries in all situations. As an example; If you have a Query 1, you created a reference from that as Query 2. You cannot use the result from Query 2 in Query 1! It will create a circular reference. You are combining a query with reference to the query itself. It is impossible!

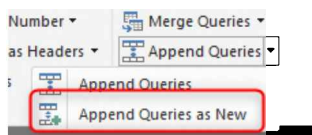


Some actions that invoke Reference or Duplicate

There are some actions in the Power Query that trigger Reference or Duplicate, let's check those options:

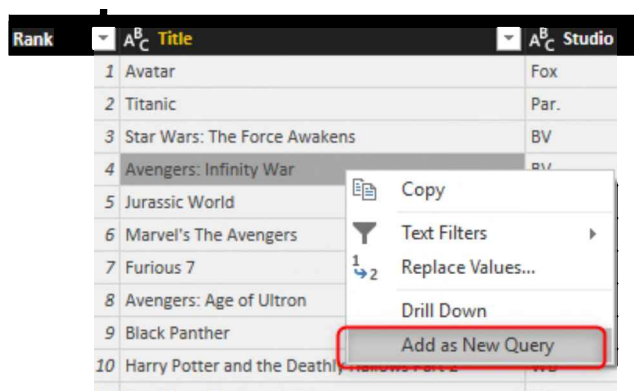
Append Queries as New / or Merge Queries as New is a Reference action

These two actions create a reference from the original query and then do Append or Merge with other queries.

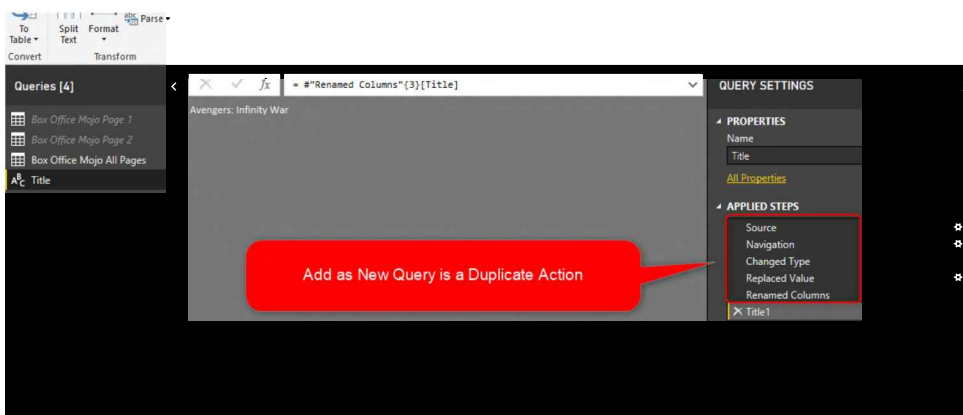


Add as New Query is a Duplicate action

Believe it or not, when you right-click on a column or cell and select Add as New Query, you are creating a duplicate of the original query.



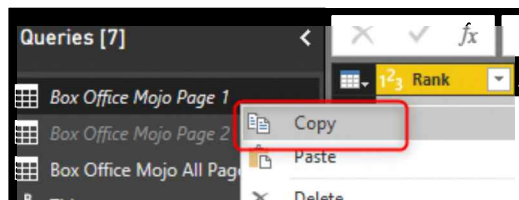
This can be misleading sometimes because you may expect the new query to source from the original, and with the change of original, this query also to change. However, the truth is that this is a duplicate action, and after this action, your original query and the new copy will be isolated from each other.



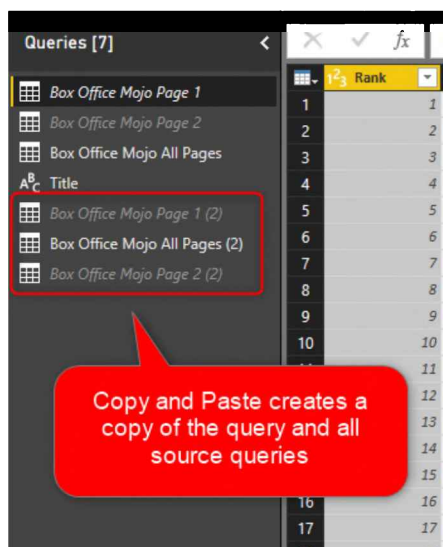
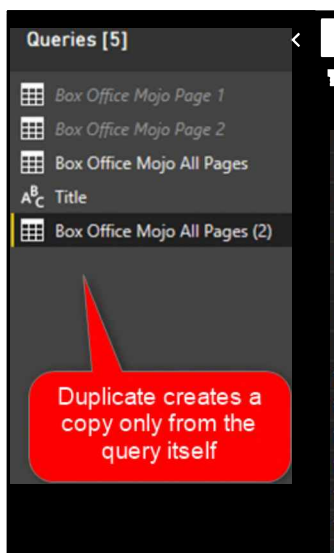
Copy and Paste is Neither Duplicate Nor Reference!

This is another misconception that Copy and Paste are similar to Duplicate. It is not, and it is not a Reference either. When you do this action on a simple query (I mean a query that is not sourced from any other queries), then you get a result similar to Duplicate.

Part 1: Combining data tables



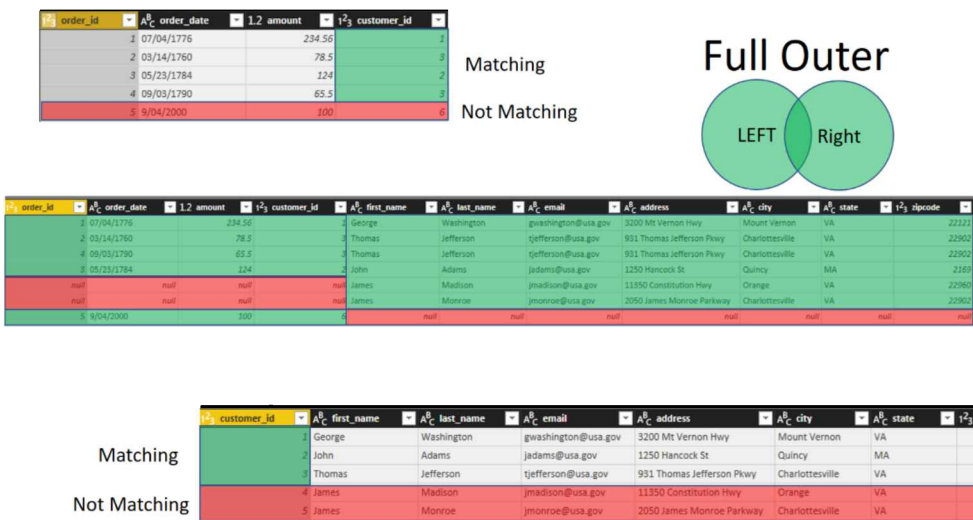
But when you do the Copy and Paste on a query sourced from other queries, the result is a copy of all original queries. Here is the result of Copy and Paste on Box Office Mojo All Pages (which is sourced from Page 1 and Page 2);



Summary

Duplicate and Reference are two different actions, and they are also different from the Copy and Paste of a query. A duplicate will give you an exact copy of the query with all steps. Reference will create a reference to the original query instead of a new query. Duplicate is a good option to choose when you want the two copies to be isolated from each other. A reference is a good option when you create different branches from one original query. There are some actions in Power Query that trigger Duplicate or Reference, as listed in this chapter. I hope this was a good chapter for you to understand the difference between these two actions clearly, and use them wisely from now on.

Chapter 3: Choose the Right Merge Join Type in Power BI



Previously, I have written a chapter explaining two ways of combining data sets: Append vs. Merge[<https://radacad.com/append-vs-merge-in-power-bi-and-power-query>]. In this chapter, I want to explain in detail what is the difference between all types of Merge Types and explaining how to choose the right merge (or Join) type. These Merge types are very similar to join types in relational databases. Because many people who work with Power BI might not have experience working with relational databases, I think this chapter is a good explanation in detail of what these types are and when to use them.

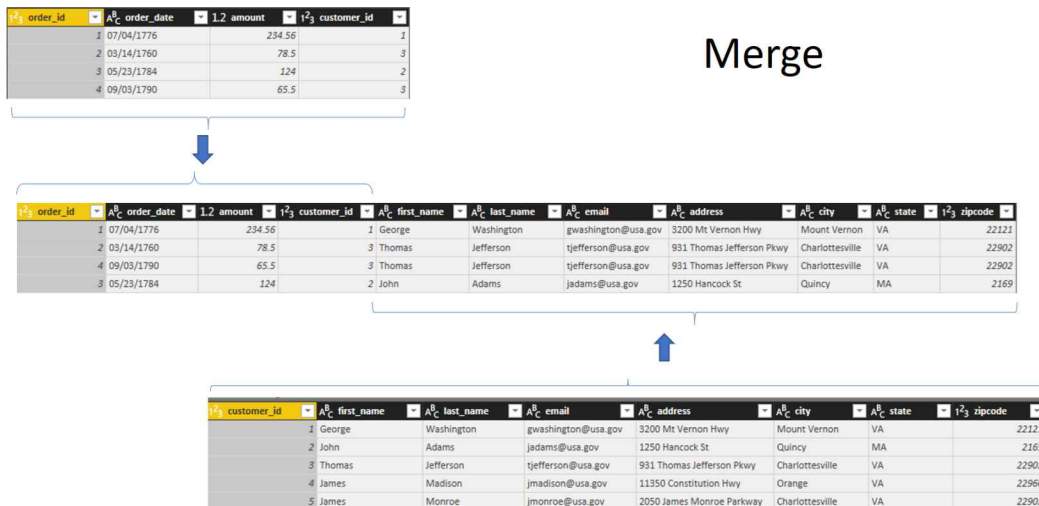
Sample Data Set

Download Data set from the book’s code files.

What is Merge?

Combining two data sets can be done in multiple ways. One of the ways of combining data sets is Merging data sets. Merge is similar to Join in relational databases. Merging two data sets requires some joining fields, and the result will be a combined set of columns from both data sets.

Part 1: Combining data tables



Let's go through it with an example;

Consider two data tables as below

Download the data files from the code files of this book.

Customers Table:

| customer_id | first_name | last_name | email | address | city | state | zipcode |
|-------------|------------|------------|----------------------|---------------------------|-----------------|-------|---------|
| 1 | George | Washington | gswashington@usa.gov | 3200 Mt Vernon Hwy | Mount Vernon | VA | 22121 |
| 2 | John | Adams | jadams@usa.gov | 1250 Hancock St | Quincy | MA | 2169 |
| 3 | Thomas | Jefferson | tjefferson@usa.gov | 931 Thomas Jefferson Pkwy | Charlottesville | VA | 22902 |
| 4 | James | Madison | jmadison@usa.gov | 11350 Constitution Hwy | Orange | VA | 22960 |
| 5 | James | Monroe | jmonroe@usa.gov | 2050 James Monroe Parkway | Charlottesville | VA | 22902 |

Orders Table:

| order_id | order_date | amount | customer_id |
|----------|------------|----------|-------------|
| 1 | 07/04/1776 | \$234.56 | 1 |
| 2 | 03/14/1760 | \$78.50 | 3 |
| 3 | 05/23/1784 | \$124.00 | 2 |
| 4 | 09/03/1790 | \$65.50 | 3 |

Merging these two tables gives you a data set with a combined set of columns like below;

| i1: order_id | A1: order_date | 1.2 amount | i2: customer_id | A2: first_name | A2: last_name | A2: email | A2: address | A2: city | A2: state | i2: zipcode |
|--------------|----------------|------------|-----------------|----------------|---------------|----------------------|---------------------------|-----------------|-----------|-------------|
| 1 | 07/04/1776 | 234.56 | 1 | George | Washington | gswashington@usa.gov | 3200 Mt Vernon Hwy | Mount Vernon | VA | 22121 |
| 2 | 03/14/1760 | 78.5 | 3 | Thomas | Jefferson | tjefferson@usa.gov | 931 Thomas Jefferson Pkwy | Charlottesville | VA | 22902 |
| 4 | 09/03/1790 | 65.5 | 3 | Thomas | Jefferson | tjefferson@usa.gov | 931 Thomas Jefferson Pkwy | Charlottesville | VA | 22902 |
| 3 | 05/23/1784 | 124 | 2 | John | Adams | jadams@usa.gov | 1250 Hancock St | Quincy | MA | 2169 |

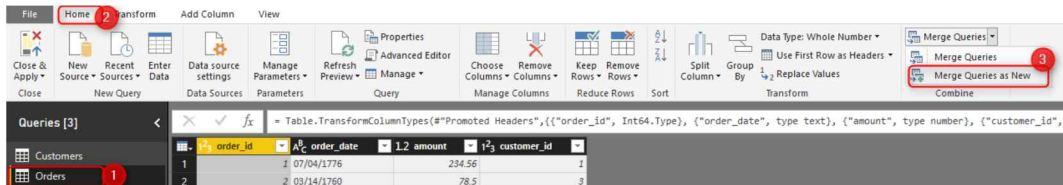
For merging data sets, you need to have some joining fields. In this example joining field is customer_id. To get a data set that includes all columns from both tables based on customer_id relationship, this is how you can join tables to each other:

How to Merge Queries

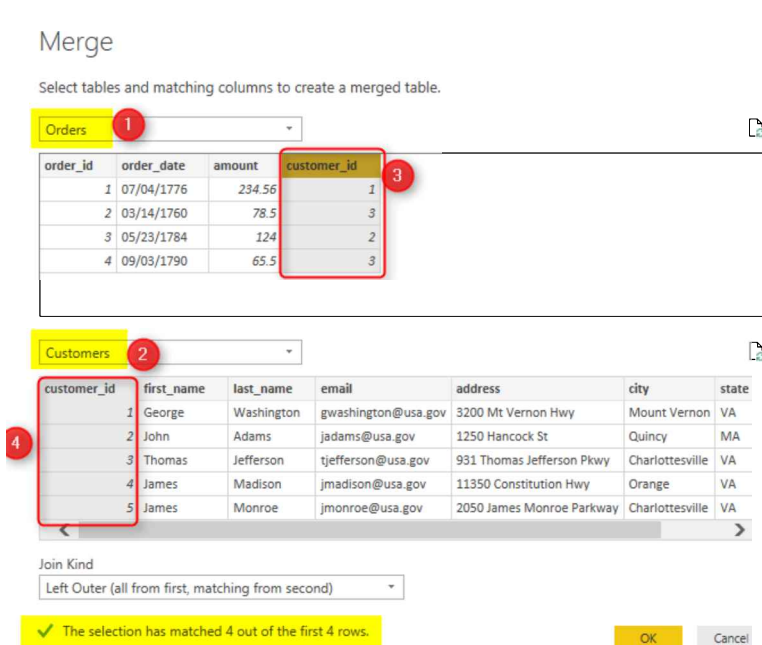
Select the first query (for this example: Orders), and then from the home tab, Merge queries. There are two options there:

- Merge Queries: This will amend the existing query (orders) to result from Merging.
- Merge Queries as New: This will not change the existing query. It will create a reference from it, and the result of merging would be another query.

Select Merge Queries as New.

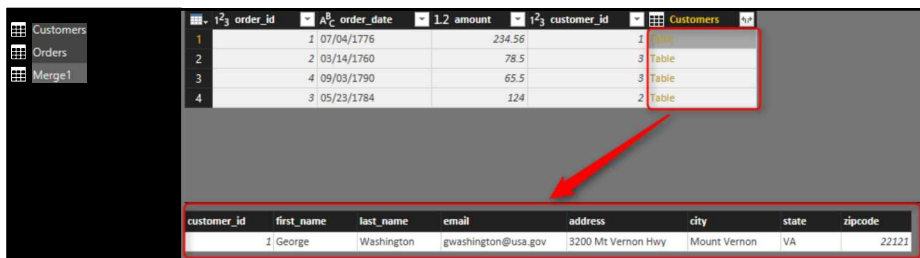


Select the second table (Customers) in the Merge configuration window, then select the joining field in each table (Customer_id). You will also see the number of matching rows as extra information there.



The result of this operation will be a new query named Merge1, which has the combined result of these two queries.

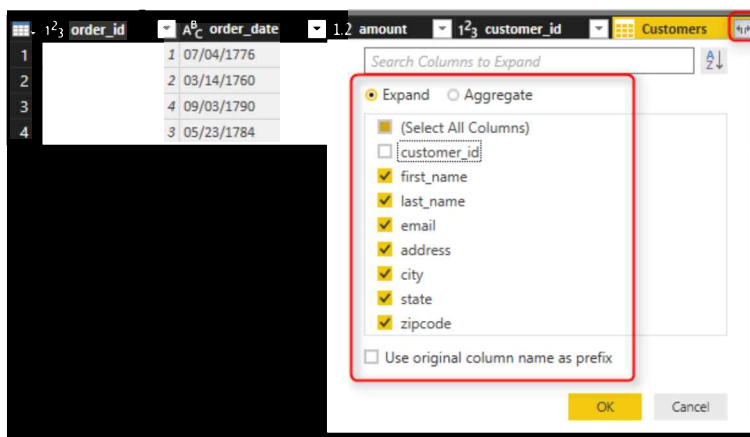
Part 1: Combining data tables



The screenshot shows a data table with columns: order_id, order_date, amount, customer_id, and a sub-table in the Customers column. The sub-table is highlighted with a red box and a red arrow pointing to it. The sub-table has columns: customer_id, first_name, last_name, email, address, city, state, zipcode. The data in the sub-table is as follows:

| customer_id | first_name | last_name | email | address | city | state | zipcode |
|-------------|------------|------------|----------------------|--------------------|--------------|-------|---------|
| 1 | George | Washington | g.washington@usa.gov | 3200 Mt Vernon Hwy | Mount Vernon | VA | 22121 |

The table in the Customers column is a sub-table from the customer table for records matching that customer_id. You can then expand it to columns you want;



and the final result will be now all columns in one query;



The screenshot shows the final merged data table with all columns from both tables. The data is as follows:

| order_id | order_date | amount | customer_id | first_name | last_name | email | address | city | state | zipcode |
|----------|------------|--------|-------------|------------|------------|----------------------|---------------------------|-----------------|-------|---------|
| 1 | 07/04/1776 | 234.56 | 1 | George | Washington | g.washington@usa.gov | 3200 Mt Vernon Hwy | Mount Vernon | VA | 22121 |
| 2 | 03/14/1760 | 78.5 | 3 | Thomas | Jefferson | t.jefferson@usa.gov | 931 Thomas Jefferson Pkwy | Charlottesville | VA | 22902 |
| 3 | 09/03/1790 | 65.5 | 3 | Thomas | Jefferson | t.jefferson@usa.gov | 931 Thomas Jefferson Pkwy | Charlottesville | VA | 22902 |
| 4 | 05/23/1784 | 124 | 2 | John | Adams | j.adams@usa.gov | 1250 Hancock St | Quincy | MA | 2169 |

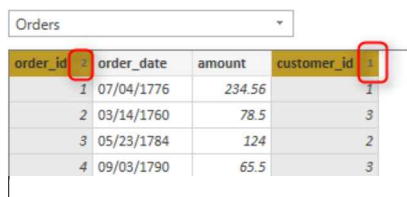
This process is called Merge Join. However, there are some configurations you can do for this;

Merging Based on Multiple Columns

You can easily use multiple columns for merging two data sets. Just select them in order by holding the Ctrl key of the keyboard.

Merge

Select tables and matching columns to create a merged ta



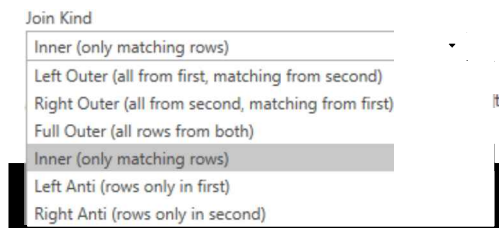
The screenshot shows the Merge dialog box with the 'Orders' table selected. The 'order_id' and 'customer_id' columns are highlighted with red boxes. The data in the table is as follows:

| order_id | order_date | amount | customer_id |
|----------|------------|--------|-------------|
| 1 | 07/04/1776 | 234.56 | 1 |
| 2 | 03/14/1760 | 78.5 | 3 |
| 3 | 05/23/1784 | 124 | 2 |
| 4 | 09/03/1790 | 65.5 | 3 |

This method can be really helpful because the relationship tab in Power BI Desktop doesn't create a relationship based on multiple columns. However, in Power Query, you can create the merge and create a unique field for the relationship in Power BI Desktop. There is a chapter about it in this book.

Merge Types

In addition to the merging column (or joining field), the type of Merge is very important. You can get a different result set by choosing a different type of merge. Here is where you can see the Join Kind (or Merge Type);



At the moment of writing this chapter, there are six types of joins. Every one of these joins gets a different result set. Let's see what their difference is.

Left and Right

To start, you need to know the concept of Left and Right tables (or queries). When you merge two data sets, the first query is considered LEFT and the second as RIGHT.

Part 1: Combining data tables

Merge

Select tables and matching columns to create a merged table.

Orders LEFT

| order_id | order_date | amount | customer_id |
|----------|------------|--------|-------------|
| 1 | 07/04/1776 | 234.56 | 1 |
| 2 | 03/14/1760 | 78.5 | 3 |
| 3 | 05/23/1784 | 124 | 2 |
| 4 | 09/03/1790 | 65.5 | 3 |

Customers

RIGHT

| customer_id | first_name | last_name | email | address |
|-------------|------------|------------|---------------------|----------------|
| 1 | George | Washington | gWASHINGTON@usa.gov | 3200 Mt Vernon |
| 2 | John | Adams | jADAMS@usa.gov | 1250 Hancock |
| 3 | Thomas | Jefferson | tJEFFERSON@usa.gov | 931 Thomas |
| 4 | James | Madison | jMADISON@usa.gov | 11350 Con |
| 5 | James | Monroe | jMONROE@usa.gov | 2050 James |

<

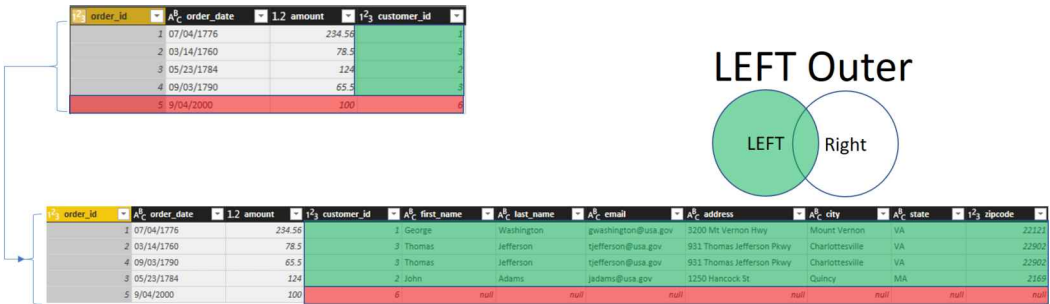
In the example above, Orders is LEFT query, and Customers is the RIGHT query. You can change them if you want, of course. Understanding this is important because most of Join Kinds works with the concept of left or right or both.

Left Outer (All from first, matching from second)

The first type of Join/Merge is Left Outer. This means the LEFT query is the important one. All records from this query (LEFT or FIRST) will be shown in the result set plus their matching rows in the right (or second table). This type of join is the default type. If you don't specify the Join Kind, it will always be Left Outer.

For example, in the first Merge example we have done, you can see that the result set is four records, representing four records from the left table (orders) and their matching rows in the customer table.

Chapter 3: Choose the Right Merge Join Type in Power BI



Matching

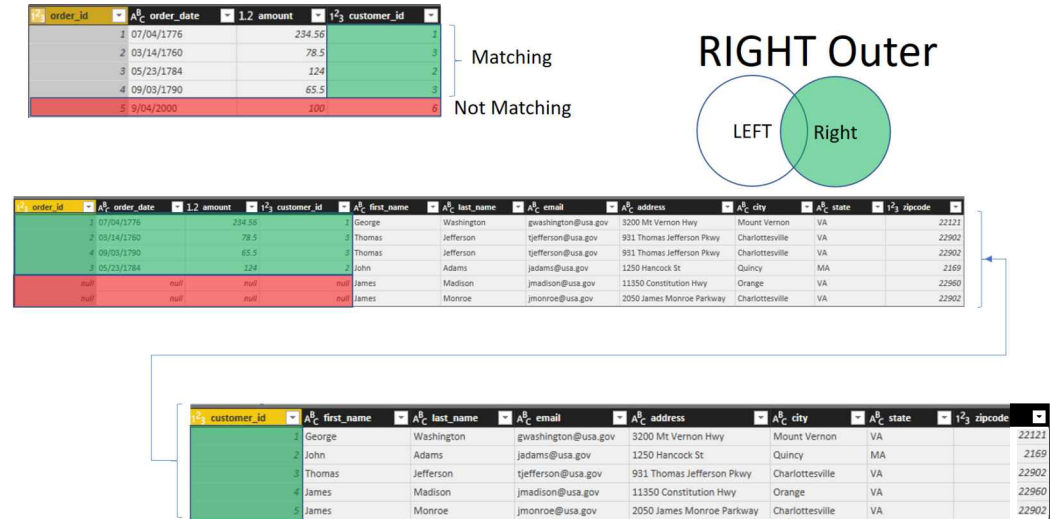
Not Matching

| customer_id | first_name | last_name | email | address | city | state | zipcode |
|-------------|------------|------------|----------------------|---------------------------|-----------------|-------|---------|
| 1 | George | Washington | gswashington@usa.gov | 3200 Mt Vernon Hwy | Mount Vernon | VA | 22121 |
| 2 | John | Adams | jadams@usa.gov | 1250 Hancock St | Quincy | MA | 2169 |
| 3 | Thomas | Jefferson | tjefferson@usa.gov | 931 Thomas Jefferson Pkwy | Charlottesville | VA | 22902 |
| 4 | James | Madison | jmadison@usa.gov | 11350 Constitution Hwy | Orange | VA | 22960 |
| 5 | James | Monroe | jmonroe@usa.gov | 2050 James Monroe Parkway | Charlottesville | VA | 22902 |

As you can see in the screenshot above, two customers won't be in the result set. Customers with id 4 and 5. because these rows do not match the customer_id field in the orders table. Only records from the Left table with matching rows of the right table will be selected in the LEFT Outer merge.

Right Outer (all rows from second, matching from first)

Sometimes you need to fetch all rows from the second table, regardless of whether they exist in the first table. In that case, you would need to use another type of Join called Right Outer. With this type of Join, you get all rows from the RIGHT (or second) table, matching rows from the left (or first table). Here is an example:



Part 1: Combining data tables

As you can see in the above screenshot, all rows from the customers' table are showed in the result set. However, only four rows of that are matching with the orders table. If a record in the Orders table doesn't match, it will come as Null (two red rows in the result set).

Full Outer (all rows from both)

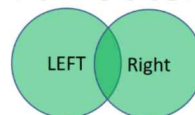
This type of join/merge is normally the safest among all. This will return all rows from both tables (matching and non-matching). You will have all rows from the first table, all rows from the second table, and all matching rows. With this method, you won't lose any records.

| order_id | order_date | amount | customer_id |
|----------|------------|--------|-------------|
| 1 | 07/04/1776 | 234.56 | 1 |
| 2 | 03/14/1760 | 78.5 | 3 |
| 3 | 05/23/1784 | 124 | 2 |
| 4 | 09/03/1790 | 65.5 | 3 |
| 5 | 9/04/2000 | 100 | 4 |

Matching

Not Matching

Full Outer



| order_id | order_date | amount | customer_id | first_name | last_name | email | address | city | state | zipcode |
|----------|------------|--------|-------------|------------|------------|----------------------|---------------------------|-----------------|-------|---------|
| 1 | 07/04/1776 | 234.56 | 1 | George | Washington | g.washington@usa.gov | 3200 Mt Vernon Hwy | Mount Vernon | VA | 22121 |
| 2 | 03/14/1760 | 78.5 | 3 | Thomas | Jefferson | tjefferson@usa.gov | 931 Thomas Jefferson Pkwy | Charlottesville | VA | 22902 |
| 4 | 09/03/1790 | 65.5 | 3 | Thomas | Jefferson | tjefferson@usa.gov | 931 Thomas Jefferson Pkwy | Charlottesville | VA | 22902 |
| 3 | 05/23/1784 | 124 | 2 | John | Adams | j.adams@usa.gov | 1250 Hancock St | Quincy | MA | 2169 |
| | | | | James | Madison | jmadison@usa.gov | 11350 Constitution Hwy | Orange | VA | 22960 |
| | | | | James | Monroe | jmonroe@usa.gov | 2050 James Monroe Parkway | Charlottesville | VA | 22902 |
| 5 | 9/04/2000 | 100 | 4 | | | | | | | |

Matching

Not Matching

| customer_id | first_name | last_name | email | address | city | state | zipcode |
|-------------|------------|------------|----------------------|---------------------------|-----------------|-------|---------|
| 1 | George | Washington | g.washington@usa.gov | 3200 Mt Vernon Hwy | Mount Vernon | VA | 22121 |
| 2 | John | Adams | j.adams@usa.gov | 1250 Hancock St | Quincy | MA | 2169 |
| 3 | Thomas | Jefferson | tjefferson@usa.gov | 931 Thomas Jefferson Pkwy | Charlottesville | VA | 22902 |
| 4 | James | Madison | jmadison@usa.gov | 11350 Constitution Hwy | Orange | VA | 22960 |
| 5 | James | Monroe | jmonroe@usa.gov | 2050 James Monroe Parkway | Charlottesville | VA | 22902 |

This result set, as you can see, is seven rows. There are four rows matching in both tables. 2 rows are only on the customer table, but not in the orders table. One row only in the orders table, but not in the customers' table.

Inner (only matching rows)

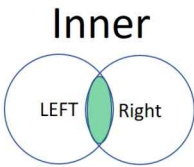
This method only selects matching rows. You will not have any record with null values (because these records were generated as a result of not matching). Here is an example;

Chapter 3: Choose the Right Merge Join Type in Power BI

| 1: order_id | 1: order_date | 1: amount | 2: customer_id |
|-------------|---------------|-----------|----------------|
| 1 | 07/04/1776 | 234.56 | 1 |
| 2 | 03/14/1760 | 78.5 | 3 |
| 3 | 05/23/1784 | 124 | 2 |
| 4 | 09/03/1790 | 65.5 | 3 |
| 5 | 9/04/2000 | 100 | 6 |

Matching

Not Matching



| 1: order_id | 1: order_date | 1: amount | 2: customer_id | 1: first_name | 1: last_name | 1: email | 1: address | 1: city | 1: state | 1: zipcode |
|-------------|---------------|-----------|----------------|---------------|--------------|----------------------|---------------------------|-----------------|----------|------------|
| 1 | 07/04/1776 | 234.56 | 1 | George | Washington | gswashington@usa.gov | 3200 Mt Vernon Hwy | Mount Vernon | VA | 22121 |
| 2 | 03/14/1760 | 78.5 | 3 | Thomas | Jefferson | tjefferson@usa.gov | 931 Thomas Jefferson Pkwy | Charlottesville | VA | 22902 |
| 3 | 05/23/1784 | 124 | 2 | John | Adams | jadams@usa.gov | 1250 Hancock St | Quincy | MA | 2169 |
| 4 | 09/03/1790 | 65.5 | 3 | Thomas | Jefferson | tjefferson@usa.gov | 931 Thomas Jefferson Pkwy | Charlottesville | VA | 22902 |

Matching

Not Matching

| 2: customer_id | 1: first_name | 1: last_name | 1: email | 1: address | 1: city | 1: state | 1: zipcode |
|----------------|---------------|--------------|----------------------|---------------------------|-----------------|----------|------------|
| 1 | George | Washington | gswashington@usa.gov | 3200 Mt Vernon Hwy | Mount Vernon | VA | 22121 |
| 2 | John | Adams | jadams@usa.gov | 1250 Hancock St | Quincy | MA | 2169 |
| 3 | Thomas | Jefferson | tjefferson@usa.gov | 931 Thomas Jefferson Pkwy | Charlottesville | VA | 22902 |
| 4 | James | Madison | jmadison@usa.gov | 11350 Constitution Hwy | Orange | VA | 22960 |
| 5 | James | Monroe | jmonroe@usa.gov | 2050 James Monroe Parkway | Charlottesville | VA | 22902 |

There are only four rows that are matching between two tables. Rows with customer_id 1, 2, and 3. all not matching rows will be excluded from the result set.

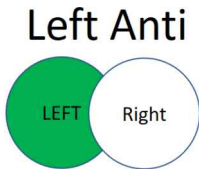
Let Anti (rows only in first)

If you are only interested in rows from the LEFT (first) table, this is the option to select. This means rows that are in the first table and DO NOT match with the second table. So, only Not matching rows from the first table. Anti options always get null for the second data set because these rows don't exist there. Anti options are good for finding rows that exist in one table but not in the other one. Here is an example:

| 1: order_id | 1: order_date | 1: amount | 2: customer_id |
|-------------|---------------|-----------|----------------|
| 1 | 07/04/1776 | 234.56 | 1 |
| 2 | 03/14/1760 | 78.5 | 3 |
| 3 | 05/23/1784 | 124 | 2 |
| 4 | 09/03/1790 | 65.5 | 3 |
| 5 | 9/04/2000 | 100 | 6 |

Matching

Not Matching



| 2: customer_id | 1: order_date | 1: amount | 2: customer_id | 1: first_name | 1: last_name | 1: email | 1: address | 1: city | 1: state | 1: zipcode |
|----------------|---------------|-----------|----------------|---------------|--------------|----------|------------|---------|----------|------------|
| 5 | 9/04/2000 | 100 | 6 | | | | | | | |

| 2: customer_id | 1: first_name | 1: last_name | 1: email | 1: address | 1: city | 1: state | 1: zipcode |
|----------------|---------------|--------------|----------------------|---------------------------|-----------------|----------|------------|
| 1 | George | Washington | gswashington@usa.gov | 3200 Mt Vernon Hwy | Mount Vernon | VA | 22121 |
| 2 | John | Adams | jadams@usa.gov | 1250 Hancock St | Quincy | MA | 2169 |
| 3 | Thomas | Jefferson | tjefferson@usa.gov | 931 Thomas Jefferson Pkwy | Charlottesville | VA | 22902 |
| 4 | James | Madison | jmadison@usa.gov | 11350 Constitution Hwy | Orange | VA | 22960 |
| 5 | James | Monroe | jmonroe@usa.gov | 2050 James Monroe Parkway | Charlottesville | VA | 22902 |

This will find the only row in the Orders table and does not match with any of the rows in the customers' table.

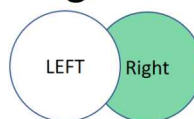
Part 1: Combining data tables

Right Anti (rows only in second)

Similar to Left Anti, this method will give you only not matching rows. However, this time from the second (Right) table. You can find out what rows in the right table are not matching with the left table. Here is the example;

| t1: order_id | A1: order_date | 1.2: amount | t2: customer_id |
|--------------|----------------|-------------|-----------------|
| 1 | 07/04/1776 | | 234.56 |
| 2 | 03/14/1760 | | 78.5 |
| 3 | 05/23/1784 | | 124 |
| 4 | 09/03/1790 | | 65.5 |
| 5 | 9/04/2000 | | 100 |

Right Anti



| t2: order_id | A1: order_date | 1.2: amount | t2: customer_id | A1: first_name | A1: last_name | A1: email | A1: address | A1: city | A1: state | t2: zipcode |
|--------------|----------------|-------------|-----------------|----------------|---------------|------------------|--------------------|-----------------|-----------|-------------|
| null | null | null | null | James | Madison | jmadison@usa.gov | 11350 Constitution | Orange | VA | 22960 |
| null | null | null | null | James | Monroe | jmonroe@usa.gov | 2050 James Monr | Charlottesville | VA | 22902 |

Matching

Not Matching

| t2: customer_id | A1: first_name | A1: last_name | A1: email | A1: address | A1: city | A1: state | t2: zipcode |
|-----------------|----------------|---------------|---------------------|---------------------------|-----------------|-----------|-------------|
| 1 | George | Washington | gWASHINGTON@usa.gov | 3200 Mt Vernon Hwy | Mount Vernon | VA | 22121 |
| 2 | John | Adams | jADAMS@usa.gov | 1250 Hancock St | Quincy | MA | 2169 |
| 3 | Thomas | Jefferson | tJEFFERSON@usa.gov | 931 Thomas Jefferson Pkwy | Charlottesville | VA | 22902 |
| 4 | James | Madison | jMADISON@usa.gov | 11350 Constitution Hwy | Orange | VA | 22960 |
| 5 | James | Monroe | JMONROE@usa.gov | 2050 James Monroe Parkway | Charlottesville | VA | 22902 |

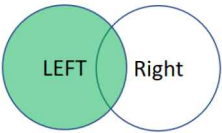
This result set is all rows from the customer table (right table) that do NOT match orders (first table).

Summary

Different Join Kinds in Merge returns a different result set. Make sure to select the right join kind to avoid any issues later. There are six types of joins as below;

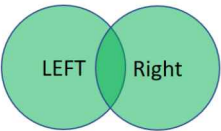
- Left Outer: Rows from left table and matching with the right
- Right Outer: Rows from right table and matching with the left
- Full Outer: Rows from both tables (matching or not matching)
- Inner: Only matching rows from both tables
- Left Anti: Not matching rows from the left table
- Right Anti: Not matching rows from the right table

LEFT Outer



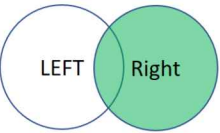
All rows from left and matching from right

Full Outer



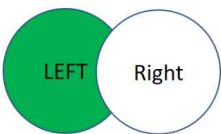
All rows from both: matching and not matching

RIGHT Outer



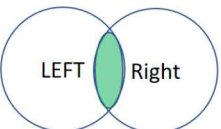
All rows from right and matching from left

Left Anti



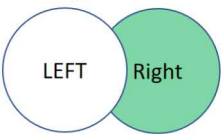
Not matching rows from left

Inner



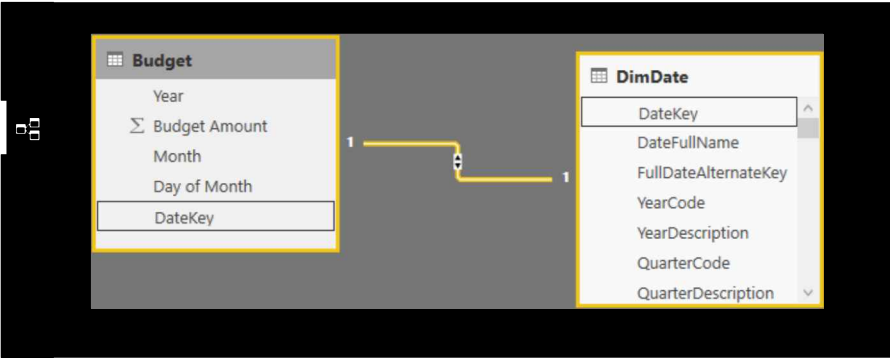
Only matching rows

Right Anti



Not matching rows from right

Chapter 4: Relationship in Power BI with Multiple Columns



You can create relationships in Power BI between tables. Relationships are useful for some functions to work across multiple tables and produce the result. The relationship between tables also makes visualization and report elements more efficient because the selection in one chart can affect another chart from a different table. However, there is a limitation in the Power BI relationship: you can't create relationships based on more than one column. In other words, if you want to create a relationship (or join two tables) with two or more columns, you cannot! Fortunately, there is a workaround that I'll explain in this chapter.

Defining the Problem

Assume that we have a budget table with the fiscal year, fiscal period, and budget amount. Here is a screenshot of this table:

| Year | Month | Budget Amount |
|------|-------|---------------|
| 2010 | Mth1 | 50000 |
| 2010 | Mth2 | 50000 |
| 2010 | Mth3 | 50000 |
| 2010 | Mth4 | 45000 |
| 2010 | Mth5 | 50000 |
| 2010 | Mth6 | 50000 |
| 2010 | Mth7 | 60000 |
| 2010 | Mth8 | 50000 |
| 2010 | Mth9 | 50000 |
| 2010 | Mth10 | 50000 |
| 2010 | Mth11 | 50000 |
| 2010 | Mth12 | 50000 |
| 2011 | Mth1 | 50000 |
| 2011 | Mth2 | 55000 |
| 2011 | Mth3 | 55000 |
| 2011 | Mth4 | 65000 |
| 2011 | Mth5 | 55000 |

Chapter 4: Relationship in Power BI with Multiple Columns

If I want to do date analysis and date-based calculations, it is best to create a relationship between the budget table and a date dimension. Here is a view of my date dimension:

| crOffYear | ISOWeekYearCode | WeekDay | WeekDayName | NZFiscalYearCode | NZFiscalYearDescription | NZFiscalQuarterCode | NZFiscalQuarterYearCode | NZFiscalQuarterDescription | NZFiscalMonthCode | NZFiscalMonthYearCode | NZFiscalMonth |
|-----------|-----------------|---------|-------------|------------------|-------------------------|---------------------|-------------------------|----------------------------|-------------------|-----------------------|---------------|
| 27 | 199027 | 3 | Tuesday | 1991 | FY 1991 | 2 | 19912 | QTR 2 | 4 | 199104 | Month 4 |
| 28 | 199028 | 3 | Tuesday | 1991 | FY 1991 | 2 | 19912 | QTR 2 | 4 | 199104 | Month 4 |
| 29 | 199029 | 3 | Tuesday | 1991 | FY 1991 | 2 | 19912 | QTR 2 | 4 | 199104 | Month 4 |
| 30 | 199030 | 3 | Tuesday | 1991 | FY 1991 | 2 | 19912 | QTR 2 | 4 | 199104 | Month 4 |
| 31 | 199031 | 3 | Tuesday | 1991 | FY 1991 | 2 | 19912 | QTR 2 | 4 | 199104 | Month 4 |
| 32 | 199032 | 3 | Tuesday | 1991 | FY 1991 | 2 | 19912 | QTR 2 | 5 | 199105 | Month 5 |
| 33 | 199033 | 3 | Tuesday | 1991 | FY 1991 | 2 | 19912 | QTR 2 | 5 | 199105 | Month 5 |
| 34 | 199034 | 3 | Tuesday | 1991 | FY 1991 | 2 | 19912 | QTR 2 | 5 | 199105 | Month 5 |
| 35 | 199035 | 3 | Tuesday | 1991 | FY 1991 | 2 | 19912 | QTR 2 | 5 | 199105 | Month 5 |
| 27 | 199127 | 3 | Tuesday | 1992 | FY 1992 | 2 | 19922 | QTR 2 | 4 | 199204 | Month 4 |
| 28 | 199128 | 3 | Tuesday | 1992 | FY 1992 | 2 | 19922 | QTR 2 | 4 | 199204 | Month 4 |
| 29 | 199129 | 3 | Tuesday | 1992 | FY 1992 | 2 | 19922 | QTR 2 | 4 | 199204 | Month 4 |
| 30 | 199130 | 3 | Tuesday | 1992 | FY 1992 | 2 | 19922 | QTR 2 | 4 | 199204 | Month 4 |
| 31 | 199131 | 3 | Tuesday | 1992 | FY 1992 | 2 | 19922 | QTR 2 | 4 | 199204 | Month 4 |
| 32 | 199132 | 3 | Tuesday | 1992 | FY 1992 | 2 | 19922 | QTR 2 | 5 | 199205 | Month 5 |
| 33 | 199133 | 3 | Tuesday | 1992 | FY 1992 | 2 | 19922 | QTR 2 | 5 | 199205 | Month 5 |
| 34 | 199134 | 3 | Tuesday | 1992 | FY 1992 | 2 | 19922 | QTR 2 | 5 | 199205 | Month 5 |
| 35 | 199135 | 3 | Tuesday | 1992 | FY 1992 | 2 | 19922 | QTR 2 | 5 | 199205 | Month 5 |
| 28 | 199228 | 3 | Tuesday | 1993 | FY 1993 | 2 | 19932 | QTR 2 | 4 | 199304 | Month 4 |

To join these two tables, I can add a day column to the budget table and then join them based on three columns: fiscal year, period, and day (month). So here is the budget table with the new day column added and month value a bit polished to remove “Mth” from the month label;

| | Year | Month | Budget Amount | Day of Month |
|----|------|-------|---------------|--------------|
| 1 | 2010 | 1 | 50000 | 1 |
| 2 | 2010 | 2 | 50000 | 1 |
| 3 | 2010 | 3 | 50000 | 1 |
| 4 | 2010 | 4 | 45000 | 1 |
| 5 | 2010 | 5 | 50000 | 1 |
| 6 | 2010 | 6 | 50000 | 1 |
| 7 | 2010 | 7 | 60000 | 1 |
| 8 | 2010 | 8 | 50000 | 1 |
| 9 | 2010 | 9 | 50000 | 1 |
| 10 | 2010 | 10 | 50000 | 1 |
| 11 | 2010 | 11 | 50000 | 1 |
| 12 | 2010 | 12 | 50000 | 1 |
| 13 | 2011 | 1 | 50000 | 1 |
| 14 | 2011 | 2 | 55000 | 1 |
| 15 | 2011 | 3 | 55000 | 1 |

If I want to create a relationship between the date dimension and budget table based on these three columns, I cannot! The create relationship dialog doesn’t allow me to select multiple columns.

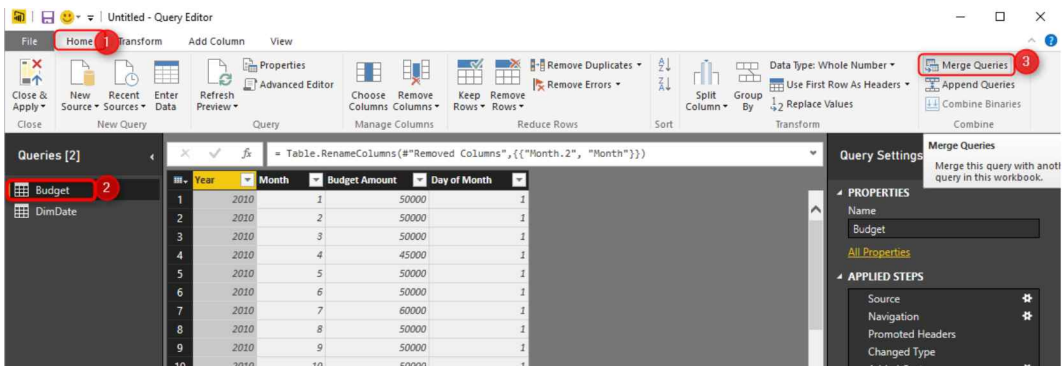
Workaround

The workaround for this problem is easy. Power BI doesn’t allow relationships in the model based on multiple columns, but Power Query can join tables with as many columns as possible. So what I can do as a workaround is to join the budget table to date dimension in Power Query and fetch the date key. Then I’ll use the date key as a single field relationship in Power BI modeling section.

First, I open Merge Queries from the Combine section of the Home tab;



Part 1: Combining data tables



Here is how I join two tables based on multiple columns: I can hold the CTRL key and select columns one by one (in the right order of joining)

Merge

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

Budget

| Year | 1 | Month | 2 | Budget Amount | Day of Month | 3 |
|------|---|-------|---|---------------|--------------|---|
| 2010 | | 1 | | 50000 | | 1 |
| 2010 | | 2 | | 50000 | | 1 |
| 2010 | | 3 | | 50000 | | 1 |
| 2010 | | 4 | | 45000 | | 1 |
| 2010 | | 5 | | 50000 | | 1 |

DimDate

| Name | NZFiscalYearCode | 1 | NZFiscalMonthCode | 2 | DayNumberOfMonth | 3 | NZFiscalMonthYearCode | NZFiscalYear |
|------|------------------|---|-------------------|---|------------------|---|-----------------------|--------------|
| | 1990 | | 10 | | 1 | | 199010 | FY 1990 |
| | 1990 | | 10 | | 2 | | 199010 | FY 1990 |
| | 1990 | | 10 | | 3 | | 199010 | FY 1990 |
| | 1990 | | 10 | | 4 | | 199010 | FY 1990 |
| | 1990 | | 10 | | 5 | | 199010 | FY 1990 |

Join Kind

Left Outer (all from first, matching from second)

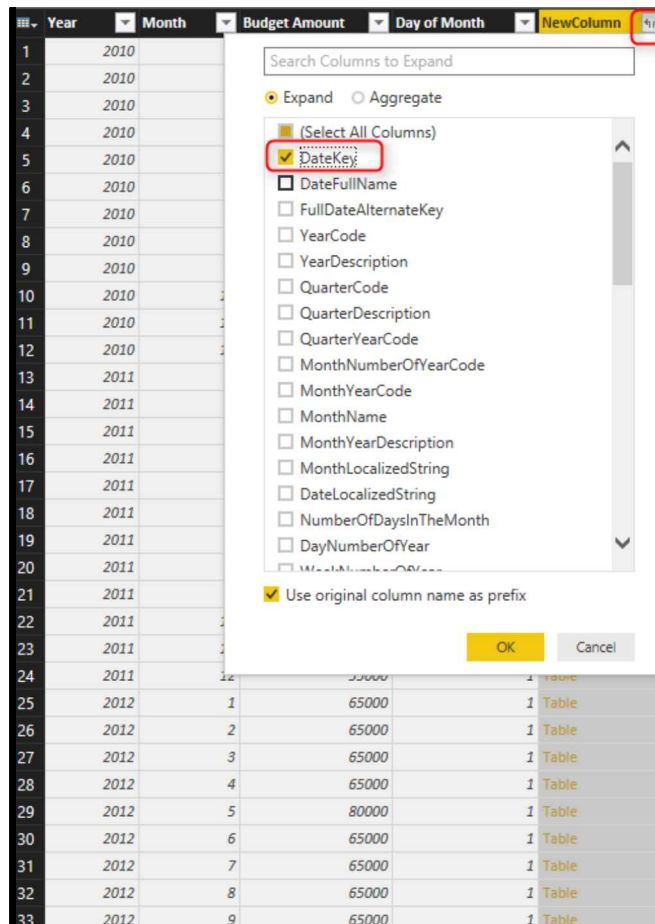
The selection has matched 48 out of the first 48 rows.

OK

Cancel

Then I'll have the new table embedded as a result of the join;

Chapter 4: Relationship in Power BI with Multiple Columns

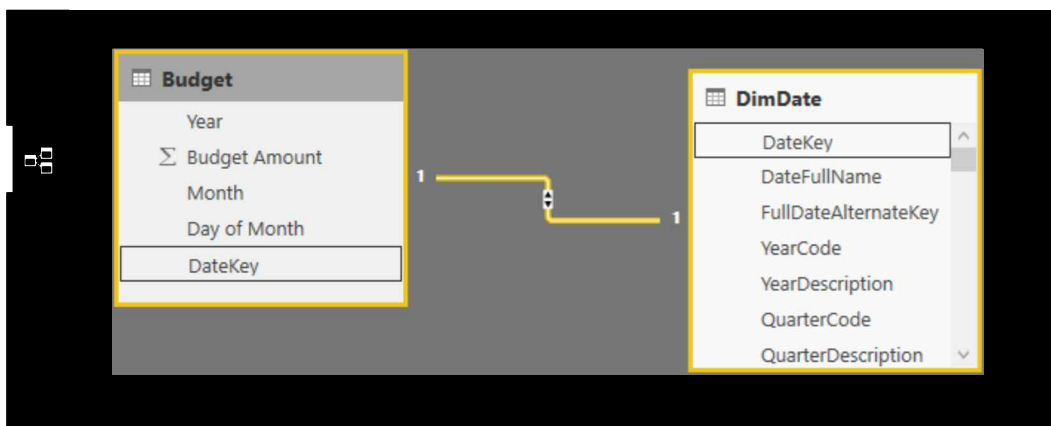


So I'll just pick the key field from the embedded table;

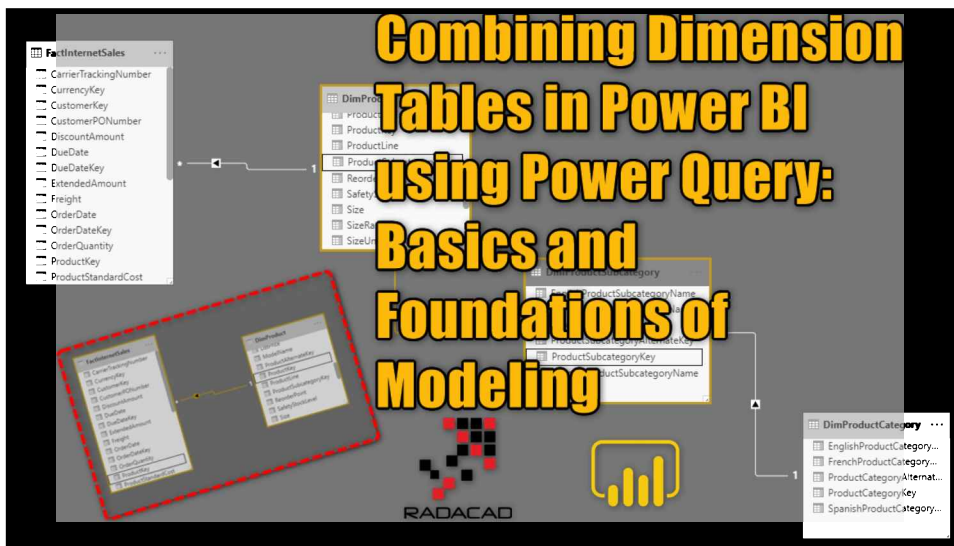
| | Year | Month | Budget Amount | Day of Month | DateKey |
|----|------|-------|---------------|--------------|----------|
| 1 | 2010 | 1 | 50000 | 1 | 20090401 |
| 2 | 2010 | 2 | 50000 | 1 | 20090501 |
| 3 | 2010 | 3 | 50000 | 1 | 20090601 |
| 4 | 2010 | 4 | 45000 | 1 | 20090701 |
| 5 | 2010 | 5 | 50000 | 1 | 20090801 |
| 6 | 2010 | 6 | 50000 | 1 | 20090901 |
| 7 | 2010 | 7 | 60000 | 1 | 20091001 |
| 8 | 2010 | 8 | 50000 | 1 | 20091101 |
| 9 | 2010 | 9 | 50000 | 1 | 20091201 |
| 10 | 2010 | 10 | 50000 | 1 | 20100101 |
| 11 | 2010 | 11 | 50000 | 1 | 20100201 |
| 12 | 2010 | 12 | 50000 | 1 | 20100301 |
| 13 | 2011 | 1 | 50000 | 1 | 20100401 |

And after save and closing the query editor window, I can create a relationship in the Power BI model based on a single column;

Part 1: Combining data tables



Chapter 5: Combining Dimension Tables in Power BI using Power Query



Is it good to have too many dimension tables? Can you combine some of those tables to build one flattened dimension table? How much should you flatten it? Should you end up with one huge table, including everything? In this chapter, I'm answering all of these questions and explaining the scenarios of combining dimensions.

Sample Dataset

The dataset for this model is the AdventureWorksDW2012.xlsx Excel file. For this example, we need these tables: DimProduct, DimProductCategory, DimProductSubcategory, and FactInternetSales.

In the dataset, we have three tables related to Product information; DimProduct has the details of every single product;

Part 1: Combining data tables

| ProductKey | ProductAlternateKey | ProductSubcategoryKey | WeightUnitMeasureCode | SizeUnitMeasureCode | EnglishProductName | Spanish |
|------------|---------------------|-----------------------|-----------------------|---------------------|-----------------------|---------|
| 1 | AR-5381 | | null | null | Adjustable Race | |
| 2 | BA-8327 | | null | null | Bearing Ball | |
| 3 | BE-2349 | | null | null | BB Ball Bearing | |
| 4 | BE-2908 | | null | null | Headset Ball Bearings | |
| 5 | BL-2036 | | null | null | Blade | |
| 6 | CA-5965 | | null | null | LL Crankarm | |
| 7 | CA-6738 | | null | null | ML Crankarm | |
| 8 | CA-7457 | | null | null | HL Crankarm | |
| 9 | CB-2903 | | null | null | Chainring Bolts | |
| 10 | CN-6137 | | null | null | Chainring Nut | |
| 11 | CR-7833 | | null | null | Chainring | |
| 12 | CR-9981 | | null | null | Crown Race | |
| 13 | CS-2812 | | null | null | Chain Stays | |
| 14 | DC-8732 | | null | null | Decal 1 | |
| 15 | DC-9824 | | null | null | Decal 2 | |
| 16 | DT-2377 | | null | null | Down Tube | |
| 17 | EC-M092 | | null | null | Mountain End Caps | |
| 18 | EC-R098 | | null | null | Road End Caps | |
| 19 | EC-T209 | | null | null | Touring End Caps | |
| 20 | FE-3760 | | null | null | Fork End | |
| 21 | FH-2981 | | null | null | Freewheel | |
| 22 | FW-1000 | | null | null | Flat Washer 1 | |

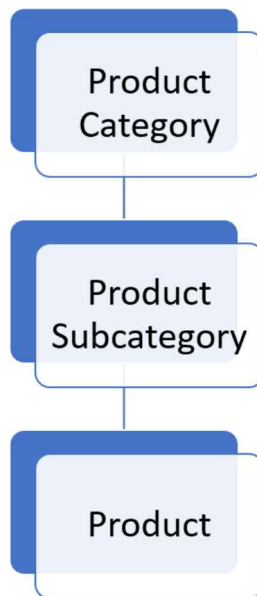
DimProductCategory has information about all categories of products;

| ProductCategoryKey | ProductCategoryAlternateKey | EnglishProductCategoryName | SpanishProductCategoryName | FrenchProductCategoryName |
|--------------------|-----------------------------|----------------------------|----------------------------|---------------------------|
| 1 | | Bikes | Bicicleta | Vélo |
| 2 | | Components | Componente | Composant |
| 3 | | Clothing | Prenda | Vêtements |
| 4 | | Accessories | Accesorio | Accessoire |

DimProductSubCategory is the second level of the hierarchy, where we have subcategories under each category;

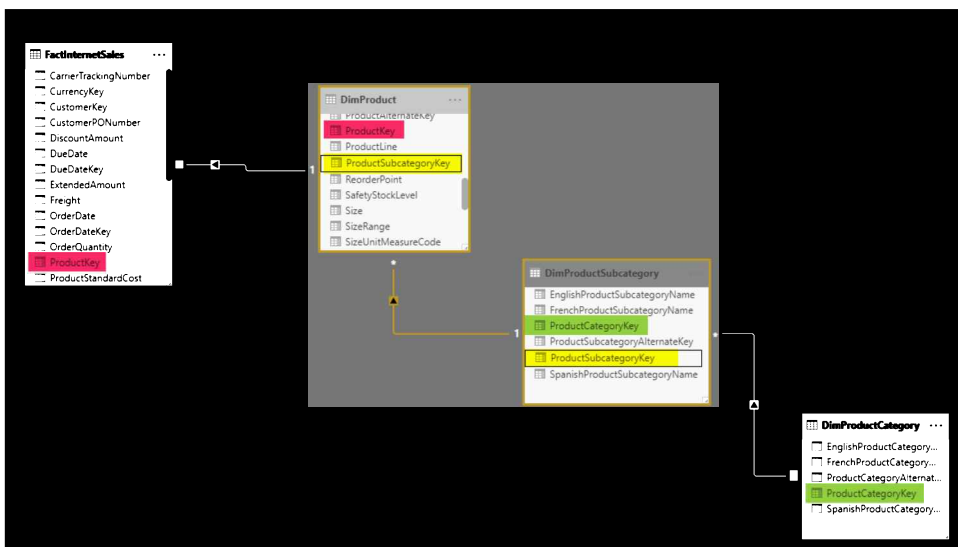
| ProductSubcategoryKey | ProductSubcategoryAlternateKey | EnglishProductSubcategoryName | SpanishProductSubcategoryName | FrenchProductSubcategoryName | ProductCategoryKey |
|-----------------------|--------------------------------|-------------------------------|-------------------------------|------------------------------|--------------------|
| 1 | | Mountain Bikes | Bicicleta de montaña | VTT | 1 |
| 2 | | Road Bikes | Bicicleta de carretera | Vélo de route | 1 |
| 3 | | Touring Bikes | Bicicleta de paseo | Vélo de randonnée | 1 |
| 4 | | Handlebars | Barra | Barre d'appui | 2 |
| 5 | | Bottom Brackets | Eje de pedalier | Axe de pédalier | 2 |
| 6 | | Brakes | Frenos | Freins | 2 |
| 7 | | Chains | Cadena | Chaîne | 2 |
| 8 | | Cranksets | Bielas | Pédalier | 2 |
| 9 | | Derailleurs | Desviador | Dérailleur | 2 |
| 10 | | Forks | Horquilla | Fourche | 2 |
| 11 | | Headsets | Dirección | Jeu de direction | 2 |
| 12 | | Mountain Frames | Cuadro de montaña | Cadre de VTT | 2 |
| 13 | | Pedals | Pedal | Pédale | 2 |
| 14 | | Road Frames | Cuadro de carretera | Cadre de vélo de route | 2 |
| 15 | | Saddles | Sillin | Selle | 2 |
| 16 | | Touring Frames | Cuadro de paseo | Cadre de vélo de randonnée | 2 |
| 17 | | Wheels | Rueda | Roue | 2 |
| 18 | | Bit-Shorts | Culote corto | Cuissards avec bretelles | 3 |
| 19 | | Caps | Gorra | Casquette | 3 |
| 20 | | Gloves | Gaantes | Gants | 3 |
| 21 | | Jerseys | Jersey | Maillot | 3 |
| 22 | | Shorts | Pantalones cortos | Cuissards | 3 |
| 23 | | Socks | Calcetines | Chaussettes | 3 |
| 24 | | Tights | Mallas | Collants | 3 |
| 25 | | Vests | Camiseta | Veste | 3 |
| 26 | | Bike Bags | Portabicietas | Porte-vélos | 4 |

As you can see, every record in the subcategory table belongs to a category, or let's say, have a value in their ProductCategoryKey column, which is then the key to find the higher level category in the DimProductCategory table. If we want to design a hierarchy of Products, this is how it would look like;



Design Challenge

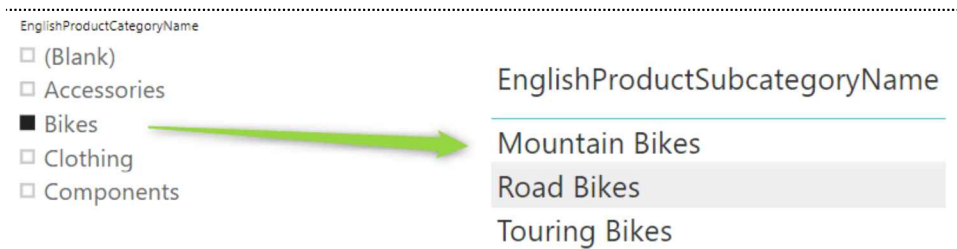
Let's say we load all of the tables above (plus the FactInternetSale table, which is our transactional sales table) into a Power BI model. This is how the relationship diagram looks like:



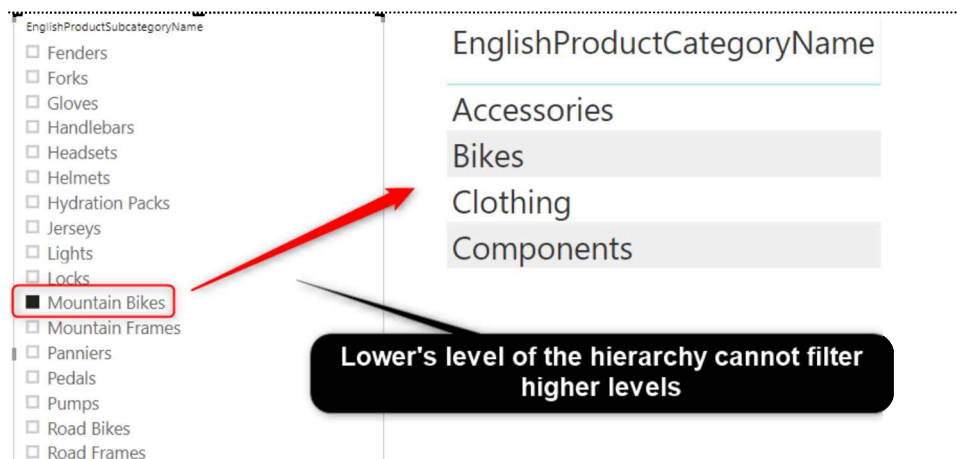
Note that Power BI automatically creates a couple of these relationships. Still, you need to create the highlighted yellow yourself, connect ProductSubcategoryKey from DimProduct to ProductSubcategoryKey in the DimProductSubcategory table.

The Need for Both-Directional Relationship Sometimes

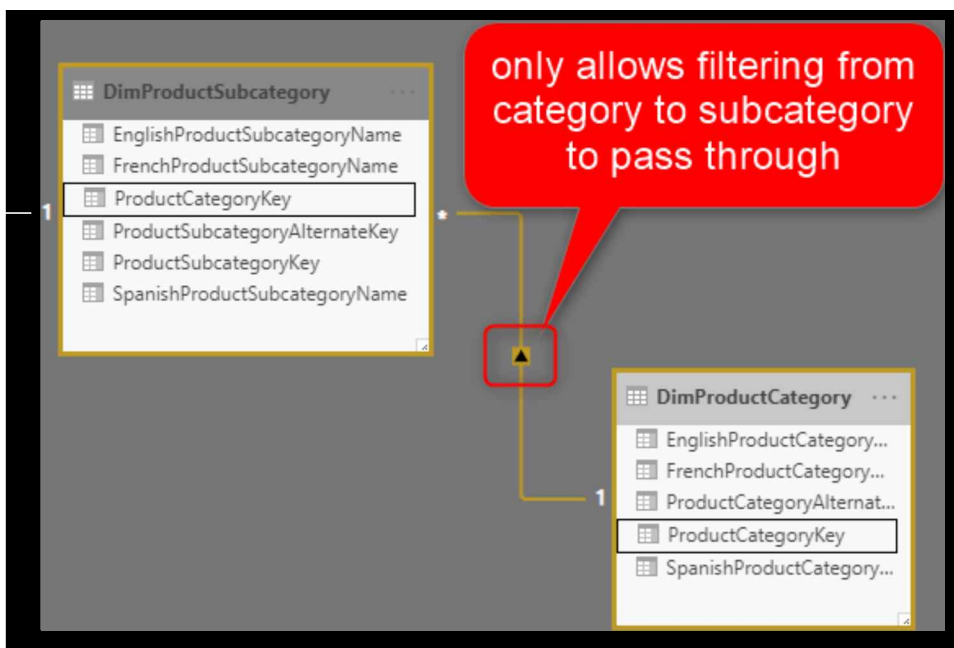
As you can see in the above diagram, the directions of filters are from category to subcategory, then to the product, and finally to the fact table. If we filter data in the fact table using a column from any of these three tables, then we get slicing and dicing working perfectly; higher levels of hierarchy filters lower levels of it easily as below;



However, what about this: we want to filter the higher level of the hierarchy using the lower level! For example, I want to know the product category that Mountain bikes belong to is?



As you can see, the visual above isn't helpful. Product Subcategory cannot filter the ProductCategory table, and the reason is obvious; the relationship is single directional between these tables from the category to the subcategory.



Depending on the requirements, the design above leads to the need for both-directional relationships, which is not recommended.

Creating a Hierarchy of all three tables

You want to make the report design consistent and create a hierarchy of all of these three fields (category, subcategory, and product) and then use it in all visuals.

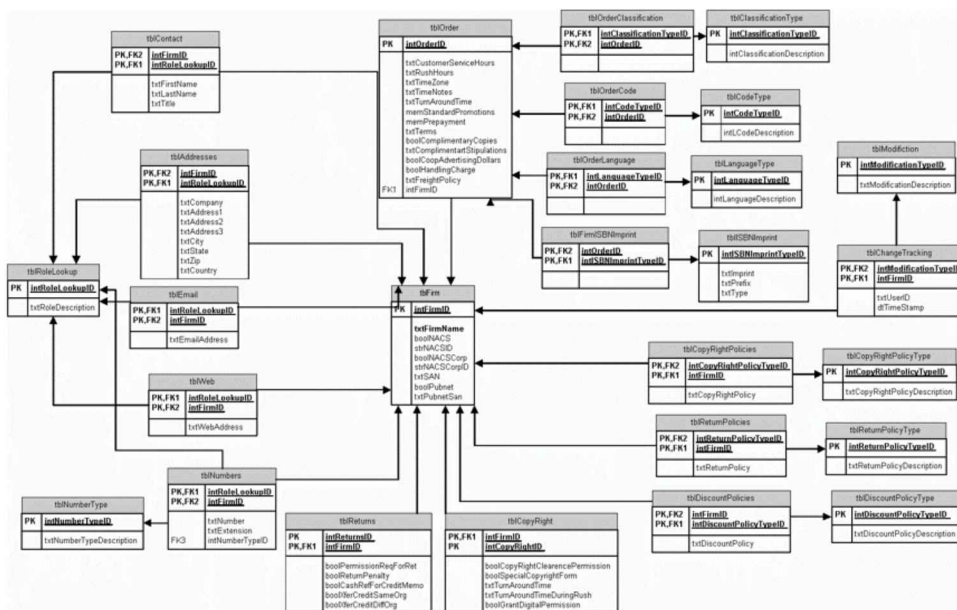
Note that it is possible to create a hierarchy of fields directly in each visual by dragging and dropping fields after each other. However, creating the hierarchy in the model makes this process more consistent, and then you can just drag the hierarchy in every visual you want.

The challenge is that you cannot create a model hierarchy between fields not on the same table! You may need to create calculated columns in the Product table using the RELATED DAX function to achieve this process. This is so much process to achieve something really simple.

You cannot create model hierarchies between fields that come from multiple tables.

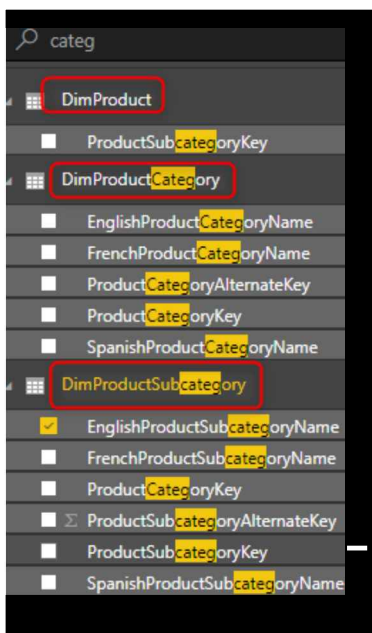
Extra Relationships! Why?!

The next challenge of the existing model is that we have so many relationships to cover something really simple. In this model so far, we have three relationships. In real-world scenarios, you don't have only four tables, you will have hundreds of tables, and if you follow this approach, you will have hundreds of relationships! Relationships will cost processing power when it comes to slicing and dicing. You won't notice it for small models with a few tables, but you will see the difference when the model grows. If you don't think much about your data model and just add tables and relationships to it, soon you will end up with a model much more complicated than this:



The Model is Confusing

Having more tables in your model will make it more confusing on the reporting side. There are too many tables having product information on it! Why such confusion for such a simple model?!



Where is this designing concept coming from?

So the design above has many challenges. Let's see where the design concept is coming from? The design you see, one table per attribute or function, comes from transactional database design. In the world of transactional databases, it is important to design the database to apply CRUD (Create, Update, Retrieve, and Delete) operations easily and fast. And one of the best ways of doing that is to separate each entity as a table; category as a table, subcategory as another table, the color of the product as a table, brand as another table, etc. in such a design, you will have probably more than ten tables storing different pieces of information about the product itself. That design is great for transactional systems, but it is not working for reporting systems. You need a different design for reporting.

Flattening Dimension Tables

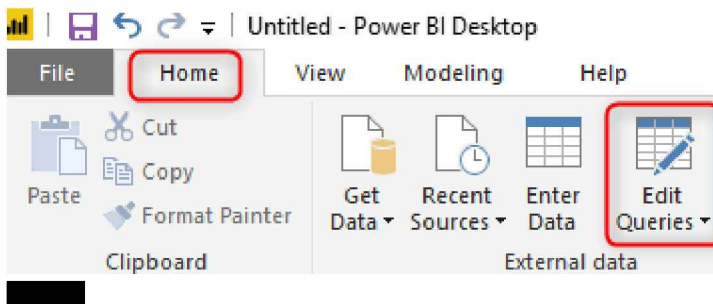
In a reporting data model, your approach should be designing a star schema and having a fair amount of dimension tables. Your dimension tables can include everything about that entity. For example, if you have a Product dimension, it can include color, brand, size, product name, subcategory, and category of the product, all in one table. This way of design will avoid all challenges and issues I mentioned earlier in this chapter and many other challenges.

So the solution to our design challenge is to flatten the Product table by combining DimProductCategory, and DimProductSubCategory with it. Let's see how it is possible;

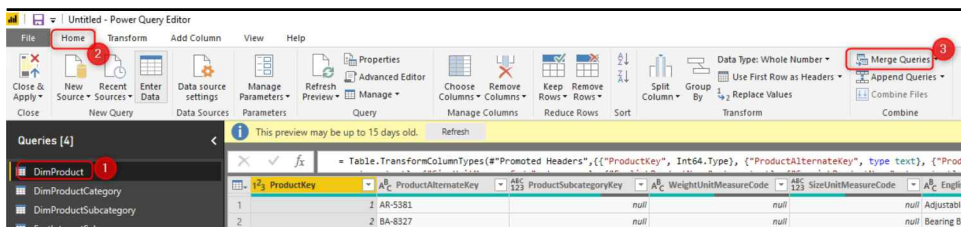
Combining or Flattening tables into one Dimension

There are different methods you can do this flattening process. You can flatten it using T-SQL if you have relational databases as a source. You can use DAX to do it or use Power Query, which is the method I am explaining here.

Go to Edit Queries in the Power BI Desktop;



In the Power Query Editor window, click on Product table, and from the Combine section, select Merge Queries;



In the Merge Queries window, check that the DimProduct is the first table, select ProductSubcategoryKey from it, select DimProductSubcategory as the second table, and the ProductSubcategoryKey column in there too, and then click on OK.

Merge

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

DimProduct

1

2

| ProductKey | ProductAlternateKey | ProductSubcategoryKey | WeightUnitMeasureCode | SizeUnitMeasureCode |
|------------|---------------------|-----------------------|-----------------------|---------------------|
| 1 | AR-5381 | null | null | null |
| 2 | BA-8327 | null | null | null |
| 3 | BE-2349 | null | null | null |
| 4 | BE-2908 | null | null | null |
| 5 | BL-2036 | null | null | null |

DimProductSubcategory

3

4

| ProductSubcategoryKey | ProductSubcategoryAlternateKey | EnglishProductSubcategoryName | SpanishProductSubcategoryName |
|-----------------------|--------------------------------|-------------------------------|-------------------------------|
| 1 | 1 | Mountain Bikes | Bicicleta de montaña |
| 2 | 2 | Road Bikes | Bicicleta de carretera |
| 3 | 3 | Touring Bikes | Bicicleta de paseo |
| 4 | 4 | Handlebars | Barra |
| 5 | 5 | Bottom Brackets | Eje de pedalier |

Join Kind

Left Outer (all from first, matching from second)

☐ Use fuzzy matching to perform the merge

Fuzzy merge options

The selection has matched 397 out of the first 606 rows.

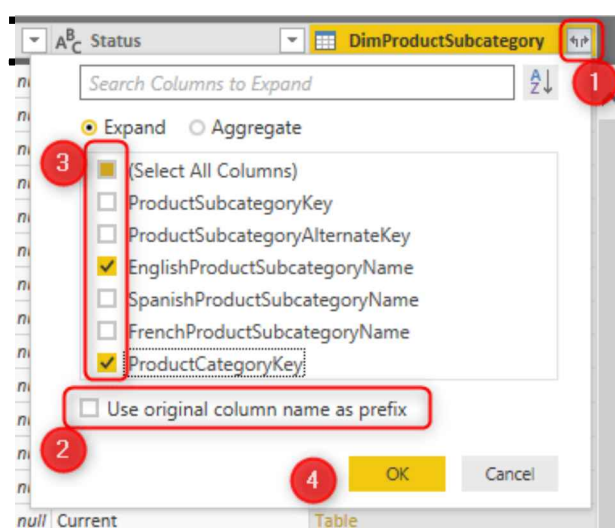
5

OK

Cancel

A merge table is an operation that flattens two tables based on matching field(s). In this operation, the default join kind would work for us. The Merge operation will create a new column in the DimProduct table (at the end of all columns), which then can be expanded into details coming from DimProductSubcategory. This way, we are combining product and subcategory;

Part 1: Combining data tables



I only selected EnglishProductSubcategoryName, and ProductCategoryKey (this one is needed to merge it at the next step with DimProductCategory). Here is the result;

| ABC 123 EndDate | ABC Status | ABC EnglishProductSubcategoryName | 123 ProductCategoryKey |
|-----------------|------------|-----------------------------------|------------------------|
| 1/1/2000 | Current | null | null |
| 1/1/2000 | Current | null | null |
| 1/1/2000 | Current | null | null |
| 1/1/2000 | Current | null | null |
| 1/1/2000 | Current | null | null |
| 1/1/2000 | Current | null | null |
| 1/1/2000 | Current | null | null |
| 1/1/2000 | Current | null | null |
| 1/1/2000 | Current | null | null |
| 1/1/2000 | Current | null | null |

As you can see, the two columns above are now part of the Product table.

We do the Merge Operation one more time, and this time will select DimProductCategory as the second table using the ProductCategoryKey in both tables to merge.

Merge

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

DimProduct

1

| escription | StartDate | EndDate | Status | EnglishProductSubcategoryName | ProductCategoryKey |
|------------|-----------------------|---------|---------|-------------------------------|--------------------|
| null | 1/06/1998 12:00:00 AM | null | Current | null | null |
| null | 1/06/1998 12:00:00 AM | null | Current | null | null |
| null | 1/06/1998 12:00:00 AM | null | Current | null | null |
| null | 1/06/1998 12:00:00 AM | null | Current | null | null |
| null | 1/06/1998 12:00:00 AM | null | Current | null | null |

2

DimProductCategory

3

| ProductCategoryKey | ProductCategoryAlternateKey | EnglishProductCategoryName | SpanishProductCategoryName |
|--------------------|-----------------------------|----------------------------|----------------------------|
| 1 | 1 | Bikes | Bicicleta |
| 2 | 2 | Components | Componente |
| 3 | 3 | Clothing | Prenda |
| 4 | 4 | Accessories | Accesorio |

4

Join Kind

Left Outer (all from first, matching from second)

☐ Use fuzzy matching to perform the merge

Fuzzy merge options

5

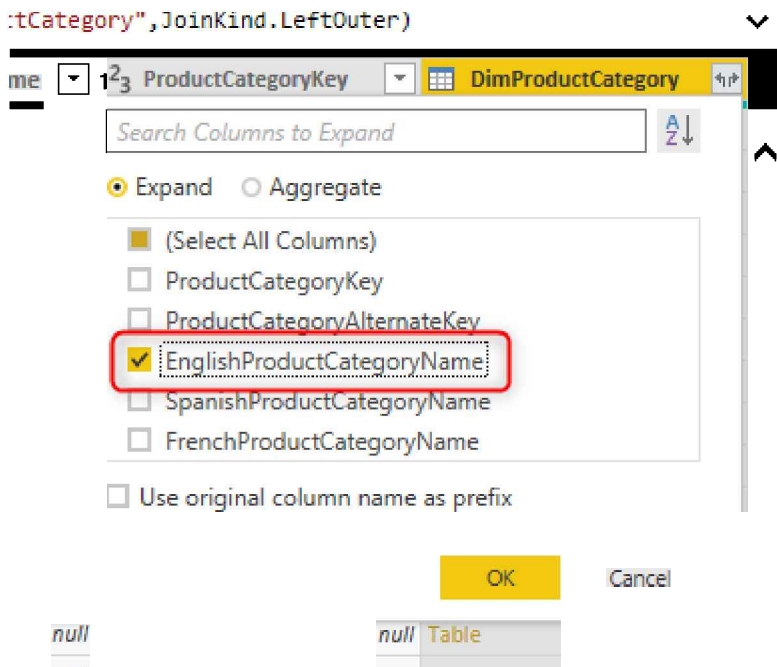
The selection has matched 397 out of the first 606 rows.

OK

Cancel

And then expanding it to the details of DimProductCategory, which in this case is only ProductSubcategoryName

Part 1: Combining data tables

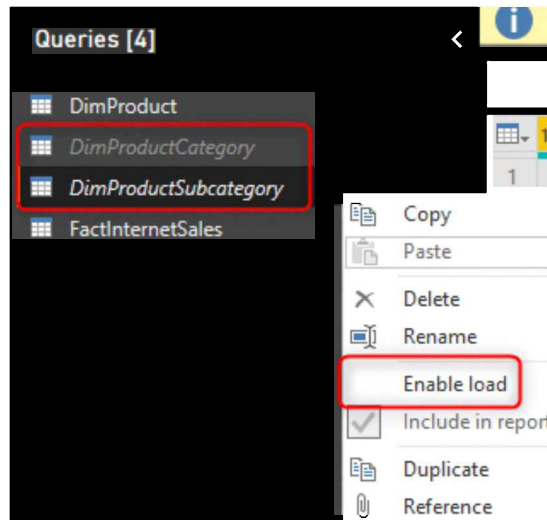


Finally, we will have all category and subcategory details in the DimProduct as below. After the second merge, I also deleted the ProductCategoryKey column, which is not needed anymore.

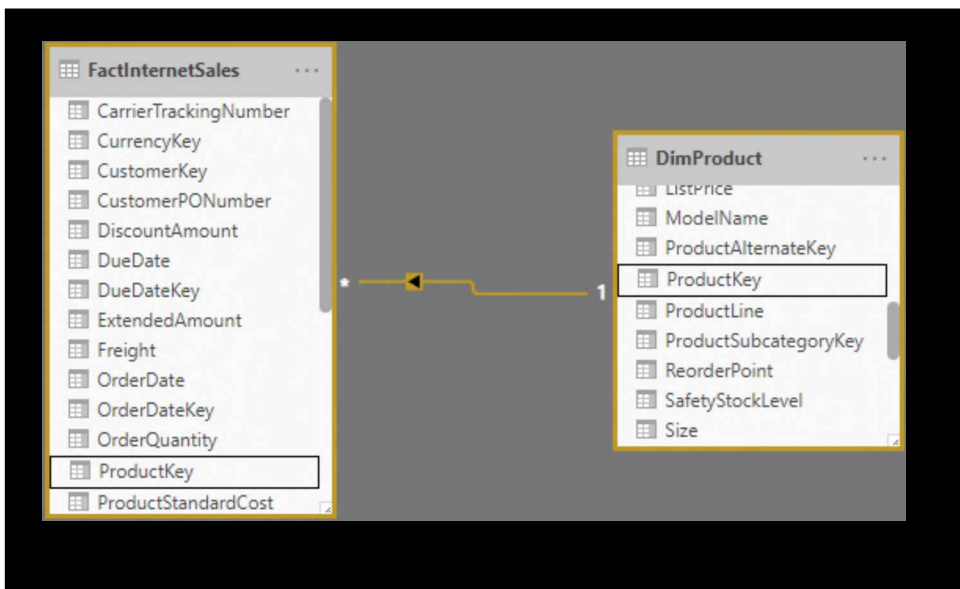
| StartDate | EndDate | Status | EnglishProductSubcategoryName | EnglishProductCategoryName |
|-----------------------|------------------------|---------|-------------------------------|----------------------------|
| 1/07/2006 12:00:00 AM | 30/06/2003 12:00:00 AM | Current | Jerseys | Clothing |
| 1/07/2007 12:00:00 AM | 30/06/2002 12:00:00 AM | Current | Jerseys | Clothing |
| 1/07/2005 12:00:00 AM | 30/06/2002 12:00:00 AM | Current | Jerseys | Clothing |
| 1/07/2006 12:00:00 AM | 30/06/2003 12:00:00 AM | Current | Jerseys | Clothing |
| 1/07/2007 12:00:00 AM | 30/06/2002 12:00:00 AM | Current | Jerseys | Clothing |
| 1/07/2005 12:00:00 AM | 30/06/2002 12:00:00 AM | Current | Jerseys | Clothing |
| 1/07/2006 12:00:00 AM | 30/06/2003 12:00:00 AM | Current | Jerseys | Clothing |
| 1/07/2007 12:00:00 AM | 30/06/2002 12:00:00 AM | Current | Jerseys | Clothing |
| 1/07/2005 12:00:00 AM | 30/06/2002 12:00:00 AM | Current | Road Frames | Components |
| 1/07/2006 12:00:00 AM | 30/06/2003 12:00:00 AM | Current | Road Frames | Components |
| 1/07/2007 12:00:00 AM | 30/06/2002 12:00:00 AM | Current | Road Frames | Components |
| 1/07/2005 12:00:00 AM | 30/06/2002 12:00:00 AM | Current | Road Frames | Components |
| 1/07/2006 12:00:00 AM | 30/06/2003 12:00:00 AM | Current | Road Frames | Components |
| 1/07/2007 12:00:00 AM | 30/06/2002 12:00:00 AM | Current | Road Frames | Components |
| 1/07/2005 12:00:00 AM | 30/06/2002 12:00:00 AM | Current | Road Frames | Components |

Do not load intermediate tables

It is important to set the “enable load” property of the two intermediate tables (DimProductCategory and DimProductSubcategory) to unchecked now. These two tables are not needed to be loaded into the Power BI model directly. They feed the data into the DimProduct, which is the only table we need in the model.

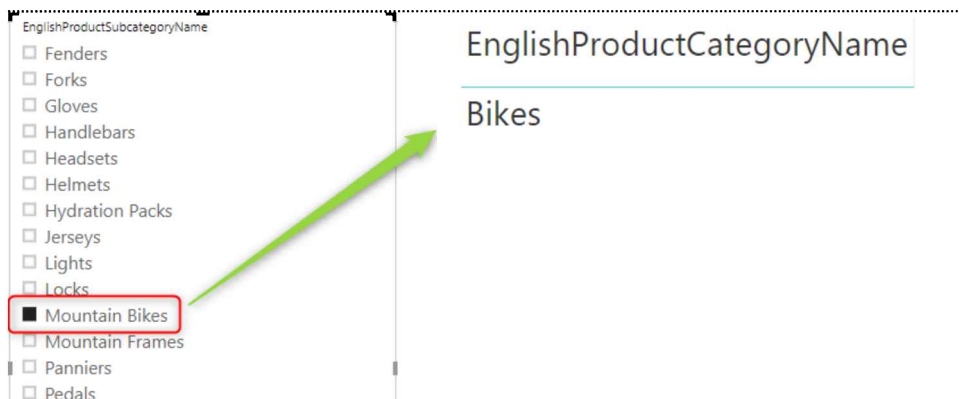


Now, if you load the data into the Power BI model, you will have a simple model such as below;

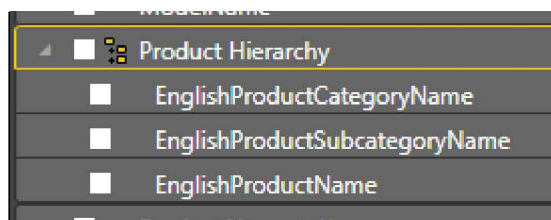


You can easily get slicing and dicing working correctly;

Part 1: Combining data tables

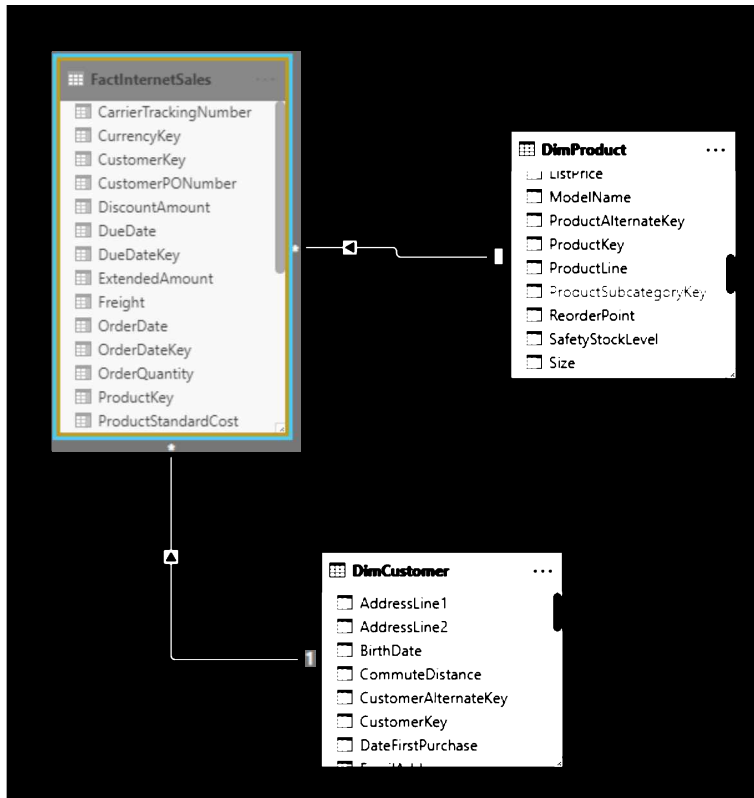


You can also build a model hierarchy and use it everywhere you want in the report;



How about other tables? Should I flatten all dimensions into one?

The method above was helpful, so you might think that let's do that for other tables. Why not combining customers and products as an example! Flattening has a golden rule; Flatten it as long as entities have a meaningful relationship with each other! The product category and product have a meaningful relationship. Every product belongs to a category. However, products and customers don't have a relationship unless there are sales transactions made! We already have a sales transaction table, which is the heart of this model, and flattening dimension tables into that, will make it huge. Also, it will reduce our flexibility if we want to connect another table later on to the product or customer. So in this scenario, we will not flatten customers and products. We will keep them as their dimensions, with a relationship to the fact table, building a star schema.



Summary

In summary, flattening dimension tables will avoid many challenges in the future in the reporting system. It is recommended to flatten attributes into one big dimension for each entity. The product dimension can contain all the colors, sizes, brands, product numbers, categories, etc. The Customer dimension can have everything about the customer's job title, education, age, etc. Through this chapter, you learned some of the challenges when dimensions are not flattened, and you learned how you could use the Power Query Merge command to combine them.

This chapter and the very next chapter in this book are examples of data modeling. Data modeling is an essential skill for every Power BI Developer. To learn more about data modeling, I strongly recommend reading my book on the Basics of Data Modeling [here\[https://www.amazon.com/gp/product/B08HWNZ7GC\]](https://www.amazon.com/gp/product/B08HWNZ7GC).

Chapter 6: Creating a Shared Dimension in Power BI Using Power Query

| Product | Date | Cost |
|--------------|---------------------------|--------------|
| PN 10 | Tuesday, 3 December 2019 | 856 |
| PN 12 | Saturday, 2 February 2019 | 465 |
| PN 12 | Thursday, 7 February 2019 | 654 |
| PN 34 | Friday, 6 December 2019 | 465 |
| PN 43 | Friday, 6 February 2019 | 9541 |
| PN 45 | Friday, 1 February 2019 | 874 |
| PN 50 | Sunday, 10 February 2019 | 654 |
| PN 60 | Monday, 4 February 2019 | 654 |
| PN 60 | Saturday, 9 February 2019 | 123 |
| PN 75 | Tuesday, 5 February 2019 | 31 |
| Total | | 14317 |

Creating a Shared Dimension in Power BI Using Power Query: Basics and Foundations of Modeling

Date from Sales table

Sales table

| Date | Product | Warehouse | Quantity |
|----------------------------|---------|-------------|------------|
| Friday, 1 February 2019 | PN 10 | Warehouse X | 23 |
| Saturday, 2 February 2019 | PN 20 | Warehouse Y | 12 |
| Sunday, 3 February 2019 | PN 13 | Warehouse Z | 43 |
| Monday, 4 February 2019 | PN 50 | Warehouse X | 1 |
| Tuesday, 5 February 2019 | PN 34 | Warehouse Y | 5 |
| Wednesday, 6 February 2019 | PN 13 | Warehouse Z | 48 |
| Thursday, 7 February 2019 | PN 50 | Warehouse X | 65 |
| Friday, 8 February 2019 | PN 34 | Warehouse Y | 91 |
| Sunday, 10 February 2019 | PN 60 | Warehouse X | 13 |
| Tuesday, 19 February 2019 | PN 13 | Warehouse Z | 23 |
| Total | | | 324 |

Manufacturing table

Inventory table

Sales table

| Date | Product | Revenue |
|-----------------------------|---------|---------|
| Friday, 1 February 2019 | PN 20 | 2 |
| Monday, 4 February 2019 | PN 12 | 3 |
| Tuesday, 5 February 2019 | PN 10 | 12 |
| Friday, 8 February 2019 | PN 45 | 523 |
| Saturday, 9 February 2019 | PN 20 | 6 |
| Sunday, 10 February 2019 | PN 12 | 114 |
| Monday, 11 February 2019 | PN 14 | 100 |
| Tuesday, 12 February 2019 | PN 14 | 35 |
| Wednesday, 13 February 2019 | PN 14 | 805 |
| Thursday, 14 February 2019 | PN 14 | 76236 |

What is a shared dimension, and why do you need that in your Power BI model? This chapter will explain how it can prevent many issues and the need for a both-directional relationship or many-to-many relationship.

Sample Dataset

To follow the example in this chapter, use the excel file named “Shared dimension data source.xlsx” In this sample dataset, we have three tables as below;

The inventory table shows the inventory details each day in each warehouse.

| Date | Product | Warehouse | Quantity |
|------------|---------|-------------|----------|
| 1/02/2019 | PN 10 | Warehouse X | 23 |
| 2/02/2019 | PN 20 | Warehouse Y | 12 |
| 3/02/2019 | PN 13 | Warehouse Z | 43 |
| 4/02/2019 | PN 50 | Warehouse X | 1 |
| 5/02/2019 | PN 34 | Warehouse Y | 5 |
| 6/02/2019 | PN 13 | Warehouse Z | 48 |
| 7/02/2019 | PN 50 | Warehouse X | 65 |
| 8/02/2019 | PN 34 | Warehouse Y | 91 |
| 19/02/2019 | PN 13 | Warehouse Z | 23 |
| 10/02/2019 | PN 60 | Warehouse X | 13 |

The sales table shows sales transactions.

| A ^B _C Product Name | Date | 1 ² ₃ Quantity | 1 ² ₃ Revenue |
|--|------------|--------------------------------------|-------------------------------------|
| PN 13 | 1/02/2019 | 10 | 1532 |
| PN 12 | 9/02/2019 | 24 | 16541 |
| PN 10 | 12/02/2019 | 35 | 5000 |
| PN 20 | 4/02/2019 | 2 | 261 |
| PN 12 | 5/02/2019 | 3 | 4165 |
| PN 10 | 6/02/2019 | 12 | 123 |
| PN 45 | 7/02/2019 | 523 | 41586 |
| PN 20 | 8/02/2019 | 6 | 123 |
| PN 12 | 9/02/2019 | 90 | 6584 |
| PN 14 | 10/02/2019 | 100 | 321 |

The manufacturing table; shows summarized information about the cost of producing each product based on date.

| Date | A ^B _C Product | 1 ² ₃ Cost |
|------------|-------------------------------------|----------------------------------|
| 1/02/2019 | PN 45 | 874 |
| 2/02/2019 | PN 12 | 465 |
| 3/12/2019 | PN 10 | 856 |
| 4/02/2019 | PN 60 | 654 |
| 5/02/2019 | PN 75 | 31 |
| 6/12/2019 | PN 34 | 465 |
| 7/02/2019 | PN 12 | 654 |
| 8/02/2019 | PN 43 | 9541 |
| 9/02/2019 | PN 60 | 123 |
| 10/02/2019 | PN 50 | 654 |

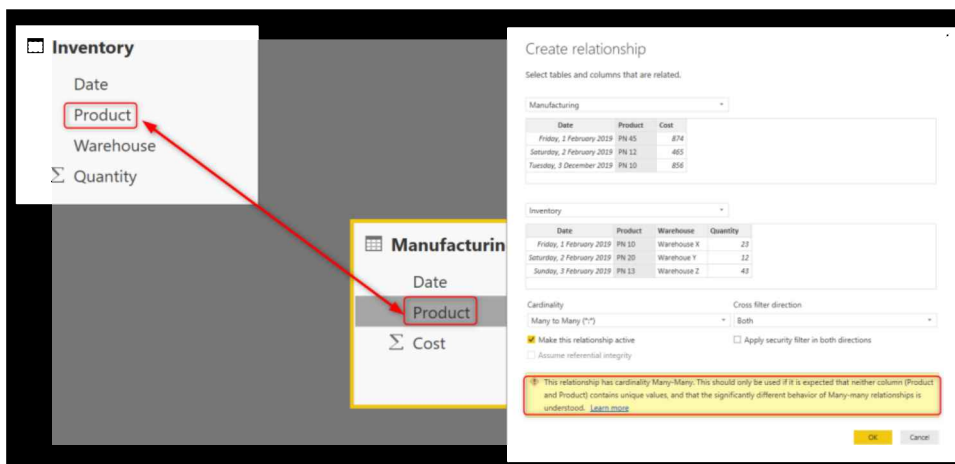
Design Challenge

When you load the three tables above in a model, you can see that they are not related.



Many-to-many Relationship Issue

Suppose you try to create the relationship yourself based on Product, for example, between the two tables, Inventory, and Manufacturing. In that case, you get the message pop up about the need for a Many-to-Many relationship!

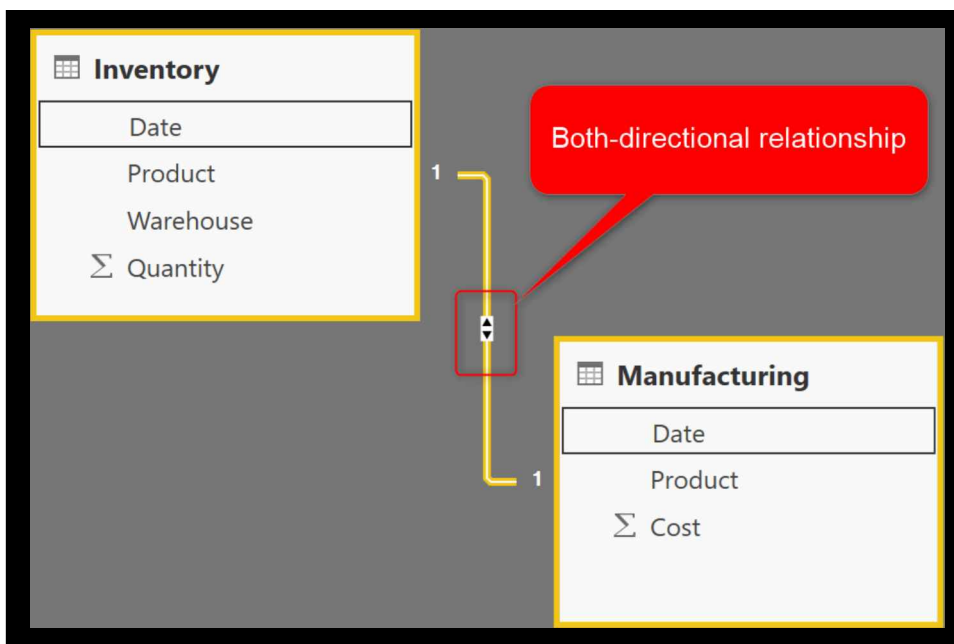


Don't click on creating the relationship. A many-to-many relationship is not a perfect type of relationship to be used. The relationship can only be created as a many-to-many relationship because the Product column in none of these tables has unique values. There is no single product table with a list of unique values of products, so there is no single table that can be used as a source of a One-to-many relationship.

When none of the two tables on both sides of a relationship doesn't have unique values for the relationship field, then many-to-many will be suggested. However, It is not recommended to use it. Read the rest of the blog chapter to learn how to fix it using a shared dimension.

Both-directional Relationship Issue

Another issue usually happens when you have a unique list of values; however, you still want to slice and dice based on both tables. You want to create the relationship between the Inventory table and the Manufacturing table, but this time is based on the Date field.



Because we want to slice and dice Inventory data by Dates selected from the Manufacturing table and vice versa, the relationship needs to be both-directional.

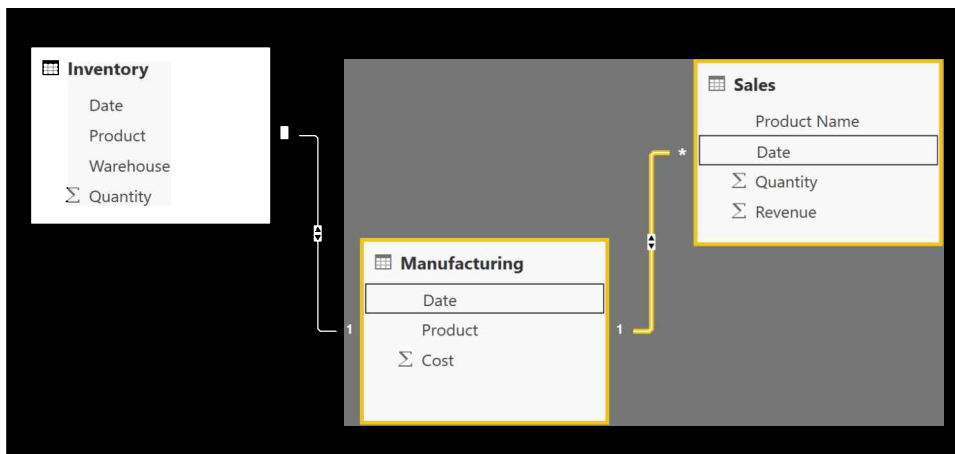
A both-directional relationship usually happens when you want to use fields from both tables for slicing and dicing. A both-directional relationship has a significant effect on performance and is not recommended. Read the rest of this chapter to learn how it can be fixed using a shared dimension.

Another issue with the both-directional relationship is that you cannot apply it to all relationships in your model because it might create a circular reference scenario!

Master List Does Not Exist!

The third issue of design with the three tables above is that there is no master list! There are some products on each table, and we do not necessarily have all products on each table. Or there are some dates in each table, and we do not necessarily have all dates in each table (Power BI will create an auto date dimension which can resolve this issue only for Date fields, but what about other fields such as Product?). To explain the issue, I've created directional relationships between all three tables to ensure that they are all filtering each other. All of the relationships are based on Date fields.

Part 1: Combining data tables



Then I created a table with the Date field from the Sales table as a slicer and three table visuals from each table. The date slicer should be able to filter all three tables based on the Date selection;

Manufacturing table

| Product | Date | Cost |
|--------------|---------------------------|--------------|
| PN 10 | Tuesday, 3 December 2019 | 856 |
| PN 12 | Saturday, 2 February 2019 | 465 |
| PN 12 | Thursday, 7 February 2019 | 654 |
| PN 34 | Friday, 6 December 2019 | 465 |
| PN 43 | Friday, 8 February 2019 | 9541 |
| PN 45 | Friday, 1 February 2019 | 874 |
| PN 50 | Sunday, 10 February 2019 | 654 |
| PN 60 | Monday, 4 February 2019 | 654 |
| PN 60 | Saturday, 9 February 2019 | 123 |
| PN 75 | Tuesday, 5 February 2019 | 31 |
| Total | | 14317 |

Inventory table

| Date | Product | Warehouse | Quantity |
|----------------------------|---------|-------------|------------|
| Friday, 1 February 2019 | PN 10 | Warehouse X | 23 |
| Saturday, 2 February 2019 | PN 20 | Warehouse Y | 12 |
| Sunday, 3 February 2019 | PN 13 | Warehouse Z | 43 |
| Monday, 4 February 2019 | PN 50 | Warehouse X | 1 |
| Tuesday, 5 February 2019 | PN 34 | Warehouse Y | 5 |
| Wednesday, 6 February 2019 | PN 13 | Warehouse Z | 48 |
| Thursday, 7 February 2019 | PN 50 | Warehouse X | 65 |
| Friday, 8 February 2019 | PN 34 | Warehouse Y | 91 |
| Sunday, 10 February 2019 | PN 60 | Warehouse X | 13 |
| Tuesday, 19 February 2019 | PN 13 | Warehouse Z | 23 |
| Total | | | 324 |

Date from Sales table

- ☐ Friday, 1 February 2019
- ☐ Monday, 4 February 2019
- ☐ Tuesday, 5 February 2019
- ☐ Wednesday, 6 February 2019
- ☐ Thursday, 7 February 2019
- ☐ Friday, 8 February 2019
- ☐ Saturday, 9 February 2019
- ☐ Sunday, 10 February 2019
- ☐ Tuesday, 12 February 2019

Sales table

| Date | Product Name | Quantity | Revenue |
|----------------------------|--------------|------------|--------------|
| Friday, 1 February 2019 | PN 13 | 10 | 1532 |
| Monday, 4 February 2019 | PN 20 | 2 | 261 |
| Tuesday, 5 February 2019 | PN 12 | 3 | 4165 |
| Wednesday, 6 February 2019 | PN 10 | 12 | 123 |
| Thursday, 7 February 2019 | PN 45 | 523 | 41566 |
| Friday, 8 February 2019 | PN 20 | 6 | 123 |
| Saturday, 9 February 2019 | PN 12 | 114 | 23125 |
| Sunday, 10 February 2019 | PN 14 | 100 | 321 |
| Tuesday, 12 February 2019 | PN 10 | 35 | 5000 |
| Total | | 805 | 76236 |

Annotations:

- "This date does not exist in the slicer!" (pointing to Tuesday, 3 December 2019 in Manufacturing table)
- "This date does not exist in the slicer!" (pointing to Tuesday, 19 February 2019 in Inventory table)
- "There is no master list/table to filter the data of all three tables" (red box)

Selecting a field in the date slicer will filter all three tables (because of both-directional relationships). However, if you look closely, the two dates mentioned in the above screenshot, and some other dates in the Inventory and Manufacturing tables, don't exist in the slicer. Because the date slicer is coming from the Sales table, and the Sales table doesn't have those dates in it! We will have the same challenge with the Product slicer if we add it.

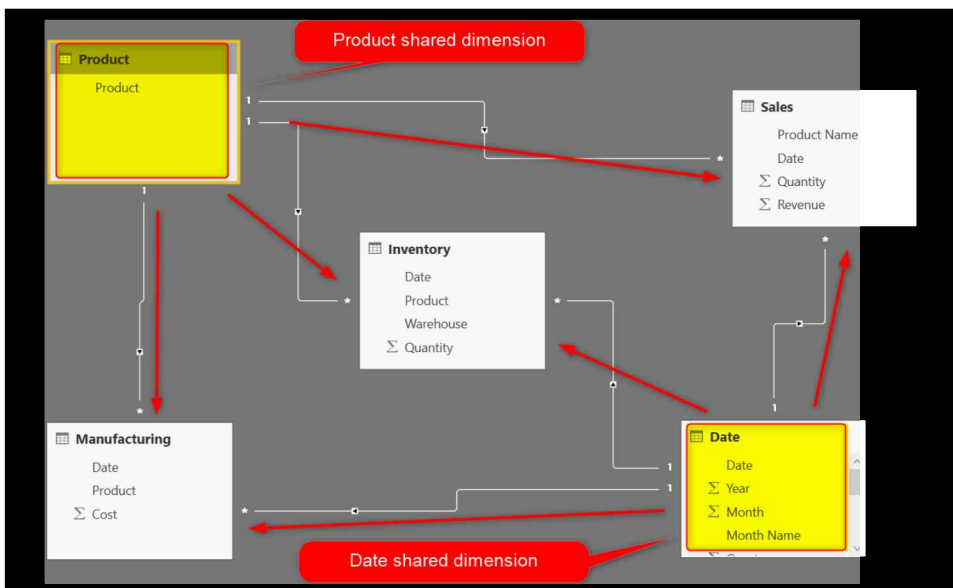
If you use a field as a slicer that doesn't have all possible values, it cannot show the right data from all tables. It is not recommended to design it this way. Read the rest of the chapter to learn how a shared dimension can fix this challenge.

Shared Dimension: Solution

I just mentioned three of the challenges you might have with a design like above. In reality, you don't have just three tables, you will have much more, and you will have many more challenges with a design such as above. The best practice to design such a model is to create a shared dimension. A shared dimension is a dimension that is shared between multiple fact tables.

Well, now, in the design above, what are our dimensions? Date and Product. What are fact tables? Sales, Inventory, and Manufacturing. The challenge is that there is no dimension table. Dimensions are fields inside the fact tables, and this creates inconsistency and design approaches. What you should do is to build the two tables separately. Because the Date and Product tables will be tables that slice and dice all of the fact tables and are related to all fact tables, we call them **shared dimensions**. A shared dimension is just a dimension that is shared between multiple fact tables.

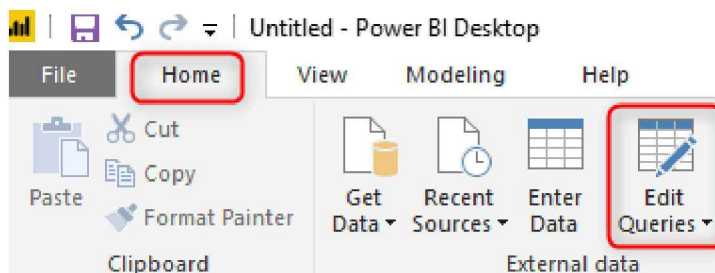
a design sketch of tables above with shared dimensions would be like this:



Creating Shared Dimension

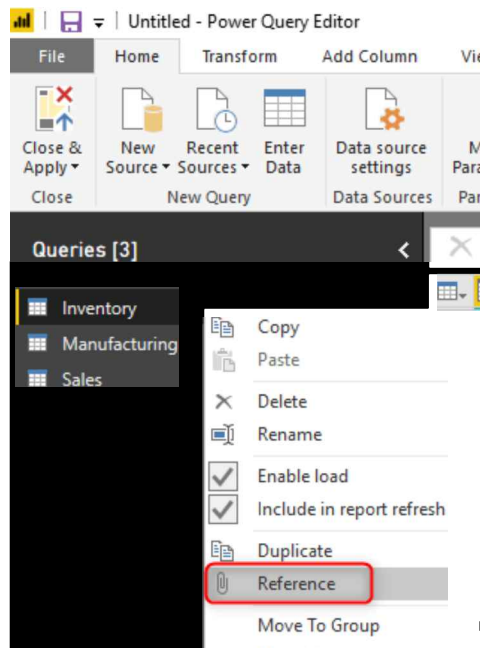
Now that you know what the shared dimension is and how it can be helpful let's see how we can add it to our design. You can build the shared dimension in many ways, using DAX calculated tables, t-SQL (if sourced from database systems), or Power Query because Power Query is applicable regardless of the data source you select. Because the data transformations step is better done in Power Query rather than DAX, I will show you how to do it in Power Query.

Go to Edit Queries in the Power BI Desktop;



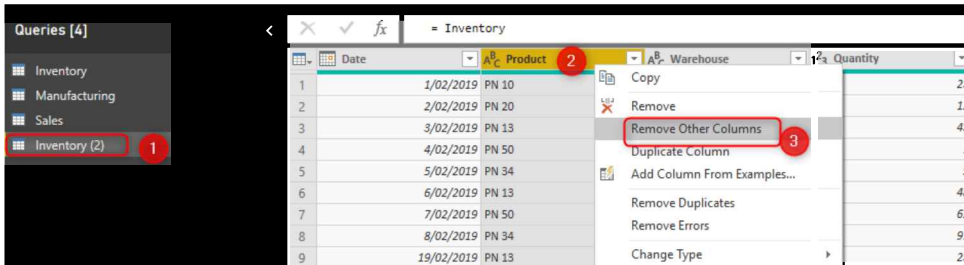
Prepare sub-tables

In the Power Query Editor window, right-click on the Inventory table, and create a reference from the query;

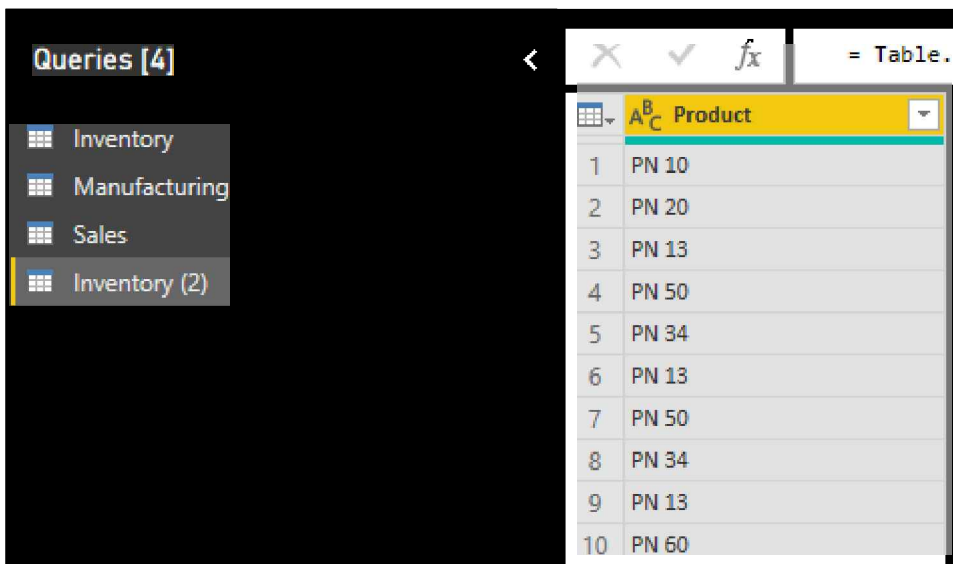


Chapter 6: Creating a Shared Dimension in Power BI Using Power Query

Reference will create a copy of the existing query, with reference to the existing query, which can now have extra steps in it. In the new query, right-click on the Product table and remove all other columns.

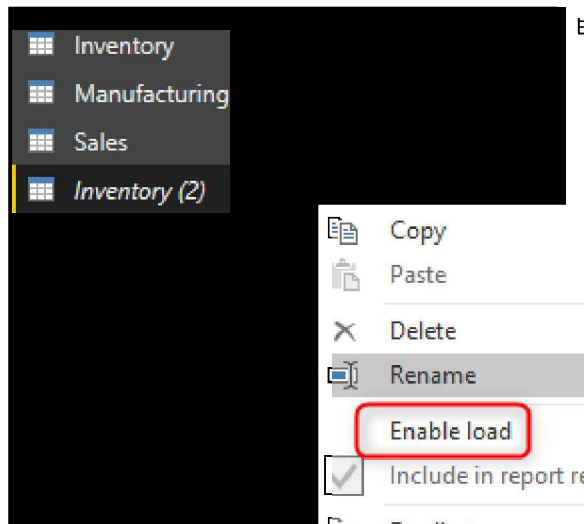


The Inventory table should now look like this:



Right-click on the Inventory (2) table and uncheck the Enable Load option for it. This is to save performance and avoid loading extra tables into the memory of the Power BI Desktop.

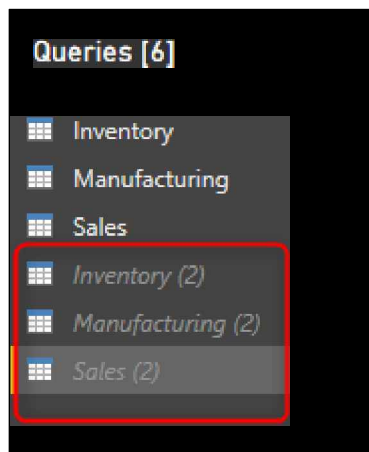
Part 1: Combining data tables



Do the same process now for the other two tables; Manufacturing and Sales;

- create a reference from each table
- Only keep the Product table and remove other columns
- uncheck the enable load in the new query

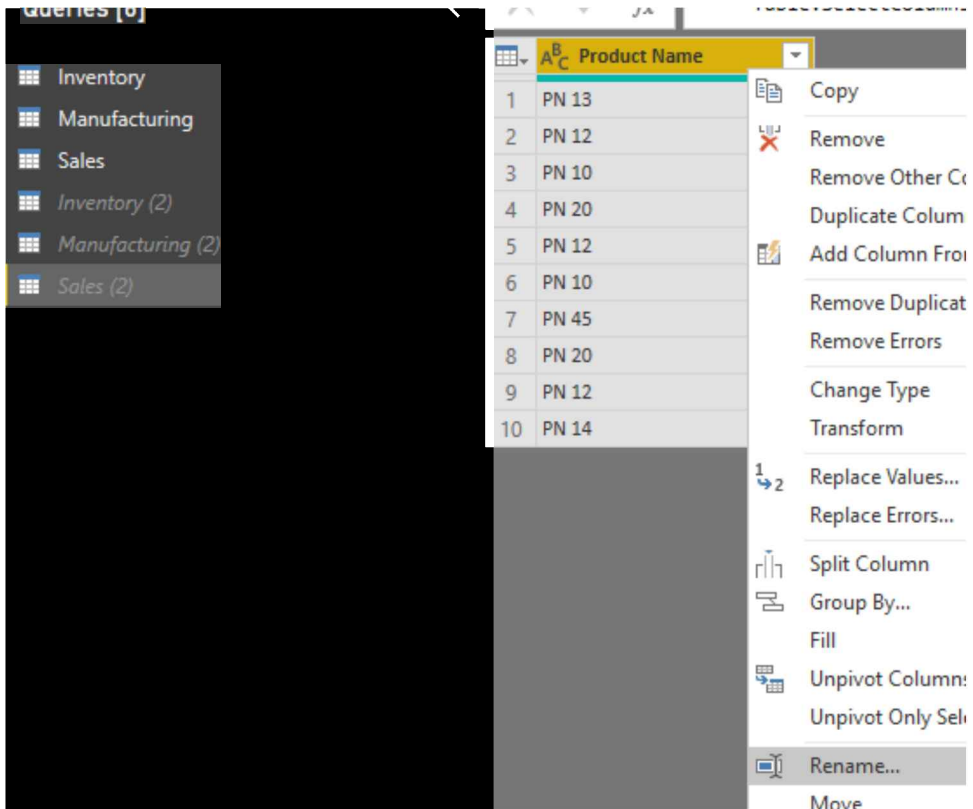
You should now have the new three tables with one Product column only in each:



Set all column names to be the same

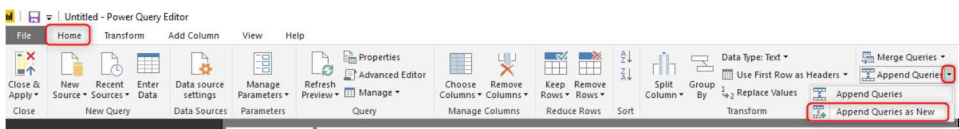
The next step is to make sure the column names are the same. Because we will append the three tables, it would create extra columns if we have different names. Names should be an exact match, and remember that Power Query is a case-sensitive language; “product” is different from “Product” in the Power Query world. In our sample model,

the two tables, Inventory, and Manufacturing, have the column name as Product, but in the Sales table, it is called Product Name, rename it to Product.



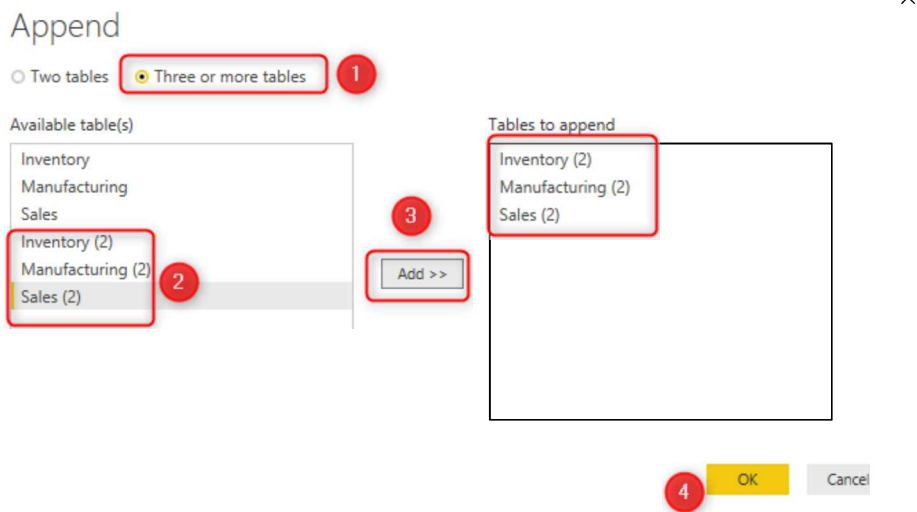
Append all three tables

Append all three tables together to create one table with all Product values in it.

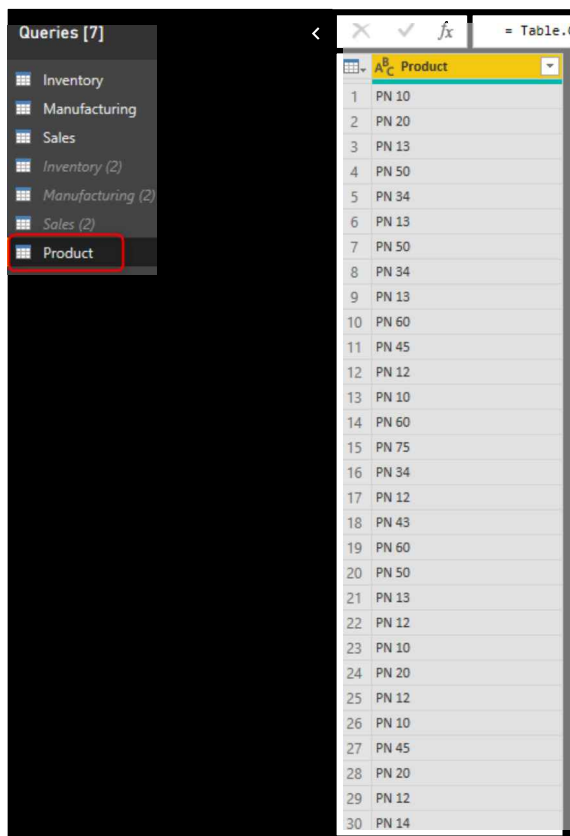


Then in the Append command window, select Three or more tables and all the new tables in it;

Part 1: Combining data tables



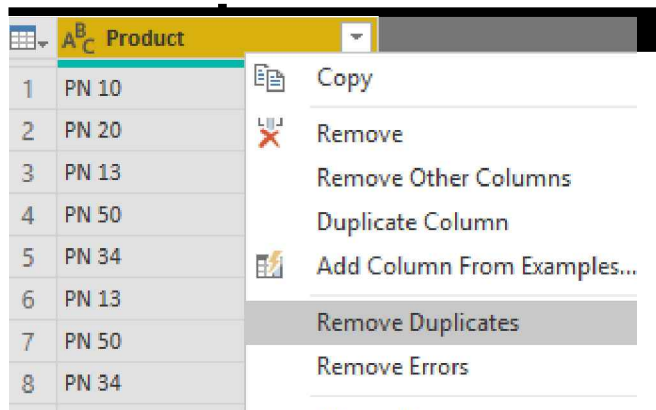
The output of the query would be one table, including all Product values; you can rename this query to Product.



Because this is a table you want to be loaded into Power BI Desktop, make sure the Enable Load of this table is checked. This table is our master list, including all product values. However, there are duplicate values in it that have to be removed.

Remove Duplicates

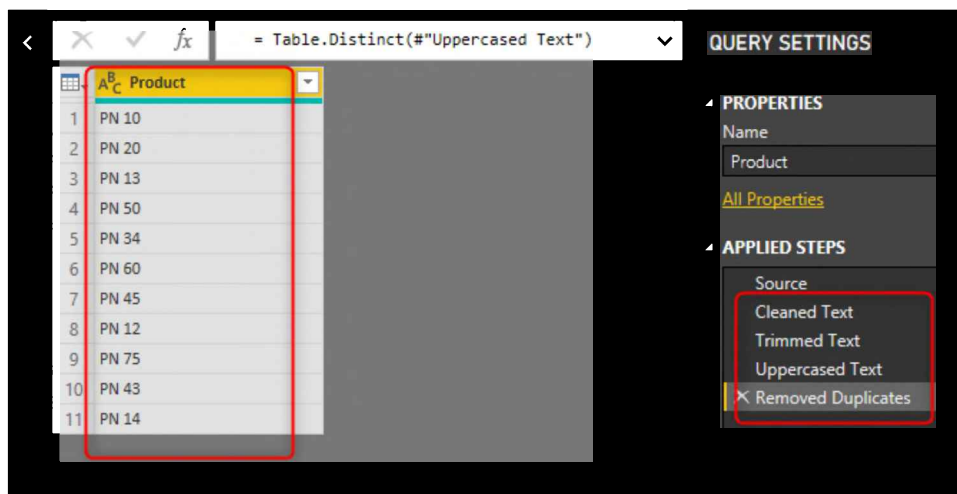
A dimension should have a unique list of values, so we need to remove duplicates for the key field here.



Before using remove duplicate, make sure to read the chapter about it earlier in this book about important tips you need to know before applying remove duplicate in the Power Query. In a nutshell, because Power Query is case-sensitive, and because space at the end of text values and other special characters may end with keeping duplicate values, this is how you would remove duplicates in few steps;

- Clean transformation
- Trim transformation
- Transform to Upper Case
- Remove Duplicates

Part 1: Combining data tables



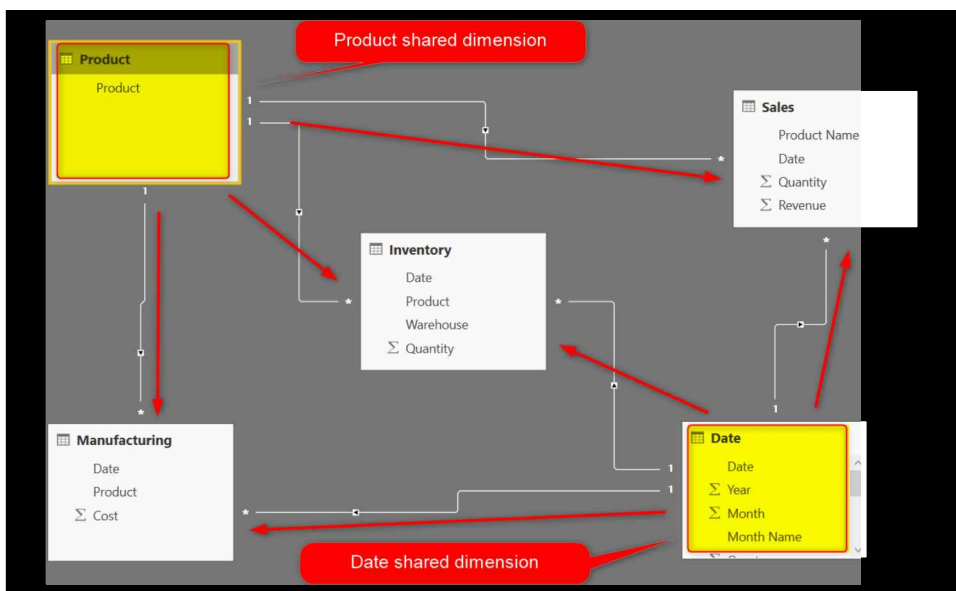
Now you have your Product shared dimension ready! Repeat this process for any other shared dimensions with the relevant fields. However, for the Date table, we would do it differently.

Date Dimension

You learned in an earlier chapter how to create a date dimension using the Power Query script.

Best Practice Design: Star Schema and Shared Dimensions

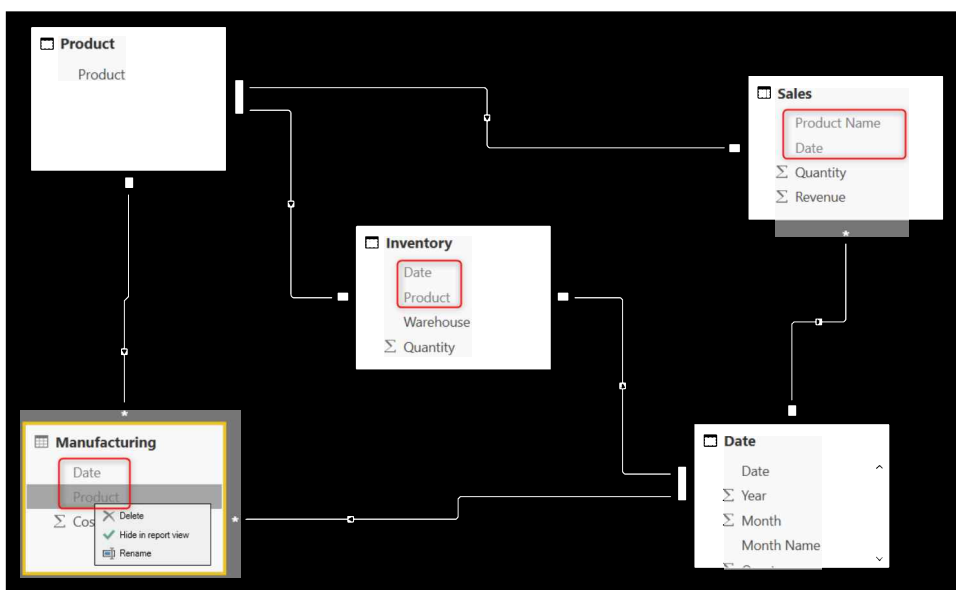
After loading the tables above, you can create a one-to-many relationship single directional from Product and Date tables to all other fact tables. This is the final design based on our example;



The above design uses two shared dimensions and avoided all challenges mentioned;

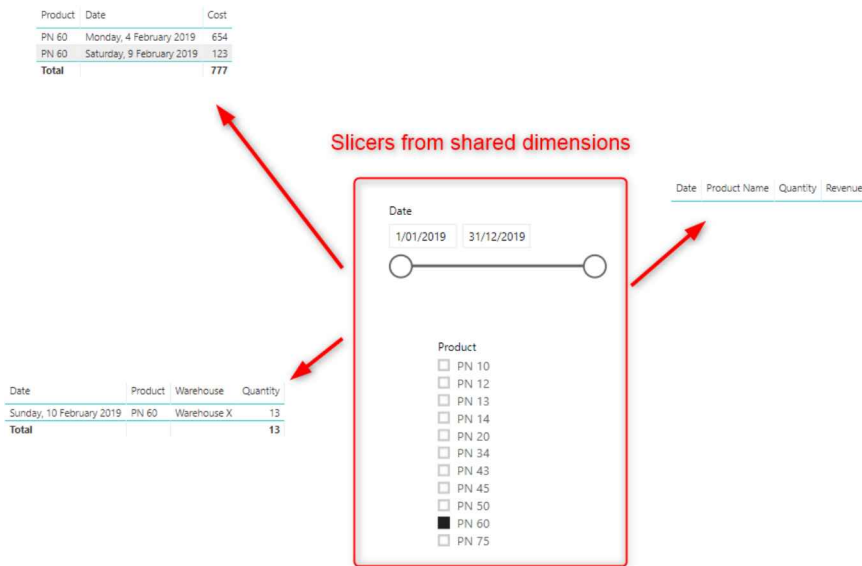
- It doesn't need both-directional relationships
- It doesn't need many-to-many relationships
- Product and Date tables are master tables that can be the source of any slicing and dicing

To make sure that you won't use incorrect fields for slicing and dicing, make sure that you hide Date and Product columns in all the three fact tables as below;



This solution can now have proper reporting capabilities as below;

Part 1: Combining data tables



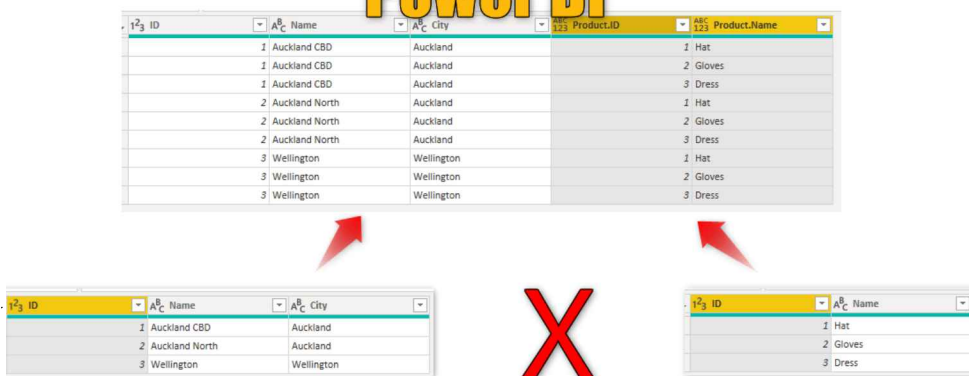
Summary

Nothing is worse than a bad data model design. Bad data model design causes using relationships that decrease the performance. It causes writing a lot of unnecessary DAX expressions to cover for the wrong design. At the end of the day, it performs slow. In this example, you learned one of the basics, but the most fundamentals of designing Power BI data models. Using a shared dimension in your model will avoid both directional and many-to-many relationships. You learned how easy it is to create such a dimension. This method can always be used in your Power BI data models.

This chapter and the previous chapter in this book are examples of data modeling. Data modeling is an essential skill for every Power BI Developer. To learn more about data modeling, I strongly recommend reading my book on the Basics of Data Modeling [here](https://www.amazon.com/gp/product/B08HWNZ7GC)[<https://www.amazon.com/gp/product/B08HWNZ7GC>].

Chapter 7: Cartesian Product in Power Query: Multiply All Sets of All Pairs in Power BI

Cartesian Product in Power Query: Multiply All Sets of All Pairs in Power BI



| ID | Name | City | ProductID | ProductName |
|----|----------------|------------|-----------|-------------|
| 1 | Auckland CBD | Auckland | 1 | Hat |
| 1 | Auckland CBD | Auckland | 2 | Gloves |
| 1 | Auckland CBD | Auckland | 3 | Dress |
| 2 | Auckland North | Auckland | 1 | Hat |
| 2 | Auckland North | Auckland | 2 | Gloves |
| 2 | Auckland North | Auckland | 3 | Dress |
| 3 | Wellington | Wellington | 1 | Hat |
| 3 | Wellington | Wellington | 2 | Gloves |
| 3 | Wellington | Wellington | 3 | Dress |

| ID | Name | City |
|----|----------------|------------|
| 1 | Auckland CBD | Auckland |
| 2 | Auckland North | Auckland |
| 3 | Wellington | Wellington |

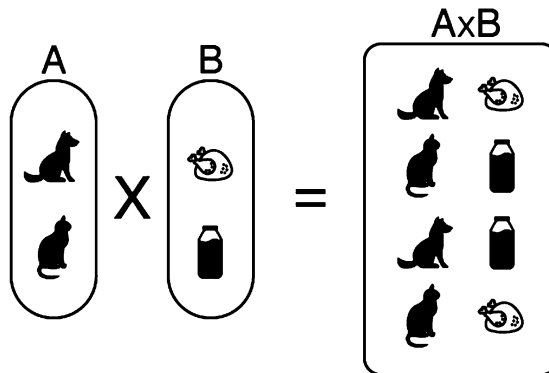
| ID | Name |
|----|--------|
| 1 | Hat |
| 2 | Gloves |
| 3 | Dress |

Sometimes, you need to create a multiplication of all sets of all pairs from two different data tables in Power BI. This action is different from Merge (Join) because there are no matching key columns and no way to relate the two tables together. However, you want to create a multiplication as a flattened table. If this process needs to be done in Power Query, then there is a simple trick. In this chapter, I'll explain how you can do it.

What is Cartesian Product

Based on a definition from [Mathstopia](https://www.mathstopia.net/sets/cartesian-product) (and that is where the below picture is also coming from)[<https://www.mathstopia.net/sets/cartesian-product>], Cartesian Product is the multiplication of two sets to form the set of all ordered pairs. The first element of the ordered pair belong to the first set and the second pair belongs to the second set. For example;

Part 1: Combining data tables



Cartesian Product of Two Sets.

Reference: <https://www.mathstopia.net/sets/cartesian-product>
<https://www.mathstopia.net/sets/cartesian-product>

Example

Let's say I have a table of products as below;

| T_3 ID | A_C^B Name |
|----------|--------------|
| 1 | Hat |
| 2 | Gloves |
| 3 | Dress |

and a Warehouse table as below;

| T_3 ID | A_C^B Name | A_C^B City |
|----------|----------------|--------------|
| 1 | Auckland CBD | Auckland |
| 2 | Auckland North | Auckland |
| 3 | Wellington | Wellington |

There is no link between the two tables above. If there was a table with information about each product in each warehouse, I could use that to merge the three tables, but that is not the case here.

The expected output is below:

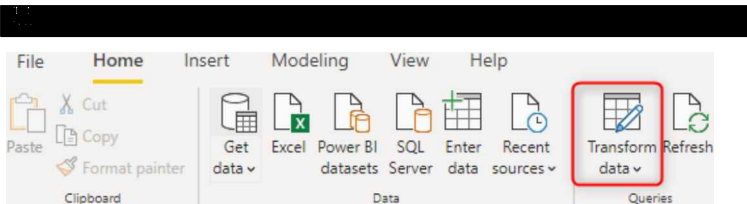
| ID | Name | City | Product.ID | Product.Name |
|----|----------------|------------|------------|--------------|
| 1 | Auckland CBD | Auckland | 1 | Hat |
| 1 | Auckland CBD | Auckland | 2 | Gloves |
| 1 | Auckland CBD | Auckland | 3 | Dress |
| 2 | Auckland North | Auckland | 1 | Hat |
| 2 | Auckland North | Auckland | 2 | Gloves |
| 2 | Auckland North | Auckland | 3 | Dress |
| 3 | Wellington | Wellington | 1 | Hat |
| 3 | Wellington | Wellington | 2 | Gloves |
| 3 | Wellington | Wellington | 3 | Dress |

For each warehouse, all the products are listed. Considering that the product table has three rows and the warehouse table has three rows, the final result is nine rows.

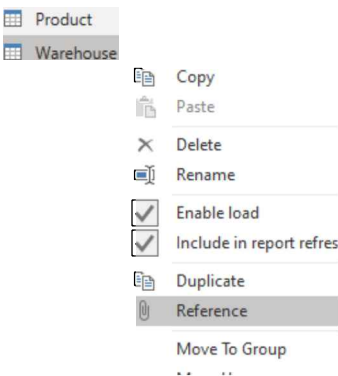
Cartesian Product in Power Query

Here are steps that you can follow to do this process in Power Query Editor.

First, click on Transform Data to get to the Power Query Editor,

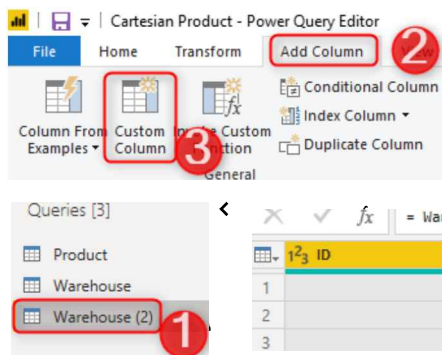


I normally prefer to keep the original tables untouched, so I create a [reference](https://radacad.com/reference-vs-duplicate-in-power-bi-power-query-back-to-basics) for the cartesian product query (you can do this in any of the existing queries if you want);

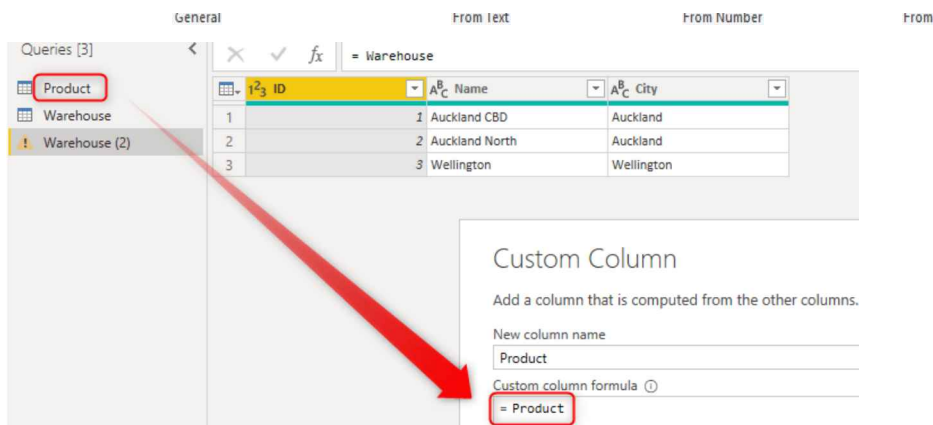


Now in this new query, you can add a column; Custom column.

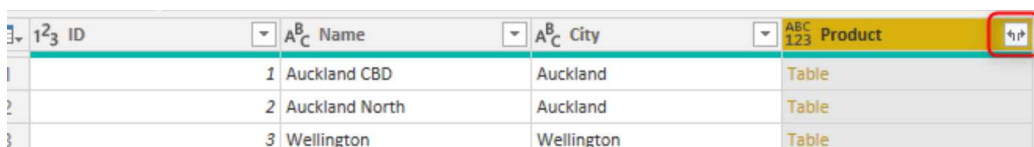
Part 1: Combining data tables



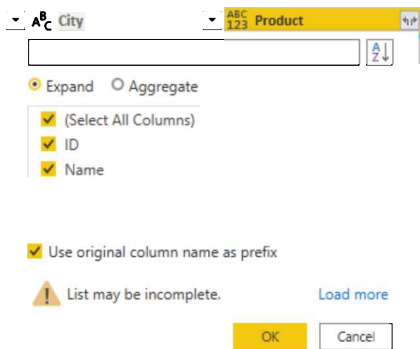
The column you add can be named as the other table (the name part is not the important part unless you want to use it as the columns' prefix after expanding it). the most important part is that the expression should be the name of the other table (the same spelling, Power Query is case-sensitive, remember that!)



You cannot drag and drop the other table name's into the expression, but if you start typing, you will see the intellisense comes to help.



Now you can expand the Product column;



And here is the results:

| ID | Name | City | Product.ID | Product.Name |
|----|----------------|------------|------------|--------------|
| 1 | Auckland CBD | Auckland | 1 | Hat |
| 1 | Auckland CBD | Auckland | 2 | Gloves |
| 1 | Auckland CBD | Auckland | 3 | Dress |
| 2 | Auckland North | Auckland | 1 | Hat |
| 2 | Auckland North | Auckland | 2 | Gloves |
| 2 | Auckland North | Auckland | 3 | Dress |
| 3 | Wellington | Wellington | 1 | Hat |
| 3 | Wellington | Wellington | 2 | Gloves |
| 3 | Wellington | Wellington | 3 | Dress |

Don't Use Cartesian Product Instead of Merge/Join

Remember! There is a big difference between a merge/join or a cartesian product. Merge/Join is useful when the two tables have logical ways to be matched together, using one or more joining fields. Cartesian Product, on the other hand, side is for tables with no existing logic of connection. It is for scenarios that you want to have a result set of all pairs regardless.

Cartesian Product can result in a huge table if the tables you use as the source are big. If table A is 1,000 rows, and table B is also 1,000 rows, the result of the cartesian product will be 1,000,000 rows. So use it carefully, and only if needed.

Chapter 8: Find Mismatch Rows with Power Query in Power BI

Merge ×

Select tables and matching columns to create a merged table.

Customer table from website

| CustomerKey | FirstName | MiddleName | LastName | EmailAddress |
|-------------|-----------|------------|--------------|--------------------------------|
| 11000 | Jon | V | Yang | jon24@adventure-works.com |
| 11001 | Eugene | L | Huang | eugene10@adventure-works.com |
| 11002 | Ruben | | null Torres | ruben35@adventure-works.com |
| 11003 | Christy | | null Zhu | christy12@adventure-works.com |
| 11004 | Elizabeth | | null Johnson | elizabeth5@adventure-works.com |

Customer table from application

| CustomerKey | FirstName | MiddleName | LastName | BirthDate | MaritalStatus | Suffix | Gender |
|-------------|-----------|------------|--------------|------------|---------------|--------|--------|
| 11000 | Jon | V | Yang | 8/04/1966 | M | | |
| 11001 | Eugene | L | Huang | 14/05/1965 | S | null | M |
| 11002 | Ruben | | null Torres | 12/08/1965 | M | null | M |
| 11003 | Christy | | null Zhu | 15/02/1968 | S | null | F |
| 11004 | Elizabeth | | null Johnson | 8/08/1968 | S | null | F |

Join Kind

Left Anti (rows only in first)

The selection has matched 18476 out of the first 18480 rows.

OK Cancel

Finding rows in one table but not the other is one of the most common scenarios in any data-related application. You may have customer records coming from two sources and want to find data rows that exist in one but not the other. In Power Query, you can use Merge to combine data tables. Merge can also be used for finding mismatch records. You will learn through this chapter how in Power Query, you can find out which records are missing with Merge and then report them in Power BI.

Sample Data Tables

I have two customer tables; one customer table comes from the website and another from an application. Here is the customer table from the website:

Chapter 8: Find Mismatch Rows with Power Query in Power BI

| CustomerKey | FirstName | MiddleName | LastName | EmailAddress |
|-------------|-----------|------------|----------|--------------------------------|
| 11000 | Jon | V | Yang | jon24@adventure-works.com |
| 11001 | Eugene | L | Huang | eugene10@adventure-works.com |
| 11002 | Ruben | | Torres | ruben35@adventure-works.com |
| 11003 | Christy | | Zhu | christy12@adventure-works.com |
| 11004 | Elizabeth | | Johnson | elizabeth5@adventure-works.com |
| 11005 | Julio | | Ruiz | julio1@adventure-works.com |
| 11006 | Janet | G | Alvarez | janet9@adventure-works.com |
| 11007 | Marco | | Mehta | marco14@adventure-works.com |
| 11008 | Rob | | Verhoff | rob4@adventure-works.com |
| 11013 | Ian | M | Jenkins | ian47@adventure-works.com |
| 11014 | Sydney | | Bennett | sydney23@adventure-works.com |
| 11015 | Chloe | | Young | chloe23@adventure-works.com |
| 11016 | Wyatt | L | Hill | wyatt32@adventure-works.com |
| 11017 | Shannon | | Wang | shannon1@adventure-works.com |

And the other table that comes from the application:

| CustomerKey | FirstName | MiddleName | LastName | BirthDate | MaritalStatus | Suffix | Gender |
|-------------|-----------|------------|----------|------------|---------------|--------|--------|
| 11000 | Jon | V | Yang | 8/04/1966 | M | | M |
| 11001 | Eugene | L | Huang | 14/05/1965 | S | | M |
| 11002 | Ruben | | Torres | 12/08/1965 | M | | M |
| 11003 | Christy | | Zhu | 15/02/1968 | S | | F |
| 11004 | Elizabeth | | Johnson | 8/08/1968 | S | | F |
| 11005 | Julio | | Ruiz | 5/08/1965 | S | | M |
| 11006 | Janet | G | Alvarez | 6/12/1965 | S | | F |
| 11007 | Marco | | Mehta | 9/05/1964 | M | | M |
| 11008 | Rob | | Verhoff | 7/07/1964 | S | | F |
| 11009 | Shannon | C | Carlson | 1/04/1964 | S | | M |
| 11010 | Jacquelyn | C | Suarez | 6/02/1964 | S | | F |
| 11017 | Shannon | | Wang | 26/06/1944 | S | | F |
| 11018 | Clarence | D | Rai | 9/10/1944 | S | | M |
| 11019 | Luke | L | Lal | 7/03/1978 | S | | M |
| 11020 | Jordan | C | King | 20/09/1978 | S | | M |
| 11021 | Destiny | | Wilson | 3/09/1978 | S | | F |

The customer table from the website has the email address field, plus name and CustomerKey. The table from the application has fields such as birthdate, gender, marital status, and name and customer key. The link field in this scenario is CustomerKey which exists in both tables.

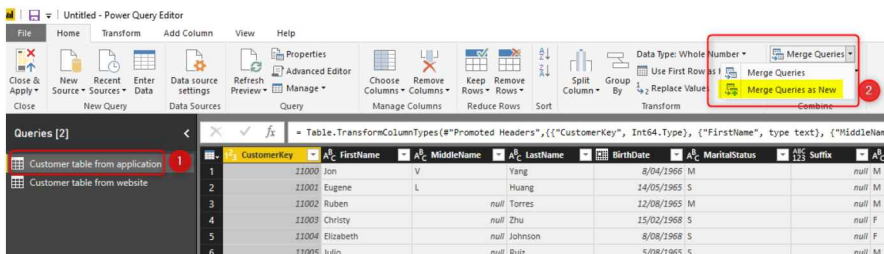
| CustomerKey | FirstName | MiddleName | LastName | BirthDate | MaritalStatus | Suffix | Gender |
|-------------|-----------|------------|-----------|------------|---------------|--------|--------|
| 11000 | Jon | V | Yang | 8/04/1966 | M | | M |
| 11001 | Eugene | L | Huang | 14/05/1965 | S | | M |
| 11002 | Ruben | | Torres | 12/08/1965 | M | | M |
| 11003 | Christy | | Zhu | 15/02/1968 | S | | F |
| 11004 | Elizabeth | | Johnson | 8/08/1968 | S | | F |
| 11005 | Julio | | Ruiz | 5/08/1965 | S | | M |
| 11006 | Janet | G | Alvarez | 6/12/1965 | S | | F |
| 11007 | Marco | | Mehta | 9/05/1964 | M | | M |
| 11008 | Rob | | Verhoff | 7/07/1964 | S | | F |
| 11009 | Shannon | C | Carlson | 1/04/1964 | S | | M |
| 11010 | Jacquelyn | C | Suarez | 6/02/1964 | S | | F |
| 11017 | Shannon | | Wang | 26/06/1944 | S | | F |
| 11018 | Clarence | D | Rai | 9/10/1944 | S | | M |
| 11019 | Luke | L | Lal | 7/03/1978 | S | | M |
| 11020 | Jordan | C | King | 20/09/1978 | S | | M |
| 11021 | Destiny | | Wilson | 3/09/1978 | S | | F |
| 11022 | Shan | G | Zhang | 12/10/1978 | M | | M |
| 11023 | Seth | M | Edwards | 11/10/1978 | M | | M |
| 11024 | Russell | | Xie | 17/09/1978 | M | | M |
| 11025 | Alejandro | | Beck | 23/12/1945 | M | | M |
| 11026 | Harold | | Sai | 3/04/1946 | S | | M |
| 11027 | Jessie | R | Zhao | 7/12/1946 | M | | F |
| 11028 | Jill | | Jimenez | 11/04/1946 | M | | F |
| 11029 | Jimmy | L | Moreno | 21/12/1946 | M | | M |
| 11030 | Bethany | G | Yuan | 22/02/1947 | M | | F |
| 11031 | Theresa | G | Ramos | 22/08/1947 | M | | F |
| 11032 | Denise | | Stone | 11/09/1947 | M | | F |
| 11033 | Aime | | Nath | 23/09/1947 | M | | M |
| 11034 | Ebony | | Gonzalez | 19/06/1947 | M | | F |
| 11035 | Wendy | | Dominguez | 26/02/1948 | M | | F |
| 11036 | Emiller | C | Russell | 18/12/1978 | M | | F |
| 11037 | Chloe | M | Garcia | 27/11/1977 | S | | F |

| CustomerKey | FirstName | MiddleName | LastName | EmailAddress |
|-------------|-----------|------------|----------|----------------------------------|
| 11000 | Jon | V | Yang | jon24@adventure-works.com |
| 11001 | Eugene | L | Huang | eugene10@adventure-works.com |
| 11002 | Ruben | | Torres | ruben35@adventure-works.com |
| 11003 | Christy | | Zhu | christy12@adventure-works.com |
| 11004 | Elizabeth | | Johnson | elizabeth5@adventure-works.com |
| 11005 | Julio | | Ruiz | julio1@adventure-works.com |
| 11006 | Janet | G | Alvarez | janet9@adventure-works.com |
| 11007 | Marco | | Mehta | marco14@adventure-works.com |
| 11008 | Rob | | Verhoff | rob4@adventure-works.com |
| 11013 | Ian | M | Jenkins | ian47@adventure-works.com |
| 11014 | Sydney | | Bennett | sydney23@adventure-works.com |
| 11015 | Chloe | | Young | chloe23@adventure-works.com |
| 11016 | Wyatt | L | Hill | wyatt32@adventure-works.com |
| 11017 | Shannon | | Wang | shannon1@adventure-works.com |
| 11018 | Clarence | D | Rai | clarence32@adventure-works.com |
| 11019 | Luke | L | Lal | luke18@adventure-works.com |
| 11020 | Jordan | C | King | jordan73@adventure-works.com |
| 11021 | Destiny | | Wilson | destiny7@adventure-works.com |
| 11022 | Shan | G | Zhang | ethan20@adventure-works.com |
| 11023 | Seth | M | Edwards | seth46@adventure-works.com |
| 11024 | Russell | | Xie | russell17@adventure-works.com |
| 11025 | Alejandro | | Beck | alejandrod45@adventure-works.com |
| 11026 | Harold | | Sai | harold13@adventure-works.com |
| 11027 | Jessie | R | Zhao | jessie16@adventure-works.com |
| 11028 | Jill | | Jimenez | jill13@adventure-works.com |
| 11029 | Jimmy | L | Moreno | jimmy9@adventure-works.com |
| 11030 | Bethany | G | Yuan | bethany18@adventure-works.com |
| 11031 | Theresa | G | Ramos | theresa13@adventure-works.com |

Part 1: Combining data tables

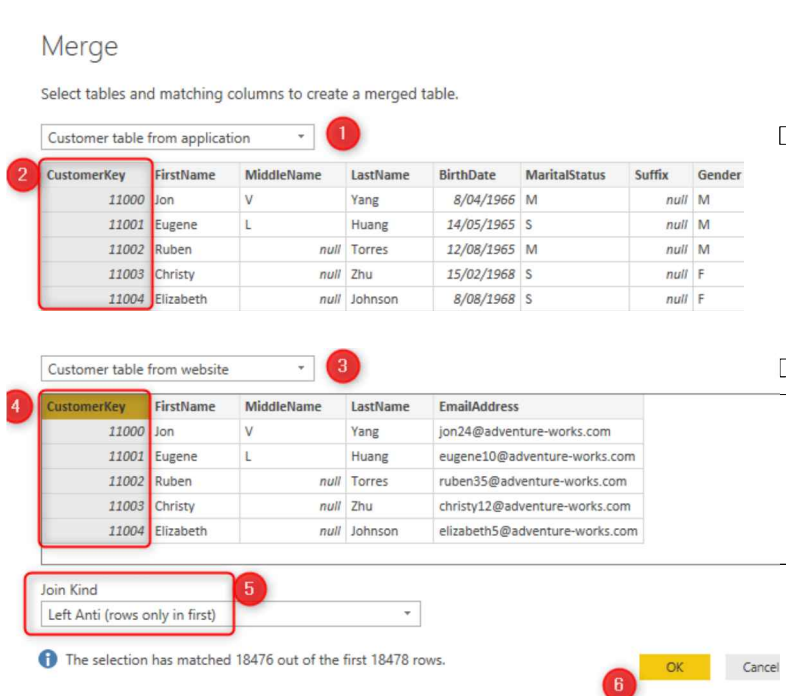
Merge in Power Query

Previously, I have explained what the Merge is and the difference of that to Append. I also explained in another chapter about different types of Merge Kind. In this example, you would find a couple of those join kinds useful; Left Anti Join and Right Anti Join. In the Below Power Query example, I read the data from both tables, and I merge them;

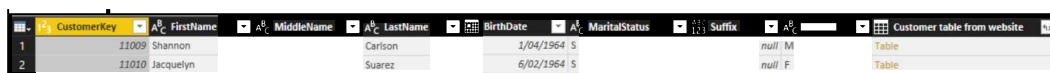


Left Anti Join: Records Only in the First Table

When I merge these two queries, I select Left Anti Join, which gives me rows that exist only in the first table (customer table from the application):



This will give you only rows that exist in the first table, “Customer table from the application.”



Then you can remove the last column in the output because the table value would not have any rows (these are mismatch rows)

| | CustomerKey | FirstName | MiddleName | LastName | BirthDate | MaritalStatus | Suffix | Gender |
|---|-------------|-----------|------------|----------|-----------|---------------|--------|--------|
| 1 | 11009 | Shannon | C | Carlson | 1/04/1964 | S | | null M |
| 2 | 11010 | Jacquelyn | C | Suarez | 6/02/1964 | S | | null F |

Right Anti Join: Records Only in the Second Table

The same approach can be used for rows that exist only in the second table, using the Right Anti Join

Merge

Select tables and matching columns to create a merged table.

Customer table from application

| CustomerKey | FirstName | MiddleName | LastName | BirthDate | MaritalStatus | Suffix | Gender |
|-------------|-----------|------------|----------|------------|---------------|--------|--------|
| 11000 | Jon | V | Yang | 8/04/1966 | M | null | M |
| 11001 | Eugene | L | Huang | 14/05/1965 | S | null | M |
| 11002 | Ruben | | Torres | 12/08/1965 | M | null | M |
| 11003 | Christy | | Zhu | 15/02/1968 | S | null | F |
| 11004 | Elizabeth | | Johnson | 8/08/1968 | S | null | F |

Customer table from website

| CustomerKey | FirstName | MiddleName | LastName | EmailAddress |
|-------------|-----------|------------|----------|--------------------------------|
| 11000 | Jon | V | Yang | jon24@adventure-works.com |
| 11001 | Eugene | L | Huang | eugene10@adventure-works.com |
| 11002 | Ruben | | Torres | ruben35@adventure-works.com |
| 11003 | Christy | | Zhu | christy12@adventure-works.com |
| 11004 | Elizabeth | | Johnson | elizabeth5@adventure-works.com |

Join Kind

Right Anti (rows only in second)

OK

Cancel

But right Anti Join will give you a result that looks a bit weird if you do not expand the table;

| | CustomerKey | FirstName | MiddleName | BirthDate | MaritalStatus | Suffix | | Customer table from website |
|---|-------------|-----------|------------|-----------|---------------|--------|------|-----------------------------|
| 1 | | null | null | null | null | null | null | null Table |

You DO NEED to expand the table for the second query to get mismatch rows when using RIGHT ANIT Join. Which is an extra step but still works fine. You can remove all columns from the first table and expand the last column;

| | CustomerKey | FirstName | MiddleName | LastName | EmailAddress |
|---|-------------|-----------|------------|----------|------------------------------|
| 1 | 11013 | Ian | M | Jenkins | ian47@adventure-works.com |
| 2 | 11014 | Sydney | | Bennett | sydney23@adventure-works.com |
| 3 | 11015 | Chloe | | Young | chloe23@adventure-works.com |
| 4 | 11016 | Wyatt | L | Hill | wyatt32@adventure-works.com |

Left Anti with Changing Order of Tables; Works similar to Right Anti

Alternatively, you can switch the order of tables at the time of Merge and use Left Anti join to get the same output;

Merge ×

Select tables and matching columns to create a merged table.

Customer table from website

| CustomerKey | FirstName | MiddleName | LastName | EmailAddress |
|-------------|-----------|------------|----------|--------------------------------|
| 11000 | Jon | V | Yang | jon24@adventure-works.com |
| 11001 | Eugene | L | Huang | eugene10@adventure-works.com |
| 11002 | Ruben | null | Torres | ruben35@adventure-works.com |
| 11003 | Christy | null | Zhu | christy12@adventure-works.com |
| 11004 | Elizabeth | null | Johnson | elizabeth5@adventure-works.com |

Customer table from application

| CustomerKey | FirstName | MiddleName | LastName | BirthDate | MaritalStatus | Suffix | Gender |
|-------------|-----------|------------|----------|------------|---------------|--------|--------|
| 11000 | Jon | V | Yang | 8/04/1966 | M | null | M |
| 11001 | Eugene | L | Huang | 14/05/1965 | S | null | M |
| 11002 | Ruben | null | Torres | 12/08/1965 | M | null | M |
| 11003 | Christy | null | Zhu | 15/02/1968 | S | null | F |
| 11004 | Elizabeth | null | Johnson | 8/08/1968 | S | null | F |

Join Kind

Left Anti (rows only in first)

The selection has matched 18476 out of the first 18480 rows.

OK Cancel

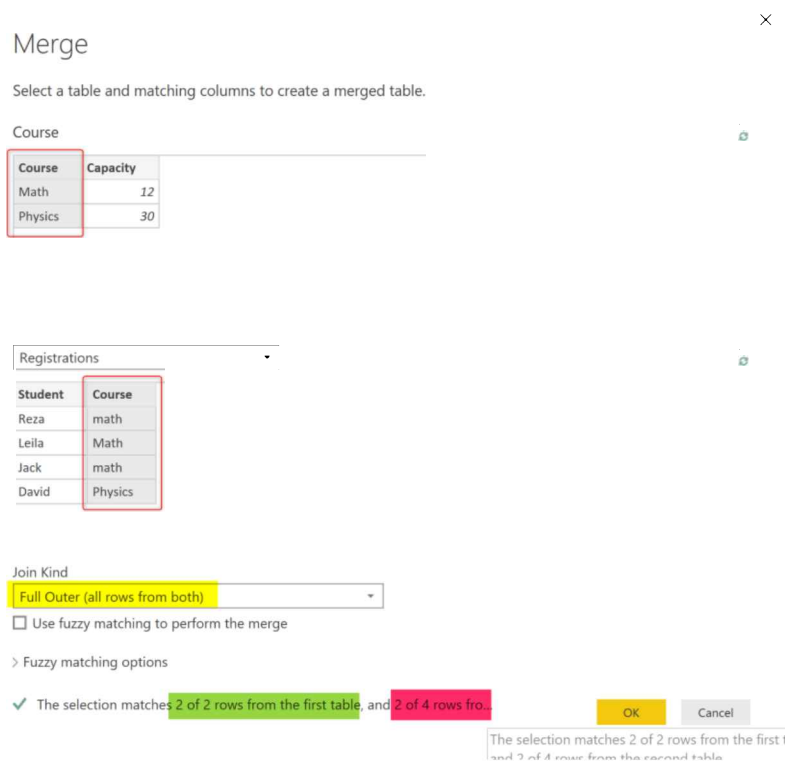
The output of this is similar to the step we have done before (it gives you the records only in the second table “customers from the website”):

| | CustomerKey | FirstName | MiddleName | LastName | EmailAddress | Customer table from application |
|---|-------------|-----------|------------|--------------|------------------------------|---------------------------------|
| 1 | 11013 | Ian | M | Jenkins | ian47@adventure-works.com | Table |
| 2 | 11014 | Sydney | | null Bennett | sydney23@adventure-works.com | Table |
| 3 | 11015 | Chloe | | null Young | chloe23@adventure-works.com | Table |
| 4 | 11016 | Wyatt | L | Hill | wyatt32@adventure-works.com | Table |

Summary

In summary, this chapter focused on two of the least common join types in Power Query; **Left Anti** join and **Right Anti** join. These two join types are very useful for finding records that exist in one of the tables but not the other. All you need is to select the right order of tables and merge type in the Merge command graphical interface.

Chapter 9: Be Careful When Merging on Text Fields in Power BI Using Power Query



Merging two tables in Power Query is one of the most useful transformations to combine data from different datasets and create a flattened data structure. However, if you use a text field for merge, then you might see some unexpected results. Here in this chapter, I share some important considerations for using a text field for merge.

What is Merge?

Merge action in Power Query is a way to have two tables with one or more joining fields to match their records and create a flattened table.

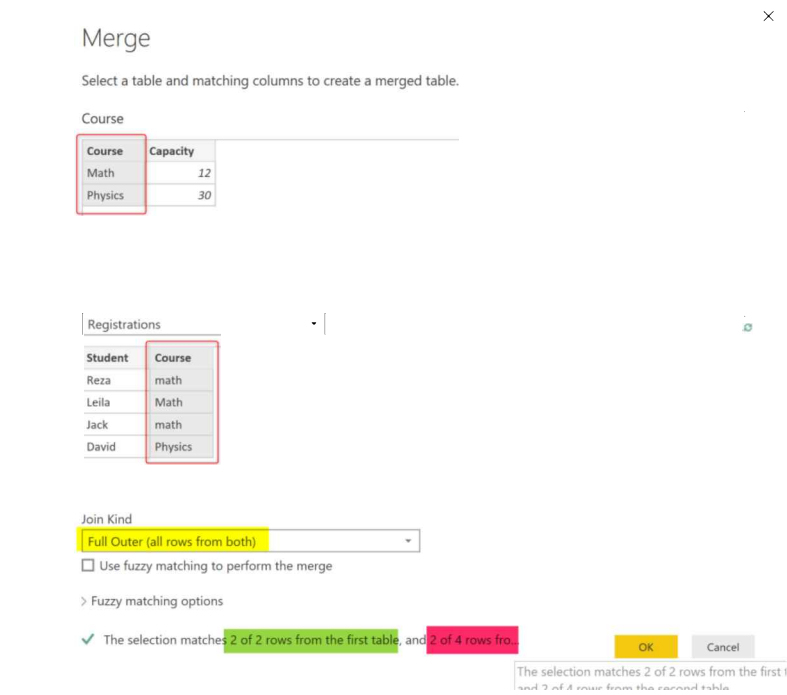
Merging on the Text Field

In an ideal data model design, you merge tables based on a numeric ID field. However, it happens that you want to merge tables with their text fields. For those scenarios, you have to remember a very important tip:

Power Query is case-sensitive. **One** and **one** won't be matched in Power Query.

When the Merge on the Text Field Doesn't Work

Below is an example of two tables matched using Power Query merge, and you can see that the match didn't work as we want it;



As you can see in the screenshot above, I have two tables; one is Courses, which has two courses: Math and Physics. The other one is Registrations, which says which student signed up for which course. There is no course value in the Course column of the registration table that does not match the Course table from our view.

I used the Full Outer join as the Join Kind, which means I will get all rows matching and not matching. However, as you can see. Some of the rows are not matching. Here is the result of the merge:

| Course | Capacity | Registrations |
|---------|----------|---------------|
| Math | 12 | Table |
| null | null | Table |
| Physics | 30 | Table |

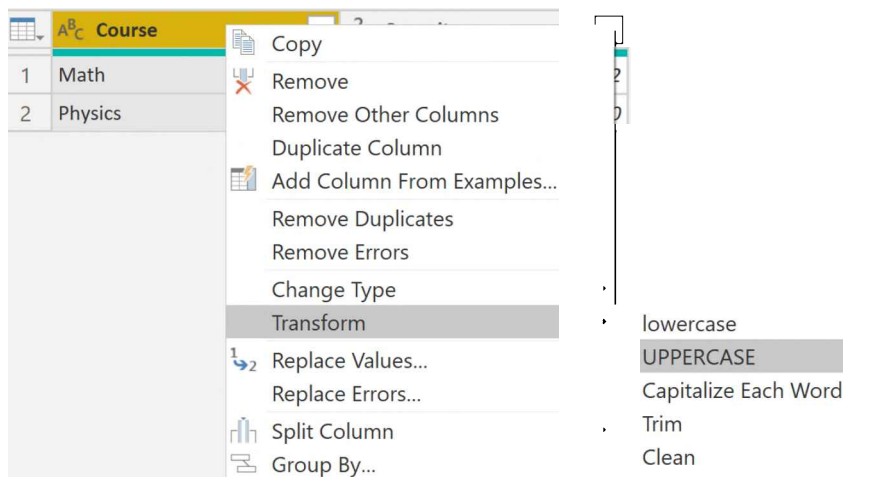
It seems there are rows in the Registration table with Course values that do not match with the course table, and it creates blank rows (rows with null values in their columns). To expand to the columns of the second table, I can use the expand column, and here is what I get:

| A ^B C Course | 1 ² 3 Capacity | A ^B C Registrations.Student | A ^B C Registrations.Course |
|-------------------------|---------------------------|--|---------------------------------------|
| Math | 12 | Leila | Math |
| null | null | Reza | math |
| null | null | Jack | math |
| Physics | 30 | David | Physics |

You can see that two rows with the course value of “math” don’t match. This is due to the fact I mentioned earlier. Power Query is case-sensitive. “Math” in the Course table doesn’t match with “math” in the Registrations table.

Change to UPPER CASE or lower case Before Merge

The easy solution for this is first to change the case of the column in both tables to either UPPER or lower case. (they both to be the same). There is also an option to capitalize each word.



After doing the same transformation in both tables before the Merge, then the result should be better:

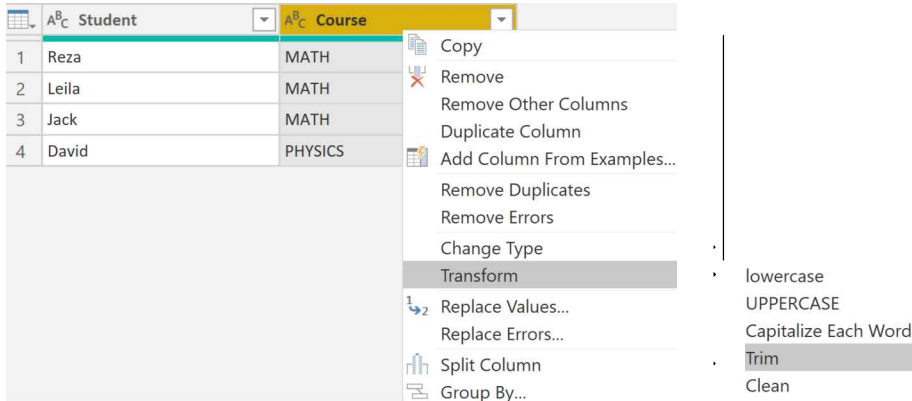
| A ^B C Course | 1 ² 3 Capacity | A ^B C Registrations.Student | A ^B C Registrations.Course |
|-------------------------|---------------------------|--|---------------------------------------|
| MATH | 12 | Reza | MATH |
| MATH | 12 | Leila | MATH |
| null | null | Jack | MATH |
| PHYSICS | 30 | David | PHYSICS |

Part 1: Combining data tables

This time I got three rows matching. However, there is still one row not matching. That is because the value in the Registration table is “MATH ” with space at the end. Which doesn’t match with “MATH.”

Trimming spaces

In the text fields, trimming is most of the time helpful. Trim the text column in both tables before the merge;



The screenshot shows a table with two columns: 'Student' and 'Course'. The 'Course' column has values 'MATH', 'MATH', 'MATH', and 'PHYSICS'. A context menu is open over the 'Course' column, and the 'Trim' option is highlighted. The menu also includes options like 'Copy', 'Remove', 'Duplicate Column', 'Add Column From Examples...', 'Remove Duplicates', 'Remove Errors', 'Change Type', 'Transform', 'Replace Values...', 'Replace Errors...', 'Split Column', and 'Group By...'. The 'Transform' sub-menu is also visible, showing options like 'lowercase', 'UPPERCASE', 'Capitalize Each Word', 'Trim', and 'Clean'.

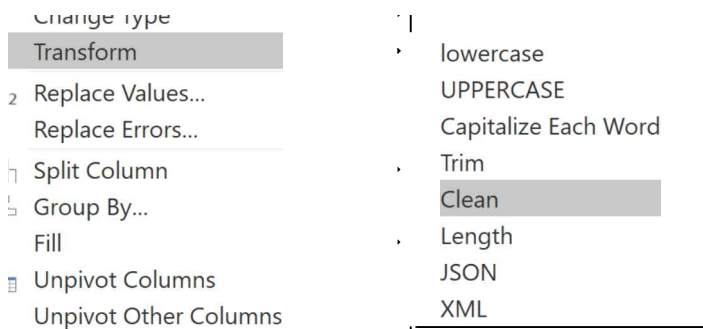
| | Student | Course |
|---|---------|---------|
| 1 | Reza | MATH |
| 2 | Leila | MATH |
| 3 | Jack | MATH |
| 4 | David | PHYSICS |

After doing the trimming in both tables, my results are now all matching;

| Course | Capacity | Registrations.Student | Registrations.Course |
|---------|----------|-----------------------|----------------------|
| MATH | 12 | Reza | MATH |
| MATH | 12 | Leila | MATH |
| MATH | 12 | Jack | MATH |
| PHYSICS | 30 | David | PHYSICS |

Other transformations to keep the two text fields the same

There are a few other transformations that can be helpful to make sure the two text fields are the same. Clean is another good transformation to use. Usually, getting the Length of the text field is useful for spotting the differences.



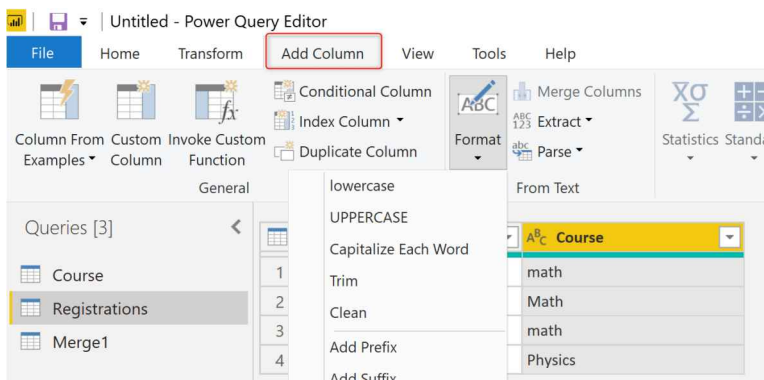
The screenshot shows a context menu with the 'Transform' option highlighted. The 'Transform' sub-menu is also visible, showing options like 'lowercase', 'UPPERCASE', 'Capitalize Each Word', 'Trim', 'Clean', 'Length', 'JSON', and 'XML'. The 'Clean' option is highlighted in the sub-menu.

| Change type |
|-----------------------|
| Transform |
| Replace Values... |
| Replace Errors... |
| Split Column |
| Group By... |
| Fill |
| Unpivot Columns |
| Unpivot Other Columns |

These facts are not just for merging text fields. Later in this book, a chapter about it explains that when you want to do Remove Duplicate on text fields, you have to be aware of these too.

Keep the Original Column Intact

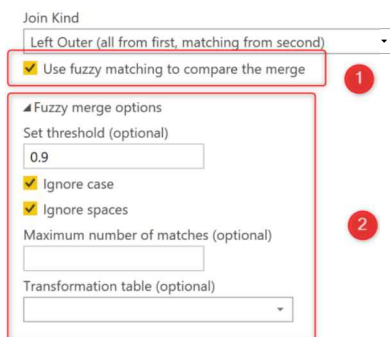
Suppose you wish to keep the original column (like Course in this example) intact and don't want it to become UPPER or lower case after the merge. Then you can have a copy of that column before doing the UPPER or lower case transformation. or choose these transformations from the Add columns tab, so it creates a new column that you use for the merge and remove it after the merge.



Consider Fuzzy Matching, But Carefully

Text values can be different, no matter what you do. Because they usually come from a data entry form, users might enter different values; sometimes, they might enter Reza Rad, Reza Raad, and many other variations.

There is a very useful transformation in Power Query called Fuzzy Merge, which matches the rows based on their similarity, and you can adjust the threshold of that similarity. I have written about that in a chapter in this book.



Part 1: Combining data tables

However, when using Fuzzy Matching, be careful of the number of rows in both tables before the merge. Fuzzy matching checks every row of one table against every row of the second table and checks for similarity. It can be slow, better to do all your normal matching and then do the fuzzy matching.

Summary

Merge in Power Query is best to be done in numeric fields. However, sometimes you don't have any other options. If you are merging based on text fields, be mindful of doing some transformations such as UPPER or lower and Trim and Clean, and use Length to make sure the fields match each other's format. You can enhance the results with Fuzzy Matching, but better to do exact matching first to get the best performance.

Chapter 10: Dates Between Merge Join in Power Query

Dates Between Merge with Matching Grains

| | CustomerID | Name | City | FromDate | ToDate |
|---|------------|------|-----------|------------|------------|
| 1 | 1 | Reza | Auckland | 1/01/2017 | 31/12/2017 |
| 2 | 2 | John | Sydney | 1/01/2017 | 19/02/2017 |
| 3 | 2 | John | Melbourne | 20/02/2017 | 31/12/2017 |

| ID | CustomerID | ProductID | SalesAmount | Date |
|----|------------|-----------|-------------|------------|
| 1 | 1 | 234 | 100 | 15/01/2017 |
| 2 | 2 | 123 | 20 | 20/01/2017 |
| 3 | 1 | 4 | 120 | 14/03/2017 |
| 4 | 2 | 23 | 60 | 25/04/2017 |

| CustomerID | Name | City | Dates |
|------------|------|-----------|------------|
| 2 | John | Sydney | 11/02/2017 |
| 2 | John | Sydney | 12/02/2017 |
| 2 | John | Sydney | 13/02/2017 |
| 2 | John | Sydney | 14/02/2017 |
| 2 | John | Sydney | 15/02/2017 |
| 2 | John | Sydney | 16/02/2017 |
| 2 | John | Sydney | 17/02/2017 |
| 2 | John | Sydney | 18/02/2017 |
| 2 | John | Sydney | 19/02/2017 |
| 2 | John | Melbourne | 20/02/2017 |
| 2 | John | Melbourne | 21/02/2017 |
| 2 | John | Melbourne | 22/02/2017 |
| 2 | John | Melbourne | 23/02/2017 |
| 2 | John | Melbourne | 24/02/2017 |
| 2 | John | Melbourne | 25/02/2017 |
| 2 | John | Melbourne | 26/02/2017 |

Using Merge in Power Query allows you to join on an EQUAL join with one or more fields between two tables. However, in some situations, you need to do the Merge Join not based on equality of values, based on other comparison options. One of the very common use cases is to Merge Join two queries based on dates between. In this example, I will show you how to use Merge Join to merge based on dates.

Download Sample Data Set

Download the data set and sample from this book’s code files.

Problem Definition

There are some situations that you need to join two tables based on dates between the not exact match of two dates. For example; consider the scenario below:

There are two tables; the Sales table includes sales transactions by Customer, Product, and Date. The Customer table has detailed information about the customer, including ID, Name, and City. Here is a screenshot of the Sales Table:

| | 1 ² ₃ ID | 1 ² ₃ CustomerID | 1 ² ₃ ProductID | 1 ² ₃ SalesAmount | Date | |
|---|--------------------------------|--|---------------------------------------|---|------|------------|
| 1 | | 1 | 1 | 234 | 100 | 15/01/2017 |
| 2 | | 2 | 2 | 123 | 20 | 20/01/2017 |
| 3 | | 3 | 1 | 4 | 120 | 14/03/2017 |
| 4 | | 4 | 2 | 23 | 60 | 25/04/2017 |

The customer’s table has the history details of changes through time. For example, customer ID 2 has a track of change. John was living in Sydney for a period of time, then moved to Melbourne after that.

Part 1: Combining data tables

| | CustomerID | Name | City | FromDate | ToDate |
|---|------------|------|-----------|------------|------------|
| 1 | 1 | Reza | Auckland | 1/01/2017 | 31/12/2017 |
| 2 | 2 | John | Sydney | 1/01/2017 | 19/02/2017 |
| 3 | 2 | John | Melbourne | 20/02/2017 | 31/12/2017 |

We are trying to solve the problem of joining these two tables based on their customer ID and finding out the City related to that specific period. We have to check the Date field from Sales Table to fit into the FromDate and ToDate of the Customer table.

Grain Matching

One of the easiest ways of matching two tables is to bring them both to the same grain. In this example, Sales Table is at the grain of Customer, Product, and Date. However, the Customer table is at the grain of Customer and a change in properties such as City. We can change the grain of the customer table to be on Customer and Date. That means having one record per every customer and every day.

Before applying this change, there is a little warning I would like to explain; by changing the grain of a table to a more detailed grain, the number of rows for that table will increase significantly. It is fine to do it as an intermediate change, but if you want to make it a final query to be loaded in Power BI, you need to think about your approach more carefully.

Step 1: Calculating Duration

The first step in this approach is to determine the duration between FromDate and ToDate in the customer table for each row. That simply can be calculated by selecting two columns (First ToDate, then FromDate), then From Add Column Tab, under Date, Subtract Days.

The screenshot shows the Power BI Query Editor interface. The 'Queries' pane on the left lists 'Sales' and 'Customer'. The 'Customer' query is selected. The main area displays a table with columns: CustomerID, Name, City, FromDate, and ToDate. The 'Add Column' tab is active, and the 'Date' dropdown menu is open, showing options like 'Age', 'Date Only', 'Parse', 'Year', 'Month', 'Quarter', 'Week', 'Day', and 'Subtract Days'. The 'Subtract Days' option is highlighted. A red box is drawn around the 'Subtract Days' option. The formula bar at the top shows the formula: = Table.TransformColumnTypes("#Promoted Headers",{"CustomerID", Int64}).

Then you will see the new column added, which is the duration between From and To dates

| | 1 ² CustomerID | A ^B C Name | A ^B C City | FromDate | ToDate | 1 ² 3 DateDifference |
|---|---------------------------|-----------------------|-----------------------|------------|------------|---------------------------------|
| 1 | | 1 Reza | Auckland | 1/01/2017 | 31/12/2017 | 364 |
| 2 | | 2 John | Sydney | 1/01/2017 | 19/02/2017 | 49 |
| 3 | | 2 John | Melbourne | 20/02/2017 | 31/12/2017 | 314 |

Step 2: Creating List of Dates

The second step is to create a list of dates for every record, starting from FromDate, adding one day at a time for the number of occurrences in the DateDifference column.

There is a generator that you can easily use to create a list of dates. List.Dates is a Power Query function that will generate a list of dates. Here is the syntax for this table;

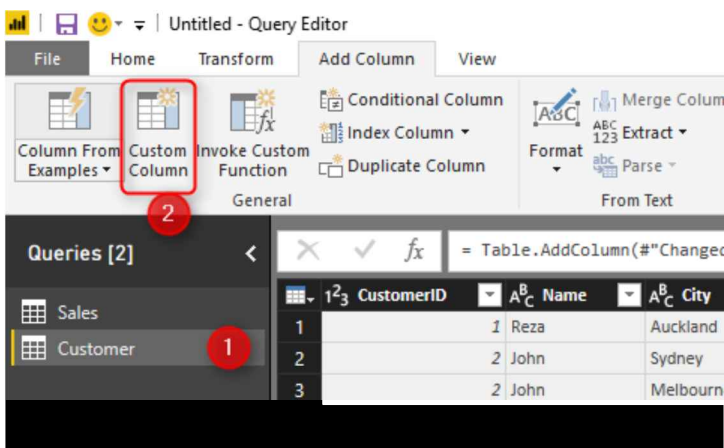
List.Dates(<start date>,<occurrence>,<duration>)

- start date in this scenario will come from the FromDate column
- Occurrence would come from DateDifference plus one.
- Duration should be in a day Level. Duration has 4 input arguments:

#duration(<day>,<hour>,<minute>,<second>)

a daily duration would be: #duration(1,0,0,0)

So, we need to add a custom column to our table;



The custom column expression can be as below;

List.Dates([FromDate],[DateDifference]+1,#duration(1,0,0,0))

I named this column as Dates.

Here is the result:

Part 1: Combining data tables

Table: AddColumn("Inserted Date Subtraction", "Dates", each List.Dates([FromDate],[DateDifference]+1,#duration(1,0,0,0)))

| CustomerID | Name | City | FromDate | ToDate | DateDifference | Dates |
|------------|------|-----------|------------|------------|----------------|-------|
| 1 | Reza | Auckland | 1/01/2017 | 31/12/2017 | 364 | List |
| 2 | John | Sydney | 1/01/2017 | 19/02/2017 | 49 | List |
| 3 | John | Melbourne | 20/02/2017 | 31/12/2017 | 314 | List |

Expanded Dates list (for CustomerID 1):

- 20/02/2017
- 21/02/2017
- 22/02/2017
- 23/02/2017
- 24/02/2017
- 25/02/2017
- 26/02/2017
- 27/02/2017
- 28/02/2017
- 1/03/2017
- 2/03/2017
- 3/03/2017
- 4/03/2017
- 5/03/2017
- 6/03/2017
- 7/03/2017
- 8/03/2017
- 9/03/2017
- 10/03/2017
- 11/03/2017

The Dates column now has a list in every row. This list is a list of dates. The next step is to expand it.

Step 3: Expand List to Day Level

The last step to change the grain of this table is to expand the Dates column. To expand, just click on Expand button.

Table: AddColumn("Inserted Date Subtraction", "Dates", each List.Dates([FromDate],[DateDifference],#duration(1,0,0,0)))

| CustomerID | Name | City | FromDate | ToDate | DateDifference | Dates |
|------------|------|-----------|------------|------------|----------------|-------|
| 1 | Reza | Auckland | 1/01/2017 | 31/12/2017 | 364 | List |
| 2 | John | Sydney | 1/01/2017 | 19/02/2017 | 49 | List |
| 3 | John | Melbourne | 20/02/2017 | 31/12/2017 | 314 | List |

Context Menu:

- Expand to New Rows
- Extract Values...

Expanding to new rows will give you a data set with all dates;

| CustomerID | Name | City | FromDate | ToDate | DateDifference | Dates |
|------------|------|----------|-----------|------------|----------------|------------|
| 1 | Reza | Auckland | 1/01/2017 | 31/12/2017 | 364 | 1/01/2017 |
| 2 | Reza | Auckland | 1/01/2017 | 31/12/2017 | 364 | 2/01/2017 |
| 3 | Reza | Auckland | 1/01/2017 | 31/12/2017 | 364 | 3/01/2017 |
| 4 | Reza | Auckland | 1/01/2017 | 31/12/2017 | 364 | 4/01/2017 |
| 5 | Reza | Auckland | 1/01/2017 | 31/12/2017 | 364 | 5/01/2017 |
| 6 | Reza | Auckland | 1/01/2017 | 31/12/2017 | 364 | 6/01/2017 |
| 7 | Reza | Auckland | 1/01/2017 | 31/12/2017 | 364 | 7/01/2017 |
| 8 | Reza | Auckland | 1/01/2017 | 31/12/2017 | 364 | 8/01/2017 |
| 9 | Reza | Auckland | 1/01/2017 | 31/12/2017 | 364 | 9/01/2017 |
| 10 | Reza | Auckland | 1/01/2017 | 31/12/2017 | 364 | 10/01/2017 |
| 11 | Reza | Auckland | 1/01/2017 | 31/12/2017 | 364 | 11/01/2017 |

Now you can remove FromDate, ToDate, and DateDifference. We don't need these three columns anymore.

| | 123 CustomerID | A ^B _C Name | A ^B _C City | A ^B _C 123 Dates |
|-----|----------------|----------------------------------|----------------------------------|---------------------------------------|
| 407 | 2 | John | Sydney | 11/02/2017 |
| 408 | 2 | John | Sydney | 12/02/2017 |
| 409 | 2 | John | Sydney | 13/02/2017 |
| 410 | 2 | John | Sydney | 14/02/2017 |
| 411 | 2 | John | Sydney | 15/02/2017 |
| 412 | 2 | John | Sydney | 16/02/2017 |
| 413 | 2 | John | Sydney | 17/02/2017 |
| 414 | 2 | John | Sydney | 18/02/2017 |
| 415 | 2 | John | Sydney | 19/02/2017 |
| 416 | 2 | John | Melbourne | 20/02/2017 |
| 417 | 2 | John | Melbourne | 21/02/2017 |
| 418 | 2 | John | Melbourne | 22/02/2017 |
| 419 | 2 | John | Melbourne | 23/02/2017 |
| 420 | 2 | John | Melbourne | 24/02/2017 |
| 421 | 2 | John | Melbourne | 25/02/2017 |
| 422 | 2 | John | Melbourne | 26/02/2017 |

The table above is the same customer table but on different grain. We can now easily see on which dates John was in Sydney and which dates in Melbourne. This table now can be easily merged with the sales table.

Merging Tables on the Same Grain

When both tables are at the same grain, then you can easily merge them.

Part 1: Combining data tables

Merge

Select tables and matching columns to create a merged table.

Sales

| ID | CustomerID | ProductID | SalesAmount | Date |
|----|------------|-----------|-------------|------------|
| 1 | 1 | 234 | 100 | 15/01/2017 |
| 2 | 2 | 123 | 20 | 20/01/2017 |
| 3 | 1 | 4 | 120 | 14/03/2017 |
| 4 | 2 | 23 | 60 | 25/04/2017 |

Customer

| CustomerID | Name | City | Dates |
|------------|------|----------|-----------|
| 1 | Reza | Auckland | 1/01/2017 |
| 1 | Reza | Auckland | 2/01/2017 |
| 1 | Reza | Auckland | 3/01/2017 |
| 1 | Reza | Auckland | 4/01/2017 |
| 1 | Reza | Auckland | 5/01/2017 |

Join Kind

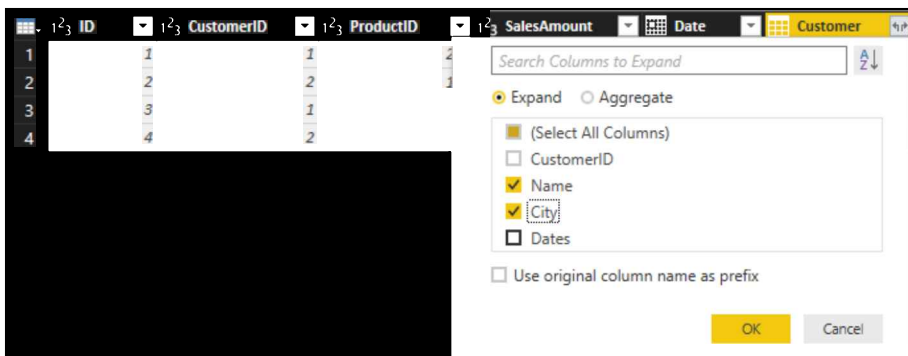
Left Outer (all from first, matching from second)

✓ The selection has matched 4 out of the first 4 rows.

OK

Cancel

Merge should be between two tables, based on CustomerID and Dates. You need to hold the Ctrl key to select more than one column. And make sure you select them in the same order in both tables. After merge then you can expand and only select City and Name from the other table;

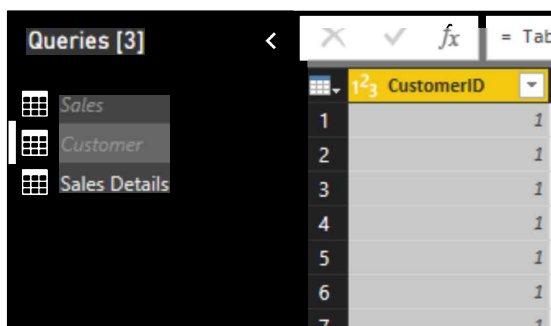


The final result shows that two sales transactions for John happened at two different times that John has been in two different cities of Sydney and Melbourne.

| ID | CustomerID | ProductID | SalesAmount | Date | Name | City |
|----|------------|-----------|-------------|------------|------------|----------|
| 1 | 1 | 234 | 100 | 15/01/2017 | Reza | Auckland |
| 2 | 3 | 1 | 4 | 120 | 14/03/2017 | Reza |
| 3 | 2 | 2 | 123 | 20 | 20/01/2017 | John |
| 4 | 4 | 2 | 23 | 60 | 25/04/2017 | John |

Final Step: Cleansing

You won't need the first two tables after merging them. You can disable their load to avoid extra memory consumption (especially for the Customer table, which should be big after grain change). To learn more about Enable Load and solve performance issues, read the chapter about it in this book.

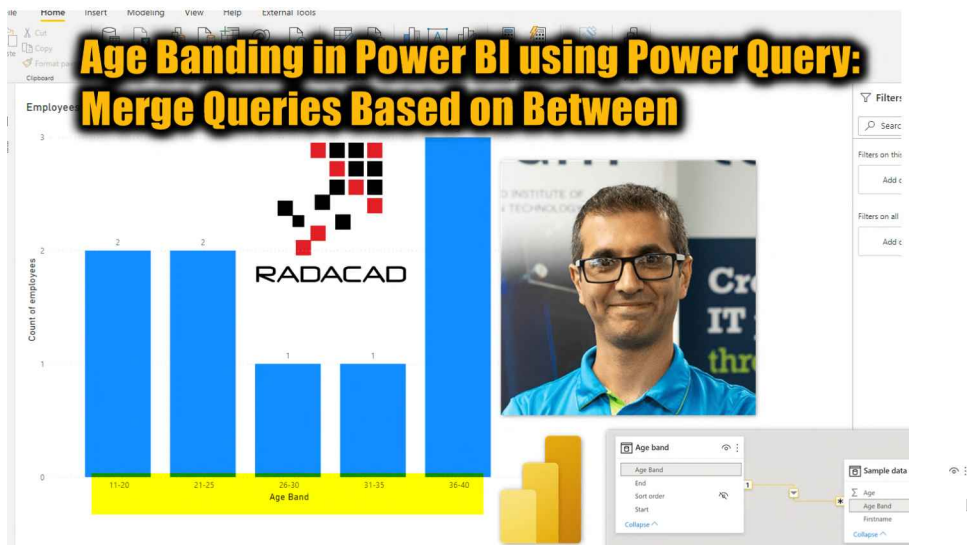


| 1^2_3 | CustomerID |
|-------|------------|
| 1 | 1 |
| 2 | 1 |
| 3 | 1 |
| 4 | 1 |
| 5 | 1 |
| 6 | 1 |
| 7 | 1 |

Summary

There are multiple ways of joining two tables based on the non-equality comparison. Matching grain is one of them and works perfectly fine, and is easy to implement. In this chapter, you've learned how to use grain matching to do this joining and get the join result based on dates between comparisons. With this method, be careful to disable the load of the table that you've changed the grain for it to avoid performance issues afterward.

Chapter 11: Age Banding in Power BI using Power Query – Merge Queries Based on Between



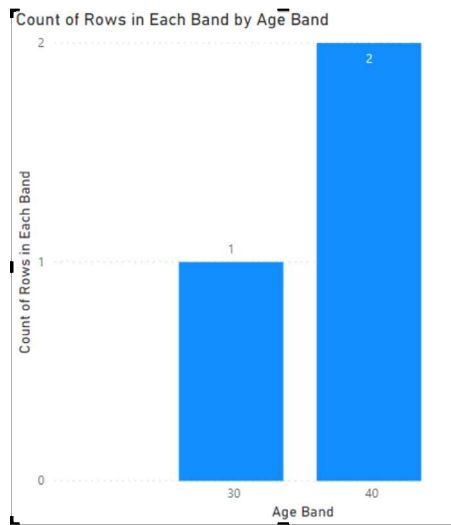
Age banding is a common scenario in analytics, and I have explained how it is possible in many different ways. One of the methods I have explained was using DAX measures and the TREATAS function. Sometimes, however, the age banding can be static, and Power Query can be used for the calculation, which would make the report faster at the end. In this chapter, I'll explain how you can have an age band table with columns of start and end and how Power Query can merge your main table with this table based on Between the start and end columns.

Sample Model

I am using a very simple data model in this example; the table below is what I use as the Sample Data;

| First Name | Last Name | Age |
|------------|-----------|-----|
| Reza | Rad | 40 |
| Mick | Peterson | 34 |
| Joe | White | 23 |

The goal is to have an age group banding for customers and count customers in each group. Something similar to this:



desired outcome

Age Band Table

As we need the banding to be the axis of the chart, we need that as a field. You can create an age band table.

and here is an example of the age band table:

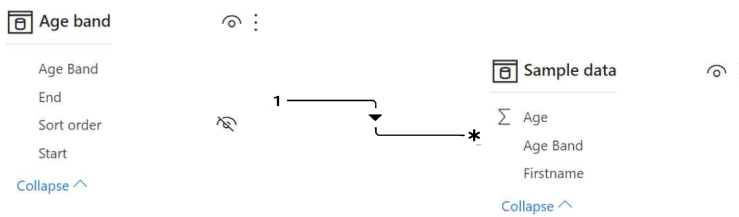
| Age Band | Start | End | Sort Order |
|----------|-------|-----|------------|
| 1-10 | 1 | 10 | 1 |
| 11-20 | 11 | 20 | 2 |
| 21-25 | 21 | 25 | 3 |
| 26-30 | 26 | 30 | 4 |
| 31-35 | 31 | 35 | 5 |
| 36-40 | 36 | 40 | 6 |

As you see, I have bands that only cover five years (31-35) and bands covering ten years (11-20).

This table can't have a relationship with the Sample Data table because if you create the relationship, it would only filter data for the start or end of each band, but not the whole range. So the tables remain unrelated.

However, our goal is to create a relationship like below to use the age band in visuals.

Part 1: Combining data tables

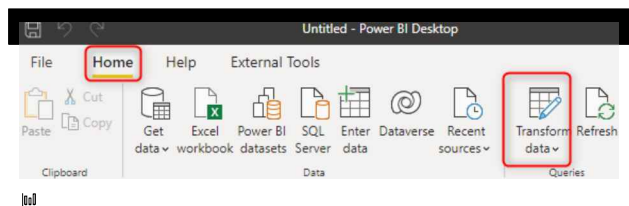


Power Query Merge Between

In Power Query, when you Merge two tables, the merge operation is always based on equal values. Unless you use Fuzzy matching. The fuzzy matching is not based on exact equal, but on a threshold close to the equal anyway.

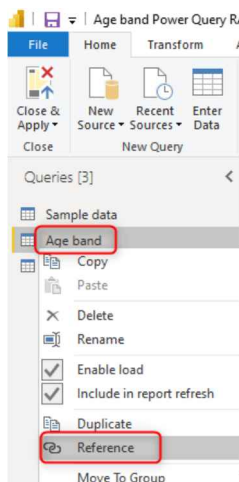
This means you cannot merge two tables if the field Age of the Sample data table is between the fields Start and End in the Age band table. But there is a trick to do that; let's see how it works.

Before starting this, you have to go to Power Query Editor window using the Transform Data in Power BI Desktop.



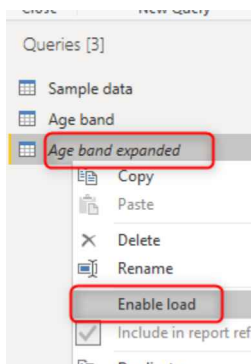
Reference table from the Age band

Doing merge-between operations in Power Query is not complicated. I have explained once how you can do that with date fields. You can use a similar approach here. The very first step is to create a reference from the age band table. We do need to create the reference because we are changing the structure of this table and will be using this new table to merge back to the main table.



reference from a Query in Power BI

Once the reference is created, you can name it as Age band expanded and disable the load of it (this is a temporary table that will not be loaded directly into Power BI)



load disabled for the referenced table

Expand the age range

In the age band extended table, you need to create one row per age. That is why I call it an expanded table. We need to create a list of numbers from the Start to the End in each row and then expand it.

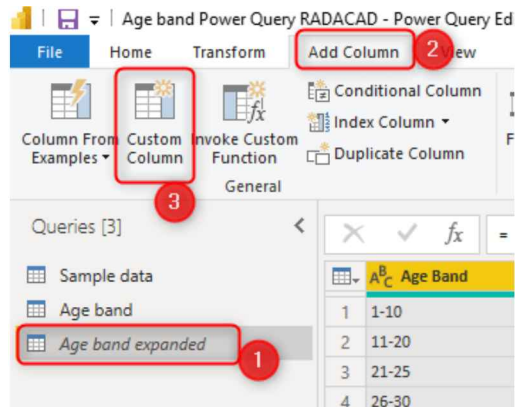
In Power Query, you can create a list using the formula below;

```
{<start number>..<end number>}
```

This will create a list from the start number to the end number incrementing one number each time.

Add a new custom column to the Age band expanded table

Part 1: Combining data tables



Add custom column in Power Query

Set the custom column name to Age, and in the formula, write the below expression;

Custom Column

Add a column that is computed from the other columns.

New column name

Custom column formula ⓘ

Available columns

- Age Band
- Sort order
- Start
- End

<< Insert

✓ No syntax errors have been detected.

OK Cancel

create a list of numbers for each row in Power Query

This will create a list of numbers for each row, from the Start to the End

Chapter 11: Age Banding in Power BI using Power Query – Merge Queries Based on Between

`= Table.AddColumn(Source, "Age", each {[Start]..[End]})`

| Age Band | Sort order | Start | End | Age |
|----------|------------|-------|-----|------|
| 1-10 | 1 | 1 | 10 | List |
| 11-20 | 2 | 11 | 20 | List |
| 21-25 | 3 | 21 | 25 | List |
| 26-30 | 4 | 26 | 30 | List |
| 31-35 | 5 | 31 | 35 | List |
| 36-40 | 6 | 36 | 40 | List |

List

21

22

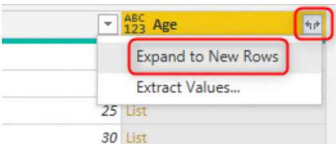
23

24

25

The list of numbers generated for each row's start-end

This means you can click on Expand on the new Age column and expand it to new rows



Expand to new rows

Once the expansion is done, your age band expanded table looks below with one row for each age.

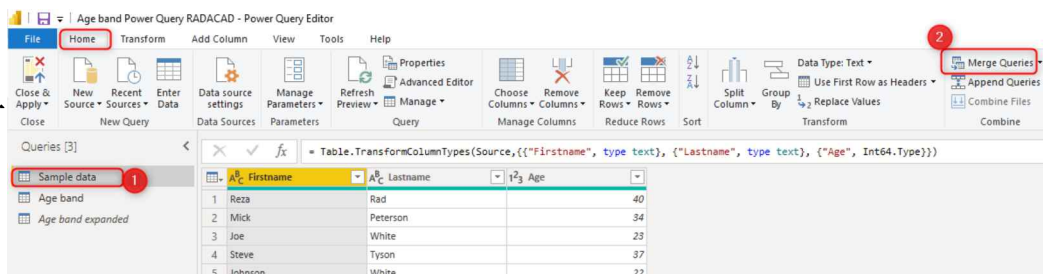
| Age Band | Sort order | Start | End | Age |
|----------|------------|-------|-----|-----|
| 1-10 | 1 | 1 | 10 | 1 |
| 1-10 | 1 | 1 | 10 | 2 |
| 1-10 | 1 | 1 | 10 | 3 |
| 1-10 | 1 | 1 | 10 | 4 |
| 1-10 | 1 | 1 | 10 | 5 |
| 1-10 | 1 | 1 | 10 | 6 |
| 1-10 | 1 | 1 | 10 | 7 |
| 1-10 | 1 | 1 | 10 | 8 |
| 1-10 | 1 | 1 | 10 | 9 |
| 1-10 | 1 | 1 | 10 | 10 |
| 11-20 | 2 | 11 | 20 | 11 |
| 11-20 | 2 | 11 | 20 | 12 |
| 11-20 | 2 | 11 | 20 | 13 |
| 11-20 | 2 | 11 | 20 | 14 |
| 11-20 | 2 | 11 | 20 | 15 |
| 11-20 | 2 | 11 | 20 | 16 |
| 11-20 | 2 | 11 | 20 | 17 |
| 11-20 | 2 | 11 | 20 | 18 |
| 11-20 | 2 | 11 | 20 | 19 |
| 11-20 | 2 | 11 | 20 | 20 |
| 21-25 | 3 | 21 | 25 | 21 |
| 21-25 | 3 | 21 | 25 | 22 |
| 21-25 | 3 | 21 | 25 | 23 |
| 21-25 | 3 | 21 | 25 | 24 |
| 21-25 | 3 | 21 | 25 | 25 |
| 26-30 | 4 | 26 | 30 | 26 |
| 26-30 | 4 | 26 | 30 | 27 |

age rows expanded

Part 1: Combining data tables

Merge with the main table

The Age band expanded table is now ready to be merged to the main table. In my case, the Sample data table can be merged into this table



merge queries

In the merge queries window, merge the two tables based on the Age column.

Merge

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

Sample data

| Firstname | Lastname | Age |
|-----------|----------|-----|
| Reza | Rad | 40 |
| Mick | Peterson | 34 |
| Joe | White | 23 |
| Steve | Tyson | 37 |
| Johnson | White | 22 |

Age band expanded

| Age Band | Sort order | Start | End | Age |
|----------|------------|-------|-----|-----|
| 1-10 | 1 | 1 | 10 | 1 |
| 1-10 | 1 | 1 | 10 | 2 |
| 1-10 | 1 | 1 | 10 | 3 |
| 1-10 | 1 | 1 | 10 | 4 |
| 1-10 | 1 | 1 | 10 | 5 |

Join Kind

Left Outer (all from first, matching from second)

☐ Use fuzzy matching to perform the merge

> Fuzzy matching options

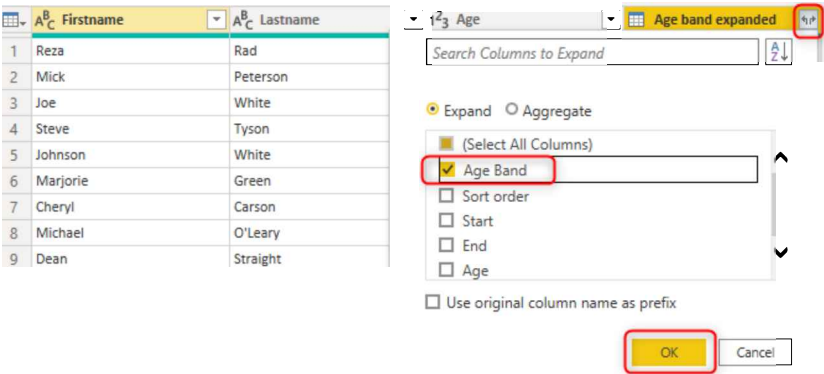
4

OK

Cancel

merge two tables based on the age column

Finally, the created table in each row can be expanded to reveal the age band of each row. You won't need other columns because we will still create a relationship to the age band table once loaded into Power BI and will have access to the columns of that table.



expand to the Age band in each row

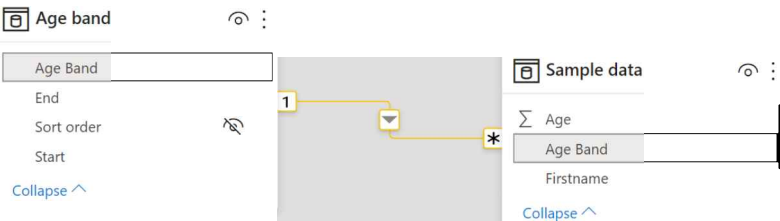
The data of the table now will have the age band in it;

| | A ^B C _C Firstname | A ^B C _C Lastname | 1 ² 3 Age | A ^B C _C Age Band |
|---|---|--|----------------------|--|
| 1 | Marjorie | Green | 17 | 11-20 |
| 2 | Michael | O'Leary | 19 | 11-20 |
| 3 | Johnson | White | 22 | 21-25 |
| 4 | Joe | White | 23 | 21-25 |
| 5 | Cheryl | Carson | 29 | 26-30 |
| 6 | Dean | Straight | 33 | 31-35 |
| 7 | Mick | Peterson | 34 | 31-35 |
| 8 | Steve | Tyson | 37 | 36-40 |
| 9 | Reza | Rad | 40 | 36-40 |

Age band added to the main table

Relationship and Modeling

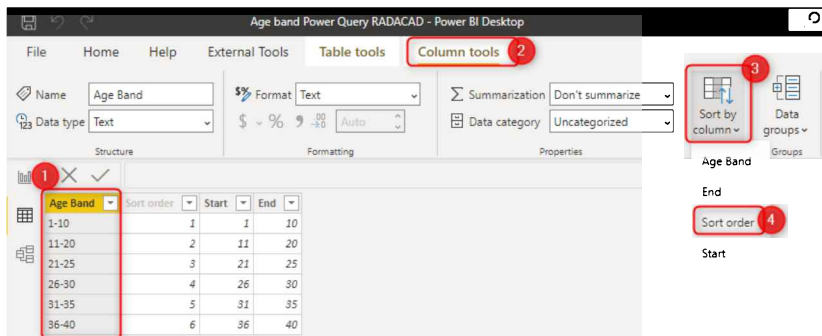
You may just need the Age band column in the main table and nothing else. In that case, you don't even need to load the Age Band table. However, based on my experience, you will often need more than just a column. You may need to have the order of age bands (sort order column) or some other columns explaining something about that age band. that is why you may need to load the Age Band table into Power BI and then create a relationship to the main table based on the Age Band.



Relationship is created based on Age Band

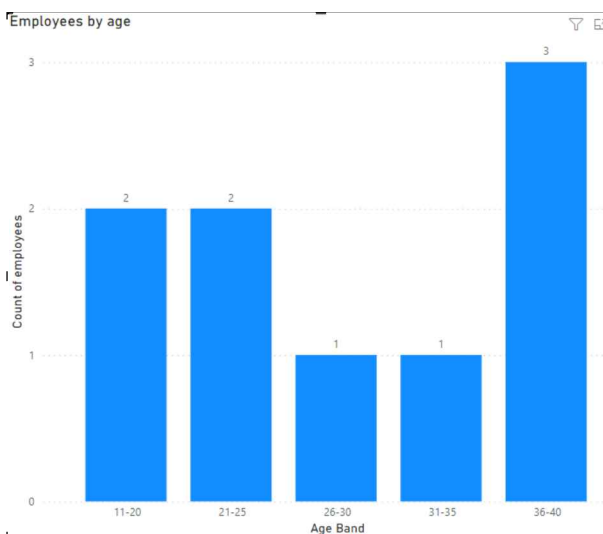
Part 1: Combining data tables

You can also [sort the Age Band column by the Sort Order column](https://radacad.com/sort-a-column-with-a-custom-order-in-power-bi)[\[https://radacad.com/sort-a-column-with-a-custom-order-in-power-bi\]](https://radacad.com/sort-a-column-with-a-custom-order-in-power-bi) and then [hide the sort order column](https://radacad.com/what-fields-to-hide-in-your-power-bi-solution)[\[https://radacad.com/what-fields-to-hide-in-your-power-bi-solution\]](https://radacad.com/what-fields-to-hide-in-your-power-bi-solution).



sort one column by another column in Power BI

After doing all these, you can visualize the data in a Power BI report



Age band visualization in Power BI

Summary

The method explained here was static age banding with custom band settings. Each age band can have a different start and end. Some bands might be smaller or larger than others. We used the Power Query Merge operation to create a connection between the main data table with the age band table. You learned a trick in Power Query to implement a merge based on BETWEEN.

Part 2: Grouping and aggregation

Chapter 12: Grouping in Power Query; Getting The Last Item in Each Group

| 1.2 CustomerKey | 1.2 Order Count | 1.2 Total Revenue | Order Details | 1.2 First SalesAmount | 1.2 Last SalesAmount |
|-----------------|-----------------|-------------------|----------------|-----------------------|----------------------|
| 1 | 21768 | 2 | 4118.26 Table | 3578.27 | 539.99 |
| 2 | 28389 | 1 | 3399.99 Table | 3399.99 | 3399.99 |
| 3 | 25863 | 5 | 4631.11 Table | 3399.99 | 2.29 |
| 4 | 14501 | 2 | 2994.082 Table | 699.0982 | 2294.99 |
| 5 | 11003 | 9 | 8139.29 Table | 3399.99 | 2.29 |
| 6 | 27645 | 5 | 6051.31 Table | 3578.27 | 54.99 |
| 7 | 16624 | 4 | 5938.25 Table | 3578.27 | 35 |
| 8 | 11005 | 6 | 8121.33 Table | 3374.99 | 2384.07 |
| 9 | 11011 | 4 | 8133.04 Table | 3399.99 | 53.99 |
| 10 | 27621 | 3 | 5971.33 Table | 3578.27 | 8.99 |
| 11 | 27616 | 5 | 5998.61 Table | 3578.27 | 2.29 |

| OrderDateKey | CustomerKey | PromotionKey | CurrencyKey | SalesTerritoryKey | SalesOrderNumber | SalesOrderLineNumber | RevisionNumber | OrderQuantity | UnitPrice | ExtendedPrice |
|--------------|-------------|--------------|-------------|-------------------|------------------|----------------------|----------------|---------------|-----------|---------------|
| 20050708 | 25863 | 1 | 100 | | 1 SO43699 | | 1 | 1 | 1 | 3399.99 |
| 20080203 | 25863 | 1 | 100 | | 1 SO62866 | | 1 | 1 | 1 | 1214.85 |
| 20080203 | 25863 | 1 | 100 | | 1 SO62866 | | 2 | 1 | 1 | 4.99 |
| 20080203 | 25863 | 1 | 100 | | 1 SO62866 | | 3 | 1 | 1 | 8.99 |
| 20080203 | 25863 | 2 | 100 | | 1 SO62866 | | 4 | 1 | 1 | 2.29 |

Power BI or Power Query in Excel (or Get Data and Transform as the new name) can do many data transformations. One of these transformations is grouping rows by the number of fields. Suppose you use the graphical user interface in Power Query for Power BI or Excel. In that case, you have a number of options to get aggregated results such as count of rows, maximum or minimum, an average of each group, or sum... But there are still many operations that you cannot do through GUI, such as getting the last item in each group or first item. Fortunately, with M (Power Query Formula Language), you can apply any of these operations you want. In this chapter, I'll show you how to benefit both; start with GUI to write the Group by command for you, and then customize it in M script to achieve what you want.

Learning Objectives for this section

By completing the example of this chapter, you will learn;

- How to perform Group By in Power Query/Power BI
- How to leverage pre-defined aggregation functions in Group By GUI window
- How to extend grouping possibilities with changes in Power Query script

Prerequisite

For this example, I will be using AdventureWorksDW sample Microsoft SQL Server database. You can download it from the book code files.

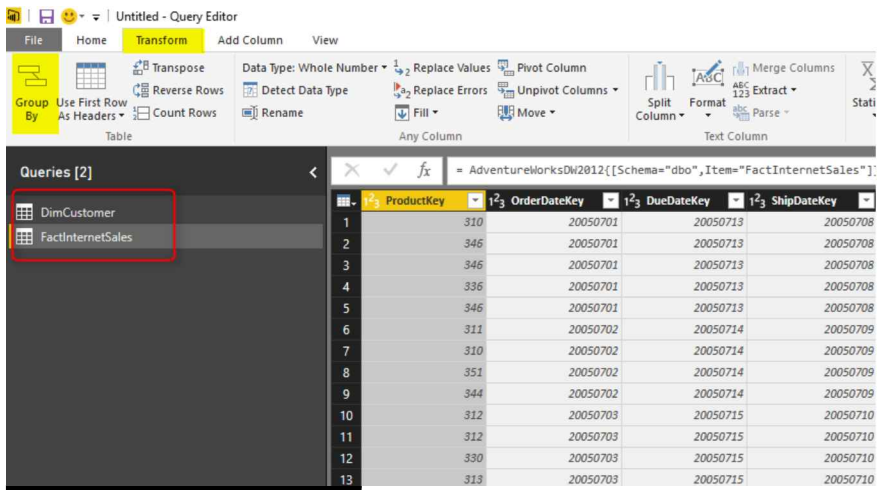
Scenario

A scenario that I want to solve as an example is this:

FactInternetSales has sales transaction information for each customer, by each product, each order date, and some other information. We want to have a grouped table by the customer, which has the number of sales transactions by each customer, total sales amount for that customer, the first, and the amount of the last sale for that customer—the first and last defined by the first and last order date for the transaction.

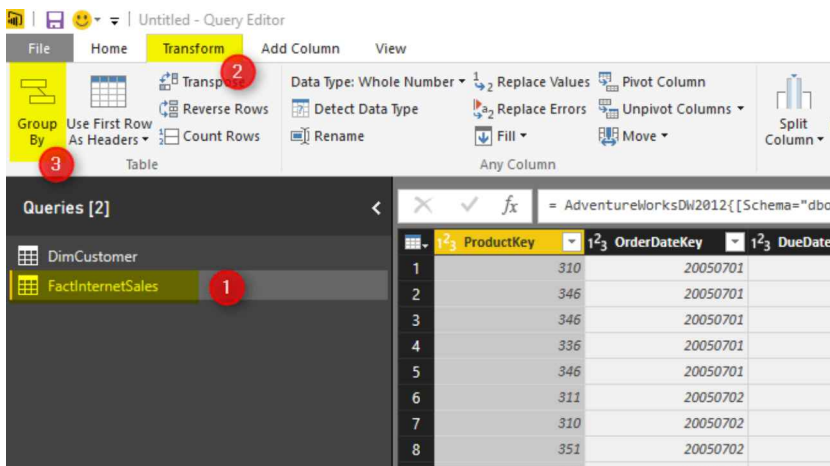
Get Data

Let's start by getting data from SQL Server, Choose AdventureWorksDW as the source database, and select DimCustomer and FactInternetSales as the only tables for this example. Click on Edit to move into the Power Query window.



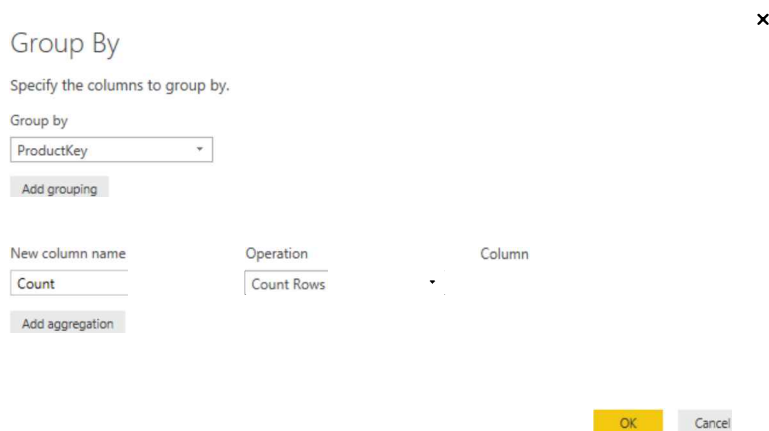
Group By Transformation

FactInternetSales table is the one we want to apply all transformations in. So Click on FactInternetSales first, then from Transform Tab, select Group By option as the first menu option.



Part 2: Grouping and aggregation

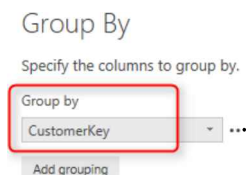
This will open the Group By dialog window with some configuration options



By default Group By happens on the selected columns. Because usually, the first column is the selected column (in our table ProductKey), then the column mentioned under group by section is also ProductKey. You can change this to another column and add or remove columns in this section.

Choose the Group By Field

Based on your final grain of the output table, the group by field will be defined. In this example, we want the final table to have one record per Customer, so CustomerKey (the identifier for each customer) should be our Group By Column.



Note that you can add as many fields as you want in the Group By section. The result would be one record per combination of unique values from all these fields.

Add Aggregation Fields

Group by result would be one record per unique combination of all fields set in the “group by ” section. In addition, you can also have some aggregated columns. Here is a list of operations you can have by default:

Group By

Specify the columns to group by.

Group by

CustomerKey

Add grouping

New column name

Count

Add aggregation

Operation

- Count Rows
- Sum
- Average
- Median
- Min
- Max
- Count Rows
- Count Distinct Rows
- All Rows

Most of the items above are self-explanatory. For example, when you want to count the number of sales transactions, you can use Count Rows. If you want the total Sales Amount for each group, you can choose Sum, and then in the Column section, choose the column as SalesAmount. All Rows will generate a sub table in each element of the aggregated table that contains all rows in that group.

Columns that I want to create in this section are:

Order Count (Count Rows), Total Revenue (Sum of Sales Amount), Order Details (All Rows)

Group By

Specify the columns to group by.

Group by

CustomerKey




Add grouping

| New column name | Operation | Column |
|-----------------|------------|-------------|
| Order Count | Count Rows | |
| Total Revenue | Sum | SalesAmount |
| Order Details | All Rows | |

Add aggregation

Adding aggregated columns is as easy as that. Now If you click on OK, you will see the result;

Part 2: Grouping and aggregation

| | 1 ² ₃ CustomerKey | 1.2 Order Count | 1.2 Total Revenue |  Order Details |  | | | |
|------------|---|-----------------|-------------------|---|---|-------------|-------------------|------------------|
| 1 | 21768 | 2 | 4118.26 | Table |  | | | |
| 2 | 28389 | 1 | 3399.99 | Table | | | | |
| 3 | 25863 | 5 | 4631.11 | Table | | | | |
| 4 | 14501 | 2 | 2994.082 | Table | | | | |
| 5 | 11003 | 9 | 8139.29 | Table | | | | |
| 6 | 27645 | 5 | 6051.31 | Table | | | | |
| 7 | 16624 | 4 | 5938.25 | Table | | | | |
| 8 | 11005 | 6 | 8121.33 | Table | | | | |
| 9 | 11011 | 4 | 8133.04 | Table | | | | |
| 10 | 27621 | 3 | 5971.33 | Table | | | | |
| 11 | 27616 | 5 | 5998.61 | Table | | | | |
| 12 | 20042 | 3 | 3150.3982 | Table | | | | |
| 13 | 16351 | 2 | 5873.26 | Table | | | | |
| 14 | 16517 | 2 | 4698.76 | Table | | | | |
| 15 | 27606 | 3 | 5997.33 | Table | | | | |
| 16 | 13513 | 4 | 5918.24 | Table | | | | |
| 17 | 27601 | 1 | 3578.27 | Table | | | | |
| 18 | 13591 | 9 | 8355.27 | Table | | | | |
| 19 | 16483 | 2 | 4118.26 | Table | | | | |
| ProductKey | OrderDateKey | DueDateKey | ShipDateKey | CustomerKey | PromotionKey | CurrencyKey | SalesTerritoryKey | SalesOrderNumber |
| 346 | 20050701 | 20050713 | 20050708 | 11003 | 1 | 6 | | 9 SO43701 |
| 361 | 20070709 | 20070721 | 20070716 | 11003 | 1 | 6 | | 9 SO51315 |
| 478 | 20070709 | 20070721 | 20070716 | 11003 | 1 | 6 | | 9 SO51315 |
| 477 | 20070709 | 20070721 | 20070716 | 11003 | 1 | 6 | | 9 SO51315 |
| 225 | 20070709 | 20070721 | 20070716 | 11003 | 1 | 6 | | 9 SO51315 |
| 564 | 20071111 | 20071123 | 20071118 | 11003 | 1 | 6 | | 9 SO57783 |
| 541 | 20071111 | 20071123 | 20071118 | 11003 | 1 | 6 | | 9 SO57783 |
| 530 | 20071111 | 20071123 | 20071118 | 11003 | 1 | 6 | | 9 SO57783 |
| 480 | 20071111 | 20071123 | 20071118 | 11003 | 1 | 6 | | 9 SO57783 |

As you can see, Order Count and Total Revenue show the aggregated result of each group, and Order Details (if you click not on the “Table” itself, but on a blank area on that cell) contains the actual rows in each group. This detailed view can be used for many other calculations or transformations later on. In many cases, you will find the All rows option useful.

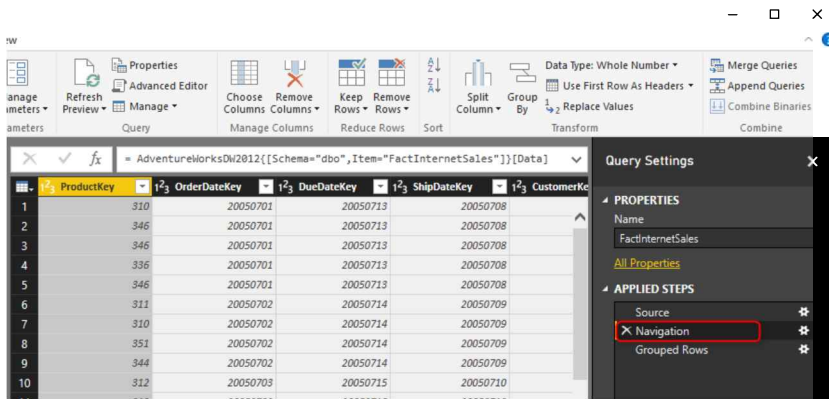
First and Last Item in each Group

Getting some default aggregation was as easy as selecting them in Group By the window. However, not all types of operations are listed there. For example, in the detailed table above, you can see that customer 11003 had nine sales transactions, and they happened on different Order dates. Getting the first and last order date is easy with Max and Min operations. However, getting the sales amount or product key associated with that record, or in other words, getting the first and last item in each group, isn’t possible through GUI. Fortunately, we can use M (Power Query formula language) to achieve this easily.

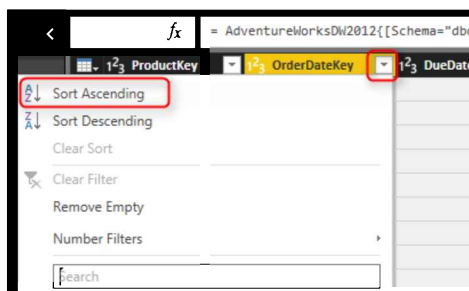
Sort By Date

I have to order the table based on that date column to get the first or last item in each group. Sorting is possible simply through GUI., and I have to apply that to the step before group by operation. So from the right-hand side applied steps list, I’ll select Navigation (which is the step before Grouped Rows);

Chapter 12: Grouping in Power Query; Getting The Last Item in Each Group



Now in this view, you can order simply by clicking on OrderDateKey and Choose Sort Ascending.



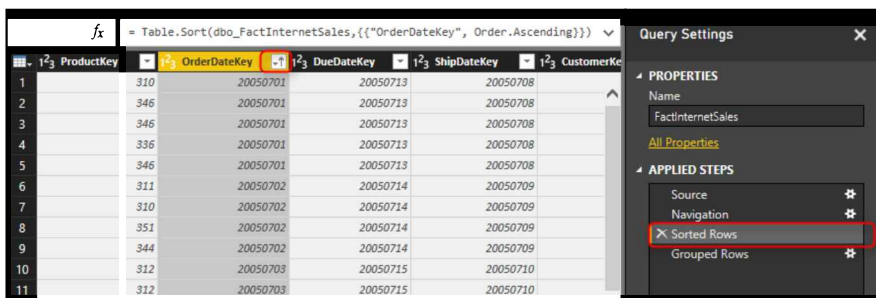
This will create another step and asks you do you want to INSERT this step here?

Insert Step

Are you sure you want to insert a step? Inserting an intermediate step may affect subsequent steps, which could cause your query to break.

Insert Cancel

Click on Insert to confirm you want to insert it here before the Grouped Rows. And then, you will see a Sorted Rows step added before Grouped Rows. This means Grouped Rows will use the output of the Sorted Rows step as the input for grouping (which is exactly what we want).



Part 2: Grouping and aggregation

Now you can go to the Grouped Rows step to see the result hasn't changed, but the sub tables are sorted now. All we need from here is to get the first and last item in the sub-table.

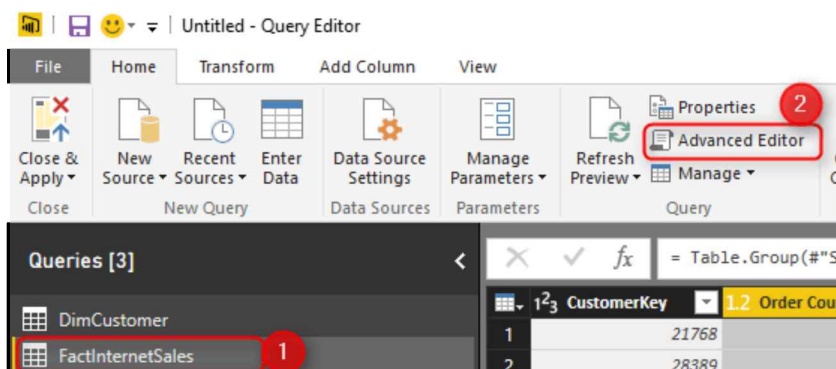
** If you want to sort based on multiple columns, simply go to the Advanced Editor and add as many sections you want to Table.Sort input parameters.

Because Power Query, by default, goes through query folding, it is recommended that you add a usage of Table after this step. Buffer, to make sure that the sorting runs before the grouping. That can be a step with this code: = Table.Buffer("#Sorted Rows")

List.First and List.Last

Fortunately, Power Query has a bunch of operations on List that we can use. List.First will return the first item in the list (based on the order defined in the list) and List.Last will return the last item. So let's use them in the Group By operation to fetch the first and last sales amount.

To make changes here, you need to go to script editor in Power Query, which can be achieved via the Advanced Editor option in the Home tab. You have to make sure that you are in the FactInternetSales Query first.



Advanced Editor will show you M script that build the output and the group by command as well.

let

```
Source = Sql.Databases("."),
```

```
AdventureWorksDW2012 = Source[{Name="AdventureWorksDW2012"}][Data],
```

Chapter 12: Grouping in Power Query; Getting The Last Item in Each Group

```
    dbo_FactInternetSales =  
AdventureWorksDW2012[{Schema="dbo",Item="FactInternetSales"}][Data],  
    #"Sorted Rows" = Table.Sort(dbo_FactInternetSales,{"OrderDateKey", Order.Ascending})),  
    Custom1 = Table.Buffer(#"Sorted Rows"),  
    #"Grouped Rows" = Table.Group(Custom1, {"CustomerKey"}, {"Order Count", each  
Table.RowCount(_), type number}, {"Total Revenue", each List.Sum([SalesAmount]), type  
number}, {"Order Details", each _, type table}})  
in  
    #"Grouped Rows"
```

The script in above code section created automatically when you did transformations through GUI. The line with Table.Group is the line that does all the grouping and aggregation. It is a long line, so let me format it better for easier understanding;

```
let  
    Source = Sql.Databases("."),  
    AdventureWorksDW2012 = Source[{Name="AdventureWorksDW2012"}][Data],  
    dbo_FactInternetSales =  
AdventureWorksDW2012[{Schema="dbo",Item="FactInternetSales"}][Data],  
    #"Sorted Rows" = Table.Sort(dbo_FactInternetSales,{"OrderDateKey", Order.Ascending})),  
    Custom1 = Table.Buffer(#"Sorted Rows"),  
    #"Grouped Rows" = Table.Group(Custom1,  
        {"CustomerKey"},  
        {  
            {"Order Count", each Table.RowCount(_), type number},  
            {"Total Revenue", each List.Sum([SalesAmount]), type number},  
            {"Order Details", each _, type table}  
        }  
    )  
in  
    #"Grouped Rows"
```

Script below is the same script. I just put some enters and tabs to format it better for reading. The above section shows the Table.Group section of the script. As you can see Table.Group gets a table as input, which is the #"Sorted Rows" table from the previous step. The group by field is "CustomerKey." and then a set of aggregated columns one after each other (which is highlighted in the code above). Each column has the name of

Part 2: Grouping and aggregation

the column, type of transformation (or aggregation), and the data type of the column. For example:

let

```
Source = Sql.Databases("."),
AdventureWorksDW2012 = Source{{Name="AdventureWorksDW2012"}}[Data],
dbo_FactInternetSales =
AdventureWorksDW2012{{Schema="dbo",Item="FactInternetSales"}}[Data],
#"Sorted Rows" = Table.Sort(dbo_FactInternetSales,{{"OrderDateKey", Order.Ascending}}),
Custom1 = Table.Buffer(#"Sorted Rows"),
#"Grouped Rows" = Table.Group(Custom1,
    {"CustomerKey"},
    {
        {"Order Count", each Table.RowCount(_), type number},
        {"Total Revenue", each List.Sum([SalesAmount]), type number},
        {"Order Details", each _, type table}
    }
)
```

in

```
#"Grouped Rows"
```

Total Revenue is the name of the column. The calculation for this column is Sum of [SalesAmount] which is one of the fields in the table, and the output is of type number.

So by now you should thinking of how each is to create a new aggregated column here; by adding similar column in the script. I add the new column after Order Details column, so I need an extra comma (,) after that line, and the new lines would be;

let

```
Source = Sql.Databases("."),
AdventureWorksDW2012 = Source{{Name="AdventureWorksDW2012"}}[Data],
dbo_FactInternetSales =
AdventureWorksDW2012{{Schema="dbo",Item="FactInternetSales"}}[Data],
#"Sorted Rows" = Table.Sort(dbo_FactInternetSales,{{"OrderDateKey", Order.Ascending}}),
Custom1 = Table.Buffer(#"Sorted Rows"),
#"Grouped Rows" = Table.Group(Custom1,
```

Chapter 12: Grouping in Power Query; Getting The Last Item in Each Group

```

        {"CustomerKey"},
    {
        {"Order Count", each Table.RowCount(_), type number},
        {"Total Revenue", each List.Sum([SalesAmount]), type number},
        {"Order Details", each _, type table},
        {"First SalesAmount", each List.First([SalesAmount]), type number},
        {"Last SalesAmount", each List.Last([SalesAmount]), type number}
    }
)

in

#"Grouped Rows"

```

Marked lines above use List.First and List.Last on the exact same structure that List.Sum worked. Because we have already sorted the table based on OrderDate, the first item would be the first sales transaction, and the last item would be the last.

Here is the output of this change:

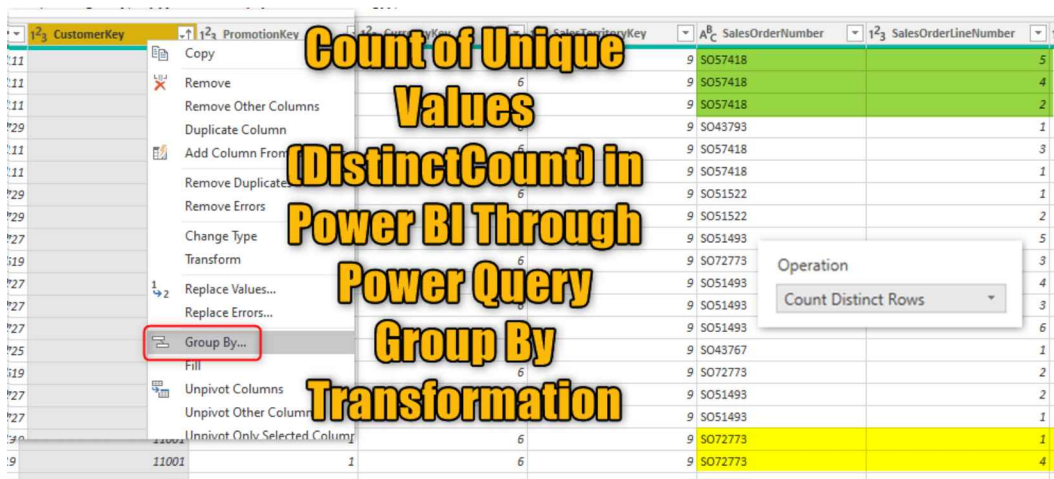
| | CustomerKey | Order Count | Total Revenue | Order Details | First SalesAmount | Last SalesAmount |
|----|-------------|-------------|---------------|---------------|-------------------|------------------|
| 1 | 21768 | 2 | 4118.26 | Table | 3578.27 | 539.99 |
| 2 | 28389 | 1 | 3399.99 | Table | 3399.99 | 3399.99 |
| 3 | 25863 | 5 | 4631.11 | Table | 3399.99 | 2.29 |
| 4 | 14501 | 2 | 2994.0882 | Table | 699.0982 | 2294.99 |
| 5 | 11003 | 9 | 8139.29 | Table | 3399.99 | 2.29 |
| 6 | 27645 | 5 | 6051.31 | Table | 3578.27 | 54.99 |
| 7 | 16624 | 4 | 5938.25 | Table | 3578.27 | 35 |
| 8 | 11005 | 6 | 8121.33 | Table | 3374.99 | 2384.07 |
| 9 | 11011 | 4 | 8133.04 | Table | 3399.99 | 53.99 |
| 10 | 27621 | 3 | 5971.33 | Table | 3578.27 | 8.99 |
| 11 | 27616 | 5 | 5998.61 | Table | 3578.27 | 2.29 |

| OrderDateKey | CustomerKey | PromotionKey | CurrencyKey | SalesTerritoryKey | SalesOrderNumber | SalesOrderLineNumber | RevisionNumber | OrderQuantity | UnitPrice | ExtendedPrice |
|--------------|-------------|--------------|-------------|-------------------|------------------|----------------------|----------------|---------------|-----------|---------------|
| 20050708 | 25863 | 1 | 100 | | 1 SO43699 | 1 | 1 | 1 | 3399.99 | |
| 20080203 | 25863 | 1 | 100 | | 1 SO62866 | | 1 | 1 | 1214.85 | |
| 20080203 | 25863 | 1 | 100 | | 1 SO62866 | | 2 | 1 | 4.99 | |
| 20080203 | 25863 | 1 | 100 | | 1 SO62866 | | 3 | 1 | 8.99 | |
| 20080203 | 25863 | 2 | 100 | | 1 SO62866 | | 4 | 1 | 2.29 | |

You can see that the first and the last SalesAmount picked correctly from each group as two new columns.

Note that by adding some changes in the script editor that are not supported through GUI, you will lose the GUI configuration window section. As a result of the change here, you cannot change the Grouping configuration in GUI anymore. If you want to change it, you have to go to Advanced Editor for this section. So if you are a GUI fan, better to apply all required configurations first and then add extra logic in the code.

Chapter 13: Count of Unique Values (DistinctCount) in Power BI Through Power Query Group By Transformation



You can have a distinct count calculation in multiple places in Power BI, through DAX code, using the Visual's aggregation on a field, or even in Power Query. If you are doing the distinct count in Power Query as part of a group by operation, however, the existing distinct count is for all columns in the table, not for a particular column. In this chapter, I'll show you a method you can use to get the distinct count of a particular column through the Group By transformation in the Power Query component of Power BI.

Why Power Query Transformation?

Before we start, it is important to know why you might consider using Power Query for a transformation such as Distinct Count in some scenarios. Having a distinct count in Power BI using DAX is great. However, sometimes this calculation can be done as a pre-calculation, and only the aggregated result is needed at the end. The details are not needed. If the dynamic calculation is not part of the requirement, then Power Query can be a good consideration for the implementation. To learn more about when to choose M (Power Query) or when to choose DAX for a calculation, read the chapter about it in later parts of this book.

What is Group By?

Group By is a transformation that groups the result based on one or more fields and provide an aggregated result from the existing table.

Part 2: Grouping and aggregation

| 1.2 CustomerKey | 1.2 Count |
|-----------------|-----------|
| 21768 | 2 |
| 28389 | 1 |
| 25863 | 5 |
| 14501 | 2 |
| 11003 | 9 |
| 27645 | 5 |
| 16624 | 4 |
| 11005 | 6 |
| 11011 | 4 |
| 27621 | 3 |
| 27616 | 5 |
| 20042 | 3 |
| 16351 | 2 |
| 16517 | 2 |

Besides the Group by field, you can also have aggregated results from the other parts of the table, which can be determined in the Group By Configuration window;

Group By

Specify the column to group by and the desired output.

☒ Basic ☐ Advanced

CustomerKey

New column name

Count

Operation

- Count Rows
- Sum
- Average
- Median
- Min
- Max
- Count Rows
- Count Distinct Rows
- All Rows

Column

OK Cancel

The Limitation of Distinct Count in Group By

The list of operations in the Group By aggregation, also useful, but it is limited. I previously wrote about a scenario that you can use to get the FIRST or LAST item in the group. The limitation that I want to point out here is the DistinctCount. The option that you see in the operations is Count Distinct Rows, which means for all columns except the Group By Column

Operation

Count Distinct Rows

This operation is most likely to return the same result as the Count Rows. It might be different if you have a duplicate row with all columns having the same value.

Chapter 13: Count of Unique Values (DistinctCount) in Power BI Through Power Query Group By Transformation

What is usually more useful is the distinct count of a specific column. For example, I am interested to know how many unique SalesOrderNumbers I have for each customer. and that cannot be achieved with a count of rows because there might be multiple records even per one SalesOrderNumber because the OrderLine information is also in the table;

You can see some examples of such a scenario in the below screenshot:

| | 123 CustomerKey | 123 PromotionKey | 123 CurrencyKey | 123 SalesTerritoryKey | A6 SalesOrderNumber | 123 SalesOrderLineNumber |
|----|-----------------|------------------|-----------------|-----------------------|---------------------|--------------------------|
| 11 | 11000 | 1 | 6 | 9 | SO57418 | 5 |
| 11 | 11000 | 1 | 6 | 9 | SO57418 | 4 |
| 11 | 11000 | 1 | 6 | 9 | SO57418 | 2 |
| 19 | 11000 | 1 | 6 | 9 | SO43793 | 1 |
| 11 | 11000 | 1 | 6 | 9 | SO57418 | 3 |
| 11 | 11000 | 1 | 6 | 9 | SO57418 | 1 |
| 19 | 11000 | 2 | 6 | 9 | SO51522 | 1 |
| 19 | 11000 | 1 | 6 | 9 | SO51522 | 2 |
| 17 | 11001 | 1 | 6 | 9 | SO51493 | 5 |
| 19 | 11001 | 1 | 6 | 9 | SO72773 | 3 |
| 17 | 11001 | 1 | 6 | 9 | SO51493 | 4 |
| 17 | 11001 | 1 | 6 | 9 | SO51493 | 3 |
| 17 | 11001 | 1 | 6 | 9 | SO51493 | 6 |
| 15 | 11001 | 1 | 6 | 9 | SO43767 | 1 |
| 19 | 11001 | 1 | 6 | 9 | SO72773 | 2 |
| 17 | 11001 | 1 | 6 | 9 | SO51493 | 2 |
| 17 | 11001 | 1 | 6 | 9 | SO51493 | 1 |
| 19 | 11001 | 1 | 6 | 9 | SO72773 | 1 |
| 19 | 11001 | 1 | 6 | 9 | SO72773 | 4 |

So the challenge is how to do Distinct Count for the SalesOrderNumber column when using Group By.

Adding Distinct Count to Group By

The Power Query function for a list of distinct values of a column is **List.Distinct**, which you can use as below:

List.Distinct(<column name>)

If you use the Count Distinct Rows in the group by;

Group By

Specify the column to group by and the desired output.

Basic

Advanced

CustomerKey

New column name

Distinct Count

Operation

Count Distinct Rows

Column

OK

Cancel

Part 2: Grouping and aggregation

It, however, uses the **Table.Distinct** function, which ends up with something like below:

`= Table.Group(#"Sorted Rows", {"CustomerKey"}, {"Distinct Count", each Table.RowCount(Table.Distinct(_)), type number}})`

| | CustomerKey | Distinct Count |
|---|-------------|----------------|
| 1 | 11000 | 8 |
| 2 | 11001 | 11 |
| 3 | 11002 | 4 |
| 4 | 11003 | 9 |
| 5 | 11004 | 6 |
| 6 | 11005 | 6 |
| 7 | 11006 | 5 |
| 8 | 11007 | 8 |

Table.Distinct is used inside the **Table.RowCount** function. The **(_)** input for the **Table.Distinct** is the input parameter from the group by action, which is the sub-table for each group.

you can replace that line of code with below;

```
= Table.Group(#'Sorted Rows', {'CustomerKey'}, {'Distinct Count', each  
Table.RowCount(List.Distinct(_[SalesOrderNumber])), type number}})
```

And that will give you the distinct count for the **SalesOrderNumber** columns.

this happens by replacing the:

`Table.Distinct(_)`

with

`List.Distinct(_[SalesOrderNumber])`

The **_** means the input table, and the **SalesOrderNumber** is the column that we want the unique values out of it, used inside the **List.Distinct**, the result is now the distinct count;

`= Table.Group(#"Sorted Rows", {"CustomerKey"}, {"Distinct Count", each Table.RowCount(List.Distinct(_[SalesOrderNumber])), type number}})`

| | CustomerKey | Distinct Count |
|---|-------------|----------------|
| 1 | 21768 | 2 |
| 2 | 28389 | 1 |
| 3 | 25863 | 2 |
| 4 | 14501 | 2 |
| 5 | 11003 | 3 |
| 6 | 27645 | 2 |
| 7 | 16624 | 2 |
| 8 | 11005 | 2 |

As I mentioned before in other chapters, once you know your way to writing M scripts (even part of the script), then Sky is the limit for the data transformation in Power BI.

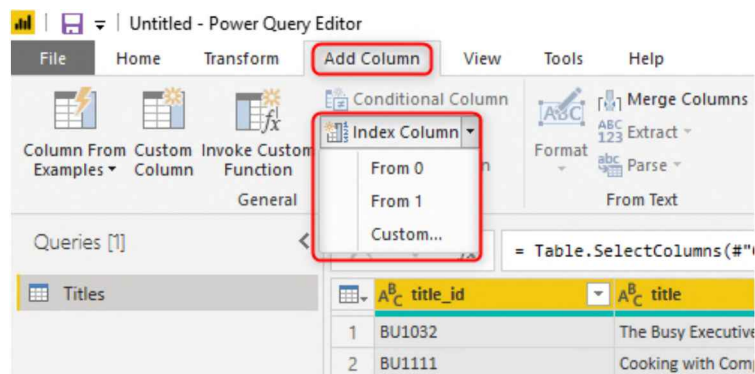
Chapter 14: Create Row Number for Each Group in Power BI using Power Query

| | A ^B _C type | A ^B _C title_id | A ^B _C title | 1 ² ₃ Index |
|---|----------------------------------|--------------------------------------|---|-----------------------------------|
| 1 | business | BU1032 | The Busy Executive's Database Guide | 1 |
| 2 | business | BU1111 | Cooking with Computers: Surreptitious Balance Sheets | 2 |
| 3 | business | BU2075 | You Can Combat Computer Stress! | 3 |
| 4 | business | BU7832 | Straight Talk About Computers | 4 |
| 5 | mod_cook | MC2222 | Silicon Valley Gastronomic Treats | 1 |
| 6 | mod_cook | MC3021 | The Gourmet Microwave | 2 |
| 7 | UNDECIDED | MC3026 | The Psychology of Computer Cooking | 1 |
| 8 | popular_comp | PC1035 | But Is It User Friendly? | 1 |
| 9 | popular_comp | PC8888 | Secrets of Silicon Valley | 2 |
| 0 | popular_comp | PC9999 | Net Etiquette | 3 |
| 1 | psychology | PS1372 | Computer Phobic AND Non-Phobic Individuals: Behavior Variations | 1 |
| 2 | psychology | PS2091 | Is Anger the Enemy? | 2 |
| 3 | psychology | PS2106 | Life Without Fear | 3 |
| 4 | psychology | PS3333 | Prolonged Data Deprivation: Four Case Studies | 4 |
| 5 | psychology | PS7777 | Emotional Security: A New Algorithm | 5 |
| 6 | trad_cook | TC3218 | Onions, Leeks, and Garlic: Cooking Secrets of the Mediterranean | 1 |
| 7 | trad_cook | TC4203 | Fifty Years in Buckingham Palace Kitchens | 2 |
| 8 | trad_cook | TC7777 | Sushi, Anyone? | 3 |

If you want to create a row-number statically as a pre-calculation in Power BI, then Power Query has a very simple way of doing it; Add Index Column. However, the Index column creates the row number regardless of any grouping or categorization. Sometimes you want to take that into account too. There is a little trick that you can add an index column with the grouping. Read the rest of this chapter to learn that method.

Add a Row Number using Add Index Column

In Power Query Editor, You can add an index column easily through the graphical interface of Power Query Editor.



There are a few options when you add the index column. The index can start from zero (default) or one, or you can select the custom starting point and the seed. Here is an example, the index started from one:

Part 2: Grouping and aggregation

| A ^B _C title_id | A ^B _C title | A ^B _C type | 1.2 Index |
|--------------------------------------|---|----------------------------------|-----------|
| BU1032 | The Busy Executive's Database Guide | business | 1 |
| BU1111 | Cooking with Computers: Surreptitious Balance Sheets | business | 2 |
| BU2075 | You Can Combat Computer Stress! | business | 3 |
| BU7832 | Straight Talk About Computers | business | 4 |
| MC2222 | Silicon Valley Gastronomic Treats | mod_cook | 5 |
| MC3021 | The Gourmet Microwave | mod_cook | 6 |
| MC3026 | The Psychology of Computer Cooking | UNDECIDED | 7 |
| PC1035 | But Is It User Friendly? | popular_comp | 8 |
| PC8888 | Secrets of Silicon Valley | popular_comp | 9 |
| PC9999 | Net Etiquette | popular_comp | 10 |
| PS1372 | Computer Phobic AND Non-Phobic Individuals: Behavior Variations | psychology | 11 |
| PS2091 | Is Anger the Enemy? | psychology | 12 |
| PS2106 | Life Without Fear | psychology | 13 |
| PS3333 | Prolonged Data Deprivation: Four Case Studies | psychology | 14 |
| PS7777 | Emotional Security: A New Algorithm | psychology | 15 |
| TC3218 | Onions, Leeks, and Garlic: Cooking Secrets of the Mediterranean | trad_cook | 16 |
| TC4203 | Fifty Years in Buckingham Palace Kitchens | trad_cook | 17 |
| TC7777 | Sushi, Anyone? | trad_cook | 18 |

If you are just after a simple row number, then the above gives you what you want. But if you are after a row number by group, you need a bit more steps to do.

Row Number By Group

If you want your index to restart at the beginning of every quarter, then you are after something like the below:

| A ^B _C type | A ^B _C title_id | A ^B _C title | 1.2 Index |
|----------------------------------|--------------------------------------|---|-----------|
| business | BU1032 | The Busy Executive's Database Guide | 1 |
| business | BU1111 | Cooking with Computers: Surreptitious Balance Sheets | 2 |
| business | BU2075 | You Can Combat Computer Stress! | 3 |
| business | BU7832 | Straight Talk About Computers | 4 |
| mod_cook | MC2222 | Silicon Valley Gastronomic Treats | 1 |
| mod_cook | MC3021 | The Gourmet Microwave | 2 |
| UNDECIDED | MC3026 | The Psychology of Computer Cooking | 1 |
| popular_comp | PC1035 | But Is It User Friendly? | 1 |
| popular_comp | PC8888 | Secrets of Silicon Valley | 2 |
| popular_comp | PC9999 | Net Etiquette | 3 |
| psychology | PS1372 | Computer Phobic AND Non-Phobic Individuals: Behavior Variations | 1 |
| psychology | PS2091 | Is Anger the Enemy? | 2 |
| psychology | PS2106 | Life Without Fear | 3 |
| psychology | PS3333 | Prolonged Data Deprivation: Four Case Studies | 4 |
| psychology | PS7777 | Emotional Security: A New Algorithm | 5 |
| trad_cook | TC3218 | Onions, Leeks, and Garlic: Cooking Secrets of the Mediterranean | 1 |
| trad_cook | TC4203 | Fifty Years in Buckingham Palace Kitchens | 2 |
| trad_cook | TC7777 | Sushi, Anyone? | 3 |

As you can see, the index is restarting at the beginning of every group. Let's see how you can do that easily with a few steps.

Before I explain the method to you: there are multiple ways of getting the row number based on a group. Some of them require writing Power Query scripts, and being fair, some of them are faster than the method I would explain here. But if you do not have so many data rows and you want an easy solution, I do recommend the below approach.

Step 1: Group the data

The first step is to Group the data, right-click on the field that you want to be your grouping field, and select Group By.

The screenshot shows the Tableau interface with a table named "Table.SelectColumns(#[\"Changed Type\"], {\"title_id\", \"title\", \"type\"})". The table has three columns: "title_id", "title", and "type". The "type" column is highlighted, and a context menu is open with the "Group By..." option selected. The table data is as follows:

| | title_id | title | type |
|----|----------|---|--------------|
| 1 | BU1032 | The Busy Executive's Database Guide | business |
| 2 | BU1111 | Cooking with Computers: Surreptitious Balance Sheets | business |
| 3 | BU2075 | You Can Combat Computer Stress! | business |
| 4 | BU7832 | Straight Talk About Computers | business |
| 5 | MC2222 | Silicon Valley Gastronomic Treats | mod_cook |
| 6 | MC3021 | The Gourmet Microwave | mod_cook |
| 7 | MC3026 | The Psychology of Computer Cooking | UNDECIDED |
| 8 | PC1035 | But Is It User Friendly? | popular_comp |
| 9 | PC8888 | Secrets of Silicon Valley | popular_comp |
| 10 | PC9999 | Net Etiquette | popular_comp |
| 11 | PS1372 | Computer Phobic AND Non-Phobic Individuals: Behavior Variations | psychology |
| 12 | PS2091 | Is Anger the Enemy? | psychology |
| 13 | PS2106 | Life Without Fear | psychology |
| 14 | PS3333 | Prolonged Data Deprivation: Four Case Studies | psychology |
| 15 | PS7777 | Emotional Security: A New Algorithm | psychology |
| 16 | TC3218 | Onions, Leeks, and Garlic: Cooking Secrets of the Mediterranean | trad_cook |
| 17 | TC4203 | Fifty Years in Buckingham Palace Kitchens | trad_cook |
| 18 | TC7777 | Sushi, Anyone? | trad_cook |

Then you will see the Group By the window; you can even group by multiple fields if you want with holding the Ctrl and selecting multiple fields, or using the Advanced tab here. For the new column to be created using the Group By, select the Operation to be All Rows. Column Name doesn't matter.

Group By

Specify the column to group by and the desired output.

☒ Basic ☐ Advanced

type

New column name

Count

Operation

All Rows

Sum

Average

Median

Min

Max

Count Rows

Count Distinct Rows

All Rows

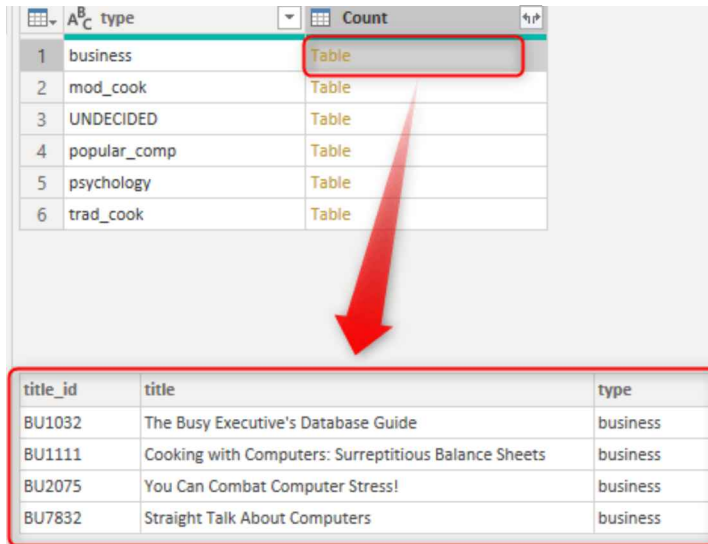
Column

OK

Cancel

Selecting All Rows means no aggregation from the group by; instead, it will populate a sub table from the original table for each group. If you click on a blank area of a cell that contains a Table in it, you can see the sub-table in the preview pane underneath.

Part 2: Grouping and aggregation

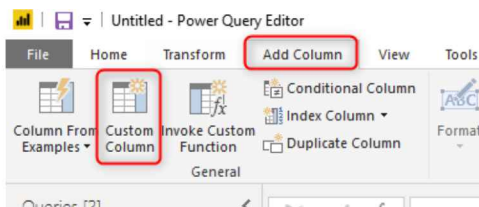


The screenshot shows a Power Query table with columns 'type' and 'Count'. The 'type' column has values: business, mod_cook, UNDECIDED, popular_comp, psychology, and trad_cook. The 'Count' column has the value 'Table' for each row. A red box highlights the 'Table' value in the first row, and a red arrow points down to a detailed sub-table below.

| title_id | title | type |
|----------|--|----------|
| BU1032 | The Busy Executive's Database Guide | business |
| BU1111 | Cooking with Computers: Surreptitious Balance Sheets | business |
| BU2075 | You Can Combat Computer Stress! | business |
| BU7832 | Straight Talk About Computers | business |

Step 2: Add Index to Sub-Table

Now that you have the sub-table of each group, you can add the index column to the sub-table. However, there is no graphical user interface for that at the moment. You can do it this way: Add a Custom Column:



You are then using the [Table.AddIndexColumn](https://docs.microsoft.com/en-us/powerquery-m/table-addindexcolumn) function in Power Query. You can add the index to the sub-table.

Chapter 14: Create Row Number for Each Group in Power BI using Power Query

Custom Column

Add a column that is computed from the other columns.

New column name

Custom

Custom column formula

= Table.AddIndexColumn([Count], "Index", 1)

Available columns

type

Count

<< Insert

Learn about Power BI Desktop formulas

✓ No syntax errors have been detected.

OK Cancel

`Table.AddIndexColumn([Count], "Index", 1)`

The [Count] in the expression above is the name of the column in my table that has sub-tables in it. If your column name is different, you should be selecting that from the list.

This newly created column now has sub-tables but with one extra column: Index

| 1 | business | Table | Table |
|---|--------------|-------|-------|
| 2 | mod_cook | Table | Table |
| 3 | UNDECIDED | Table | Table |
| 4 | popular_comp | Table | Table |
| 5 | psychology | Table | Table |
| 6 | trad_cook | Table | Table |

| title_id | title | type | Index |
|----------|--|----------|-------|
| BU1032 | The Busy Executive's Database Guide | business | 1 |
| BU1111 | Cooking with Computers: Surreptitious Balance Sheets | business | 2 |
| BU2075 | You Can Combat Computer Stress! | business | 3 |
| BU7832 | Straight Talk About Computers | business | 4 |

Step 3: Expand

Now you can remove all other columns and only keep the last column you have created.

| 1 | business | Table | Table |
|---|--------------|-------|-------|
| 2 | mod_cook | Table | Table |
| 3 | UNDECIDED | Table | Table |
| 4 | popular_comp | Table | Table |
| 5 | psychology | Table | Table |
| 6 | trad_cook | Table | Table |

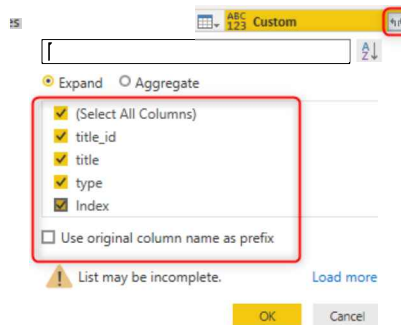
| title_id | title | type | Index |
|----------|--|----------|-------|
| BU1032 | The Busy Executive's Database Guide | business | 1 |
| BU1111 | Cooking with Computers: Surreptitious Balance Sheets | business | 2 |
| BU2075 | You Can Combat Computer Stress! | business | 3 |
| BU7832 | Straight Talk About Computers | business | 4 |

- Copy
- Remove
- Remove Other Columns
- Duplicate Column
- Add Column From Examples...
- Remove Errors

Part 2: Grouping and aggregation

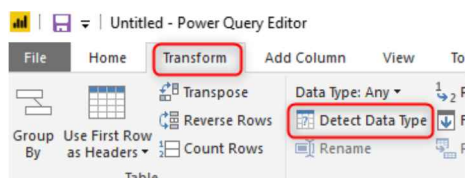
You don't need those columns because even the group column (type in the example above) is in the sub-table.

Then expand the column:



Optional: Change the Data Type

Usually, after expansion, columns get the ANY data type. You can select all columns (using Ctrl+A), and then under Transform, use Detect Data Types.



The Detect Data Type automatically sets the data type of all columns simply.

and here is the final results with the Index column added:

| title_id | title | type | Index |
|----------|---|--------------|-------|
| BU1032 | The Busy Executive's Database Guide | business | 1 |
| BU1111 | Cooking with Computers: Surreptitious Balance Sheets | business | 2 |
| BU2075 | You Can Combat Computer Stress! | business | 3 |
| BU7832 | Straight Talk About Computers | business | 4 |
| MC2222 | Silicon Valley Gastronomic Treats | mod_cook | 1 |
| MC3021 | The Gourmet Microwave | mod_cook | 2 |
| MC3026 | The Psychology of Computer Cooking | UNDECIDED | 1 |
| PC1035 | But Is It User Friendly? | popular_comp | 1 |
| PC8888 | Secrets of Silicon Valley | popular_comp | 2 |
| PC9999 | Net Etiquette | popular_comp | 3 |
| PS1372 | Computer Phobic AND Non-Phobic Individuals: Behavior Variations | psychology | 1 |
| PS2091 | Is Anger the Enemy? | psychology | 2 |
| PS2106 | Life Without Fear | psychology | 3 |
| PS3333 | Prolonged Data Deprivation: Four Case Studies | psychology | 4 |
| PS7777 | Emotional Security: A New Algorithm | psychology | 5 |
| TC3218 | Onions, Leeks, and Garlic: Cooking Secrets of the Mediterranean | trad_cook | 1 |
| TC4203 | Fifty Years in Buckingham Palace Kitchens | trad_cook | 2 |
| TC7777 | Sushi, Anyone? | trad_cook | 3 |

Part 3: Fuzzy operations

Chapter 15: Fuzzy Matching in Power BI and Power Query; Match based on Similarity Threshold

| 1 ² ID | A ^B _C Name | A ^B _C Department | A ^B _C Department Name |
|-------------------|----------------------------------|--|---|
| 1 | 1 Mike Anderson | Information Technology | Information Technology |
| 2 | 2 Antonio Martin | informatica tecnologica | Information Technology |
| 3 | | | Management |
| 4 | | | null |
| 5 | 5 Paolo Adrian | Sales | Sales |
| 6 | 6 Antony Page | Sale | Sales |
| 7 | 7 Brian Farmer | Managmnt | Management |

Have you ever wanted to match two tables together but not on exact matches, but also on a threshold of similarity? If your answer to this question is yes, then this feature is built for you. Let's explore in detail how fuzzy matching works in Power BI.

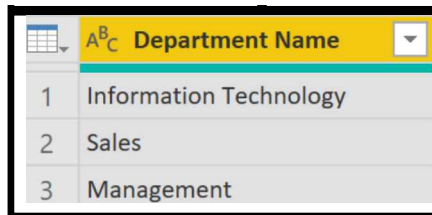
Sample Dataset

for this example; I will be using a sample dataset which has two very simple tables below;

A "source" table is the data of employees and their departments. Notice that the Department field has data quality issues. We have department values such as "Sales" and "Sale." Or another example is "Managmnt" and "Management."

| 1 ² ID | A ^B _C Name | A ^B _C Department |
|-------------------|----------------------------------|--|
| 1 | 1 Mike Anderson | Information Technology |
| 2 | 2 Antonio Martin | informatica tecnologica |
| 3 | 3 John Jefferson | Management |
| 4 | 4 Joe McCarthy | Mangmt |
| 5 | 5 Paolo Adrian | Sales |
| 6 | 6 Antony Page | Sale |
| 7 | 7 Brian Farmer | Managmnt |

A "Department" table which has a list of all departments;



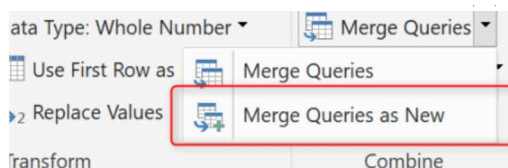
| | Department Name |
|---|------------------------|
| 1 | Information Technology |
| 2 | Sales |
| 3 | Management |

As you can see, the list of Department Names is clean in this table, and this is the table that should be used to clean the “source” table. Now let’s see how this is possible?

Fuzzy Merge

Fuzzy Merge is joining two tables together, not on exact matching criteria, but the similarity threshold. If you want to learn the Merge operation itself and the difference of that with Append, read the chapter related to that earlier in this book. If you want to learn more details about Merge and the different types of join or merge, read the chapter related to that earlier in this book. Merge or Join is simply the act of combining two tables with different structures, but with link/join columns, to access columns from one of the tables in the other one.

To use Merge operation on the “source” query, You can click on the Merge Queries as New option in the Home tab of the Power Query Editor window.



```
{"Name", type text}, {"Department", type text}}
```

Then you can select the second table and choose Department as the joining field

Part 3: Fuzzy operations

Merge ×

Select tables and matching columns to create a merged table.

source ▼

| ID | Name | Department |
|----|----------------|-------------------------|
| 1 | Mike Anderson | Information Technology |
| 2 | Antonio Martin | informatica tecnologica |
| 3 | John Jefferson | Management |
| 4 | Joe McCarthy | Mangmt |
| 5 | Paolo Adrian | Sales |

Department ▼

| Department Name |
|------------------------|
| Information Technology |
| Sales |
| Management |

Join Kind
Left Outer (all from first, matching from second) ▼

☐ Use fuzzy matching to compare the merge

> Fuzzy merge options

OK Cancel

This process will give you the output below: (result below is after expanding the merge's column output);

| | 1 ² 3 ID | A ^B C Name | A ^B C Department | A ^B C Department Name |
|---|---------------------|-----------------------|-----------------------------|----------------------------------|
| 1 | 1 | Mike Anderson | Information Technology | Information Technology |
| 2 | 2 | Antonio Martin | informatica tecnologica | null |
| 3 | 3 | John Jefferson | Management | Management |
| 4 | 4 | Joe McCarthy | Mangmt | null |
| 5 | 5 | Paolo Adrian | Sales | Sales |
| 6 | 6 | Antony Page | Sale | null |
| 7 | 7 | Brian Farmer | Managmnt | null |

You can see that the Merge operation only finds the EXACT Matching scenarios. Department "Sale" doesn't match with the Department table because it is missing an "S" at the end to match with the "Sales."

Now, let's see how the Fuzzy match works here. To use the Fuzzy Merge, just select the checkbox under the Merge tables dialog box;

Join Kind
Left Outer (all from first, matching from second)

☒ Use fuzzy matching to compare the merge

▲ Fuzzy merge options

Set threshold (optional)
0.9

☒ Ignore case

☒ Ignore spaces

Maximum number of matches (optional)

Transformation table (optional)

When you enable the fuzzy matching, you can configure it in the “fuzzy merge-operations.” you can leave everything optional. Or set values. Let’s first see the sample output of this operation and then see what the options are. This is the sample output of Fuzzy Merge:

| 1 ² 3 ID | A ^B C Name | A ^B C Department | A ^B C Department Name |
|---------------------|-----------------------|-----------------------------|----------------------------------|
| 1 | 1 Mike Anderson | Information Technology | Information Technology |
| 2 | 2 Antonio Martin | informatica technologica | Information Technology |
| 3 | 3 John Jefferson | Management | Management |
| 4 | 4 Joe McCarthy | Mangmt | null |
| 5 | 5 Paolo Adrian | Sales | Sales |
| 6 | 6 Antony Page | Sale | Sales |
| 7 | 7 Brian Farmer | Managmnt | Management |

You can see the three highlighted records, which were not recognized as the exact match in the normal merge operation, are not matching the output of the fuzzy merge. Fuzzy merge will check the similarity between joining fields, and if their similarity is more than the threshold configuration, it will pass it as a successful match. You can see that “Managmnt” can match with “Management” with this threshold configuration, but the “Mangmt” doesn’t. It shows that the threshold of similarity is higher than the similarity rate of these two text values with each other.

You can play with Options of Fuzzy Merge and get different outputs. Here is an explanation of these options:

Power Query Functions

| Option | Acceptable Value | Description |
|---------------------------|---|--|
| Threshold | a value between 0.00 to 1.00 | if the similarity of the two text values is more than the threshold, it will be considered as a successful match. Value 1.00 means exact match. |
| Ignore Case | true/false | If you want the similarity algorithm to work regardless of the upper or lower case letters, then select this option. |
| Ignore Space | true/false | If you want the similarity algorithm to work regardless of the number of spaces in the text, then select this option. |
| Maximum Number of Matches | a positive numeric value, between 0 to 2147483647 | The number of rows that can be matched to one value. |
| Transformation Table | table | This is like a mapping table. Let's check it out a bit later in this chapter. It gives you the option to use your own mapping table. This table should have at least two columns of "To" and "From." |

In addition to the option added in the graphical interface of Power Query, we also have two Power Query Functions that do the Fuzzy Merge. Functions are:

Table.FuzzyJoin

Table.FuzzyNestedJoin

Functions above both do have the same fuzzy configurations, and their only difference is that one of them gives you the expanded output (FuzzyJoin), the other one gives you the same output as the one that you see in the graphical interface with the table column output after merge (FuzzyNestedJoin). If you use these two functions directly in the M script, you will have a couple of more parameters to set, which are for concurrency and culture settings.

These are parameters of the two functions above;

table1

key1 (optional)

table2

key2 (optional)

newColumnName

Example: abc

joinKind (optional)

JoinOptions (optional)

ConcurrentRequests (optional)

Example: 123

Culture (optional)

Example: abc

IgnoreCase (optional)

Example: true

IgnoreSpace (optional)

Example: true

NumberOfMatches (optional)

Example: 123

Threshold (optional)

Example: 123

TransformationTable (optional)

Transformation Table

Sometimes in the merge operation, you need a mapping table. This table is called here a Transformation Table. Here is an example of a mapping table:

| | A ^B _C From | A ^B _C To |
|---|----------------------------------|--------------------------------|
| 1 | Information Technology | IT |
| 2 | Sales | Sales |
| 3 | Management | Board |

Note that this table should have at least the two-column of “To” and “From.” And don’t forget that Power Query is case-sensitive!

Now you can select this table in your Merge operation in the Fuzzy configuration as below;



Part 3: Fuzzy operations

Join Kind
Left Outer (all from first, matching from second) ▼

☒ Use fuzzy matching to compare the merge

▲ Fuzzy merge options

Set threshold (optional)
0.1

☒ Ignore case

☒ Ignore spaces

Maximum number of matches (optional)

Transformation table (optional)
mapping ▼

This process is like merging the “source” table, which is the first table in our Merge, with the “Department” table based on the “Department” and then “Department Name” column, then merging it with the “mapping” table, based on the “To” column and “Department Name.” The output will bring the “To” column of the mapping table. Here is the sample output:

| ID | Name | Department | Department Name |
|----|----------------|--------------------------|-----------------|
| 1 | Mike Anderson | Information Technology | IT |
| 2 | Antonio Martin | informatica technologica | null |
| 3 | John Jefferson | Management | Board |
| 4 | Joe McCarthy | Mangmt | null |
| 5 | Paolo Adrian | Sales | Sales |
| 6 | Antony Page | Sale | Sales |
| 7 | Brian Farmer | Managmt | null |

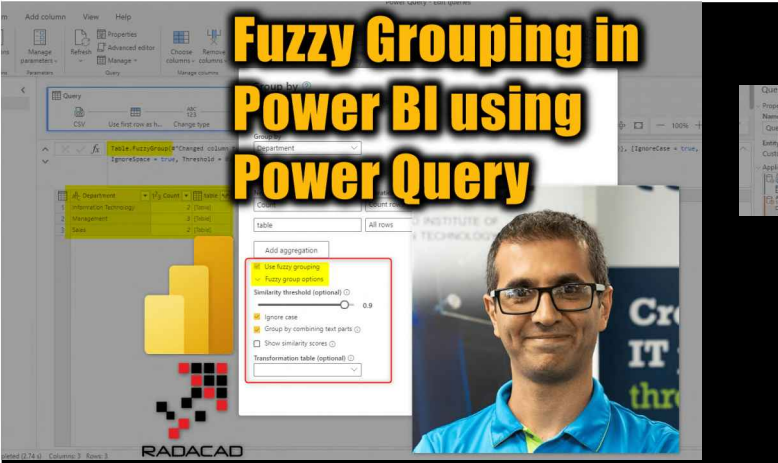
Transformation Table

| From | To |
|------------------------|-------|
| Information Technology | IT |
| Sales | Sales |
| Management | Board |

Summary

Matching based on similarity threshold or Fuzzy matching is a fantastic feature added to Power Query and Power BI. However, it is still a preview feature, and it may have some more configurations coming up. Please try it in your dataset, and let me know if you have any questions in the comment below.

Chapter 16: Fuzzy Grouping in Power BI using Power Query



I have explained previously what fuzzy matching in Power BI is and how to use it. In this chapter, I'll explain Fuzzy grouping. Fuzzy grouping, in short, means grouping text values by their similarity based on a threshold rather than exactly equal values. This option at the moment is available in Power Query online (dataflow), but it will be available soon in Power Query in Power BI Desktop or Excel too.

Sample input and requirement

Let's say we have a table like below;

"source" table, which is the data of employees and their departments. Notice that the Department field has data quality issues. We have department values such as "Sales" and "Sale." Or another example is "Managmnt" and "Management."

| ID | Name | Department |
|----|----------------|-------------------------|
| 1 | Mike Anderson | Information Technology |
| 2 | Antonio Martin | informatica tecnologica |
| 3 | John Jefferson | Management |
| 4 | Joe McCarthy | Mangmt |
| 5 | Paolo Adrian | Sales |
| 6 | Antony Page | Sale |
| 7 | Brian Farmer | Managmnt |

If the requirement is to group items based on the Department field. However, because every value in the Department column is different, we will have seven groups as a result;

Part 3: Fuzzy operations

| | Department | Count | |
|---|--------------------------|-------|-----------|
| 1 | Information Technology | | 1 [Table] |
| 2 | informatica technologica | | 1 [Table] |
| 3 | Management | | 1 [Table] |
| 4 | Mangmnt | | 1 [Table] |
| 5 | Sales | | 1 [Table] |
| 6 | Sale | | 1 [Table] |
| 7 | Managmnt | | 1 [Table] |

data grouped based on the exact matching

Our requirement is to have similar items grouped. We want an output like this:

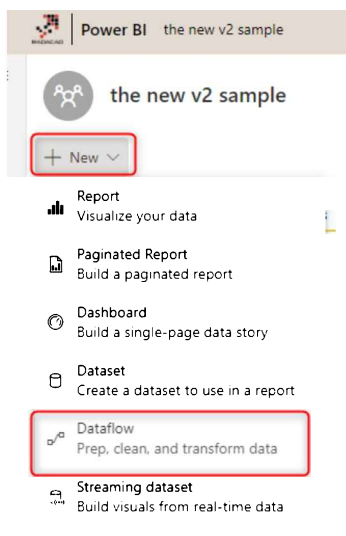
| | Department | Count | |
|---|------------------------|-------|-----------|
| 1 | Information Technology | 2 | 1 [Table] |
| 2 | Management | 3 | 1 [Table] |
| 3 | Sales | 2 | 1 [Table] |

desired outcome

We do not have a mapping table that tells us “Managmnt” means “Management.” This is something that can be done using Fuzzy Grouping.

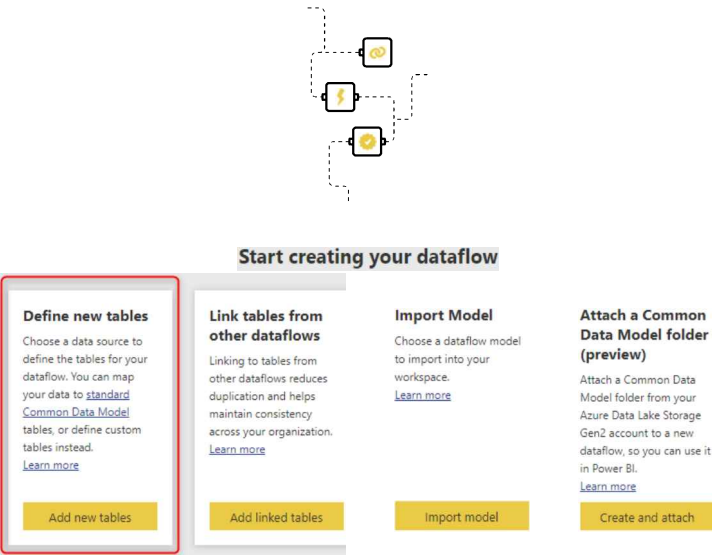
Power Query Online

At the time of writing this chapter, this option is only available in Power Query online. It surely will be available in other Power Query options on desktop such as Power BI Desktop and Excel in the future. But if you want to try it and online is the only option, it means you need to create a new dataflow inside an organizational workspace for it;



creating dataflow in Power BI Service

Once you created a dataflow, you can use the add new table

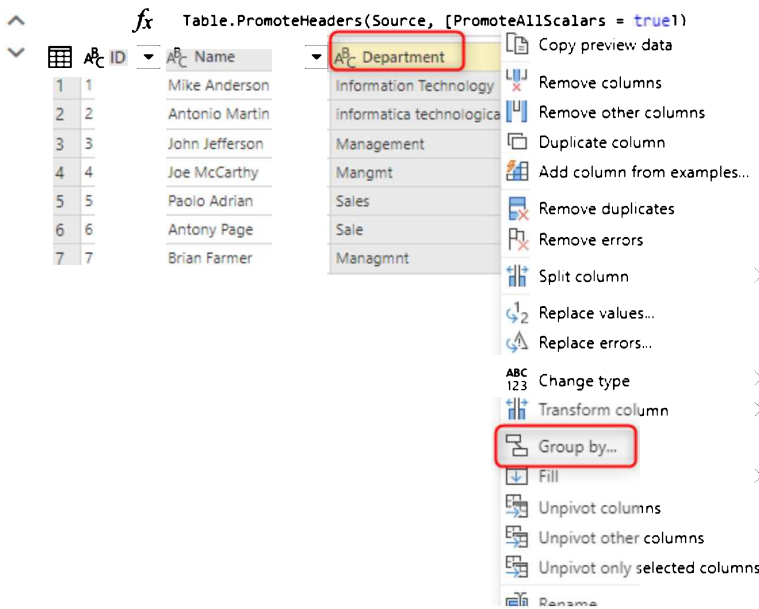


defining new tables inside Power BI dataflow

This would get you to the Get Data experience to connect to any data sources you want.

Fuzzy Grouping

Performing fuzzy grouping is very similar to the normal grouping in Power Query. You have to choose Group By on the column(s) you want.



Group by in Power Query

Part 3: Fuzzy operations

In the Group-by window, you will see the “Use Fuzzy Grouping” option.

Group by ?

Specify the column to group by and the desired output.

☒ Basic ☐ Advanced

Group by

Department

New column name

Count

Operation

Count rows

Column

☒ Use fuzzy grouping
> Fuzzy group options

OK

Cancel

Fuzzy Grouping in Power Query and Power BI

Enabling the fuzzy grouping will result in the grouping as below;

| 1 | Department | Count | Table |
|---|------------------------|-------|---------|
| 1 | Information Technology | 2 | [Table] |
| 2 | Management | 3 | [Table] |
| 3 | Sales | 2 | [Table] |

| ABC ID | ABC Name | ABC Department |
|--------|-----------------|----------------|
| 3 | John Jeffers... | Management |
| 4 | Joe McCarthy | Mangmt |
| 7 | Brian Farmer | Managmnt |

Fuzzy grouping's result in Power Query

As you can see in the above screenshot, the three values of “Management,” “Mangmt,” and “Managmnt” are all grouped into one.

Similarity Algorithm

This is possible because Fuzzy grouping uses an algorithm to find the similarity threshold of text values. The algorithm is based on **Jaccard Index**, which is explained [here](https://en.wikipedia.org/wiki/Jaccard_index).

Performance Aspect

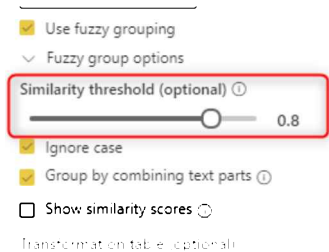
Be careful whenever you use Fuzzy Merge or Fuzzy Grouping. Any fuzzy operation on the dataset will bring a huge performance impact to the data processing. The reason is that every text value has to be compared with every other text value in the table, the

similarity threshold of the two has to be calculated (based on the algorithm above), and then if passes the similarity threshold merge into a group. This process is a very time-consuming process. Especially with more rows in the data table, you will feel it much more.

My suggestion is to first perform normal grouping on the items that match and then, for the non-matching items, perform the fuzzy operation.

Fuzzy Grouping Options

The default threshold for Fuzzy Grouping is 0.8, which means 80% similarity.



Similarity threshold for Fuzzy grouping in Power Query

You can change the options such as Ignore case or Similarity threshold. For example, if I change the similarity threshold to 1, It means 100% matching. This will result in seven groups.

^

v

```
fx Table.FuzzyGroup(#"Changed column type", {"Depi
IgnoreSpace = true, Threshold = 1})
```

| | Department | Count | table |
|---|--------------------------|-------|-----------|
| 1 | Information Technology | | 1 [Table] |
| 2 | Informatica technologica | | 1 [Table] |
| 3 | Management | | 1 [Table] |
| 4 | Mangmt | | 1 [Table] |
| 5 | Sales | | 1 [Table] |
| 6 | Sale | | 1 [Table] |
| 7 | Managmnt | | 1 [Table] |

The lower the similarity threshold, the more matching of non-similar values. For example, 0.92 similarity threshold will give me the below

Part 3: Fuzzy operations

```
fx Table.FuzzyGroup("#Changed column type", {"De  
IgnoreSpace = true, Threshold = 0.92})
```

| | Department | Count | |
|---|------------------------|-------|-----------|
| 1 | Information Technology | | 2 [Table] |
| 2 | Management | | 2 [Table] |
| 3 | Sales | | 2 [Table] |
| 4 | Mangmt | | 1 [Table] |

setting the similarity threshold for fuzzy grouping

If you want to set other settings for the Fuzzy Grouping, here is what they mean;

Power Query Functions

| Option | Acceptable Value | Description |
|-------------------------------|------------------------------|--|
| Threshold | a value between 0.00 to 1.00 | if the similarity of the two text values is more than the threshold, it will be considered a successful match. Value 1.00 means exact match. |
| Ignore Case | true/false | If you want the similarity algorithm to work regardless of the upper or lower case letters, then select this option. |
| Group by combining text parts | true/false | If you want the similarity algorithm to work regardless of the number of spaces in the text, then select this option. |
| Transformation Table | table | This is like a mapping table. Let’s check it out a bit later in this chapter. It gives you the option to use your mapping table. This table should have at least two columns of “To” and “From.” |

In addition to the option added in the graphical interface of Power Query, we also have a Power Query Function that does the Fuzzy Grouping:

Table.FuzzyGroup

If you use these two functions directly in the M script, you will have some parameters to set;

fx

Table.FuzzyGroup

Table.FuzzyGroup

Groups rows in the table based on fuzzy matching of keys.

Enter parameters

table *

null

key *

aggregatedColumns *

null

options

null

Invoke

Clear

function (table as table, key as any, aggregatedColumns as list, optional options as record) as table

Table.FuzzyGroup Function in M

Transformation Table

The transformation table works as a mapping table. Read more about it in the first chapter of this part of the book on Fuzzy matching.

| ID | Name | Department | Department Name |
|----|----------------|-------------------------|-----------------|
| 1 | Mike Anderson | Information Technology | IT |
| 2 | Antonio Martin | informatica tecnologica | null |
| 3 | John Jefferson | Management | Board |
| 4 | Joe McCarthy | Mangmt | null |
| 5 | Paolo Adrian | Sales | Sales |
| 6 | Antony Page | Sale | Sales |
| 7 | Brian Farmer | Managmnt | null |

Transformation Table

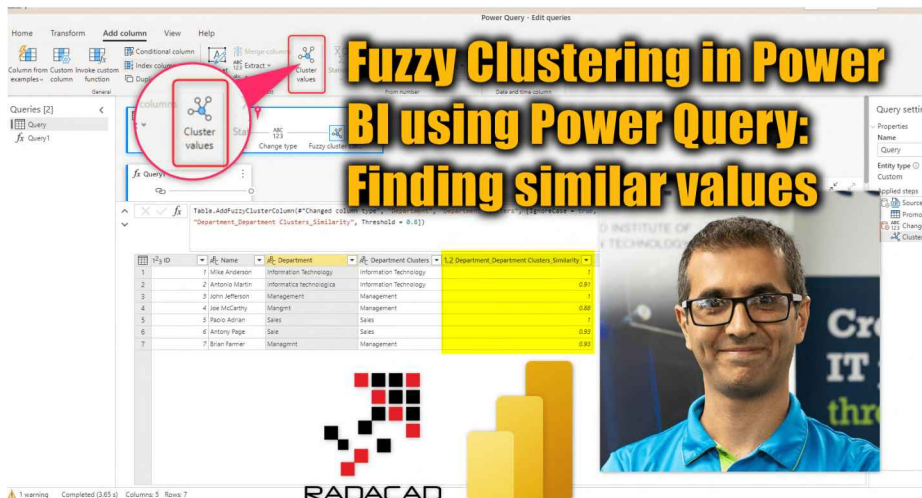
| From | To |
|------------------------|-------|
| Information Technology | IT |
| Sales | Sales |
| Management | Board |

Summary

Fuzzy Grouping is a very helpful option if you are dealing with dirty data. Fuzzy grouping is easy to apply. However, it comes with a performance cost. Ideally, you have to make sure the data comes clean before doing the transformation, which means applying the right process to get the clean data. However, sometimes when you need the transformation, you can use it. In this chapter, you learned about that.

147

Chapter 17: Fuzzy Clustering in Power BI using Power Query: Finding similar values



I have written two chapters about fuzzy operations in Power BI and Power Query; fuzzy matching in Power BI[<https://radacad.com/fuzzy-matching-in-power-bi-and-power-query-match-based-on-similarity-threshold>] and fuzzy grouping[<https://radacad.com/fuzzy-grouping-in-power-bi-using-power-query>]. The methods above involve transforming the data with a fuzzy operation. However, sometimes you just want to know the similarity of values or, let's say, in other words, find the clusters. Finding the clusters may help you in further transformation. This operation can be done using Fuzzy Clustering in Power Query inside Power BI. Let's see how it works.

Sample input and requirement

Let's say we have a table like below;

"source" table, which is the data of employees and their departments. Notice that the Department field has data quality issues. We have department values such as "Sales" and "Sale." Or another example is "Managmnt" and "Management."

| | 1 ² ₃ ID | A ^B _C Name | A ^B _C Department |
|---|--------------------------------|----------------------------------|--|
| 1 | 1 | Mike Anderson | Information Technology |
| 2 | 2 | Antonio Martin | informatica tecnologica |
| 3 | 3 | John Jefferson | Management |
| 4 | 4 | Joe McCarthy | Mangmt |
| 5 | 5 | Paolo Adrian | Sales |
| 6 | 6 | Antony Page | Sale |
| 7 | 7 | Brian Farmer | Managmnt |

We want to find out how similar values are to each other, or let’s say, how many clusters of similar values we have.

Power Query Online

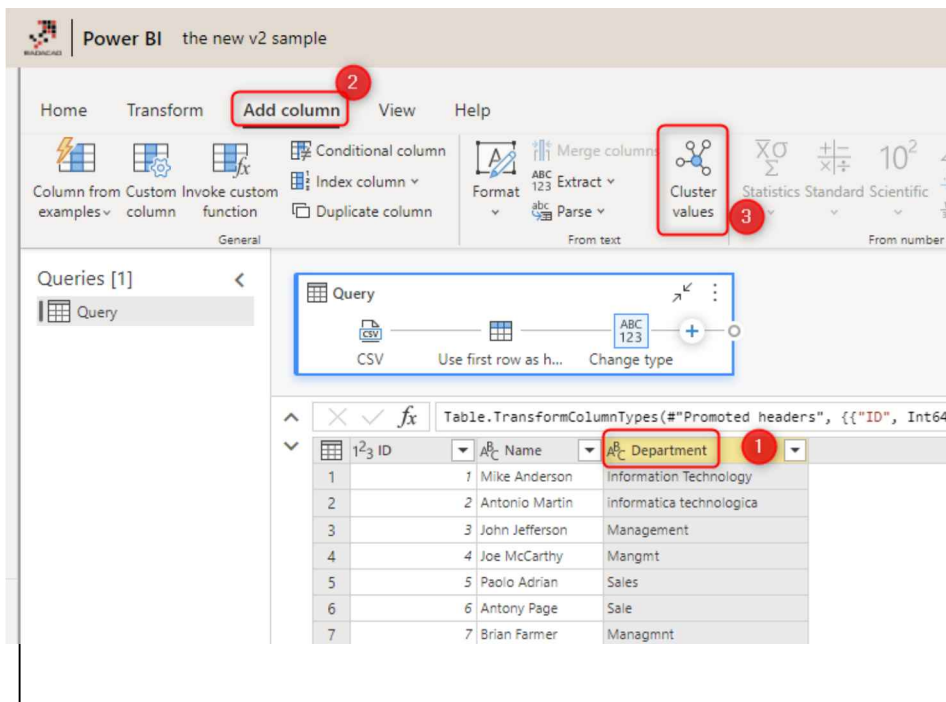
At the time of writing this chapter, this option is only available in Power Query online. It surely will be available in other Power Query options on desktop such as Power BI Desktop and Excel in the future. But if you want to try it and online is the only option, you need to create a new dataflow inside an organizational workspace for it, then create a new entity. Instruction for that is available in the previous chapter about Fuzzy Grouping.

Fuzzy Clustering

Performing fuzzy clustering is very similar to fuzzy grouping, with the difference that the output is not grouped.

To perform a fuzzy clustering, go to Add Columns in the Power Query Editor and select the Cluster values. Remember that you have to select the column on which you want to perform clustering beforehand.

Part 3: Fuzzy operations



Add a fuzzy cluster column in Power Query

You can simply then set a name for this new column and click on OK.

Cluster values ?

Specify the column to create the clusters from.

Column

Department

New column name

Department Clusters

Fuzzy cluster options

OK

Cancel

Fuzzy cluster column name

This simply gives you an output as below with a new column showing the cluster values;

123 ID

| | AB C Name | AB C Department | AB C Department Clusters |
|---|----------------|-------------------------|--------------------------|
| 1 | Mike Anderson | Information Technology | Information Technology |
| 2 | Antonio Martin | informatica tecnologica | Information Technology |
| 3 | John Jefferson | Management | Management |
| 4 | Joe McCarthy | Mangmt | Management |
| 5 | Paolo Adrian | Sales | Sales |
| 6 | Antony Page | Sale | Sales |
| 7 | Brian Farmer | Managmnt | Management |

cluster value column added to the table

I have color-coded the result above so that you can understand it simpler. There are three cluster values in the Department Clusters column; Information Technology, Management, and Sales. Power Query finds out that the values in the Department column are similar to these three main clusters.

Similarity Algorithm

This is possible because Fuzzy clustering uses an algorithm to find the similarity threshold of text values. The algorithm is based on **Jaccard Index**, which is explained [here\[https://en.wikipedia.org/wiki/Jaccard_index\]](https://en.wikipedia.org/wiki/Jaccard_index).

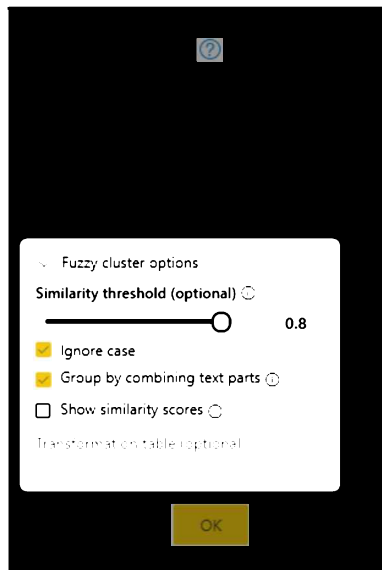
Performance Aspect

Be careful whenever you use Fuzzy Merge, Fuzzy Grouping, or Fuzzy Clustering. Any fuzzy operation on the dataset will bring a huge performance impact to the data processing. The reason is that every text value has to be compared with every other text value in the table. The similarity threshold of the two has to be calculated (based on the algorithm above). This process is a very time-consuming process. Especially with more rows in the data table, you will feel it much more.

Fuzzy cluster options

If you want to customize the fuzzy clustering's setting, you can expand the fuzzy options.

Part 3: Fuzzy operations



Fuzzy cluster options

Show Similarity Scores

This option is perhaps the most useful thing in fuzzy clustering. You like to know the cluster values, but most importantly, you like to know the similarity of each value to the cluster value. That is why the show similarity scores will do.



show similarity scores in fuzzy clustering

It will add a new column to your output with a similarity score between 0 to 1.

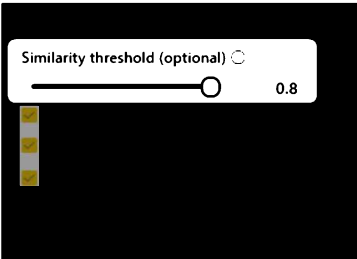
| 123 ID | AB Name | | AB Department | | AB Department Clusters | | 1,2 Department_Department Clusters_Similarity |
|--------|---------|----------------|-------------------------|------------------------|------------------------|--|---|
| | | | | | | | |
| 1 | 1 | Mike Anderson | Information Technology | Information Technology | Information Technology | | 1 |
| 2 | 2 | Antonio Martin | informatica tecnologica | Information Technology | Information Technology | | 0.91 |
| 3 | 3 | John Jefferson | Management | Management | Management | | 1 |
| 4 | 4 | Joe McCarthy | Mangmt | Management | Management | | 0.88 |
| 5 | 5 | Paolo Adrian | Sales | Sales | Sales | | 1 |
| 6 | 6 | Antony Page | Sale | Sales | Sales | | 0.93 |
| 7 | 7 | Brian Farmer | Managmnt | Management | Management | | 0.93 |

Similarity scores based on fuzzy clustering

The similarity scores are very helpful. It helps you to fine-tune the next similarity threshold based on your data values. I often use this option myself before applying fuzzy merge or fuzzy grouping.

Similarity threshold

The default threshold for Fuzzy clustering is 0.8, which means 80% similarity.



Similarity threshold for Fuzzy clustering in Power Query

You can change the options such as Ignore case or Similarity threshold. For example, if I change the similarity threshold to 1, It means 100% matching.

```
fx Table.AddFuzzyClusterColumn("#Changed column type", "Department", "Department Clusters", [IgnoreCase = true, IgnoreSpace = true, SimilarityColumnName = "Department_Department Clusters_Similarity", Threshold = 1])
```

| ID | Name | Department | Department Clusters | 1,2 Department_Department Clusters_Similarity |
|----|----------------|--------------------------|--------------------------|---|
| 1 | Mike Anderson | Information Technology | Information Technology | 1 |
| 2 | Antonio Martin | Informatica technologica | informatica technologica | 1 |
| 3 | John Jefferson | Management | Management | 1 |
| 4 | Joe McCarthy | Mangmt | Mangmt | 1 |
| 5 | Paolo Adrian | Sales | Sales | 1 |
| 6 | Antony Page | Sale | Sale | 1 |
| 7 | Brian Farmer | Managmnt | Managmnt | 1 |

The lower the similarity threshold, the more matching of non-similar values. For example, 0.92 similarity threshold will give me the below

```
fx Table.AddFuzzyClusterColumn("#Changed column type", "Department", "Department Clusters", [IgnoreCase = true, IgnoreSpace = true, SimilarityColumnName = "Department_Department Clusters_Similarity", Threshold = 0.92])
```

| ID | Name | Department | Department Clusters | 1,2 Department_Department Clusters_Similarity |
|----|----------------|--------------------------|------------------------|---|
| 1 | Mike Anderson | Information Technology | Information Technology | 1 |
| 2 | Antonio Martin | Informatica technologica | Information Technology | 0.92 |
| 3 | John Jefferson | Management | Management | 1 |
| 4 | Joe McCarthy | Mangmt | Mangmt | 1 |
| 5 | Paolo Adrian | Sales | Sales | 1 |
| 6 | Antony Page | Sale | Sales | 0.93 |
| 7 | Brian Farmer | Managmnt | Management | 0.93 |

setting the similarity threshold for fuzzy grouping

If you want to set other settings for the Fuzzy clustering, here is what they mean;

Power Query Functions

In addition to the option added in the graphical interface of Power Query, we also have a Power Query Function that does the Fuzzy clustering:

Table.AddFuzzyClusterColumn

If you use these two functions directly in the M script, you will have some parameters to set;



 Table.AddFuzzyClusterColumn


Table.AddFuzzyClusterColumn


Adds a new column with representative values obtained by fuzzy grouping values of the specified column in the table.


Enter parameters

 table *

null

 columnName *

 newColumnName *

 options

null

function (table as table, columnName as text, newColumnName as text, optional options as record) as table

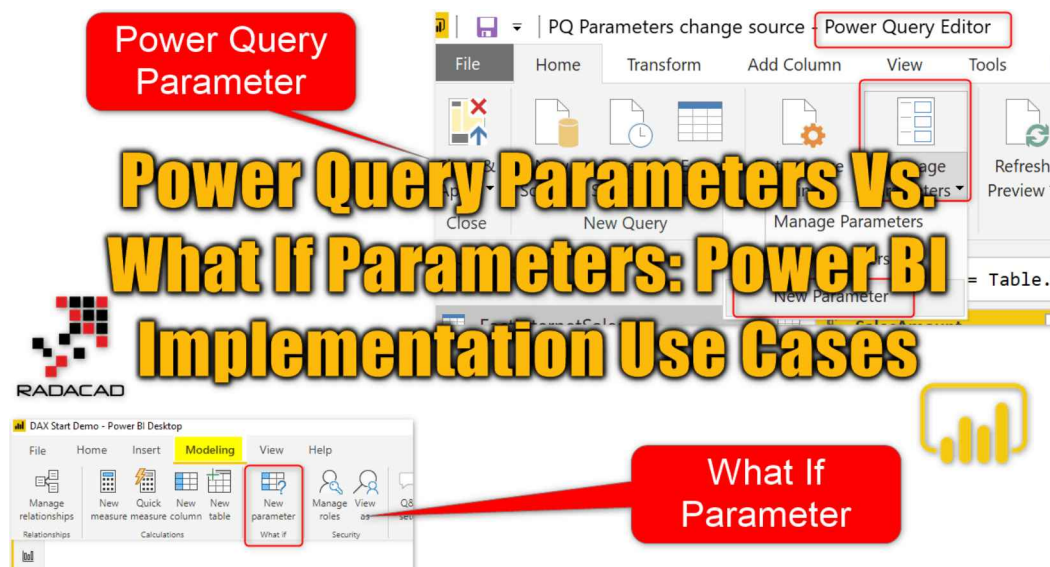
Table.AddFuzzyClusterColumn Function in M

Summary

Fuzzy Clustering is normally the step I do myself before fuzzy grouping or fuzzy matching. The showing similarity scores can help you to fine-tune the similarity threshold and then use that to perform further operations. Fuzzy clustering is easy to apply. However, it comes with a performance cost. Ideally, you have to make sure the data comes clean before doing the transformation, which means applying the right process to get the clean data. However, sometimes when you need the transformation, you can use it. In this chapter, you learned about that.

Part 4: Parameters and Custom functions

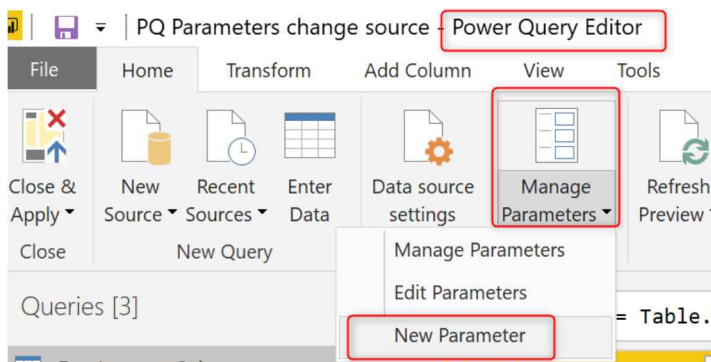
Chapter 18: Power Query Parameters Vs. What If Parameters: Power BI Implementation Use Cases



Many Power BI users are unaware that Power Query parameters and What-if parameters serve different purposes. In this chapter, I am explaining the differences between these two types of parameters and the use case scenarios for each.

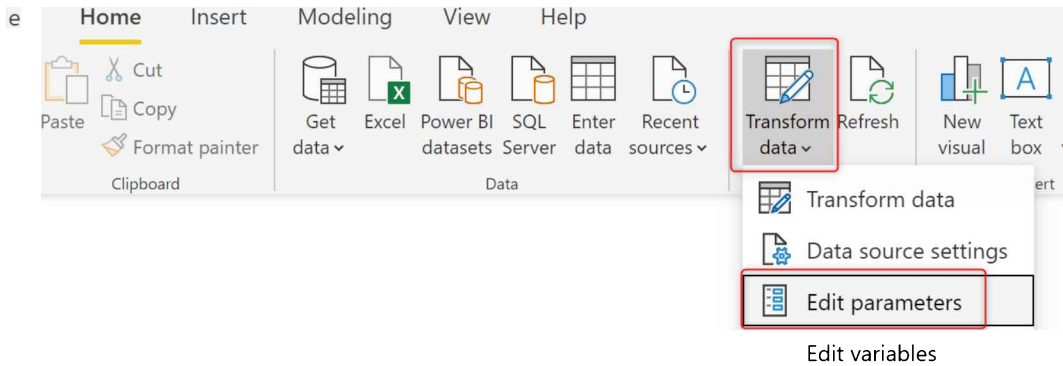
Power Query Parameters

Power Query parameters can be created through Power Query Editor



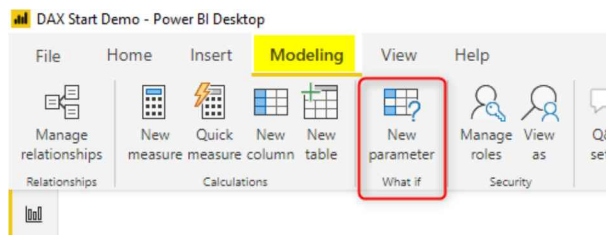
However, their value can be changed even through Power BI Desktop (or Power BI Service)

Chapter 18: Power Query Parameters Vs. What If Parameters: Power BI Implementation Use Cases



What If Parameters

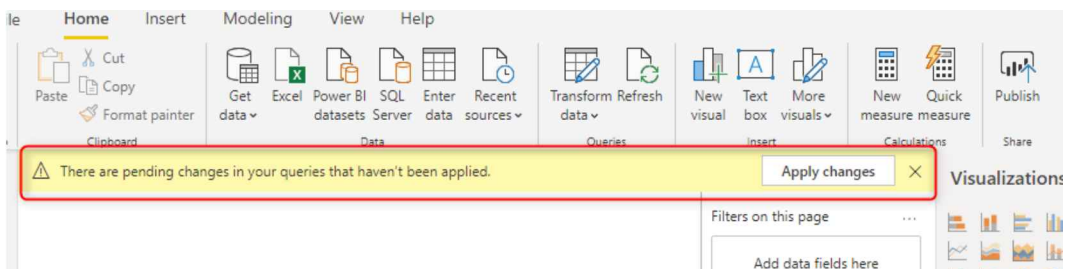
What If Parameters can be created only through Power BI Desktop, using the Modeling Tab



Values of What If parameters can be changed using the user selection of the slicer value.

Power Query Parameters Need Refresh Data

To see the result of a change in a Power Query parameter, you need to **Refresh** the data or **Apply Changes** (which runs the refresh data). Without refreshing the data, you won't change the parameter value applied to the dataset.



What If Parameters Are Dynamic. No Refresh Needed

What If parameters, on the other hand, are dynamic. The user doesn't need to refresh the data after making changes. After changing the value, the effect immediately applies.

Power Query Parameters are Suitable for Changes in the Data Transformation Layer

If you want to apply any changes dynamically in the data transformation layer, then Power Query parameters are the answer. There are example use cases of using parameters through this part of the book in the following chapters.

Here are a few scenarios that you can easily achieve using the Power Query parameters:

- Changing the database name, the Excel file path, or address of the data source
- Changing table names, Excel sheets, etc
- Changing a set of transformations using a value coming from another table
- or any other things in Power Query that you want to be dynamic.

What If parameters are for User Interaction

What If parameters, on the other hand, are useful for **user interaction**. You can easily create a parameter that changes a value in a DAX expression using What If parameters.

Here are a few scenarios of What if Parameters use cases:

- Change a DAX measure calculation based on the selection of the user.
- Calculate rolling 12 months sales, rolling six months, rolling two months, etc
- , changing the selected measure visualized in the visualization using a slicer
- or changing anything in DAX measure expression dynamically by the end-user.

Not Recommended Use Cases

It is recommended to use Power Query parameters to make the data transformation dynamic and the What-if parameter to make DAX expression dynamic. It is **not recommended** to do it the other way around. Here are some bad practices:

- You are using What If parameters to change the server address from Dev to Test or Prod. Because that means you need to load the entire data first into Power BI for all the environments, then apply what-if parameters on it. This is something you need to do with Power Query parameters.
- I am using What if parameters to change the source Excel file or folder. Exactly similar to the above example.
- Using Power Query Parameters for user interaction. Because after changing the value, the user needs to refresh the dataset, and often they don't have access to do this action.

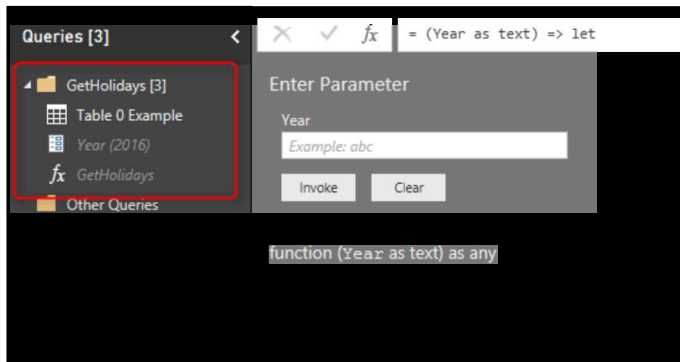
Two Different Types of Parameters, Two different Purposes

The bottom line is that there is no BETTER type of parameter here. I use these two types of parameters in every Power BI solution.

Power Query parameters are for applying changes in the data source or data transformation dynamically. What If parameters are for user interaction to change a calculation dynamically.

You cannot use one instead of the other. It is like saying you would build a house with only a roof and no walls, or only with walls and no roof. Neither is possible. You need both to build a complete dynamic solution.

Chapter 19: Custom Functions Made Easy in Power BI Desktop



With recent updates of the Power BI Desktop, creating custom functions is made easier and easier every month. In this chapter, you will learn how easy it is to create a custom function now, the benefits of doing it in this way, and the limitations of it.

What is Custom Function?

Custom Function, in simple definition, is a query that is run by other queries. The main benefit of having a query run by other queries is that you can repeat many steps on the same data structure. Let's see that as an example: The website below listed public holidays in New Zealand:

<http://publicholiday.co.nz/>

Every year, there is a page, and pages are similar to each other. Each page contains a table of dates and descriptions of holidays. here is, for example, public holidays of 2016:

<http://publicholiday.co.nz/nz-public-holidays-2016.html>



You can simply use Power BI Get Data from the Web menu option of Power Query to get the public holidays of this page. You can also make some changes in the date format to make it a proper Date data type. Then you probably want to apply the same steps on all other pages for other years (2015, 2017, 2018...), So instead of repeating the process, you can reuse an existing query. Here is where Custom Function comes to help.

Benefits of Custom Function

- Re-Use of Code
- Increasing Consistency
- Reducing Redundancy

With a Custom function, you are able to re-use a query multiple times. If you want to change part of it, there is only one place to make that change instead of multiple copies. You can call this function from everywhere in your code. And you are reducing redundant steps, which normally cause extra maintenance of the code.

How to Create Custom Function?

Well, that's the question I want to answer in this chapter. Previously (about a year ago), creating custom functions was only possible through M scripting with lambda expressions. That method still works, but you need to be comfortable with writing M

script to use that method (To be honest, I am still a fan of that method). Recently Power BI Desktop changed a lot. You can now create a function without writing any single line of code.

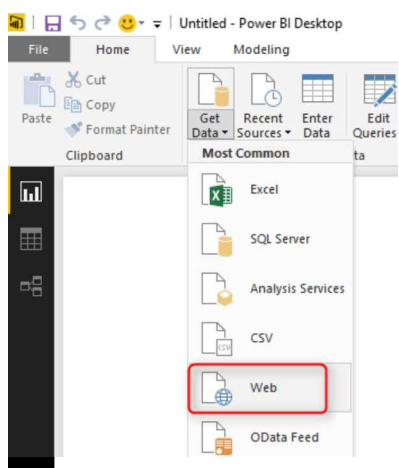
Through an example, I'll show you how to create a custom function. This example is fetching all public holidays from the website above and appending them all in a single table in Power Query. We want the process to be dynamic, so if a new year's data appear on that page, that will be included as well. Let's see how it works.

Building the Main Query

For creating a custom function, you always need to build the main query and then convert that to a function. Our main query is a query that will be repeated later on by calling from other queries. In this example, our main query is the query that processes the holidays' table on every page and returns that in proper format of Date data type and Text for the description of the holiday. We can do that for one of the years (it doesn't matter which one). I start with the year 2016, which has this URL:

<http://publicholiday.co.nz/nz-public-holidays-2016.html>[\[http://publicholiday.co.nz/nz-public-holidays-2016.html\]](http://publicholiday.co.nz/nz-public-holidays-2016.html)

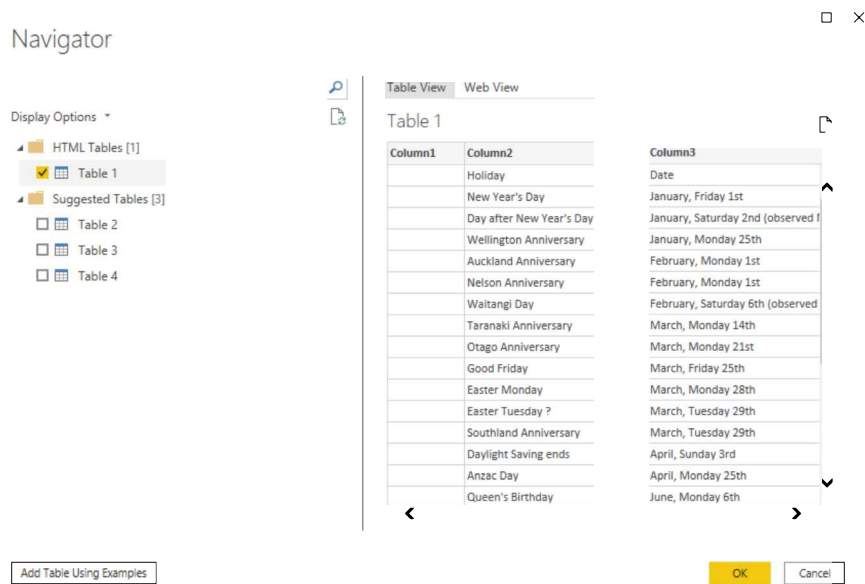
Open a Power BI Desktop and start by Get Data from Web



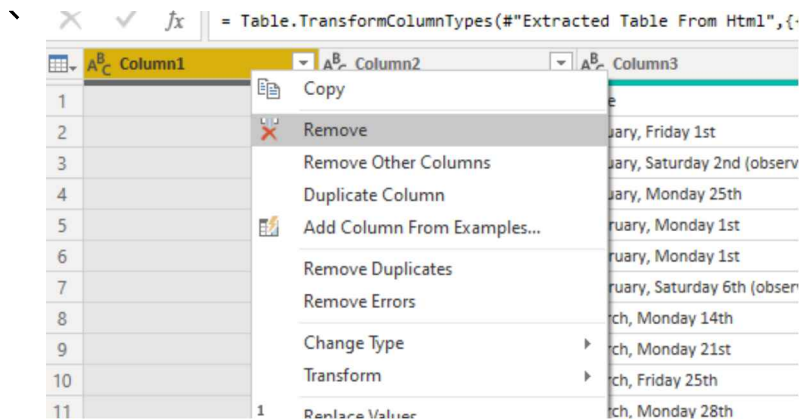
Use the 2016's web address in the From Web page;



In the Navigator, you can see that Table 1 is the table containing the data we are after. Select this table and click Edit.



This will open the Query Editor Window for you. You can now make some changes to the query. For example, remove the first column.



The first column was an empty column which we removed. Then you can Use the First Row as Headers (from the Transform tab). Now you can see two columns; Holiday (the description of the holiday) and Date.

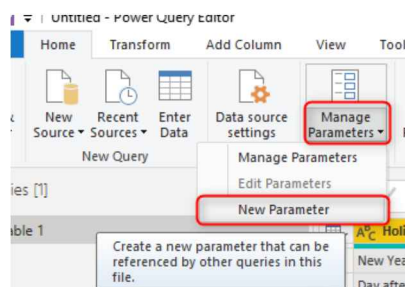
Part 4: Parameters and Custom functions

| | A ^B _C Holiday | A ^B _C Date |
|---|-------------------------------------|---|
| 1 | New Year's Day | January, Friday 1st |
| 2 | Day after New Year's Day | January, Saturday 2nd (observed Monday 4th) |
| 3 | Wellington Anniversary | January, Monday 25th |
| 4 | Auckland Anniversary | February, Monday 1st |
| 5 | Nelson Anniversary | February, Monday 1st |
| 5 | Waitangi Day | February, Saturday 6th (observed Monday 8th) |
| 7 | Taranaki Anniversary | March, Monday 14th |
| 8 | Otago Anniversary | March, Monday 21st |
| 9 | Good Friday | March, Friday 25th |
| 0 | Easter Monday | March, Monday 28th |
| 1 | Easter Tuesday ? | March, Tuesday 29th |
| 2 | Southland Anniversary | March, Tuesday 29th |
| 3 | Daylight Saving ends | April, Sunday 3rd |
| 4 | Anzac Day | April, Monday 25th |
| 5 | Queen's Birthday | June, Monday 6th |
| 6 | Daylight Saving starts | September, Sunday 25th |
| 7 | South Canterbury Anniversary | September, Monday 26th |
| 8 | Hawke's Bay Anniversary | October, Friday 21st |
| 9 | Labour Day | October, Monday 24th |
| 0 | Marlborough Anniversary | October, Monday 31st |
| 1 | Canterbury Anniversary | November, Friday 11th |
| 2 | Westland Anniversary | November, Monday 28th |
| 3 | Chatham Islands Anniversary | November, Monday 28th |
| 4 | Christmas Day | December, Sunday 25th (observed Tuesday 27th) |
| 5 | Boxing Day | December, Monday 26th |

The date column is the Text data type, and it doesn't have the year part in it. If you try to convert it to the Date data type, you will either get an error in each cell or an incorrect date as a result (depends on the locale setting of your computer). To convert this text to a date format, we need to bring a Year value in the query. The year value for this query can be statistically set to 2016. But because we want to make it dynamic so let's use a Parameter. This Parameter later will be used as input of the query.

Parameter Definition

Parameters are ways to pass values to other queries. Normally for custom functions, you need to use parameters. Click on the Manage Parameters menu option in Query Editor, and select New Parameter.



There are different types of parameters you can use, but to keep it simple, create a parameter of type Text, with all default selections. Set the Current Value to be 2016. and name it as Year.

Manage Parameters

New **×**

Name
Year

Description

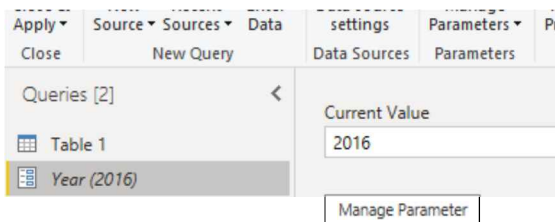
☒ **Required**

Type
Text

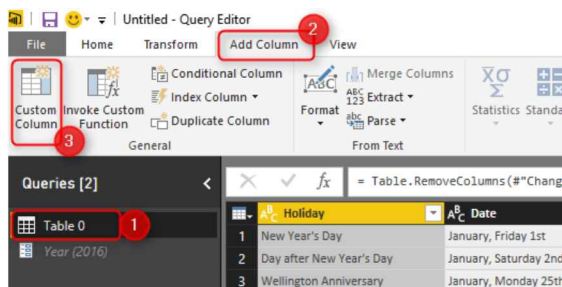
Suggested Values
Any value

Current Value
2016

After creating the Parameter, you can see that in the Queries pane with the specific icon for the parameter.



Now we can add a column in Table 1 with the value from this parameter. Click on Table 1, and from Add Column menu option, click on Add Custom Column.



Name the Custom column as Year, and write the expression to be equal Year (remember names in Power Query are case sensitive)

Part 4: Parameters and Custom functions

Add Custom Column

New column name

Year → Name of Column

Custom column formula:

= Year → Parameter Name

Now you can see the year value added to the table. I have also changed its data type to be Text

| | A ^B _C Holiday | A ^B _C Date | A ^B _C Year |
|----|-------------------------------------|--|----------------------------------|
| 1 | New Year's Day | January, Friday 1st | 2016 |
| 2 | Day after New Year's Day | January, Saturday 2nd (observed Monday 4th) | 2016 |
| 3 | Wellington Anniversary | January, Monday 25th | 2016 |
| 4 | Auckland Anniversary | February, Monday 1st | 2016 |
| 5 | Nelson Anniversary | February, Monday 1st | 2016 |
| 6 | Waitangi Day | February, Saturday 6th (observed Monday 8th) | 2016 |
| 7 | Taranaki Anniversary | March, Monday 14th | 2016 |
| 8 | Otago Anniversary | March, Monday 21st | 2016 |
| 9 | Good Friday | March, Friday 25th | 2016 |
| 10 | Easter Monday | March, Monday 28th | 2016 |
| 11 | Easter Tuesday ? | March, Tuesday 29th | 2016 |
| 12 | Southland Anniversary | March, Tuesday 29th | 2016 |
| 13 | Daylight Saving ends | April, Sunday 3rd | 2016 |
| 14 | ANZAC Day | April, Monday 25th | 2016 |
| 15 | Queen's Birthday | June, Monday 6th | 2016 |
| 16 | Daylight Saving starts | September, Sunday 25th | 2016 |
| 17 | South Canterbury Anniversary | September, Monday 26th | 2016 |
| 18 | Hawke's Bay Anniversary | October, Friday 21st | 2016 |
| 19 | Iskane Day | October, Monday 24th | 2016 |

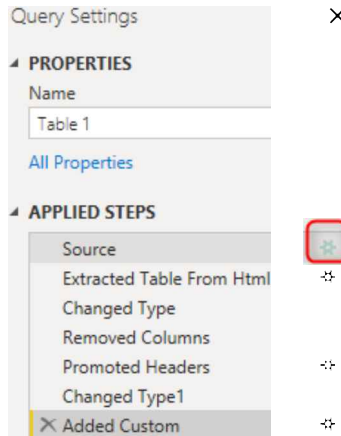
Now that we have created Parameter, we can also use that parameter as an input to the query's source URL.

URL Parameterization

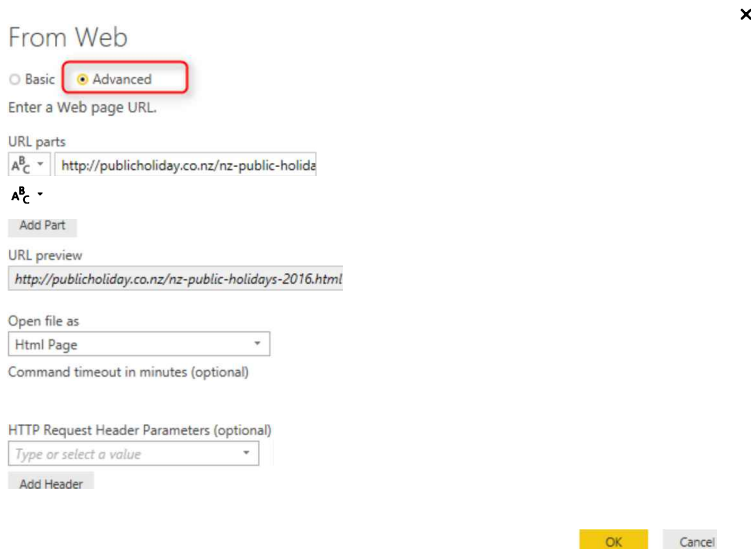
One of the main benefits of Parameters is that you can use that in a URL. In our case, the URL, which contains 2016 as the year, can be dynamic using this parameter. Also for converting a query to a custom function using parameters is one of the main steps. Let's add Parameter in the source of this query then;

While Table 1 query is selected, in the list of Steps, click on the Setting icon for Source step

Chapter 19: Custom Functions Made Easy in Power BI Desktop

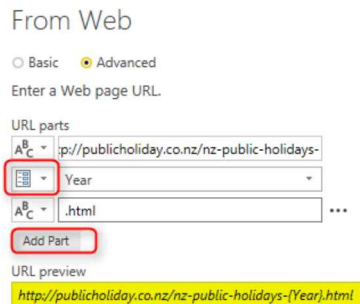


This will bring the very first step of the query where we get data from the web and provided the URL. Not in the top section, change the From Web window to Advanced.



The advanced option gives you the ability to split the URL into portions. What we want to do is to put Text portions for the beginning and end of the string and make the year part of it dynamic coming from the URL. So Add another part, and put the setting as below;

Part 4: Parameters and Custom functions



From Web

☐ Basic ☒ Advanced

Enter a Web page URL.

URL parts

A_C

☒

A_C

URL preview

`http://publicholiday.co.nz/nz-public-holidays-{Year}.html`

The configuration above means that in the first part of the URL, we put everything before 2016, which is:

`http://publicholiday.co.nz/nz-public-holidays-`

The second part of the URL is coming from parameter, and we use the Year parameter for that

the third part of the URL is the remaining part after the year, which is:

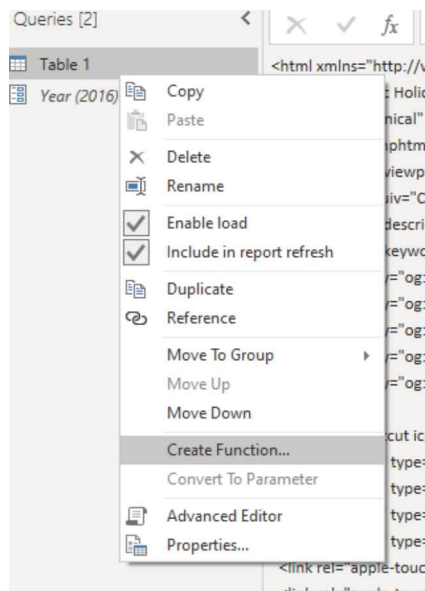
`.html`

Altogether these will make the URL highlighted above which instead of {Year} it will have 2016, or 2015 or other values.

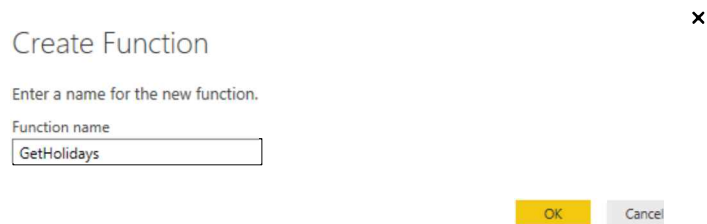
Click on OK. You won't see any changes yet, even if you click on the last step of this query, because we have used the same year code as the parameter value. If you change the parameter value and refresh the query, you will see changes, but we don't want to do it in this way.

Convert Query to Function

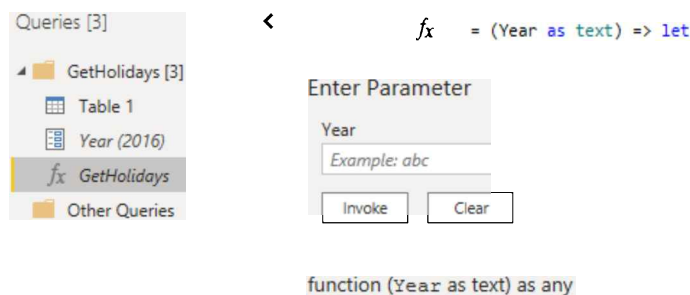
After using parameters in the source of the query, we can convert it to function—Right-click on the Table 0 query and select Create Function.



Name the function as GetHolidays and click on OK.



You will now see a group (folder) created with the name of GetHolidays, including three objects; main query (Table 1), Parameter (year), and function (GetHolidays).



The function itself is marked with the *fx* icon, and that is the function we will use to call from other queries. However, the main query and parameter are still necessary for making changes in the function. I will explain this part later.

Part 4: Parameters and Custom functions

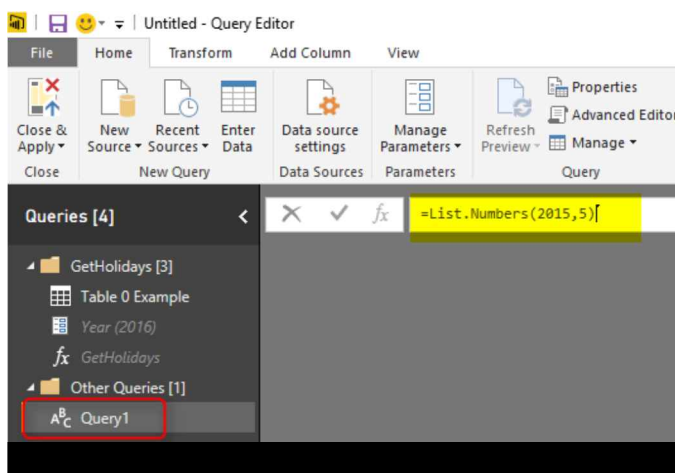
All that happened here is that there is a copy of the Table 0 query created as a function. And every time you call this function with an input parameter (which will be year value), this will give you the result (which is the public holidays' table for that year). Let's now consume this table from another query, but before that, let's create a query that includes a list of years.

Using Generator

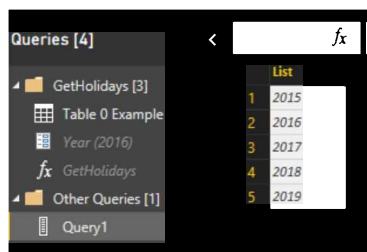
Generators are a topic of their own and can't be discussed in this chapter. All I can tell you for now is that Generators are functions that generate a list. For this example, I want to create a list of numbers from 2015 for five years. So I'll use the List—numbers generator function for that. In your Query Editor Window, create a New Source from the Home tab, and choose Blank Query.

This will create a Query1 for you. Click on Query1 in the Queries pane, and in the Formula bar type in the below script:

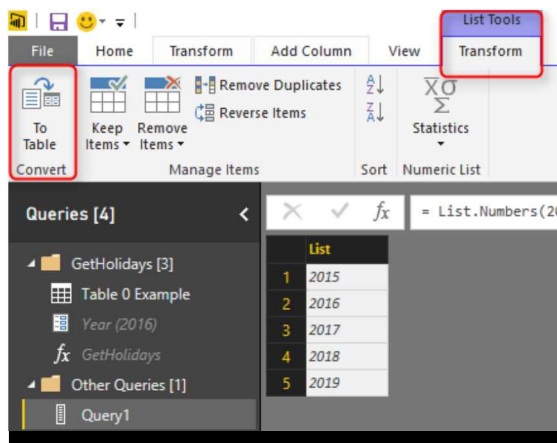
`=List.Numbers(2015,5)`



After entering the expression, press Enter, and you will see a list generated from 2015 for five numbers. That's the work done by a generator function.



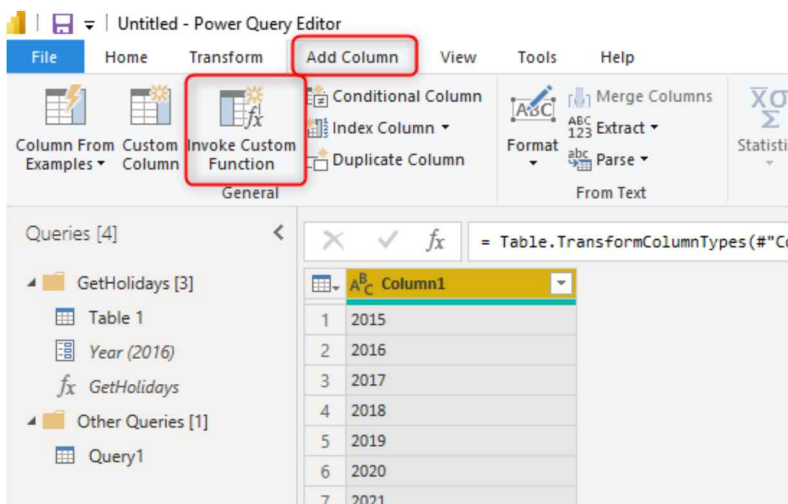
This is a List and can be converted to Table simple from the List Tools menu option.



Convert this to Table with all default settings, and now you can see a table with Column1, which is the year value. Because the value is a whole number, I have changed it to Text as well (to match the data type of parameter).

Consuming Function

It is easy to consume a function in Query Editor from a table. Go to Add Columns, and click on Invoke Custom Function option.



In the Invoke Custom Function window, choose the function (named GetHolidays), the input parameter is from the table column name Column1, and name the output column as Holidays.

Part 4: Parameters and Custom functions

×

Invoke Custom Function

Invoke a custom function defined in this file for each row.

New column name

Function query

Year

Now when you click on OK, you may see a warning about data privacy. Click on continue;

Function | Duplicate Column | Parse | Information | Ar

General | From Text | From Number | From Date & Time

= Table.AddColumn("#Changed Type", "GetHolidays", each GetHolidays([Column1]))

Information is required about data privacy.

This is because you combining multiple data sources, click on Ignore the privacy level, and continue.

×

Privacy levels

The privacy level is used to ensure data is combined without undesirable data transfer. Incorrect privacy levels may lead to sensitive data being leaked outside of a trusted scope. More information on privacy levels can be found [here](#).

☒ Ignore Privacy Levels checks for this file. Ignoring Privacy Levels could expose sensitive or confidential data to an unauthorized person.

You will see a new column added with a table in each cell. These tables are results of calling that function with the input parameter, the value of Column1 in each row. If you click on a blank area of a cell with a Table hyperlink, you will see the table structure below.

| | Column1 | Holidays |
|---|---------|----------|
| 1 | 2015 | Table |
| 2 | 2016 | Table |
| 3 | 2017 | Table |
| 4 | 2018 | Table |
| 5 | 2019 | Table |

| Holiday | Date | Year |
|------------------------------|--|------|
| New Year's Day | January, Sunday 1st (observed Tuesday 3rd) | 2017 |
| Day after New Year's Day | January, Monday 2nd | 2017 |
| Wellington Anniversary | January, Monday 23rd | 2017 |
| Auckland Anniversary | January, Monday 30th | 2017 |
| Nelson Anniversary | January, Monday 30th | 2017 |
| Waitangi Day | February, Monday 6th | 2017 |
| Taranaki Anniversary | March, Monday 13th | 2017 |
| Otago Anniversary | March, Monday 20th | 2017 |
| Daylight Saving ends | April, Sunday 2nd | 2017 |
| Good Friday | April, Friday 14th | 2017 |
| Easter Monday | April, Monday 17th | 2017 |
| Easter Tuesday ? | April, Tuesday 18th | 2017 |
| Southland Anniversary | April, Tuesday 18th | 2017 |
| ANZAC Day | April, Tuesday 25th | 2017 |
| Queen's Birthday | June, Monday 5th | 2017 |
| Daylight Saving starts | September, Sunday 24th | 2017 |
| South Canterbury Anniversary | September, Monday 25th | 2017 |
| Hawke's Bay Anniversary | October, Friday 20th | 2017 |
| Labour Day | October, Monday 23rd | 2017 |
| Marlborough Anniversary | October, Monday 30th | 2017 |

Interesting, isn't it? It was so easy. All done from GUI, not a single line of code to run this function or pass parameters, things made easy with custom functions.

Editing Function

If you want to make modifications in function, you can simply modify the main query (Table 1 Example). For example, let's create a full date format from that query in this way;

Click on Table 1 Example query and split the Date column with delimiter Comma, you will end up having a column now for Month values and another for Day (Note that I have renamed these columns respectively);

Part 4: Parameters and Custom functions

| | A ^B _C Holiday | A ^B _C Month | A ^B _C Day | A ^B _C Year |
|----|-------------------------------------|-----------------------------------|-------------------------------------|----------------------------------|
| 1 | New Year's Day | January | Friday 1st | 2016 |
| 2 | Day after New Year's Day | January | Saturday 2nd (observed Monday 4th) | 2016 |
| 3 | Wellington Anniversary | January | Monday 25th | 2016 |
| 4 | Auckland Anniversary | February | Monday 1st | 2016 |
| 5 | Nelson Anniversary | February | Monday 1st | 2016 |
| 6 | Waitangi Day | February | Saturday 6th (observed Monday 8th) | 2016 |
| 7 | Taranaki Anniversary | March | Monday 14th | 2016 |
| 8 | Otago Anniversary | March | Monday 21st | 2016 |
| 9 | Good Friday | March | Friday 25th | 2016 |
| 10 | Easter Monday | March | Monday 28th | 2016 |
| 11 | Easter Tuesday ? | March | Tuesday 29th | 2016 |
| 12 | Southland Anniversary | March | Tuesday 29th | 2016 |
| 13 | Daylight Saving ends | April | Sunday 3rd | 2016 |
| 14 | ANZAC Day | April | Monday 25th | 2016 |
| 15 | Queen's Birthday | June | Monday 6th | 2016 |
| 16 | Daylight Saving starts | September | Sunday 25th | 2016 |
| 17 | South Canterbury Anniversary | September | Monday 26th | 2016 |
| 18 | Hawke's Bay Anniversary | October | Friday 21st | 2016 |
| 19 | Labour Day | October | Monday 24th | 2016 |
| 20 | Marlborough Anniversary | October | Monday 31st | 2016 |
| 21 | Canterbury Anniversary | November | Friday 11th | 2016 |
| 22 | Westland Anniversary | November | Monday 28th | 2016 |
| 23 | Chatham Islands Anniversary | November | Monday 28th | 2016 |
| 24 | Christmas Day | December | Sunday 25th (observed Tuesday 27th) | 2016 |
| 25 | Boxing Day | December | Monday 26th | 2016 |

Now, if you go back to Query1, you will see changes in the results table immediately.

| ABC Column1 | | ABC 123 Holidays | |
|-------------|------|------------------|--|
| 1 | 2015 | Table | |
| 2 | 2016 | Table | |
| 3 | 2017 | Table | |
| 4 | 2018 | Table | |
| 5 | 2019 | Table | |

| Holiday | Month | Day | Year |
|------------------------------|-----------|-----------------------------------|------|
| New Year's Day | January | Sunday 1st (observed Tuesday 3rd) | 2017 |
| Day after New Year's Day | January | Monday 2nd | 2017 |
| Wellington Anniversary | January | Monday 23rd | 2017 |
| Auckland Anniversary | January | Monday 30th | 2017 |
| Nelson Anniversary | January | Monday 30th | 2017 |
| Waitangi Day | February | Monday 6th | 2017 |
| Taranaki Anniversary | March | Monday 13th | 2017 |
| Otago Anniversary | March | Monday 20th | 2017 |
| Daylight Saving ends | April | Sunday 2nd | 2017 |
| Good Friday | April | Friday 14th | 2017 |
| Easter Monday | April | Monday 17th | 2017 |
| Easter Tuesday ? | April | Tuesday 18th | 2017 |
| Southland Anniversary | April | Tuesday 18th | 2017 |
| ANZAC Day | April | Tuesday 25th | 2017 |
| Queen's Birthday | June | Monday 5th | 2017 |
| Daylight Saving starts | September | Sunday 24th | 2017 |
| South Canterbury Anniversary | September | Monday 25th | 2017 |
| Hawke's Bay Anniversary | October | Friday 20th | 2017 |
| Labour Day | October | Monday 23rd | 2017 |
| Marlborough Anniversary | October | Monday 30th | 2017 |

Limitations

Edit Script for the Function

Changes are fine as long as you don't want to edit the M script of the function. If you want to do so, then the function definition and the query definition split apart. And you will get the below message:

×

Edit Function

The definition of function 'GetHolidays' is updated whenever query 'Table 0 Example' is updated. However, updates will stop if you directly modify function 'GetHolidays'. Are you sure you want to continue?

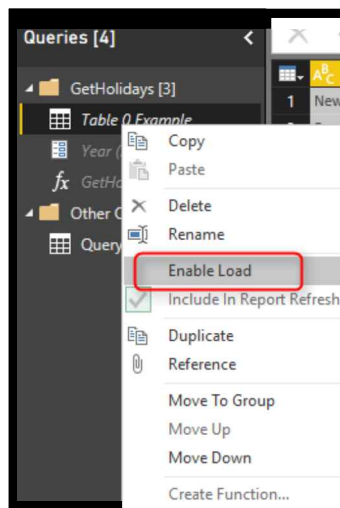
OK

Cancel

It is alright to make changes in the advanced Editor of the source query, and then the function will be updated based on that, but if you want to change the function itself, then the query will be separated.

Disable Load of the Source Query

If you have read the chapter related to the Enable load later in this book, you already know that queries that are not used in the model should not be loaded. By default, the source query (in this example named as Table 0 Example) will be loaded into the model. This means one extra table and consuming more memory. So remember to uncheck the Enable Load for this query;

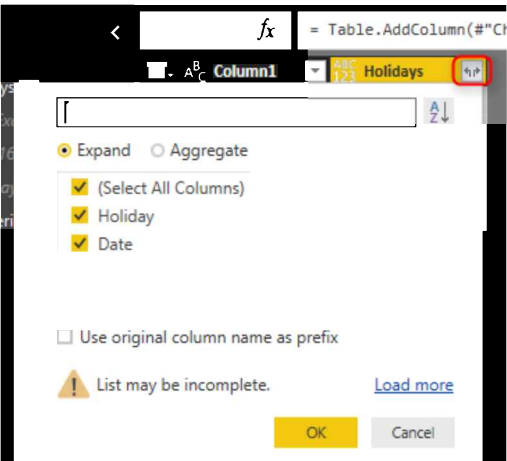


Parameterized URLs

Custom Functions that use parameterized URLs (like this example) cannot be scheduled to refresh in Power BI. That's a big limitation which I hope to be lifted very quickly.

The Example at the End

I have made some other changes and changed the data type to the Date format. Now the final query, which is an expanded table from all underlying queries, includes all public holidays. I'll leave that part to you to continue and build the rest of the example. For that functionality, you need to use Expand;



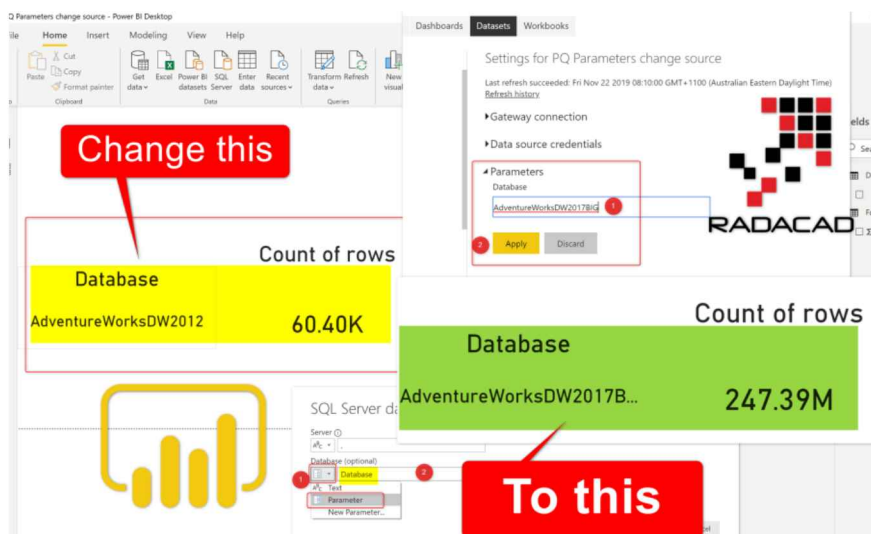
and here is the final result;

| | Holiday | Date |
|----|-----------------------------|------------|
| 47 | Westland Anniversary | 11/28/2016 |
| 48 | Chatham Islands Anniversary | 11/28/2016 |
| 49 | Christmas Day | 12/27/2016 |
| 50 | Boxing Day | 12/26/2016 |
| 51 | New Year's Day | 1/3/2017 |
| 52 | Day after New Year's Day | 1/2/2017 |
| 53 | Wellington Anniversary | 1/23/2017 |
| 54 | Auckland Anniversary | 1/30/2017 |
| 55 | Nelson Anniversary | 1/30/2017 |
| 56 | Waitangi Day | 2/6/2017 |
| 57 | Taranaki Anniversary | 3/13/2017 |
| 58 | Otago Anniversary | 3/20/2017 |
| 59 | Daylight Saving ends | 4/2/2017 |
| 60 | Good Friday | 4/14/2017 |
| 61 | Easter Monday | 4/17/2017 |
| 62 | Easter Tuesday ? | 4/18/2017 |

Summary

In this chapter, you have learned how easy it is to create a custom function from a query, all from the graphical interface. You have seen that I haven't written a single line of code to do it. And all happened through GUI. You have seen that you can change the definition of function simply by changing the source query. And you also have seen how easy it is to call/consume a function from another query. This method can be used a lot in real-world Power Query scenarios.

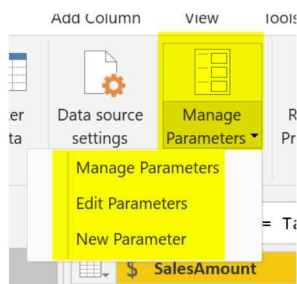
Chapter 20: Change the Source of Power BI Datasets Dynamically Using Power Query Parameters



Parameters in Power Query are useful for many scenarios when you want to do something dynamic in the data transformation process. I have previously explained how helpful they are in creating a custom function. In this chapter, I am showing you another useful way of using Parameters to create dynamic datasets, that you can change the source or anything else using it instead of opening your Power BI file each time and republish.

Parameters in Power Query

Parameters in Power Query are a useful way to change values dynamically in your Get Data and Transform process. Parameters can be used to change values without opening the Power Query (Transform Data) window in the Power BI Desktop, and they are helpful even in the Power BI Service in a way that you can change values manually without the need to open PBIX file in the Desktop and re-publish it.



Parameters in Power Query can be used in many different scenarios. One of the most common examples of using parameters is to use them for creating custom functions. Let's in this chapter focus on one of the challenges that can be easily resolved with parameters.

Changing Data Source Dynamically

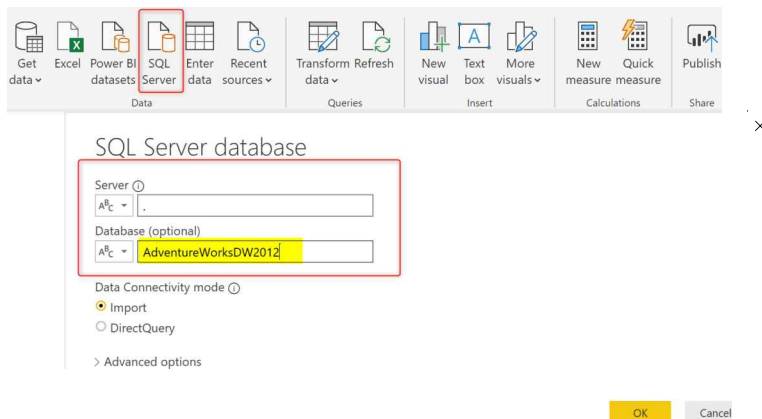
Imagine a scenario like this: You have connected to a data source using Power BI. That data source can be anything (a SQL Server or Oracle database, a folder, a file, a web API address, etc.). You created your Power BI report and then published the file to the service, and now you want to change the data source of the same type. However, the new data source is exactly similar to the old one in terms of structure. So all you need to do is just change part of the source connection (the database name, or the file name, or folder path, or the API URL, etc.). You can do these with changing values in Power Query Editor window. But that means you need to open the file in Power BI Desktop, change the value, save and re-publish it into the service. To avoid these extra steps, you can use Parameters. Let me walk you through it.

The process below is explained on a database source, but the same process can be used for ANY type of data source with slight modifications.

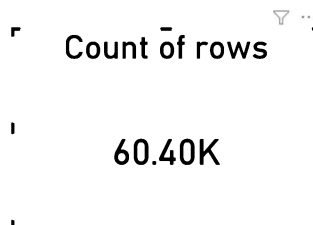
Getting Data from SQL Server Database

I have a Power BI report reading data from a SQL Server database called AdventureWorksDW2012.

Part 4: Parameters and Custom functions

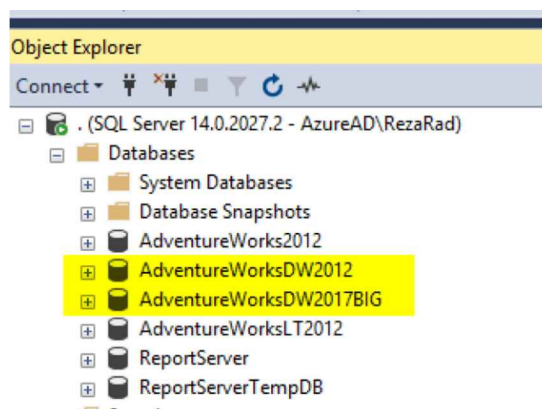


I have select FactInternetSales from this database and showed the count of rows from this table in a visual in the Power BI report:



Making the database name Parametric

I have a similar database to this one under the same server, but with a different name, which has a FactInternetSales table under it, but a much bigger version of it:

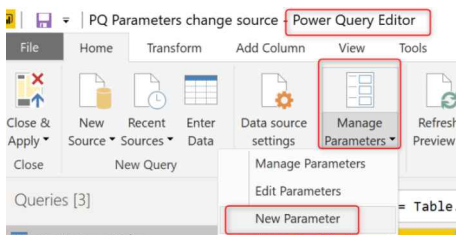


I build my report on the AdventureWorksDW2012, which is much smaller (or you might do it on your DEV database server to make the process of development faster). But after

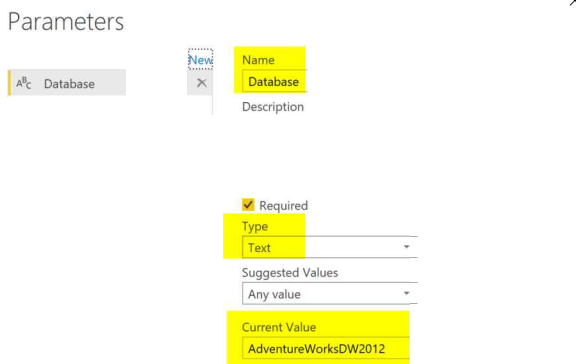
publishing it, I want the report to be plugged into the AdventureWorksDW2017BIG database and get data from there. So I will use parameters like below.

Create the parameter for the database name

The first step is to create the parameter under the Power Query Editor window (this parameter is different from What IF parameters in Power BI Desktop)

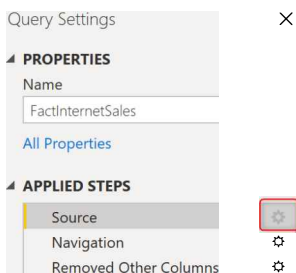


Then set the parameter name, data type, and the default value:



Using the parameter as the source database

Now you can go to the query that you want the source of that to be dynamically changing, click on the source step to set the database name.



You can then change the database to come from a parameter

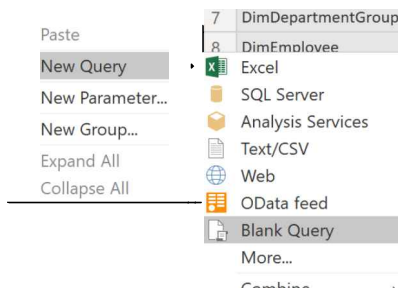
Part 4: Parameters and Custom functions



And then set the parameter you have created in the previous step there. That's it all done. Easy and fast.

Bonus Step: Showing the database name in the report

If you also like to show the database name on the report page, you can create a new blank query,

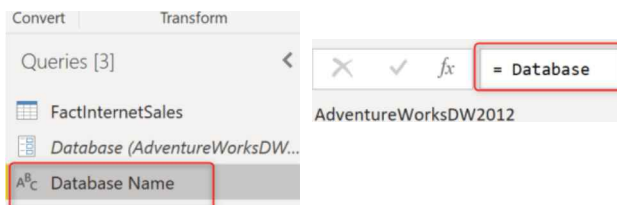


Write the expression below in the formula bar:

`=Database`

The “Database” in the above script is my parameter name. If your parameter name has space or other special characters in it, then you might try this way: `#” Database name.”` For more information, read the chapter about the basics of M scripting.

Now you have a new query with the value of this parameter in it.

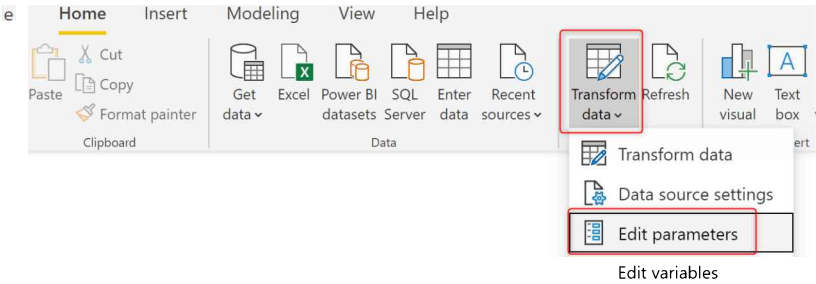


You can then use this value in a card visual in Power BI report like the below example:

| Database | Count of rows |
|----------------------|---------------|
| AdventureWorksDW2012 | 60.40K |

Changing the value of parameters from Power BI Desktop

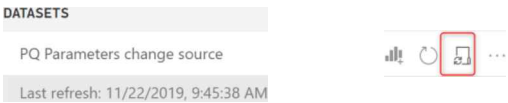
If you ever want to change the value of parameters in Power BI Desktop, you don't need to open Power Query Editor for that. You can change it easily by clicking on Transform Data and Edit Parameters



Please note that after changing the parameter value, you have to click on “Apply Changes” to get the dataset refreshed and see the changes applied.

Changing the value of parameters in the Power BI Service

After publishing your Power BI file to the service, you can go to the dataset settings (under schedule refresh)



You can then expand the Parameters section under the dataset setting page and change the value:

Part 4: Parameters and Custom functions

Dashboards **Datasets** Workbooks

Settings for PQ Parameters change source

Last refresh succeeded: Fri Nov 22 2019 08:10:00 GMT+1100 (Australian Eastern Daylight Time)
[Refresh history](#)

► Gateway connection

► Data source credentials

Parameters

Database

AdventureWorksDW2017BIG

Apply Discard

► Scheduled refresh

► Featured Q&A questions

► Endorsement

Extra Step: Sometimes, you might need to set up a Gateway connection too

If your data source is an on-premises data source, then you need to [set up the gateway configuration](https://radacad.com/the-power-bi-gateway-all-you-need-to-know) for that.

► Gateway connection

To use a data gateway, make sure the computer is online and the data source is added in [Manage Gateways](#). If you're using an On-premises data gateway (standard model), please select the corresponding data sources and then click apply.

Use a data gateway

| Gateway | Department | Contact information | Status | Actions |
|--------------|------------|----------------------|-------------------------|---------|
| Reza Surface | | Reza@RADACAD.onmi... | Running on REZA-SURFACE | |

Data sources included in this dataset:

SqlServer("server";"database";"adventureworksdw2017big")

Maps to:

- AdventureWorksDW2017
- AdventureWorksDW2017BIG
- Add to gateway

You have no personal gateways installed. [Learn more](#)

Install now

Testing the Result

Now you can either refresh the Power BI dataset in the service manually or schedule it to refresh. Either way, you will get your report connected to a different data source without editing your Power BI file!

| Database | Count of rows |
|--------------------------|---------------|
| AdventureWorksDW2017B... | 247.39M |

Summary

The method I showed here is useful in many scenarios. When you have the same data structure of the same data source type, but on a different server, different database, folder, file, API URL or etc. You can change that value easily with parameters without the need to edit your Power BI file.

This method is also used a lot for scenarios of changing connections from DEV, UAT, TEST, Prod environment without the need to make a lot of changes in the source code.

Part 5: Performance tuning

Chapter 21: Watch Your Steps! Power Query Performance Caution for Power BI



I work with Power Query transformations every day, and I want to share one simple but critical caution with you. If you use Power Query a lot, this tip can improve the performance of your transformation significantly. The number of steps that you add in a query counts in the performance of your data transformation (if you have too many steps). I like to show this to you through an example.

Too Many Variables

This is a sample Power Query file in which that I do a very simple transformation. The transformation is adding one to the existing number. However, in this sample, we are doing it over thousands of steps! one step at a time, we are adding thousands to a number. The main reason to do it this way is to show you the performance you get when you have too many variables (or, let's say, steps) in Power Query.

Here is my sample query:

let

a0= 0,

a1=a0+1,

a2=a1+1,

a3=a2+1,

... //

... // each variable used in the next variable with a plus one

Part 5: Performance tuning

... //

a1808=a1807+1,

a1809=a1808+1,

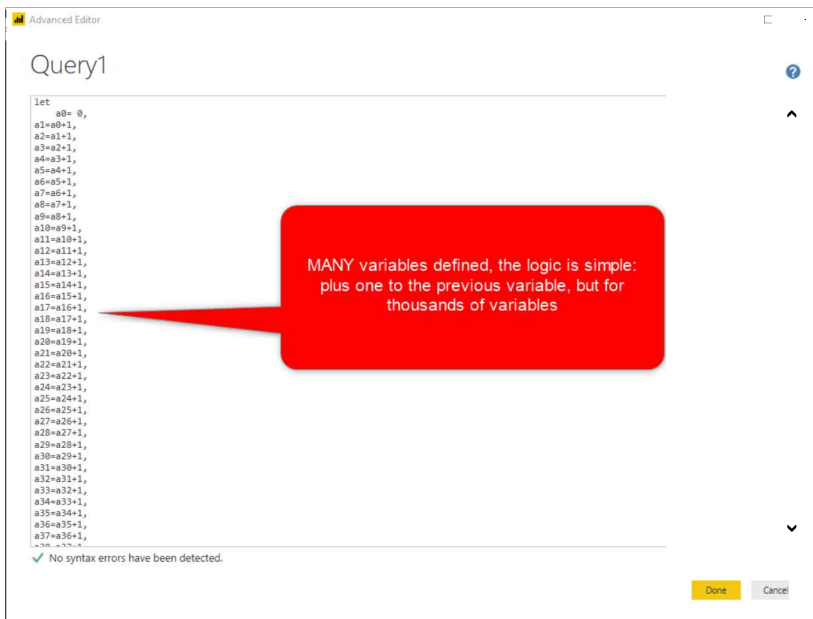
a1810=a1809+1,

a1811=a1810+1,

a1812=a1811+1

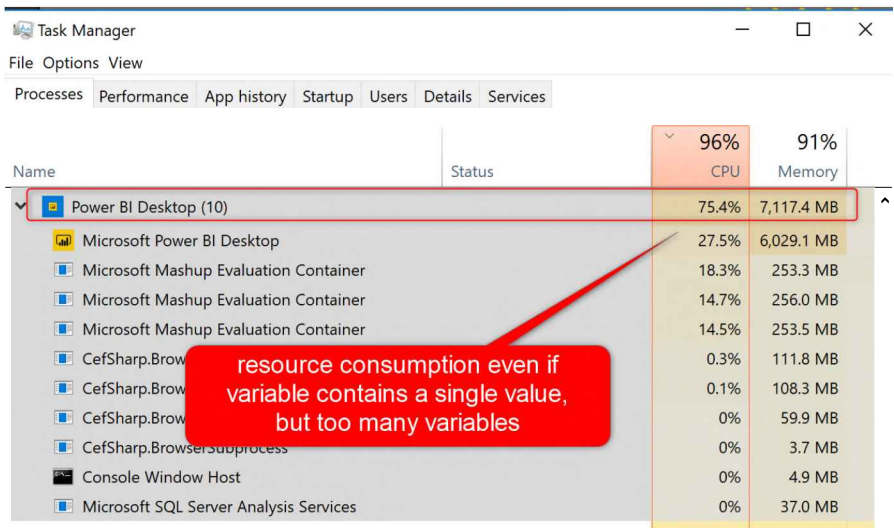
in

a1812



The query above takes 15 minutes to run on my Surface Book 2 machine with a Core i7 CPU and 16GB memory! The 15 minutes that you cannot touch Power BI through it. It will not respond to your actions, and you have to wait for that time to see the result after that.

Oh! That is a long time for a query that just adds one value in every step, Isn't it? Let's see the resource consumption in the system with Power BI Desktop and Power Query running. Here it is only halfway through:



Uh! Even with a simple calculation like adding one to a number, I had over 10GB memory consumption and 70% CPU usage for a long period! What do you think caused it? The number of variables, of course. There is nothing else in this query.

Having too many variables or too many steps cause performance issues. Power Query allocates memory for every variable, and memory consumption raises significantly. The processor also takes lots of time to process that number of variables through the Power Query Engine.

Of course, this example was an exaggerated example of too many variables. You would never have 1,800 variables. However, this example also was on a variable with a single simple numeric value. In most cases, your variables are tables with many data rows and columns. So this can happen in a real-world scenario for you even with hundreds of steps or variables. You need to watch the number of variables, in other words.

Too Many Variables will cause performance issues!

What is the Solution?

If the number of variables is the cause of the issue, the remedy would be reducing the number of variables in the first place. If I change the query above to this one below, I get the result in less than a second!

Part 5: Performance tuning

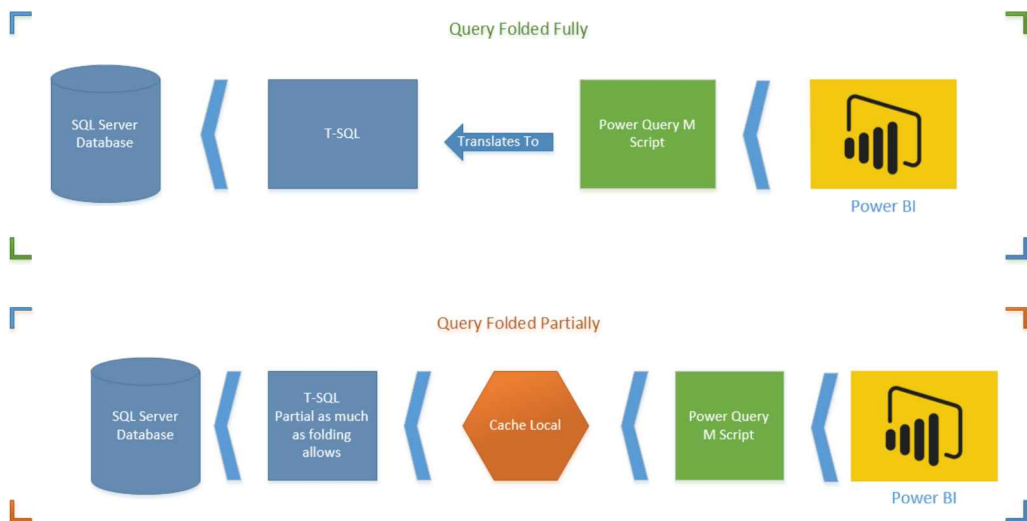
[illegible]

The query above runs in less than a second, vs. the previous query that took 15 minutes to run! The only difference is the number of variables. So here is the takeaway for this chapter:

Watch the number of variables (steps) in Power Query. If too many variables, then best to combine some of them

Important Note: I am not saying that you have to combine all your variables. You will not usually see performance issues with less than a hundred variables. This problem starts to appear when the number goes beyond that. One critical point you need to consider is that if you combine variables together, then debugging or troubleshooting, or maintenance of the code would be harder as well. So use this solution only if needed.

Chapter 22: Not Folding; the Black Hole of Power Query Performance



Have you ever thought does your Power Query Transformations happens on the data source side (server-side) or the local memory (client-side)? When you use a relational data source, the query can be run on the data source, but it depends on transformations. Some transformations can be translated to query language of the data source, some not. For example, recently (I believe from the last few releases), Power BI Desktop added a feature called Merge Columns. Merge Columns concatenate columns to either create a new column or replace them with the new concatenated result. Previously you could do the concatenation by adding concatenation simply with **&** character. What you've done was adding a new custom column and writing M expression to concatenate columns. With the new Merge Column, this is much easier; you select columns and apply Merge Columns. This easiness does come with a price, a high price, I'd say, price of reducing the performance of Power Query and as a result, Power BI! Merge Columns doesn't support query folding, which means it will affect performance badly. This chapter will show you how this causes the performance issue and how it can be solved. Note that Merge Columns is an example here; this situation might happen with some other transformations as well.

Query Folding

I can't start talking about the issue without explaining what Query Folding is, so let's start with that. Query Folding means translating Power Query (M) transformations into the native query language of the data source (for example, T-SQL). In other words, when

Part 5: Performance tuning

you run Power Query script on top of a SQL Server database, query folding will translate the M script into T-SQL statements and fetch the final results.

Here is an example of M Script:

let

```
Source = Sql.Databases("."),  
AdventureWorks2012 = Source[{Name="AdventureWorks2012"}][Data],  
Sales_SalesOrderHeader =  
AdventureWorks2012[{Schema="Sales",Item="SalesOrderHeader"}][Data],  
#"Added Conditional Column" = Table.AddColumn(Sales_SalesOrderHeader, "Custom", each if  
[SubTotal] >= 100 then "0" else "1" )
```

in

```
#"Added Conditional Column"
```

And here is the folded version of that translated to native T-SQL query:

```
select [_.SalesOrderID] as [SalesOrderID],  
[_.RevisionNumber] as [RevisionNumber],  
[_.OrderDate] as [OrderDate],  
[_.DueDate] as [DueDate],  
[_.ShipDate] as [ShipDate],  
[_.Status] as [Status],  
[_.OnlineOrderFlag] as [OnlineOrderFlag],  
[_.SalesOrderNumber] as [SalesOrderNumber],  
[_.PurchaseOrderNumber] as [PurchaseOrderNumber],  
[_.AccountNumber] as [AccountNumber],  
[_.CustomerID] as [CustomerID],  
[_.SalesPersonID] as [SalesPersonID],  
[_.TerritoryID] as [TerritoryID],  
[_.BillToAddressID] as [BillToAddressID],  
[_.ShipToAddressID] as [ShipToAddressID],  
[_.ShipMethodID] as [ShipMethodID],  
[_.CreditCardID] as [CreditCardID],  
[_.CreditCardApprovalCode] as [CreditCardApprovalCode],  
[_.CurrencyRateID] as [CurrencyRateID],
```

```

[_.[SubTotal] as [SubTotal],
[_.[TaxAmt] as [TaxAmt],
[_.[Freight] as [Freight],
[_.[TotalDue] as [TotalDue],
[_.[Comment] as [Comment],
[_.[rowguid] as [rowguid],
[_.[ModifiedDate] as [ModifiedDate],
case
    when [_.[SubTotal] >= 100
    then '0'
    else '1'
end as [Custom]
from
(
    select [$Table].[SalesOrderID] as [SalesOrderID],
        [$Table].[RevisionNumber] as [RevisionNumber],
        [$Table].[OrderDate] as [OrderDate],
        [$Table].[DueDate] as [DueDate],
        [$Table].[ShipDate] as [ShipDate],
        [$Table].[Status] as [Status],
        [$Table].[OnlineOrderFlag] as [OnlineOrderFlag],
        [$Table].[SalesOrderNumber] as [SalesOrderNumber],
        [$Table].[PurchaseOrderNumber] as [PurchaseOrderNumber],
        [$Table].[AccountNumber] as [AccountNumber],
        [$Table].[CustomerID] as [CustomerID],
        [$Table].[SalesPersonID] as [SalesPersonID],
        [$Table].[TerritoryID] as [TerritoryID],
        [$Table].[BillToAddressID] as [BillToAddressID],
        [$Table].[ShipToAddressID] as [ShipToAddressID],
        [$Table].[ShipMethodID] as [ShipMethodID],
        [$Table].[CreditCardID] as [CreditCardID],
        [$Table].[CreditCardApprovalCode] as [CreditCardApprovalCode],
        [$Table].[CurrencyRateID] as [CurrencyRateID],

```

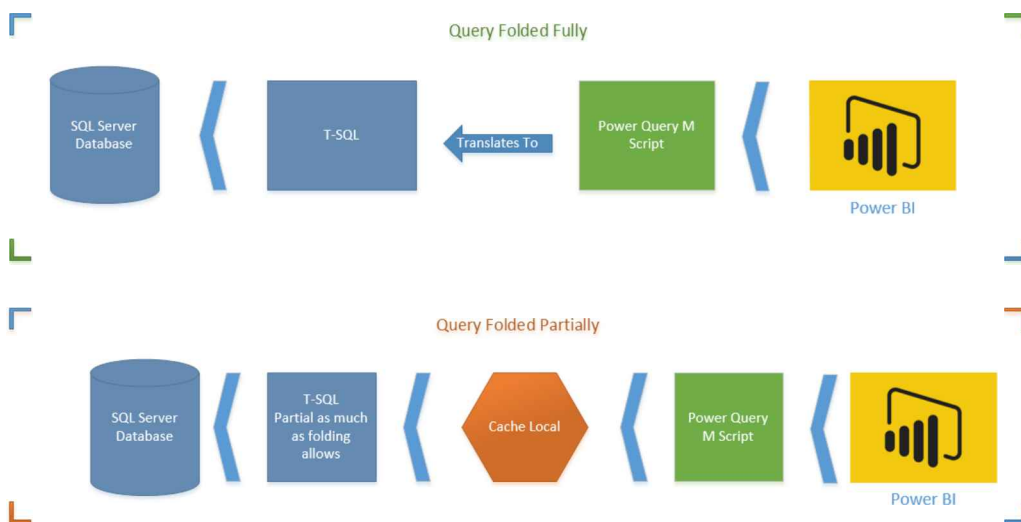
Part 5: Performance tuning

```
[ $Table ].[ SubTotal ] as [ SubTotal ],  
[ $Table ].[ TaxAmt ] as [ TaxAmt ],  
[ $Table ].[ Freight ] as [ Freight ],  
[ $Table ].[ TotalDue ] as [ TotalDue ],  
[ $Table ].[ Comment ] as [ Comment ],  
convert(nvarchar(max), [ $Table ].[ rowguid ]) as [ rowguid ],  
[ $Table ].[ ModifiedDate ] as [ ModifiedDate ]  
from [ Sales ].[ SalesOrderHeader ] as [ $Table ]  
) as [ _ ]
```

You can see how the conditional column script in M translated to the Case statement in T-SQL.

Is Query Folding Good or Bad?

Good obviously. Why? Because performance is much higher to run transformations on billions of records in the data source, rather than bringing millions of records into cache and applying some transformations on it.



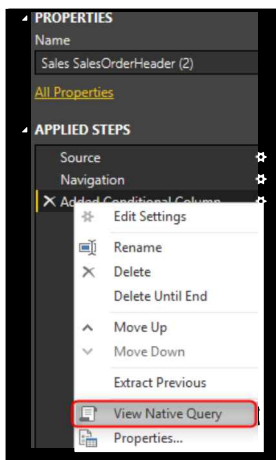
The first diagram shows an M script fully folded (translated to T-SQL). This is the best situation. The server-side operation applies all transformations on the data set and returns only desired result set.

The second diagram shows Query Folding partially supported (only up to a specific step). In this case, T-SQL brings the data before that step. And data will be loaded in the local

cache, and the rest of the transformations happen on the M engine side. You have to avoid this option as much as possible.

Can I see the Native Query?

The question that might come into your mind right now is; Can I see the Native Query that M script translates to it? The answer is Yes. If Query Folding is supported on a step, you can right-click on that step and click on View Native Query.



So Why Not Query Folding?

Query Folding is enabled by default. However, in some cases, it is not supported. For example, if you are doing some transformations on a SQL Server table in Power Query and then join it with a web query folding stops from the time you bring the external data source. That means transformations will happen on the data of the SQL Server table. Then before joining to web query, it will be fetched into cache, and then the rest of the steps happens by M engine. You would need to bring data from different data sources in Power BI, and this is the ability that Power Query gives to you. So sometimes, you have to step beyond query folding, and there might be no better way of doing that.

There are also some Transformations in Power Query that Query Folding doesn't support. Example? Merge Columns! Fortunately, there are workarounds for this situation. Let's dig into this more in detail.

Example: Merge Columns

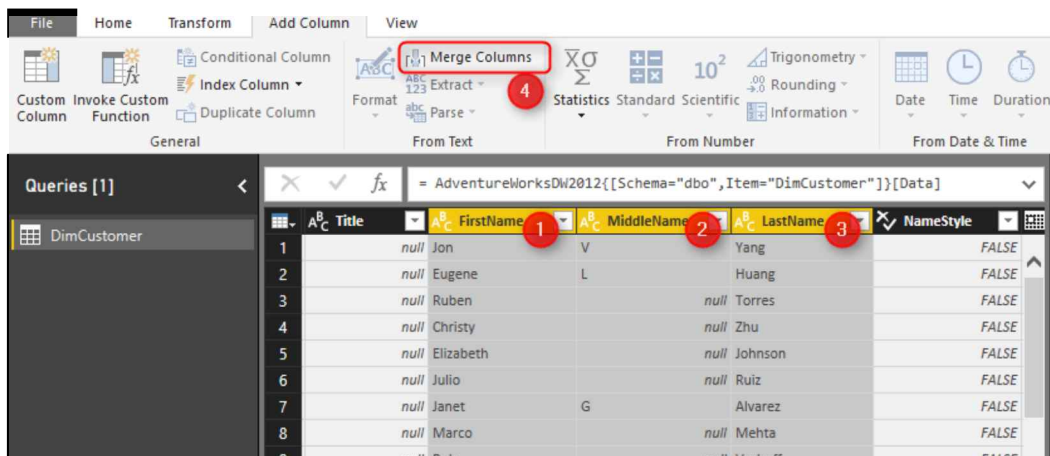
Merge Columns concatenates columns to each other in the order of selection of columns. You can also specify the delimiter character(s). So simply, if you want to create a full name from first name and last name, you can select them in the right order, and from either Transform tab or Add Column tab choose Merge Columns. Let's see this through an example;

Prerequisite for running the example

You need to have the AdvanetureWorksDW database installed on SQL Server. You can use any table in SQL Server that has two string columns that can be concatenated.

Merge Column Transformation

Create a new Power BI file in Power BI Desktop, and Get Data from SQL Server with Import Data Mode. Get Data from DimCustomer table only, and click on Edit. When in Query Editor. Select First Name, Middle Name, and Last Name in the correct order, and then from Add Column Select Merge Columns.



In the Merge, Columns Specify the Separator to be space and name the new column to be Full Name.

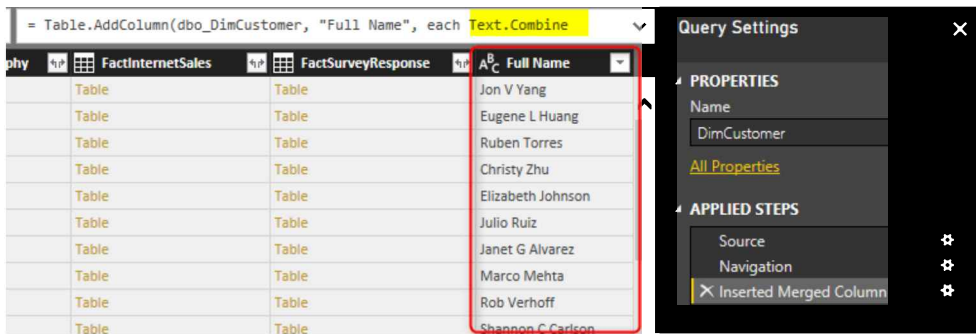
Merge Columns

Choose how to merge the selected columns.

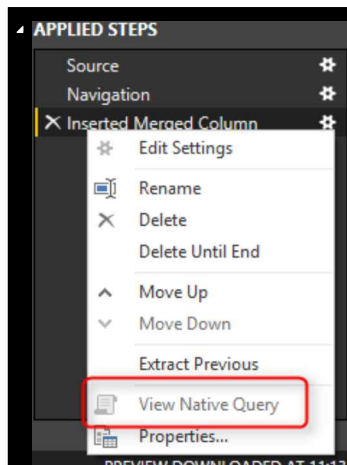
Separator
Space

New column name (optional)
Full Name

You will see the new column generates simply and adds to the end of all columns. You also may notice that Merge Columns uses Text. Combine Power Query function to concatenate columns to each other.



Now to see the problem with Query Folding, right-click on Inserted Merge Column step in the Applied Steps section. You will see that View Native Query is disabled.



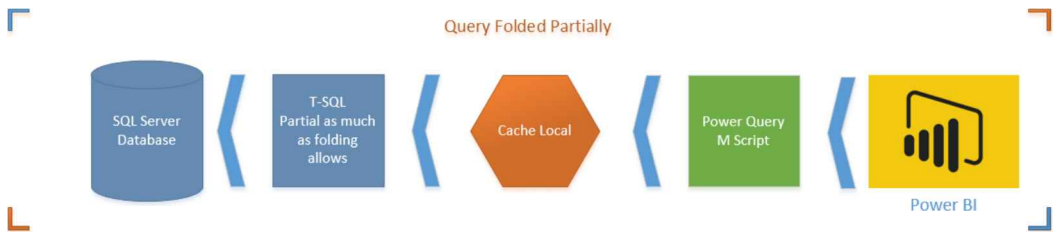
When View Native Query is not enabled, that step might be still folded, Ideally you have to check something like SQL Query Profiler to catch the query sent to the data source. In this case I did it and I know the query doesn't fold. that means the data will be loaded into cache up the step before this step. and then rest of the operation will happen locally. To understand how it works, right click on the step before which was Navigation. You will see the View Native Query. Click on that, and you can see the T-SQL query for that which is as below;

```
select [$Table].[CustomerKey] as [CustomerKey],
       [$Table].[GeographyKey] as [GeographyKey],
       [$Table].[CustomerAlternateKey] as [CustomerAlternateKey],
       [$Table].[Title] as [Title],
       [$Table].[FirstName] as [FirstName],
       [$Table].[MiddleName] as [MiddleName],
       [$Table].[LastName] as [LastName],
       [$Table].[NameStyle] as [NameStyle],
```

Part 5: Performance tuning

```
[{$Table}].[BirthDate] as [BirthDate],  
[{$Table}].[MaritalStatus] as [MaritalStatus],  
[{$Table}].[Suffix] as [Suffix],  
[{$Table}].[Gender] as [Gender],  
[{$Table}].[EmailAddress] as [EmailAddress],  
[{$Table}].[YearlyIncome] as [YearlyIncome],  
[{$Table}].[TotalChildren] as [TotalChildren],  
[{$Table}].[NumberChildrenAtHome] as [NumberChildrenAtHome],  
[{$Table}].[EnglishEducation] as [EnglishEducation],  
[{$Table}].[SpanishEducation] as [SpanishEducation],  
[{$Table}].[FrenchEducation] as [FrenchEducation],  
[{$Table}].[EnglishOccupation] as [EnglishOccupation],  
[{$Table}].[SpanishOccupation] as [SpanishOccupation],  
[{$Table}].[FrenchOccupation] as [FrenchOccupation],  
[{$Table}].[HouseOwnerFlag] as [HouseOwnerFlag],  
[{$Table}].[NumberCarsOwned] as [NumberCarsOwned],  
[{$Table}].[AddressLine1] as [AddressLine1],  
[{$Table}].[AddressLine2] as [AddressLine2],  
[{$Table}].[Phone] as [Phone],  
[{$Table}].[DateFirstPurchase] as [DateFirstPurchase],  
[{$Table}].[CommuteDistance] as [CommuteDistance],  
[{$Table}].[FullName] as [FullName]  
from [dbo].[DimCustomer] as [{$Table}]
```

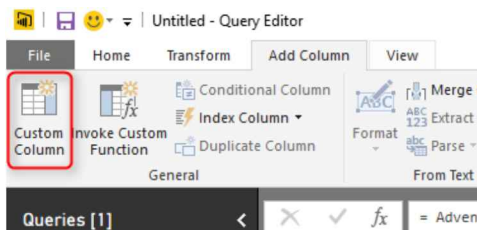
You can see that this is a simple query from DimCustomer Table. What will happen here in this scenario is that Power Query cannot translate Text.Combine to T-SQL Query. So data up to step before will be loaded into the cache. It means the query for a step before (above query) will run on the database server, the result will come to the cache, and then Text.Combine will happen on the local memory in cache. Here is a diagram of how it works;



In this example, the data set is small, but if the data set is big, then not folding causes performance issues. It takes much longer to load the whole data set into cache and then apply transformations, rather than doing transformations in the data source and just loading the result into Power BI.

Solution: Simple Concatenate with Add Column

Now remove the step for Inserted Merged Column, and go to Add Column Tab, and select Custom Column



In the Add Custom Column, write the below expression to generate Full Name;

```
=
[FirstName]
&
" "
&
(if [MiddleName]=null then "" else [MiddleName])
&
" "
&[LastName]
```

Part 5: Performance tuning

Add Custom Column

New column name
Full Name

Custom column formula:
=
[FirstName]
&
" "
&
(if [MiddleName]=null then "" else [MiddleName])
&
" "
&[LastName]

This expression used the concatenation character, which is **&**, and also checked if Middle Name is null or not. The result in the Power Query side is the same, and it generates the Full Name column like the previous example;

The screenshot shows the Power Query Editor interface. At the top, the formula bar displays: `Table.AddColumn(dbo_DimCustomer, "Full Name", each [FirstName]`. Below the formula bar, a table preview shows columns for **FactInternetSales**, **FactSurveyResponse**, and a newly added **Full Name** column. The **Full Name** column contains concatenated values like "Jon V Yang", "Eugene L Huang", etc. On the right, the **Query Settings** pane is open, showing the **PROPERTIES** section with the name "DimCustomer" and the **APPLIED STEPS** section with "Source", "Navigation", and "Added Custom".

However, it is different for Query Folding. Right Click on the Added Custom step, and you will see the Native Query this time.

The screenshot shows the context menu for the "Added Custom" step in the **APPLIED STEPS** list. The menu options include: "Edit Settings", "Rename", "Delete", "Delete Until End", "Move Up", "Move Down", "Extract Previous", "View Native Query" (highlighted with a red box), and "Properties...".

Query is simply the same with a new concatenated column added.

```
select [_.CustomerKey] as [CustomerKey],  
       [_.GeographyKey] as [GeographyKey],
```

```

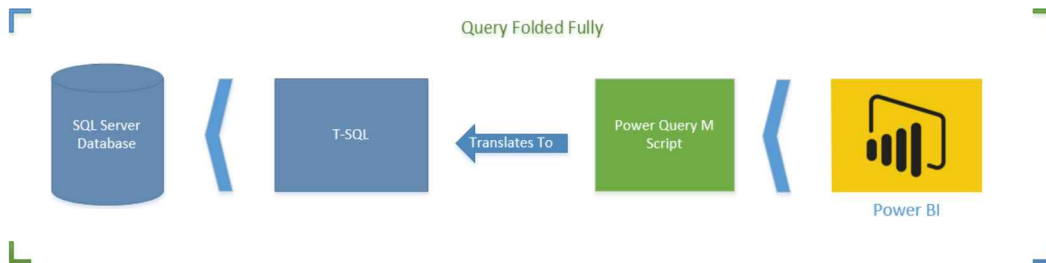
[].[CustomerAlternateKey] as [CustomerAlternateKey],
[].[Title] as [Title],
[].[FirstName] as [FirstName],
[].[MiddleName] as [MiddleName],
[].[LastName] as [LastName],
[].[NameStyle] as [NameStyle],
[].[BirthDate] as [BirthDate],
[].[MaritalStatus] as [MaritalStatus],
[].[Suffix] as [Suffix],
[].[Gender] as [Gender],
[].[EmailAddress] as [EmailAddress],
[].[YearlyIncome] as [YearlyIncome],
[].[TotalChildren] as [TotalChildren],
[].[NumberChildrenAtHome] as [NumberChildrenAtHome],
[].[EnglishEducation] as [EnglishEducation],
[].[SpanishEducation] as [SpanishEducation],
[].[FrenchEducation] as [FrenchEducation],
[].[EnglishOccupation] as [EnglishOccupation],
[].[SpanishOccupation] as [SpanishOccupation],
[].[FrenchOccupation] as [FrenchOccupation],
[].[HouseOwnerFlag] as [HouseOwnerFlag],
[].[NumberCarsOwned] as [NumberCarsOwned],
[].[AddressLine1] as [AddressLine1],
[].[AddressLine2] as [AddressLine2],
[].[Phone] as [Phone],
[].[DateFirstPurchase] as [DateFirstPurchase],
[].[CommuteDistance] as [CommuteDistance],
[].[FullName] as [FullName],
((([].[FirstName] + ' ') + (case
    when [].[MiddleName] is null
    then ''
    else [].[MiddleName]
end))) + ' ') + [].[LastName] as [Full Name]

```

Part 5: Performance tuning

from [dbo].[DimCustomer] as [__]

This time there won't be an intermediate cache. Transformation happens in the data source with the T-SQL query, and the result will be loaded into Power BI.



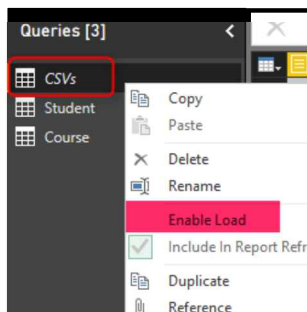
How Do I know Which Transformations Folds?

Great Question. It is important to understand which step/transformation folds and which doesn't. To understand that, you need to monitor the data source and check the queries sent there based on the transformations applied on the Power Query. Also, Note that Query Folding is not supported for data sources such as web query, CSV, or things like that. Query Folding at the moment is only supported for data stores that support a native query language. For Web, Folder, CSV... there is no native query language, so you don't need to worry about Query Folding.

***Important Note:** At the time of writing this chapter, Merge Columns doesn't support Query Folding. I have reported this to Power Query Team, and they are working on it to solve the issue. The Merge Columns is very likely to support query folding very soon as the result of a bug fix. However, there are always some other transformations that don't support query folding. This chapter is written to give you an understanding of what kind of issue might happen and how to resolve it.

My advice to you as a performance best practice is that when working with a relational data source (such as SQL Server). Always check the query folding. Sometimes it is not supported, so use another approach for the transformation. Don't fall into the black hole of not folding; otherwise, your query might take ages to run.

Chapter 23: Performance Tip for Power BI; Enable Load Sucks Memory Up



In the area of performance tuning a Power BI model, many things have to be considered, most of them around the consumption of the CPU and RAM. One of the most basic but important considerations is minimizing the usage of memory. By default, all queries from Query Editor will be loaded into the memory of the Power BI Model. In this chapter, I'll show you an example to disable the load for some queries, especially queries used as intermediate transformation to produce the final query for the model. This is a very basic tip but very important when your model grows big.

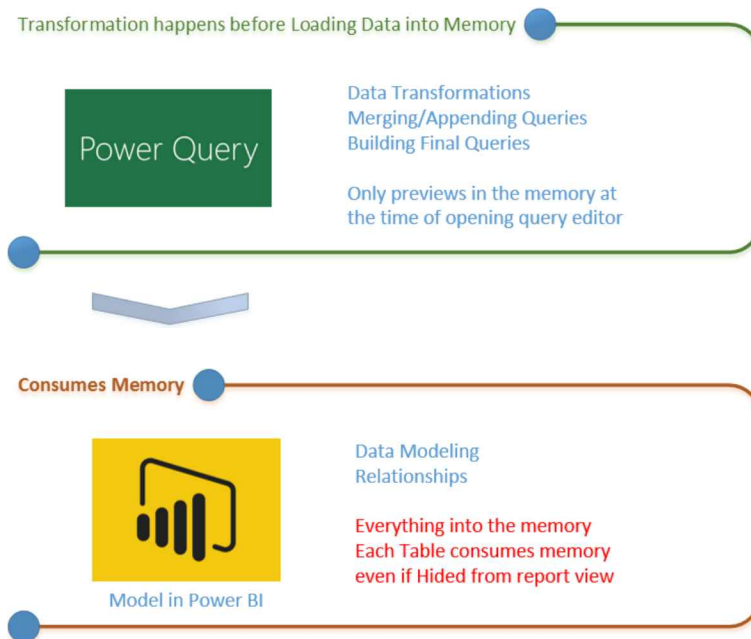
Prerequisite

For running examples of this book, you need to download the ZIP file from this book's code files.

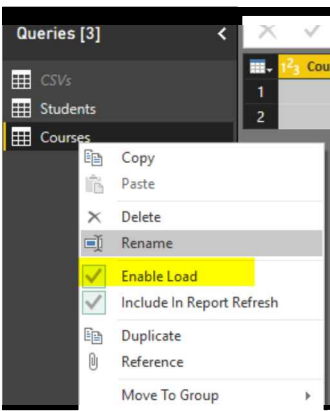
Load Mechanism for Power Query

By default, all queries in Power Query will be loaded into the Power BI model. This behavior might be the desired behavior if you connect to a proper star schema modeled data warehouse because normally, you don't need to make many changes in the structure of queries. However, this brings issues if you are connected to a transactional data store, some files, web source, and many other non-star schema modeled data sources. Normally when you get data from a data source, you apply transformations for rows and columns and merge queries or append them, and you might end up having five tables out of 10 queries as final queries. By default, when you close and apply your query editor window, all queries will be loaded into the model no matter if you want to use them in your final model or not.

Part 5: Performance tuning

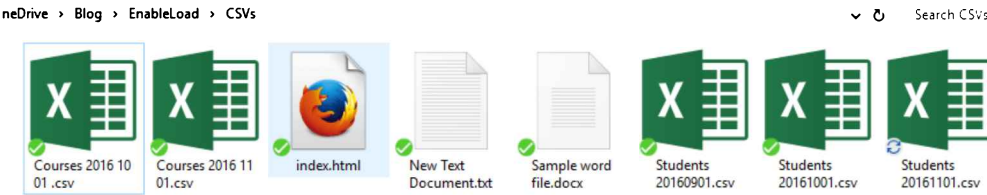


Every query that loads into model memory will be consumed. And Memory is our asset in the Model. Less memory consumption leads to better performance in most cases. I have seen lots of models that people Hide the unwanted query from the model. This approach doesn't help the performance because the hidden query will still consume the memory. The best approach is to disable loading before closing the query editor. Disabling Load doesn't mean the query won't be refreshed. It only means the query won't be loaded into the memory. When you click on the Refresh model in Power BI, or when a scheduled refresh happens, queries marked as Disable Load will be refreshed. Still, their data will be used as an intermediate source for other queries instead of loading directly into the model. This is a basic performance tuning tip but very important when your Power BI model grows bigger and bigger. Let's look at this through an example.



Example Scenario

In this example Scenario, I want to get a list of all files from a directory. There are two types of CSV files in the directory, some Students' files and some course files; both are CSV files but different structured. Also, there might be other files in the directory, such as Word files or HTML files, that I don't want to process.



The aim is to load data rows of all students and courses as two separate tables into the model. Instead of fetching files from the folder twice, we use one query to fetch the files and then use it as a reference for other queries. The referenced query itself doesn't need to be loaded into the model. Let's build the solution and see how it works in action.

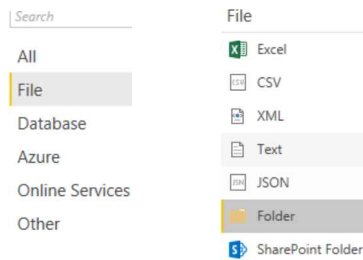
Build the Transformations

Get Data from Folder

Open a new Power BI file, and start by Getting Data from Folder

Part 5: Performance tuning

Get Data



Enter the path of the folder that contains all files (Files in this folder can be downloaded from ZIP file up in the prerequisite section of this chapter)

Folder

Choose a folder.

Folder Path

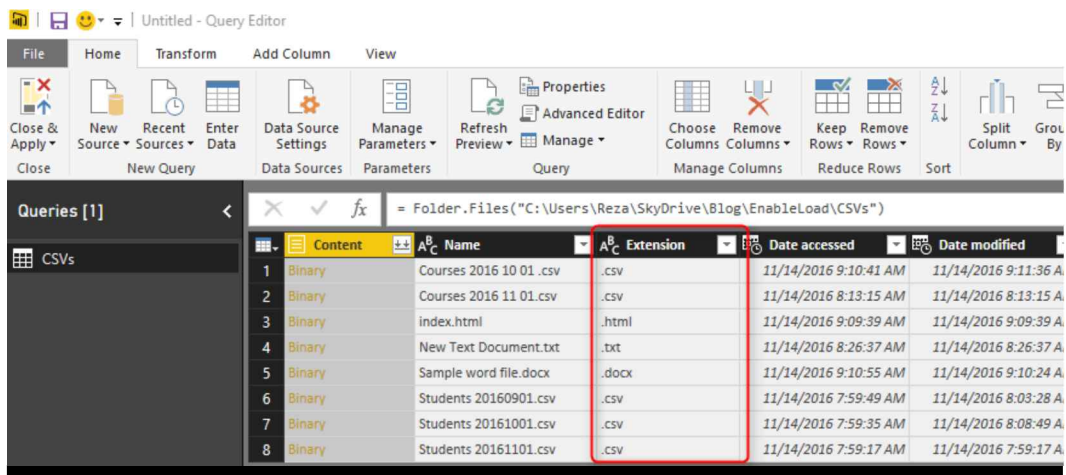
C:\Users\Reza\SkyDrive\Blog\EnableLoad\CSVs

Browse...

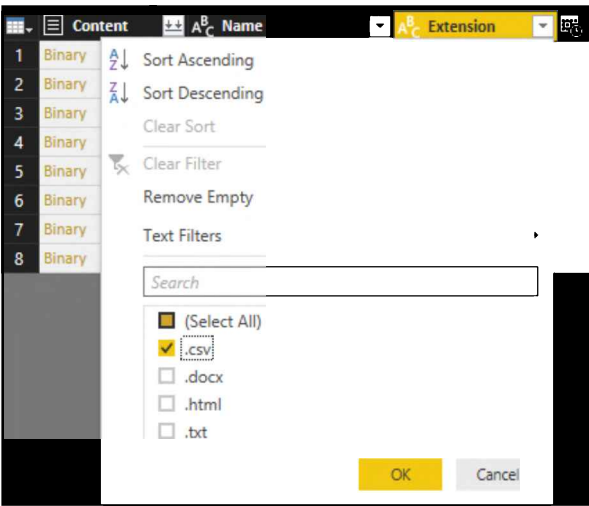
OK

Cancel

Click on Edit in the preview showed in the navigator window to open the folder content in Query Editor. As you see, there is a number of different files in the folder.



Filter the Extension to only .csv. Note that both Course and Student files are .CSV files which is what we need.



Now the subset includes Course files and Students files which are different in the structure. We have to apply transformations to each set individually.

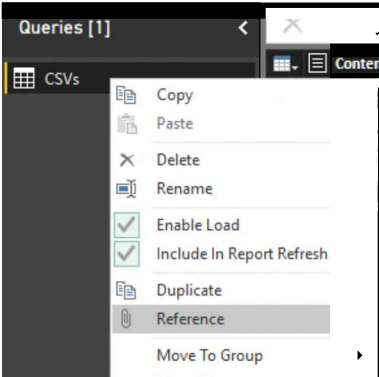
fx

= Table.SelectRows(Source, each ([Extension] = ".csv"))

| | Content | Name | Extension | Date accessed | Date modified |
|---|---------|-------------------------|-----------|-----------------------|-----------------------|
| 1 | Binary | Courses 2016 10 01 .csv | .csv | 11/14/2016 9:10:41 AM | 11/14/2016 9:11:36 AM |
| 2 | Binary | Courses 2016 11 01.csv | .csv | 11/14/2016 8:13:15 AM | 11/14/2016 8:13:15 AM |
| 3 | Binary | Students 20160901.csv | .csv | 11/14/2016 7:59:49 AM | 11/14/2016 8:03:28 AM |
| 4 | Binary | Students 20161001.csv | .csv | 11/14/2016 7:59:35 AM | 11/14/2016 8:08:49 AM |
| 5 | Binary | Students 20161101.csv | .csv | 11/14/2016 7:59:17 AM | 11/14/2016 7:59:17 AM |

Students Query

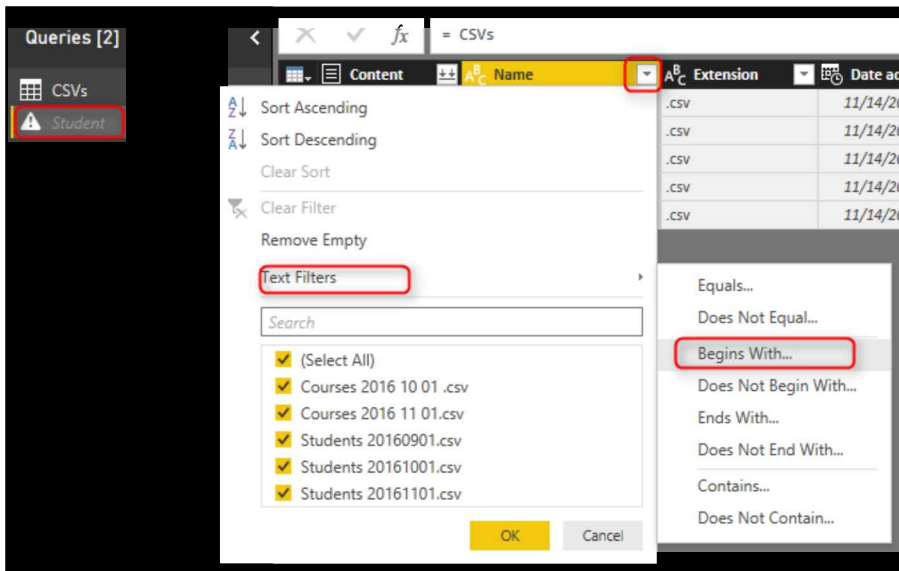
Because I don't want to repeat the process of getting all files from a folder, I want to split the data into two data sets; one for Students and another for Courses. I'll generate a REFERENCE from the main query. Right-click on the query (which is called CSVs in my example) and select Reference.



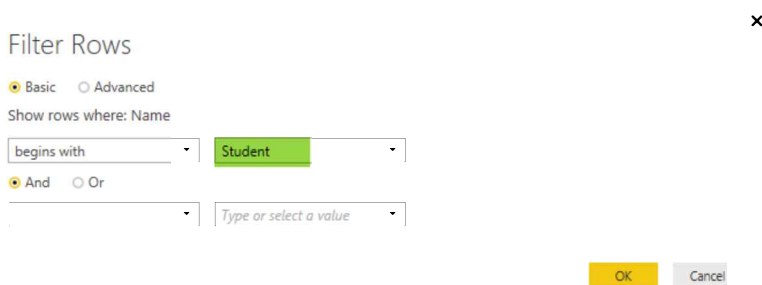
This will generate a new query named as CSVs (2). This new query is NOT A COPY of the first query. This is only a reference from the first query. This means if the first query

Part 5: Performance tuning

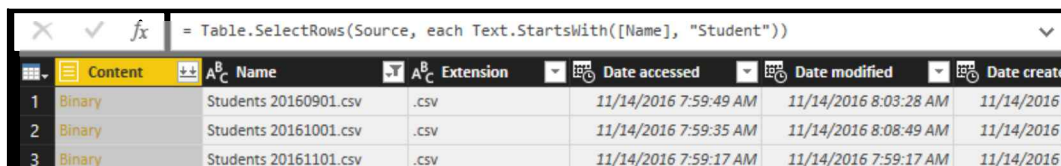
changes, the source for this query will also change. Rename this new query to be Student. Filter the Name column to everything starting (or begins with) "Student."



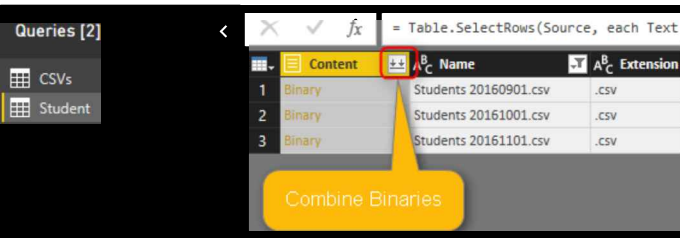
I don't type in the search box and use Text Filters specifically because the search box will filter the data statistically based on values that exist in the current data set. If in the future new values (file names) comes in the folder, this won't consider that. However, the Text Filter will apply to any data set because it will be filtered dynamically; in the Filter Rows window, type in Student as a filter.



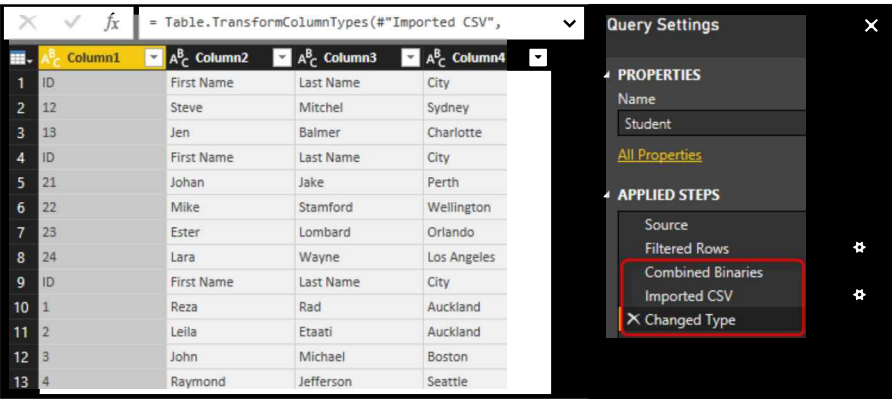
Now you will see only Students files.



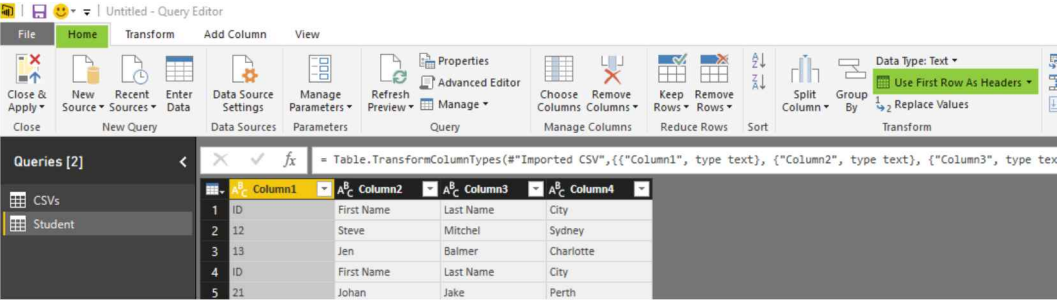
Click on the Combine Binaries icon on the header of the Content column to combine all CSV files into one.



After combining binaries, Power Query will also import the CSV into the table and do automatic data type conversion. You can see all three steps in the Query Editor.



The data table needs a couple of changes before being ready. First, set the column names. The first row has column names. So use the menu option of “Use First Row As Headers” to promote the first row to be column headers.



This simply brings column headers;

Part 5: Performance tuning

fx = Table.PromoteHeaders("#Changed Type")

| | ID | First Name | Last Name | City |
|----|----|------------|-----------|-------------|
| 1 | 12 | Steve | Mitchel | Sydney |
| 2 | 13 | Jen | Balmer | Charlotte |
| 3 | ID | First Name | Last Name | City |
| 4 | 21 | Johan | Jake | Perth |
| 5 | 22 | Mike | Stamford | Wellington |
| 6 | 23 | Ester | Lombard | Orlando |
| 7 | 24 | Lara | Wayne | Los Angeles |
| 8 | ID | First Name | Last Name | City |
| 9 | 1 | Reza | Rad | Auckland |
| 10 | 2 | Leila | Etaati | Auckland |
| 11 | 3 | John | Michael | Boston |
| 12 | 4 | Raymond | Jefferson | Seattle |

Also, you need to remove all extra header rows from the combined data set. Some rows with "ID, First Name, Last Name, and City" as their values should be removed from the data set. You can simply remove them with a Text Filter of Does Not Equal to on the ID column. Values should not be equal to "ID" because every row with "ID" in the first column's value is a header row and should be removed.

Filter Rows

☒ Basic ☐ Advanced

Show rows where: ID

does not equal ID

☒ And ☐ Or

Type or select a value

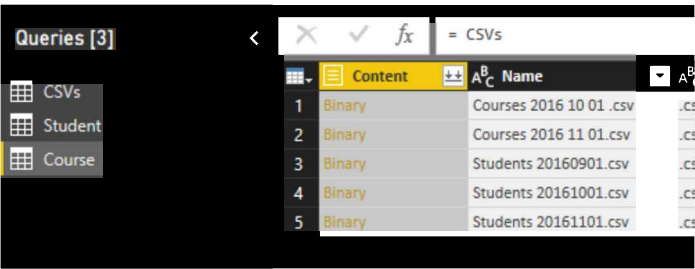
OK Cancel

Now you have cleaned the data set for Students. Note that I have also applied a data type conversion for column ID to type the whole number;

| | ID | First Name | Last Name | City |
|----|----|------------|-----------|-------------|
| 1 | 12 | Steve | Mitchel | Sydney |
| 2 | 13 | Jen | Balmer | Charlotte |
| 3 | 21 | Johan | Jake | Perth |
| 4 | 22 | Mike | Stamford | Wellington |
| 5 | 23 | Ester | Lombard | Orlando |
| 6 | 24 | Lara | Wayne | Los Angeles |
| 7 | 1 | Reza | Rad | Auckland |
| 8 | 2 | Leila | Etaati | Auckland |
| 9 | 3 | John | Michael | Boston |
| 10 | 4 | Raymond | Jefferson | Seattle |

Courses Query

The other entity is Course which we need to create as a reference from the CSVs query. Create another reference from CSVs, and name it Course.



| | Content | Name |
|---|---------|-------------------------|
| 1 | Binary | Courses 2016 10 01 .csv |
| 2 | Binary | Courses 2016 11 01 .csv |
| 3 | Binary | Students 20160901.csv |
| 4 | Binary | Students 20161001.csv |
| 5 | Binary | Students 20161101.csv |

Create a Text Filter for Begins on the Name Column with the value of “Course.”

Filter Rows

Basic Advanced

Show rows where: Name

begins with Course

And you'll have only Course files. Combine Binaries on that.

| | Column1 | Column2 | Column3 | Column4 | Column5 |
|---|---------------|------------|----------------------------|----------|------------|
| 1 | Course Number | Department | title | Capacity | Date |
| 2 | 2 | BI | SSIS Training | 30 | 2016/10/1 |
| 3 | 3 | Database | Execution Plans | 30 | 2016/10/15 |
| 4 | Course Number | Department | title | Capacity | Date |
| 5 | 12 | BI | Power BI Training | 30 | 2016/11/1 |
| 6 | 13 | Database | High Availability in a Day | 30 | 2016/12/1 |

Same as Student query, apply transformations in this order;

- Promote headers with “Use First Row As Headers.”
- Create a Text Filter on the Course Number Column that values does not equal to “Course Number.”
- Apply data type conversion on Course Number and Capacity to Whole Number and Date to Date.

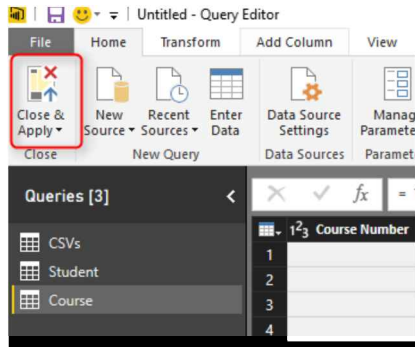
Here is the final data set for the course;

| | Course Number | Department | title | Capacity | Date |
|---|---------------|-------------|----------------------------|----------|------------|
| 1 | | 2 BI | SSIS Training | 30 | 10/1/2016 |
| 2 | | 3 Database | Execution Plans | 30 | 10/15/2016 |
| 3 | | 12 BI | Power BI Training | 30 | 11/1/2016 |
| 4 | | 13 Database | High Availability in a Day | 30 | 12/1/2016 |

Default Behavior: Enable Load

Now to see the problem, without any changes in the default load behavior, Close the Query Editor and Apply changes.

Part 5: Performance tuning

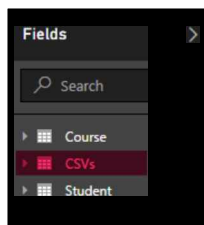


You will see that after the refresh, three queries load in the model; Student, Course, and CSVs.

Apply Query Changes

- CSVs
Creating connection in model...
- Student
Creating connection in model...
- Course
Creating connection in model...

Student and Course are expected tables in the model. However, CSVs are not useful. We already fetched everything we wanted from the query, and this is used as an intermediate query for loading Student and Course. Having CSVs as a separate table in our model has two problems;

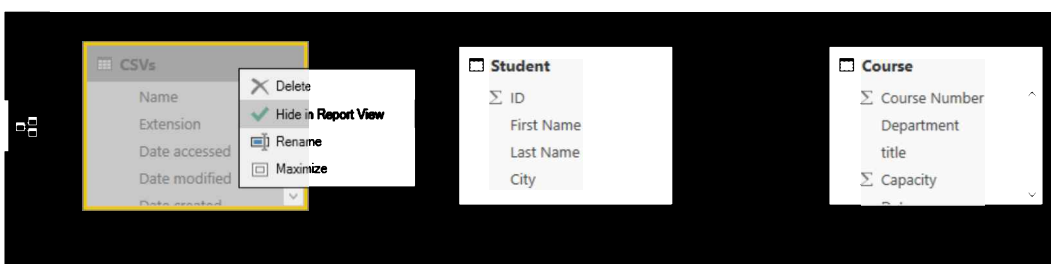


- one extra table; a confusion for users
- extra memory consumption

The main issue is the memory consumption for this extra table. Normally, memory consumption will reduce the performance and bring a heavier load to the model. In this example, the CSVs table only has few rows. But this is just sample data; in real-world examples, you might have intermediate tables with millions of rows. You need to remove every unused table from the Power BI model to enhance memory consumption.

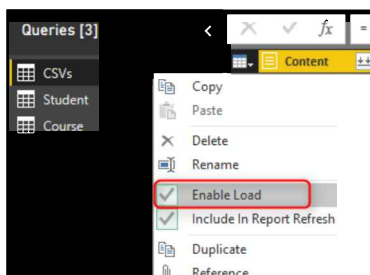
What about Hiding from Report?

The first issue, “Confusion for users,” can be solved by hiding the table from the report view. You can do that in the Relationship tab of the model, right-click on the CSVs table, and click on Hide from report view. This method HIDEs the table from the report view. However, the table still exists, and it consumes memory! It is just hidden. Hiding a table from the report view is good for tables that you need for the modeling. For example, a relationship table (that creates many to many relationships) should be kept in the model but be hidden. Because it creates a relationship, but it is not useful for the report viewer. Any other tables that are not used for relationships and are not used for reporting should be removed before loading into the model.

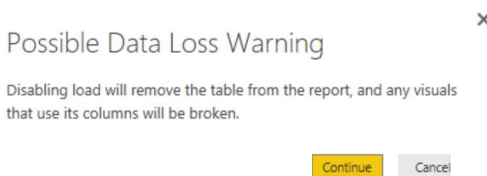


Disable Load to Save Memory

Go back to Query Editor and right-click on CSVs query. You will see that by default, every query is checked as Enable Load.

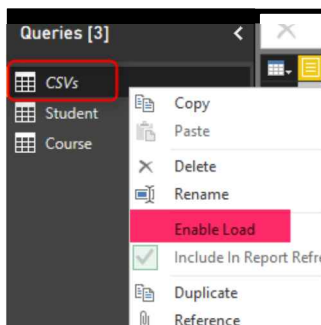


Click on the Enable Load to disable it. You will see a message saying that any visuals attached to this query will not work after this change, which is fine because this query is not used (and won't be) for any visualization. The message is: “Disabling load will remove the table from the report, and any visuals that use its columns will be broken.”

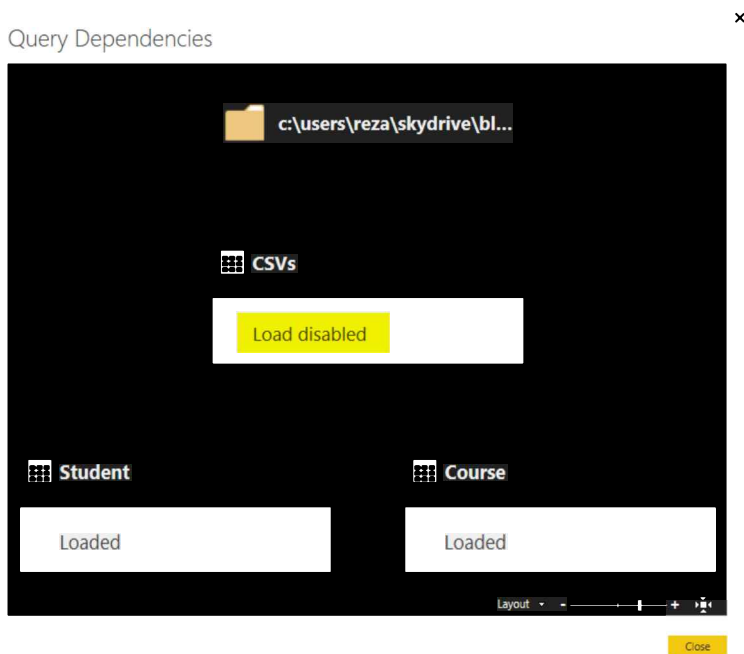


Part 5: Performance tuning

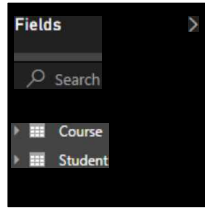
Click on Continue, and you will see the load is disabled for the query now. The query name will also be in *Italic* font, illustrating that it won't be loaded into the model. Note that the query will be refreshed, and if new files come to the directory, it will pass it on to Course and Student queries. It just won't be loaded into the model itself.



You can also see in the View tab of Query Editor on the Query Dependencies that this query is the source of the other two queries. And in that view, it also shows which query is load disabled.

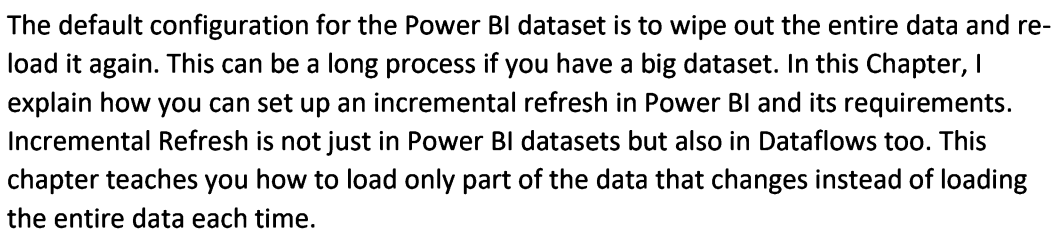


Now close and apply the Query Editor. This time you see that CSVs query won't be loaded into the model. Your relationship tab and the report tab only contain two tables; Course and Student.



Summary

In summary, you've seen how an easy tip can save memory and enhance the performance of the Power BI model. Always remember that Query Editor is your ETL (Extract, Transform, Load) engine. It will apply all transformations before loading the data into the model. Still, once it is finished with the transformation, all queries will be loaded into the model, and they take memory. By default, all queries are Enabled to Load into the model. Simply change that for queries that are not required in the model.



What is Incremental Refresh?

When you load data from the source into the destination (Power BI), there are two methods: Full Load or Incremental Refresh. Full Load means fetching the entire dataset each time and wiping out the previous data. When I say the entire dataset, I mean after all Power Query transformations, because there might be some filtering in Power Query, whatever data that loads into the Power BI dataset in this sentence is considered as the entire data.

If the dataset is small, or the refresh process is not taking a long time, then the full load is not a problem at all. The problem happens when the dataset is big, or the refresh process takes longer than an acceptable timeframe. Consider you have a large dataset including 20 years of data. From that dataset, probably the data of 20 years ago won't change at all anymore, or even the data for five years ago, sometimes even a year ago. So why re-processing it again? Why re-loading data that doesn't update? Incremental Refresh is the process of loading only part of the data that might change and adding it to the previous dataset, which is not changing anymore.

Requirements: What do you need to set up?

Table with Date Field(s)

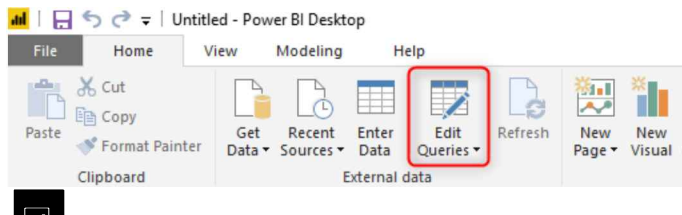
For setting up the incremental refresh, you do need to have a table (or more) with date field(s) in it. The date field is the field that will have an impact on the partial refresh of the data. For example; Let's say you have a FactSales table. You want to load all Sales which made earlier than a year ago just once, but everything from a year ago to now regularly. So you do need to have a date field on your table for it. Most of the time, this field can be created date, modified date, order date, publish date, etc. You do need to have a field like that as of date data type.

How to Setup Incremental Refresh

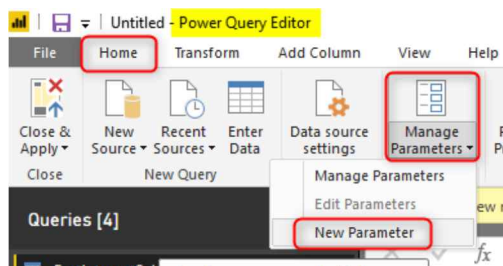
Setting up Incremental Refresh has some steps in Power BI Desktop and then in the Power BI Service. Let's check them one by one.

Parameters in Power Query

For setting up the incremental refresh in Power BI, you need to use Power Query parameters. You need to create two parameters with the reserved names of RangeStart and RangeEnd (Note that Power Query is a case-sensitive language). Go to Edit Queries in your Power BI Desktop solution,



And then click on New Parameters.



Then create two parameters of **DateTime** data type, with names of **RangeStart** and **RangeEnd**, set a default value for each too. The default value can be anything, but the name and the data type should be mentioned here.

Part 5: Performance tuning

Parameters

New

Name: RangeStart

Description:

RangeStart

RangeEnd

☒ Required

Type: Date/Time

Suggested Values: Any value

Current Value: 1/01/2010 12:00:00 AM

OK Cancel

Filter Data Based on Parameters

After creating the two parameters, you need to filter the data of the date field based on these two parameters. For example, in the FactInternetSales, I can filter the OrderDate using the column filtering as below;

gNumber ABC 123 CustomerPONumber OrderDate DueDate Ship

Sort Ascending

Sort Descending

Clear Sort

Clear Filter

Remove Empty

Date/Time Filters

Search

(Select All)

1/07/2005 12:00:00 AM

2/07/2005 12:00:00 AM

3/07/2005 12:00:00 AM

4/07/2005 12:00:00 AM

Between...

Equals...

Before...

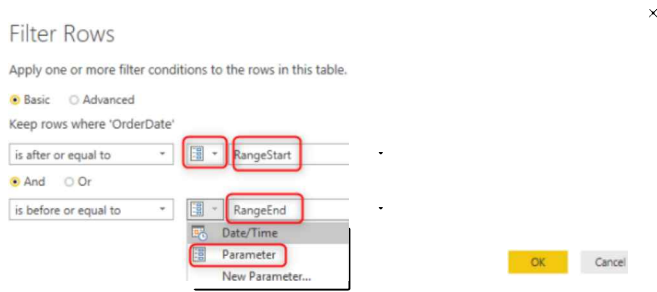
After...

In the Next...

In the Previous...

| | |
|------------------------|-----|
| 13/07/2005 12:00:00 AM | 8, |
| 13/07/2005 12:00:00 AM | 8, |
| 13/07/2005 12:00:00 AM | 8, |
| 13/07/2005 12:00:00 AM | 8, |
| 13/07/2005 12:00:00 AM | 8, |
| 14/07/2005 12:00:00 AM | 9, |
| | 9, |
| | 9, |
| | 10, |
| | 10, |
| | 10, |
| | 10, |
| | 10, |
| | 10, |

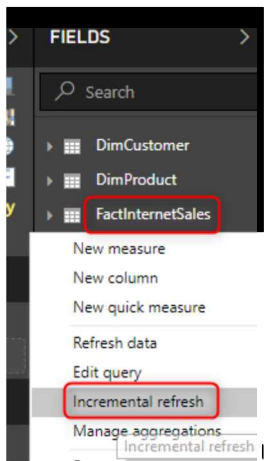
You can use the Between filter using the RangeStart and RangeEnd parameters as below;



After this action, your data in the table will be filtered based on the default values you set for the RangeStart and RangeEnd parameters. However, don't worry about that. These two parameters will be overwritten with the configuration you make in the Incremental Refresh setting of the Power BI Desktop.

Power BI Desktop Incremental Refresh Setup

The final step in Power BI Desktop is to close&apply the Power Query Editor window and set up the incremental refresh setting for the table. You have to right-click on the table in the Power BI Desktop and select Incremental Refresh.



In the Incremental Refresh settings window, you can choose the table first. If the table is a table that doesn't have the two parameters of RangeStart and RangeEnd used in filter criteria, then you won't be able to do the setting for it. For example, DimCustomer won't give me the option to do the incremental refresh settings.

Part 5: Performance tuning

Incremental refresh

⚠ Before you can set up incremental refresh on this table, you need to set up parameters. [Learn more](#)

You can improve the speed of refresh for large tables by using incremental refresh in Premium workspaces. This setting will apply once you've published a report to the Power BI service.

Table: **DimCustomer** Incremental refresh: **Off**

Store rows in the last:

Enter value: | **First value**

Refresh rows in the last:

Enter value: | **Select value**

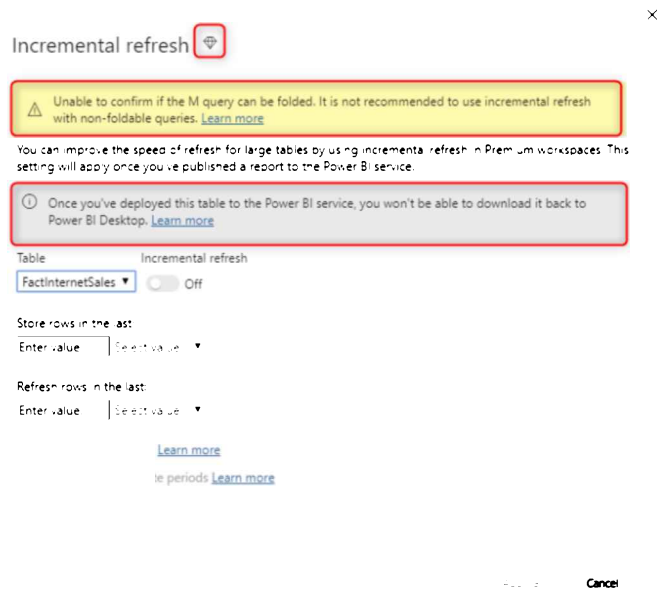
[Learn more](#)

ie periods [Learn more](#)

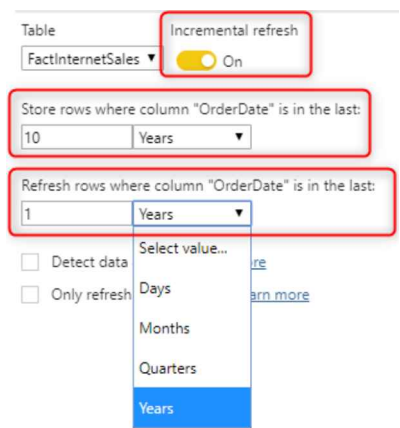
Cancel

However, I can do the settings for FactInternetSales because I did filter the OrderDate field of this table based on the parameters. There are two things to notice at this step: If your data source is coming from a data source that supports query folding, then Incremental load works as it best. If not, it is not recommended to use it. Most of the time, a huge data source is coming from a relational data store system, which supports query folding, by the way.

Another thing to note is that Incremental Refresh requires Premium licensing. And finally, you have to know that you cannot download a PBIX file from the service if it has an incremental refresh setup.



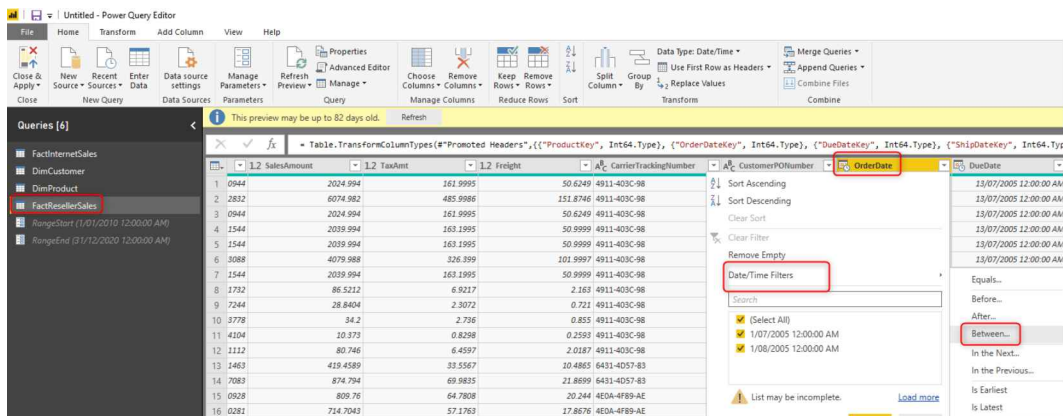
Configuration for the incremental refresh is easy. You just set up the amount rows to Store (load only once, and store it) and the amount rows to Refresh (re-load every time);



Incremental Refresh for Multiple Tables

You can set up the incremental refresh for multiple tables. You don't need more parameters; the two parameters of RangeStart and RangeEnd are enough. You just need to set up the filter in any other tables you want as well.

Part 5: Performance tuning



Then you need to set up the configuration in the Power BI Desktop for each table;

Table Incremental refresh

FactResellerSales ☒ On

Store rows where column "OrderDate" is in the last:

5 Months

Refresh rows where column "OrderDate" is in the last:

1 Months

Note that although we are using the same parameters of RangeStart and RangeEnd, you can have different configurations of Store and Refresh for the incremental refresh setting for each table.

Detect Data Changes

Incremental Refresh will make the part of the dataset to refresh much smaller, and as a result, the process would be much faster. However, There is still one better way to do that. If you have a modified DateTime (or updated DateTime) in your table, then the incremental refresh process can monitor that field and only get rows that their date/time is after the latest date/time in that field in the previous refresh. To enable this process, you can enable the Detect Data Changes and then choose the modified date or update date from the table. Notice that this is different from the OrderDate or transaction date. And not all tables have such a field.

☒ Detect data changes [Learn more](#)

Only refresh data in the last 1 year if the maximum value of this datetime column changes:

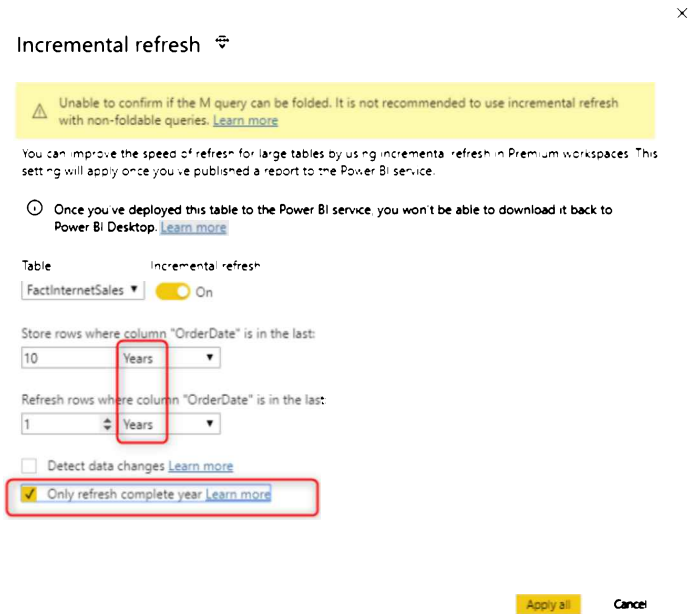
DueDate

OrderDate

ShipDate

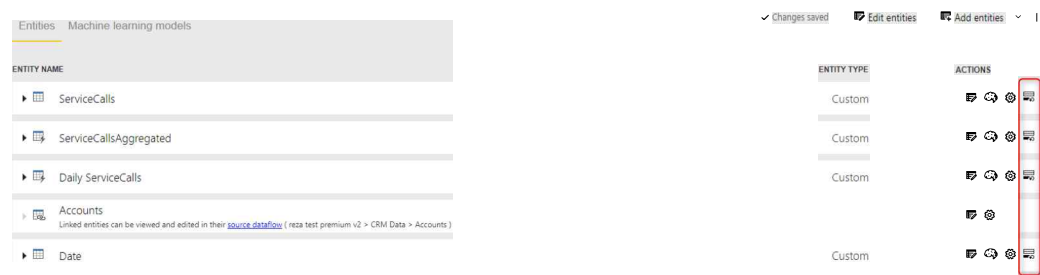
Only Refresh Complete Period

Depends on the period you selected when you set up the incremental refresh, you can choose to only refresh when the period is complete. For example, in the FactInternetSales, I set the refresh period to Year. And then I can set the option to Only Refresh the Complete Period, which means even If we were in Feb 2019, it would only refresh the data up to Dec 2018 (because that is the latest date that we have a full complete year in it);



Incremental Refresh for Dataflows

[Dataflows\[https://radacad.com/what-are-the-use-cases-of-dataflow-for-you-in-power-bi\]](https://radacad.com/what-are-the-use-cases-of-dataflow-for-you-in-power-bi) are ETL processes in the cloud for the Power BI service. You can also set up the Incremental Refresh for Dataflows. And it is even easier to do it for the dataflow. You don't need to create the RangeStart and RangeEnd parameters there. Just go to the dataflow Incremental Refresh setting directly.



Part 5: Performance tuning

You can see that the same setup of the incremental refresh setting is available for the dataflow;

Incremental refresh settings

ServiceCalls

Incremental refresh updates only data that's changed, to speed refresh, reduce capacity usage, and store historic data. [Learn more](#)

☒ On

Choose a DateTime field to filter by

callDate ▼

Store rows from the past

5 Years ▼

Refresh rows from the past

1 Year ▼

☐ Detect data changes [Learn more](#)

Only refresh data if the maximum value in this field changes

Choose a field ▼

☐ Only refresh complete years [Learn more](#)

When you save these settings, data from the past 5 years will be loaded to your dataflow storage the next time this dataflow is refreshed. Subsequent refreshes will update only data that's changed in the past 1 year.

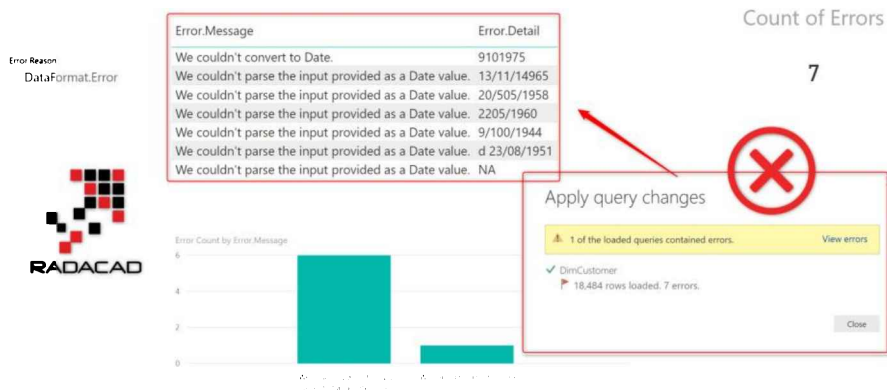
Summary

Setting up the incremental refresh in Power BI means loading only part of the data regularly and storing the consistent data. This process will make your refresh time much faster; however, there are some requirements for it. You need to have a date field in your table, and at the moment, there is a licensing requirement for Power BI premium. You can do this configuration on a Power BI dataset or in Power BI dataflows. If you do this on a dataset, then after publishing the dataset, you cannot download the PBIX file.

Part 6: Error handling, Exception reporting, and data profiling

Chapter 25: Exception Reporting in Power BI: Catch the Error Rows in Power Query

Exception Report: Error Rows

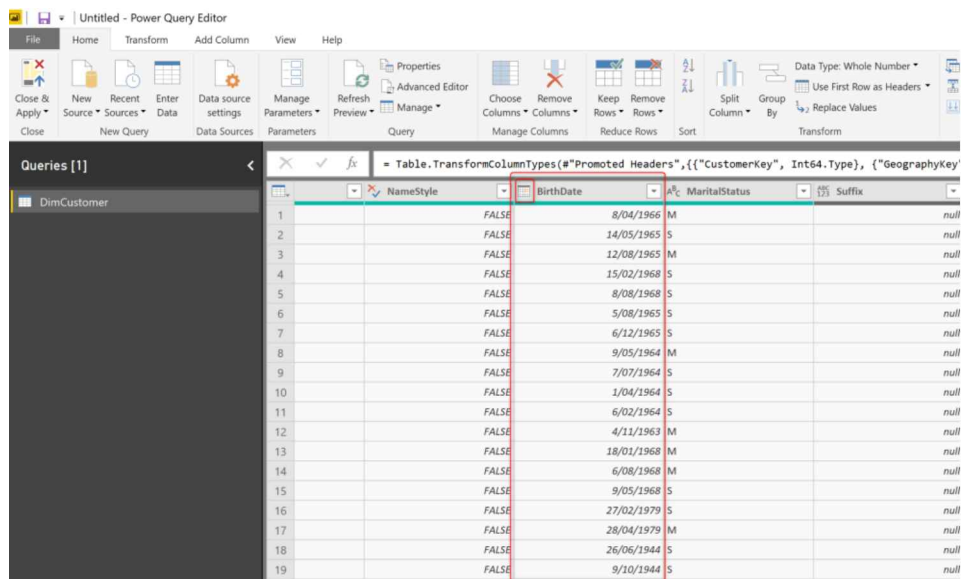


To build a robust BI system, you need to cater to errors and handle errors carefully. If you build a reporting solution that the refresh of that fails every time an error occurs, it is not a robust system. Errors can happen for many reasons. In this chapter, I'll show you how to catch potential errors in Power Query and build an exception report page to visualize the error rows for further investigation. The method that you learn here will save your model from failing at the time of refresh. This means you get the dataset updated, and you can catch any rows that caused the error in an exception report page.

Sample Dataset

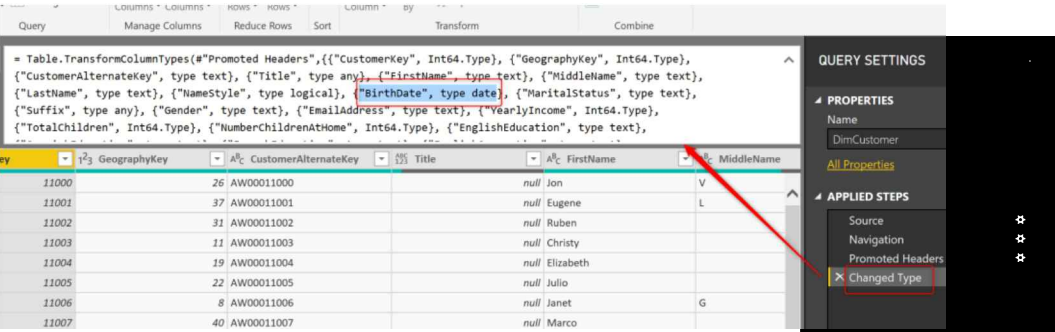
I will use a sample Excel file as a data source that has 18,484 customer rows in it. In the sample Dataset, we have a BirthDate field beside all other fields, which are supposed to have a date value. Here is what the data looks like when I bring it into Power Query:

Chapter 25: Exception Reporting in Power BI: Catch the Error Rows in Power Query



Error Happens

When I get this dataset in my Power Query Editor window (as you see in the above screenshot), Power Query automatically converts the data type of the BirthDate column to Date. You can see this automatic data type conversion in the list of Steps;



Of course, you can turn off this automatic data type detection of Power Query, but that is not my point. I want the dataset to fail to show you how to deal with it. Errors happen in Power Query in the real world, and I'm here to show you how to find them.

As you can see in the Power Query Editor, I see no errors for this data type change, and everything looks great;

Part 6: Error handling, Exception reporting, and data profiling

Power Query Editor interface showing the 'Transform' tab. The ribbon includes options like 'NameStyle', 'BirthDate', and 'MaritalStatus'. The 'Queries [1]' pane on the left shows 'DimCustomer'. The main area displays a data preview with columns 'NameStyle', 'BirthDate', and 'MaritalStatus'. The 'BirthDate' column has a data type of 'Date' and a 'Valid' status of 100%. The 'MaritalStatus' column has a data type of 'Text' and a 'Valid' status of 100%. The 'NameStyle' column has a data type of 'Text' and a 'Valid' status of 100%.

Now I load this dataset into Power BI by using Close and Apply in the Query Editor window, and I expect everything to load successfully. However, this is coming out of the blue!

Apply query changes

1 of the loaded queries contained errors. [View errors](#)

✓ DimCustomer
✗ 18,484 rows loaded. 7 errors.

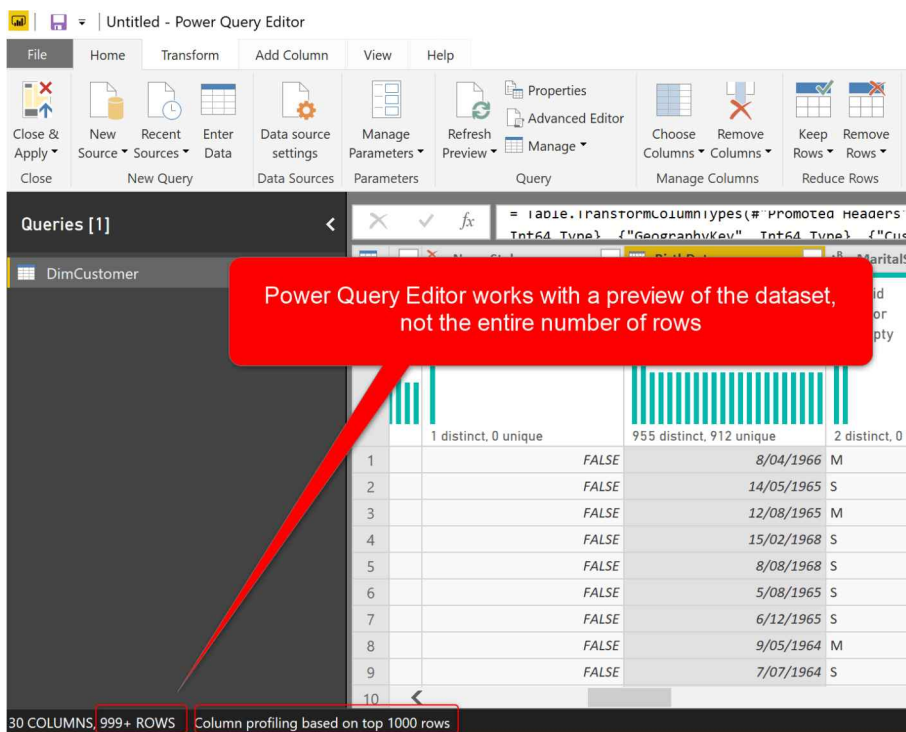
Close

Does this sound familiar? Yes, if you have worked with Power BI for a while, you might have experienced it. No errors in Power Query Editor, but when we load the data into Power BI, there are errors! How is that possible? Let's first find out why this happens.

Why didn't Power Query Editor Catch the Error?

Power Query Editor always works with a preview of the dataset. The size of the preview depends on how many columns you have, and sometimes it is 1000 rows, sometimes 200 rows. If you click on a Query in the Power Query editor window, you can, in fact, see this stated down below in the status bar;

Chapter 25: Exception Reporting in Power BI: Catch the Error Rows in Power Query



The reason for Power Query to use the preview dataset is mainly because of speeding up the transformation development process. Imagine if you have a table with 10 million rows, every single transformation you want to apply on that dataset would take a long time, and you have to wait for it before you start doing the next step. The wait for the response each time will slow down your development process. This is the reason why working on a preview on the dataset is a preferred option. You can apply all transformations you want on the preview, and when you are happy with it, then apply it to the entire dataset. Usually, the first 1000 rows or the first 200 rows are a good sample of the entire dataset, and you can expect to see most of the data challenges there. Usually, not always, of course.

How will the transformation be applied to the entire dataset then? When you load the data into Power BI. That means when you click on Close and APPLY in the Power Query Editor window. That APPLY means apply those transformations now on the entire dataset. That is the reason why the load process may take longer, especially if the dataset is big.

Power Query Editor always works with a preview of the data to make the development process fast. When you load the data in Power BI, transformations will be applied to the entire dataset.

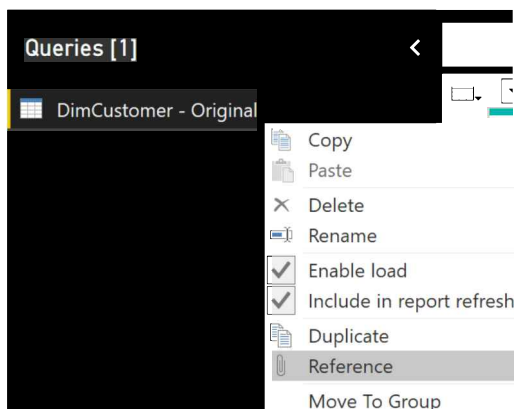
Now that you know how Power Query Editor deals with the data preview, you can guess why the error above happened? The reason is; The preview of the data (which was about 1000 rows) had no issues with the transformations applied (automatic data type change to Date for the BirthDate column). However, the entire dataset (which is about 18K rows) has problems with that transformation! When you see the error above in the Power BI Desktop, then you can click on View errors and go to Power Query editor and see those rows, and deal with them somehow, and fix them. However, that is not enough.

What if the error doesn't happen in Power BI Desktop but happens in a scheduled refresh in the Power BI Service?

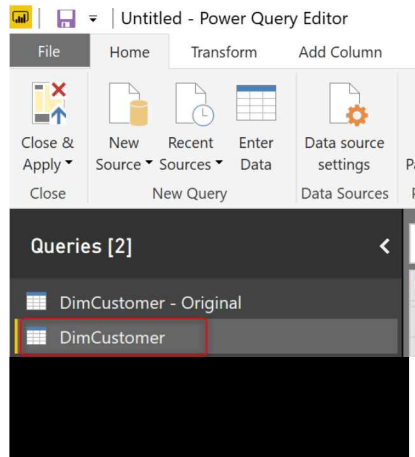
True! Fixing errors in Power BI Desktop is easy, but consider that the error didn't happen in the desktop too, and you got your Power BI report published to the website and scheduled it to refresh. Then the next day, you see the report failed to refresh with an error! You have to learn how to deal with the error rows beforehand before it causes the scheduled refresh to fail. Let's see how to deal with it then.

Dealing with Errors: Catching the Error Rows

To deal with errors, you have to catch the error before it loads into Power BI. One way to do it is to create two references of the same table, one as the final query and one as Error Rows.



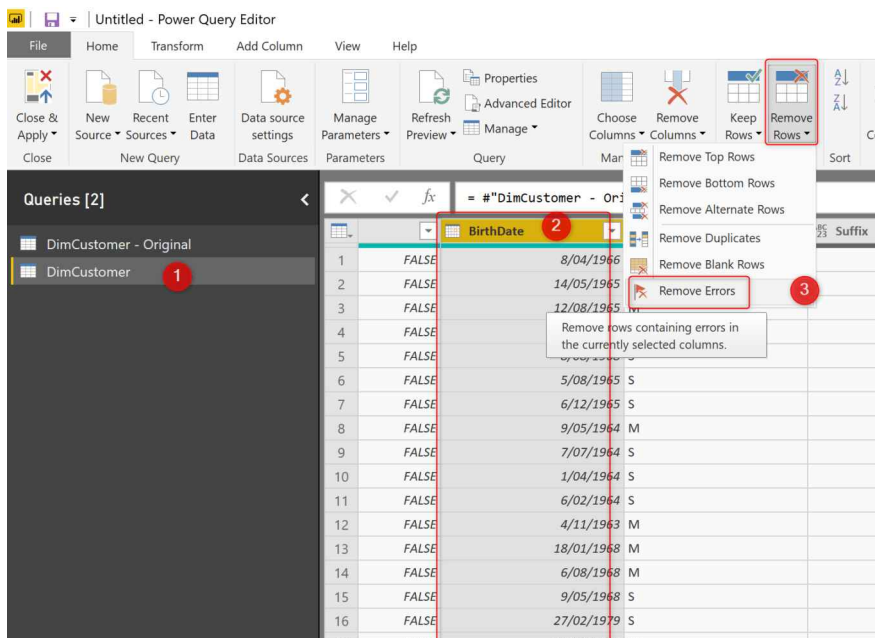
In the screenshot above, I renamed the DimCustomer table to DimCustomer – Original and then created a reference. If you would like to learn what Reference is, read my article about Reference vs. Duplicate earlier in this book. The new referenced query can be called as DimCustomer. This would be the clean query with no errors (we will remove errors from it in the next step);



The new table is the table that would be clean with no errors, and we can use it in the report. Let's clean this from any errors

Remove Errors from the Table loading into Power BI

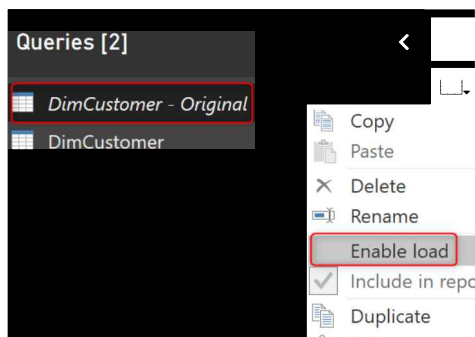
As DimCustomer would be the final query for us, I want to remove errors from it. Removing errors is a simple option in the Home tab, under Reduce Rows -> Remove Rows -> Remove Errors. Make sure that you select the BirthDate column before that.



You can also do this for all columns if you want; by selecting all columns and then using Remove Errors. This chapter is just a sample on one column and can be extended to all for sure.

Part 6: Error handling, Exception reporting, and data profiling

Remove Errors will be a step in the data transformations step, and it means that when you click on APPLY, it will apply to the entire dataset, so as a result when the data type change causes an error, the next step after that which is Remove Errors, will wipe the rows that caused the error. But the DimCustomer – Original still may cause the error, so we have to uncheck the Enable Load of that query.



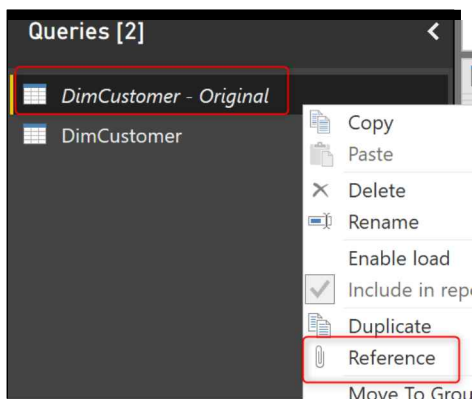
Now we have successfully removed the errors, and if we load the data into Power BI. There would be no errors happening.

But wait! what about those error rows? How can we catch them? We need to catch those rows and investigate what happened, and think about an action plan to fix them, right? So, we need another query reference from the original query to keep the error rows.

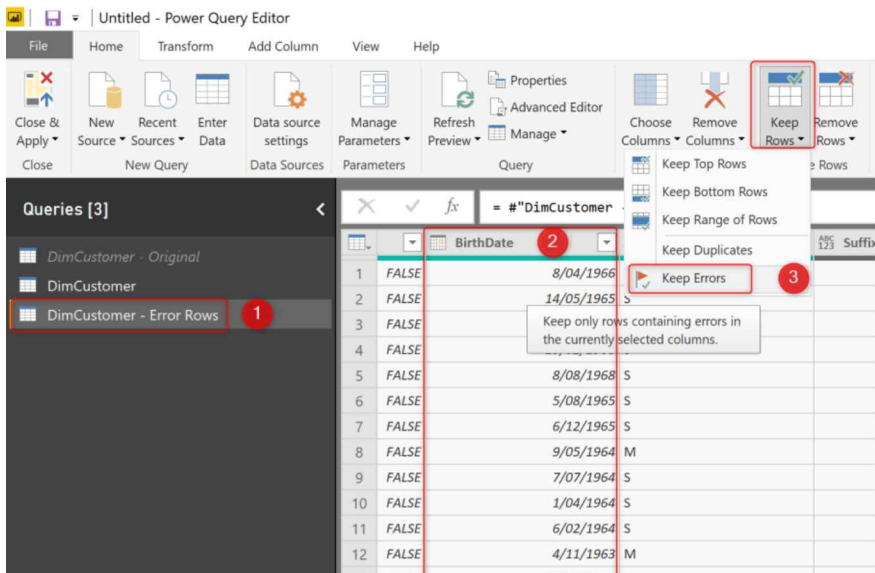
Keep Errors in the Exception Table

Similar to the Remove Errors option, there is also an option to Keep Errors. If you have seen this option before, you may have wondered what the use of such a thing is? Well, here is the exact use case scenario. Keep Errors will help to catch the error rows in an exception table.

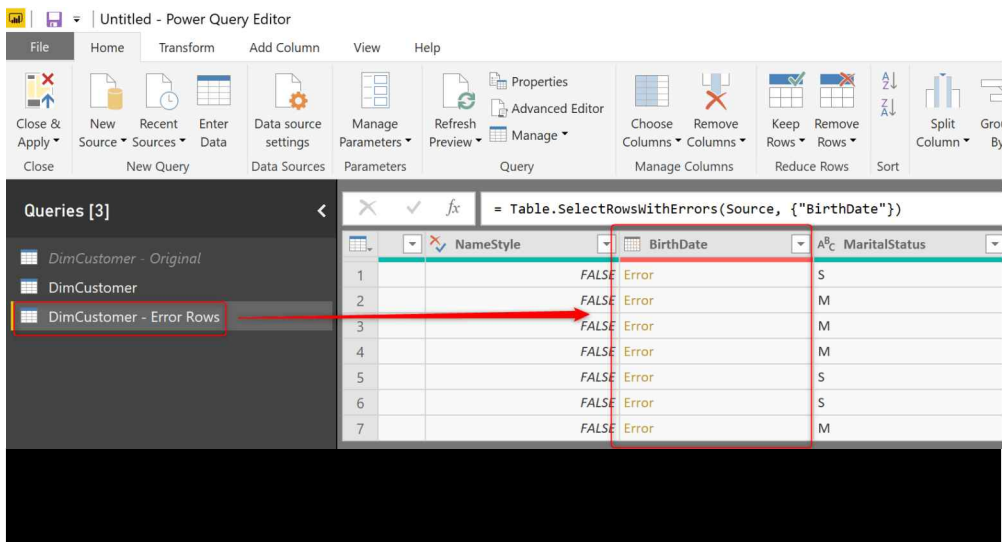
Create another reference from the DimCustomer – Original.



Rename this new query as DimCustomer Error Rows. For this query, we have to Keep Errors, which can be found close to where the Remove Errors is, but under Keep Rows.



Now this table would only keep rows that cause an error. Here is a sample set;

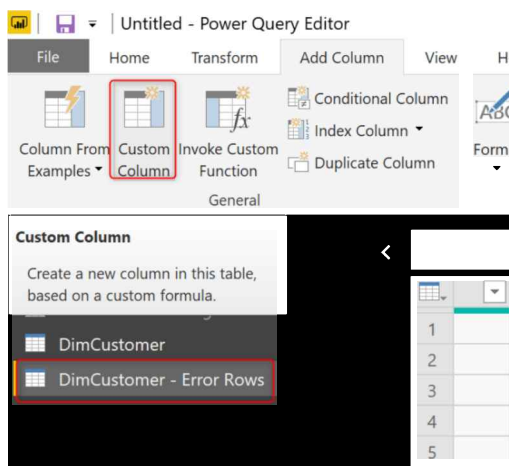


This is not the end of the story. If you load this new table of DimCustomer – Error Rows in Power BI, you will end up with the same error again. Why? well, because this query is going to return error rows! You need to remove the Error that occurred from this dataset.

Getting Error Details

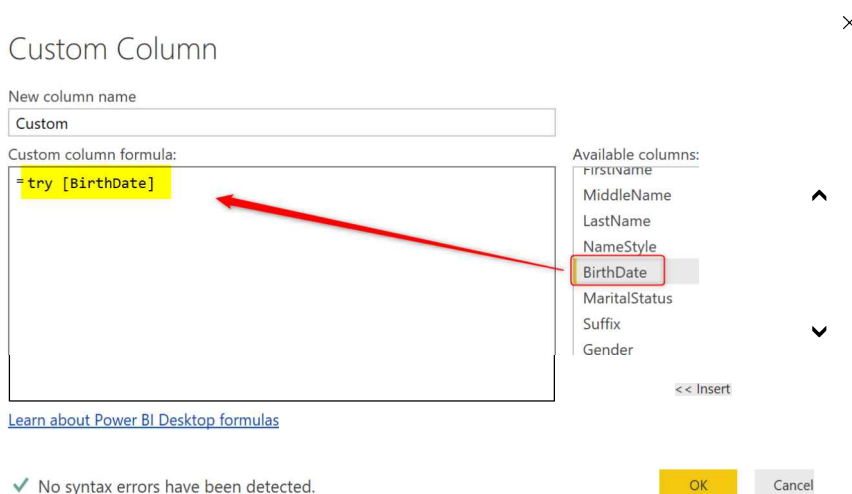
If you remove the error column from the exception table we have created, you would have no details about the error that happened, and it would be hard to track it back and troubleshoot. The best would be catching the error details. The error message and the value that caused the error are important details that you don't want to miss. Follow the steps below to get that information.

In the Error Rows table, add a Custom Column.



In the Custom Column editor, write **try** and then after a space, name of the field that caused the error. In our example: BirthDate;

try [BirthDate]



Try (all lowercase) is a keyword in M that will catch the error details. Instead of returning just an error, it will return a record containing the error details, such as the source value and the error message. The below screenshot shows how the output of *try* would come;

Table.AddColumn("#Kept Errors", "Custom", each try [BirthDate])

| | dressLine2 | Phone | DateFirstPurchase | CommuteDistance | FullName | Custom |
|---|------------|---------------------|-------------------|-----------------|------------------|--------|
| 1 | null | 161-555-0125 | 19/09/2007 | 10+ Miles | John White | Record |
| 2 | null | 334-555-0173 | 23/08/2007 | 10+ Miles | Nathan Hughes | Record |
| 3 | null | 178-555-0116 | 19/05/2006 | 2-5 Miles | Logan Turner | Record |
| 4 | null | 1 (11) 500 555-0166 | 11/09/2007 | 2-5 Miles | Morgan Sanchez | Record |
| 5 | null | 1 (11) 500 555-0190 | 7/06/2006 | 5-10 Miles | Dustin Goldstein | Record |
| 6 | null | 391-555-0176 | 28/06/2006 | 0-1 Miles | Hailey Cox | Record |
| 7 | null | 1 (11) 500 555-0118 | 13/10/2007 | 0-1 Miles | Ian Bell | Record |

HasError: TRUE
Error: Record

The Record output of the “try” will have two fields; HasError (which we already know will be true) and the Error. The Error is another record with more details. Click on Expand on the Custom column, and just select Error.

Search Columns to Expand

☒ (Select All Columns)

☐ HasError

☒ Error

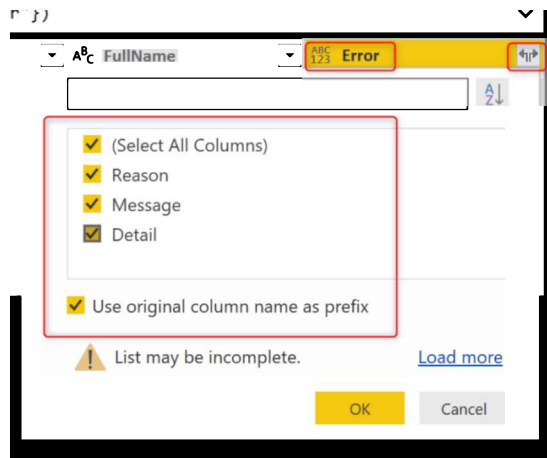
☐ Use original column name as prefix

List may be incomplete. [Load more](#)

OK Cancel

In the output column named Error, click on Expand again and this time select all columns;

Part 6: Error handling, Exception reporting, and data profiling



It is good to have the original column name as a prefix because you would know that these are error detail columns.

Now you would get the full details of the error as below;

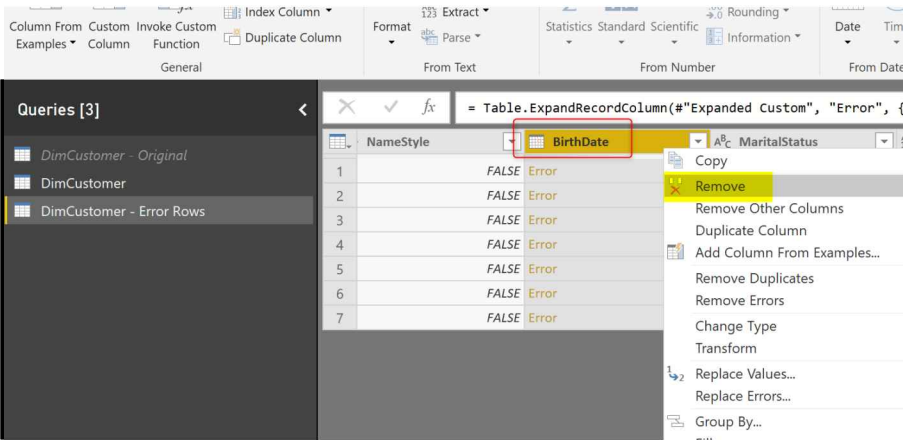
'Expanded Custom', "Error", {"Reason", "Message", "Detail"}, {"Error.Reason",

| Error.Reason | Error.Message | Error.Detail |
|------------------|---|--------------|
| DataFormat.Error | We couldn't parse the input provided as a Date value. | 9/100/1944 |
| DataFormat.Error | We couldn't parse the input provided as a Date value. | d 23/08/1951 |
| DataFormat.Error | We couldn't parse the input provided as a Date value. | 13/11/14965 |
| DataFormat.Error | We couldn't parse the input provided as a Date value. | 2205/1960 |
| DataFormat.Error | We couldn't convert to Date. | 9101975 |
| DataFormat.Error | We couldn't parse the input provided as a Date value. | 20/505/1958 |
| DataFormat.Error | We couldn't parse the input provided as a Date value. | NA |

The information above is your most valuable asset for exception reporting.

Remove Error Column

The last step before loading the data into Power BI is to remove the column that causes the Error. In our example; the BirthDate Column should be removed (otherwise, the refresh will fail again);

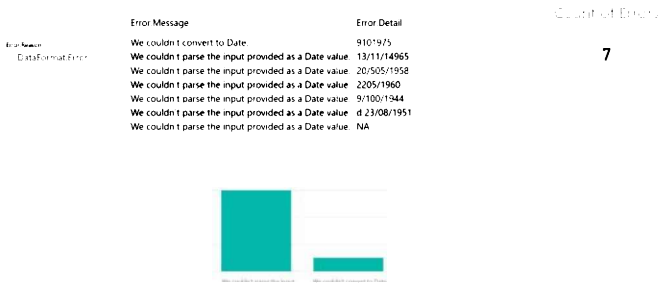


Exception Report

You can now load the data into Power BI. You will have two tables; DimCustomer, and DimCustomer – Error Rows. DimCustomer is the table that you can use for your normal reporting. DimCustomer – Error Rows is the table that you can use for exception reporting. The exception report is the report that can be used for troubleshooting and will list all the errors to users for further investigation. Make sure that there is no relationship between these two tables.

Here is a sample report visual I created that shows the errors;

Exception Report: Error Rows

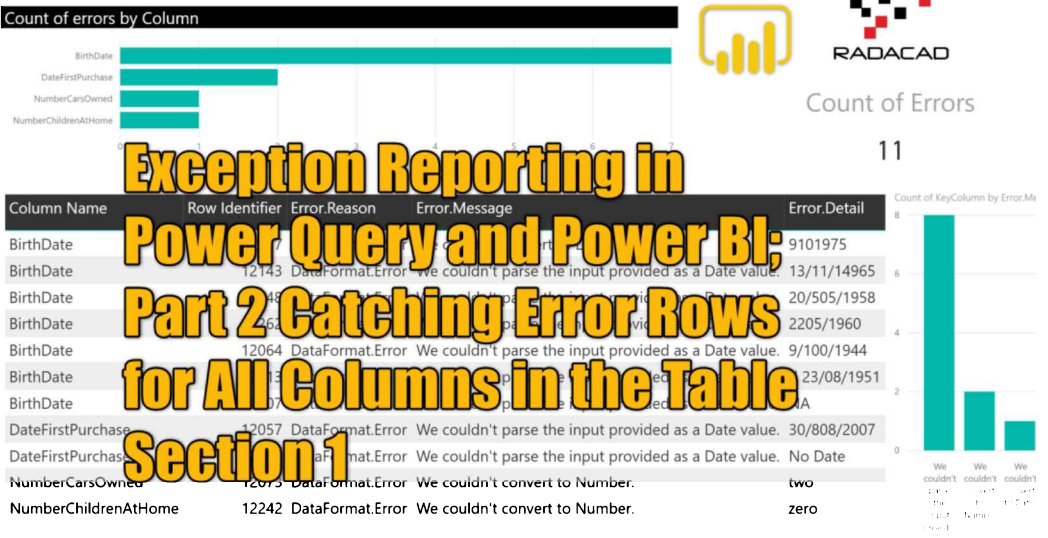


Summary

Errors happen, and you have to deal with them. Instead of waiting for the error to happen and then finding it a month after it caused, it is better to catch them as soon as they happen. In this chapter, you learned a way to deal with error rows. In RADACAD, we always create an exception report for Power BI reports. That way, we are always sure that the refresh won't fail because of the error, and we would also have a place for investigating the errors, which is called the Exception Report.

Chapter 26: Exception Reporting in Power Query and Power BI; Part 2 Catching Error Rows for All Columns in the Table

Exception Report: showing all errors happened across all columns



In the previous chapter, you learned about using Power Query transformations and functions to create an Exception report for Power BI. However, the method I mentioned there was only doing the exception handling based on one column. In this chapter, I explain how to do exception reporting for all columns in the entire table. There are a few more steps and learnings when you do the whole process for the entire table.

Prerequisite

It would be beneficial to read the previous chapter. You can download the dataset used for this from this book's code files.

Introduction

The Exception Reporting method that you learned in the previous chapter focused on handling errors for one column. That method works best if you already know which column is causing the error. However, in many scenarios, you don't know which column caused (or will cause) the error. So, this chapter then aims for that scenario.

Important Note: Exception handling is a process that makes your Refresh time a bit longer. It is faster to handle exceptions for one column than doing it for all columns in the entire table. I strongly suggest using the method mentioned in Part 1 if you already know the column prone to error in the table.

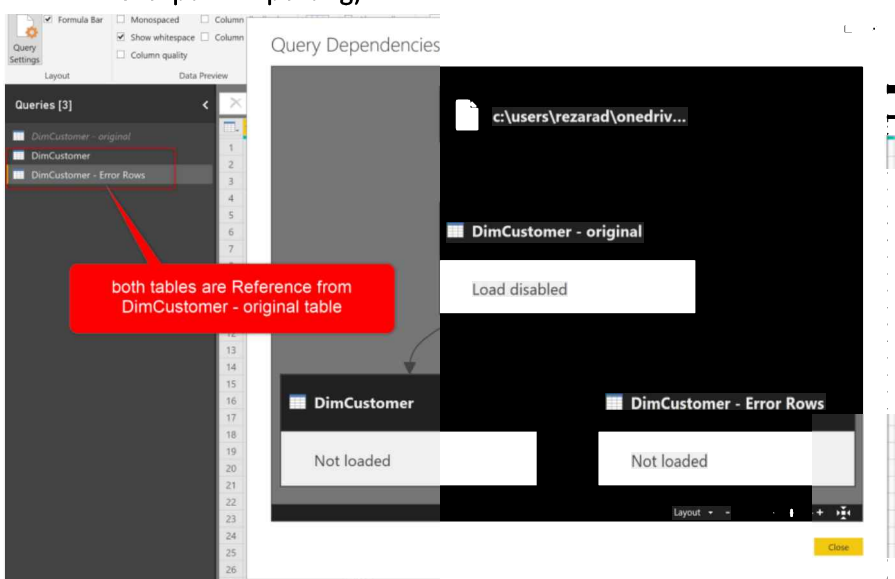
Starting from basics

In the first part of this chapter, I mentioned that to have proper error handling; you would need to have three tables; One table, which is the original source, another table referenced from the original, and error rows are removed. And the last table (which we call the error table) with error rows kept. Now let's start with that plan.

If you haven't done part 1, you can start by Get Data from the file mentioned above, then click on Edit to go to Power Query Editor, and then rename the DimCustomer, to DimCustomer – original. Disable the load for this table by right-clicking and uncheck the Enable load.

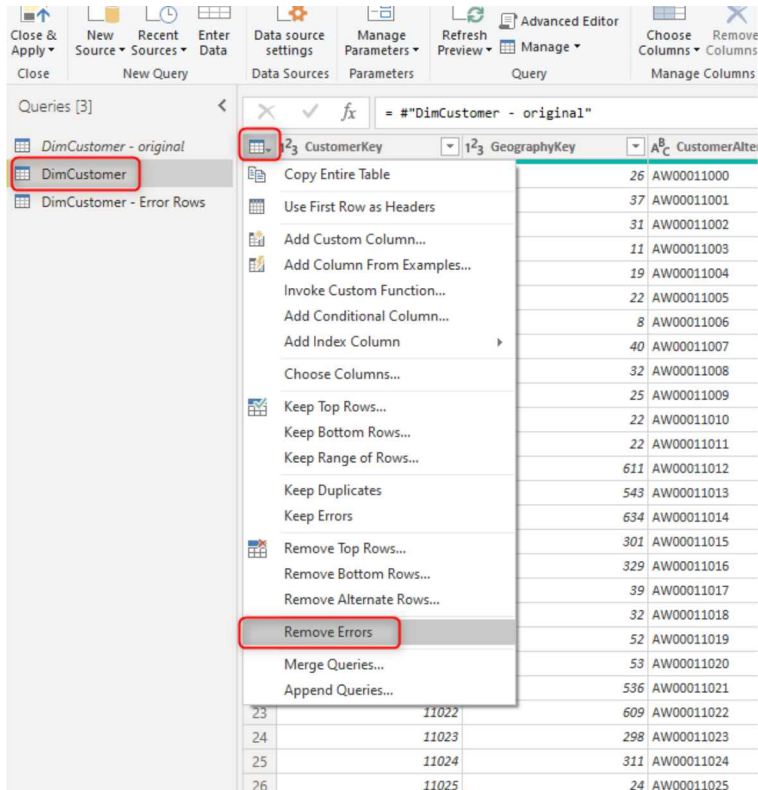
DimCustomer – original is the table that is sourced from the data source and has some data transformations which might cause an error. This table is not loading into Power BI (Enable load is unchecked). Start by creating two references from this table:

- DimCustomer (a table which will be used in normal reporting in Power BI)
- DimCustomer – Error rows (the table which will be used as the source of exception reporting).



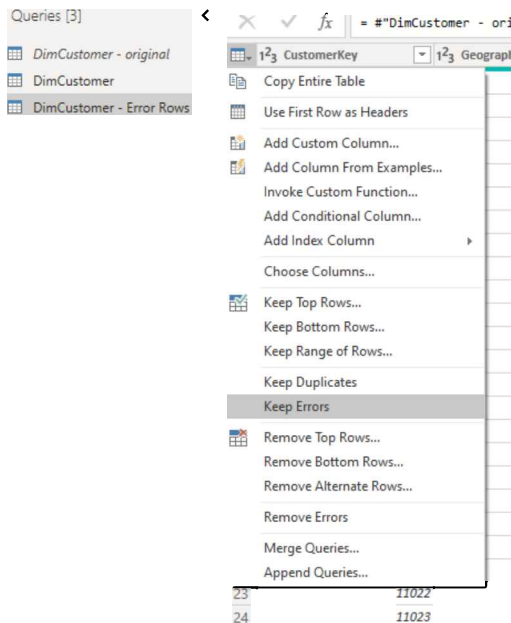
Remove rows with Errors

The DimCustomer table is the table with no errors that we use in normal reporting. To remove errors rows, you need to select the Remove Errors from the Table selector, as it showed in the screenshot below. This way, the Remove errors apply to all columns of the table.



Keep Rows with Errors

For DimCustomer – Error Rows table, from the table selector, select Keep Errors.



Now this table has all error rows, which might have occurred in any of the columns. Our next step is to find in each row; what is the column that caused an error. Because not all column's caused an error in each row. To be able to find it and then point at it, we need a row identifier. A way to be able to distinguish that row later on. We need a Key Column for the table.

Note: Key Column can be multiple columns as well, but for simplicity, I keep it as one for this example.

Key Column: the row identifier

To find the row that has the error, later on, we need to have a key column. And we need to make sure that the key column has no errors itself; otherwise, how you are going to find out which row has the issue. Let's assume for this example that the key column is CustomerKey. Let's rename it to make it easier to use later on. You can simply rename the column with right-click or double-click on the column name.

A screenshot of the Power Query interface. The formula bar at the top shows the formula: `Table.RenameColumns(#"Kept Errors",{{"CustomerKey", "KeyColumn"}})`. Below the formula bar, the table header is visible with columns: 'KeyColumn', 'GeographyKey', 'CustomerAlternateKey', and 'Title'. The 'KeyColumn' column is highlighted in yellow. The table contains 7 rows of data.

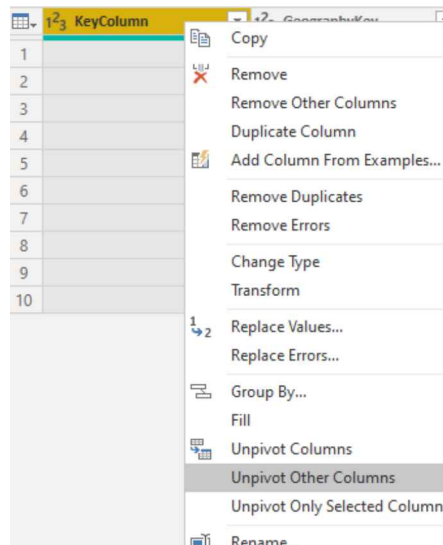
| | KeyColumn | GeographyKey | CustomerAlternateKey | Title |
|---|-----------|--------------|----------------------|-------|
| 1 | 12057 | 627 | AW00012057 | |
| 2 | 12064 | 539 | AW00012064 | |
| 3 | 12073 | 334 | AW00012073 | |
| 4 | 12111 | 322 | AW00012111 | |
| 5 | 12113 | 51 | AW00012113 | |
| 6 | 12143 | 632 | AW00012143 | |
| 7 | 12242 | 29 | AW00012242 | |

Unpivot: Columns as Rows

Now we need to have columns in a structure like below:

| KeyColumn | ColumnName | Value |
|-----------|------------|----------|
| 1 | Firstname | Reza |
| 1 | Lastname | Rad |
| 1 | Age | error |
| 1 | city | Auckland |
| 2 | Firstname | Leila |
| 2 | Lastname | Etaati |

Instead of having columns, we are going to have rows, and then we can filter only for those that their value is an error. One of the best transformations for this job is Unpivot. Right-click on the KeyColumn and Unpivot other columns.



And the result now looks like this;

Chapter 26: Exception Reporting in Power Query and Power BI; Part 2 Catching Error Rows for All Columns in the Table

Queries [3] fx = Table.UnpivotOtherColumns(#"Renamed Columns", {"KeyColumn"}, {"Attribute", "Value"})

| | KeyColumn | Attribute | Value |
|----|-----------|----------------------|----------------------------|
| 1 | 12064 | GeographyKey | 539 |
| 2 | 12064 | CustomerAlternateKey | AW00012064 |
| 3 | 12064 | FirstName | John |
| 4 | 12064 | MiddleName | D |
| 5 | 12064 | LastName | White |
| 6 | 12064 | NameStyle | FALSE |
| 7 | 12064 | BirthDate | Error |
| 8 | 12064 | MaritalStatus | S |
| 9 | 12064 | Gender | M |
| 10 | 12064 | EmailAddress | john50@adventure-works.com |
| 11 | 12064 | YearlyIncome | 70000 |
| 12 | 12064 | TotalChildren | 4 |
| 13 | 12064 | NumberChildrenAtHome | 0 |
| 14 | 12064 | EnglishEducation | Bachelors |
| 15 | 12064 | SpanishEducation | Licenciatura |
| 16 | 12064 | FrenchEducation | Bac + 4 |
| 17 | 12064 | EnglishOccupation | Management |
| 18 | 12064 | SpanishOccupation | Gestión |
| 19 | 12064 | FrenchOccupation | Direction |
| 20 | 12064 | HouseOwnerFlag | 1 |
| 21 | 12064 | NumberCarsOwned | 2 |
| 22 | 12064 | AddressLine1 | 244 La Cadena |
| 23 | 12064 | Phone | 161-555-0125 |
| 24 | 12064 | DateFirstPurchase | 19/09/2007 |
| 25 | 12064 | CommuteDistance | 10+ Miles |
| 26 | 12064 | FullName | John White |
| 27 | 12113 | GeographyKey | 51 |
| 28 | 12113 | CustomerAlternateKey | AW00012113 |
| 29 | 12113 | FirstName | Nathan |
| 30 | 12113 | MiddleName | B |
| 31 | 12113 | LastName | Hughes |
| 32 | 12113 | NameStyle | FALSE |

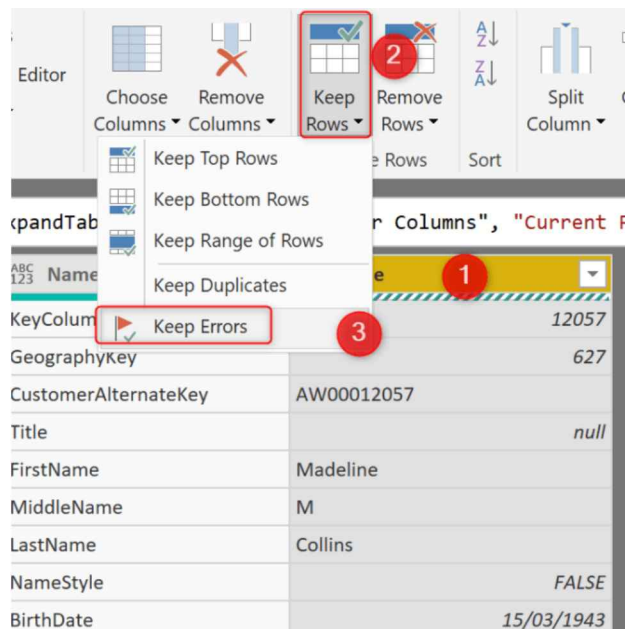
The first column in this table is the KeyColumn for each row. Now for each row in the previous table structure, we have multiple rows here in this new structure, one row per column. In each row, in addition to the KeyColumn, you can see the column Attribute (Name) and the row value in that cell (Value). And you can see that that row value sometimes has an error.

Keep Errors

Remember that we want only to keep those columns that caused an error. Now our columns are rows, so we need to keep only those rows that caused an error. And the error only can occur in the Value column because that is where the cell value exists. Select the Value column and then Keep Rows -> Keep Errors.



Part 6: Error handling, Exception reporting, and data profiling



The output table is a list of all rows with the only columns that caused an error! You may notice that I renamed the Attribute column to Name.

✓ fx = Table.SelectRowsWithErrors("#Expanded Current Row as a Table", {"Value"})

| | KeyColumn | Name | Value |
|----|-----------|----------------------|-------|
| 1 | 12057 | DateFirstPurchase | Error |
| 2 | 12064 | BirthDate | Error |
| 3 | 12073 | NumberCarsOwned | Error |
| 4 | 12111 | DateFirstPurchase | Error |
| 5 | 12113 | BirthDate | Error |
| 6 | 12143 | BirthDate | Error |
| 7 | 12242 | NumberChildrenAtHome | Error |
| 8 | 12262 | BirthDate | Error |
| 9 | 12337 | BirthDate | Error |
| 10 | 12448 | BirthDate | Error |
| 11 | 12507 | BirthDate | Error |

Get the error details: try

Congratulations! You have done the hard part. Now the rest is as it was in part 1. You need to get the error details, and adding a custom column with the try as the expression will give you that:

try [Value]

Chapter 26: Exception Reporting in Power Query and Power BI; Part 2 Catching Error Rows for All Columns in the Table

Custom Column

Add a column that is computed from the other columns.

New column name

Custom

Custom column formula

= try [Value]

Available columns

KeyColumn

Name

Value

[Learn about Power BI Desktop formulas](#)

<< Insert

✓ No syntax errors have been detected.

OK

Cancel

The new column will have a Record with error details in each row;

Table.AddColumn(#"Kept Errors1", "Custom", each try [Value])

| 123 | KeyColumn | ABC 123 | Name | ABC 123 | Value | ABC 123 | Custom |
|-----|-----------|---------|----------------------|---------|-------|---------|--------|
| 1 | | 12057 | DateFirstPurchase | Error | | | Record |
| 2 | | 12064 | BirthDate | Error | | | Record |
| 3 | | 12073 | NumberCarsOwned | Error | | | Record |
| 4 | | 12111 | DateFirstPurchase | Error | | | Record |
| 5 | | 12113 | BirthDate | Error | | | Record |
| 6 | | 12143 | BirthDate | Error | | | Record |
| 7 | | 12242 | NumberChildrenAtHome | Error | | | Record |
| 8 | | 12262 | BirthDate | Error | | | Record |
| 9 | | 12337 | BirthDate | Error | | | Record |
| 10 | | 12448 | BirthDate | Error | | | Record |
| 11 | | 12507 | BirthDate | Error | | | Record |

The new column, after expanding, have two columns inside, which we are only interested in the Error column;

Value Custom

Search Columns to Expand

(Select All Columns)

☐ HasError

☒ Error

☐ Use original column name as prefix

⚠ List may be incomplete.

[Load more](#)

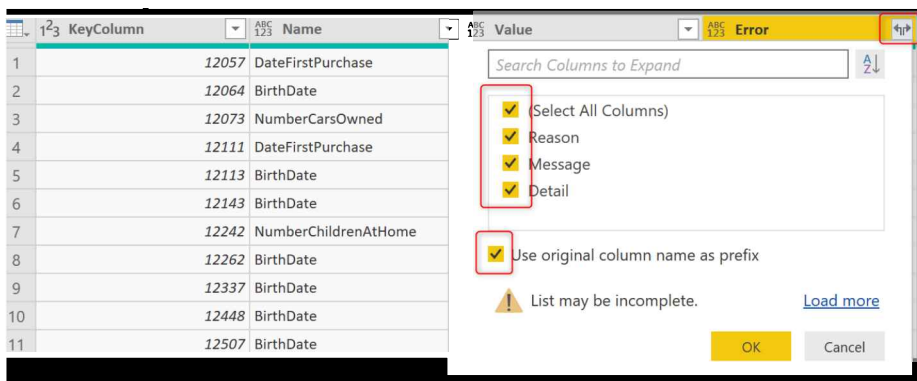
OK

Cancel

As the next step, expand the Error column to get the three-column output of the error details:



Part 6: Error handling, Exception reporting, and data profiling



And here is the output:

| | Value | Error.Reason | Error.Message | Error.Detail |
|----|-------|--------------|---|--------------|
| 1 | Error | | We couldn't parse the input provided as a Date value. | 30/808/2007 |
| 2 | Error | | We couldn't parse the input provided as a Date value. | 9/100/1944 |
| 3 | Error | | We couldn't convert to Number. | two |
| 4 | Error | | We couldn't parse the input provided as a Date value. | No Date |
| 5 | Error | | We couldn't parse the input provided as a Date value. | d 23/08/1951 |
| 6 | Error | | We couldn't parse the input provided as a Date value. | 13/11/14965 |
| 7 | Error | | We couldn't convert to Number. | zero |
| 8 | Error | | We couldn't parse the input provided as a Date value. | 2205/1960 |
| 9 | Error | | We couldn't convert to Date. | 9101975 |
| 10 | Error | | We couldn't parse the input provided as a Date value. | 20/505/1958 |
| 11 | Error | | We couldn't parse the input provided as a Date value. | NA |

And as the final step, remove the column with errors in it, which is the Value column. (If you don't do that step, you will get an error when you load data into Power BI).

I have not explained the process of using **"try"** and getting the error details because I explained it in detail in the previous chapter.

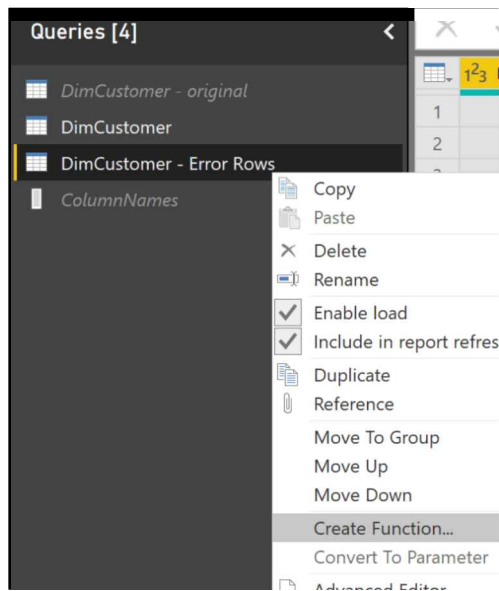
A Custom Function: Re-use your code

To get the best output, I want to create a function from this process. A function that gets the table as the input, and also the KeyColumn name, and gives me all error rows happening across all columns as a table output.

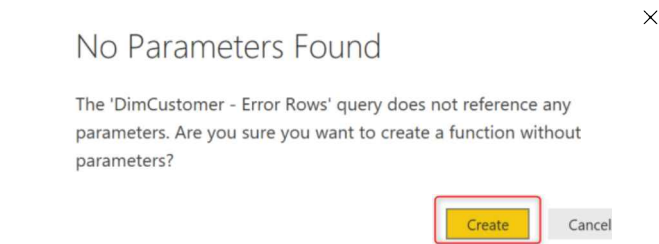
To create a function like that, you would need a parameter of the type table. However, using the graphical interface of Power Query, at the time of writing this chapter, it is not possible to create a parameter of type table. I show you another easy way to do it;

Create a function with no parameters!

The first step is to create a function from what you did in DimCustomer – Error Rows. Right-click on that table and then create a function;



You will get a warning that this function you are about to create doesn't have any input parameters. Are you fine with that? Confirm it by clicking on Create.



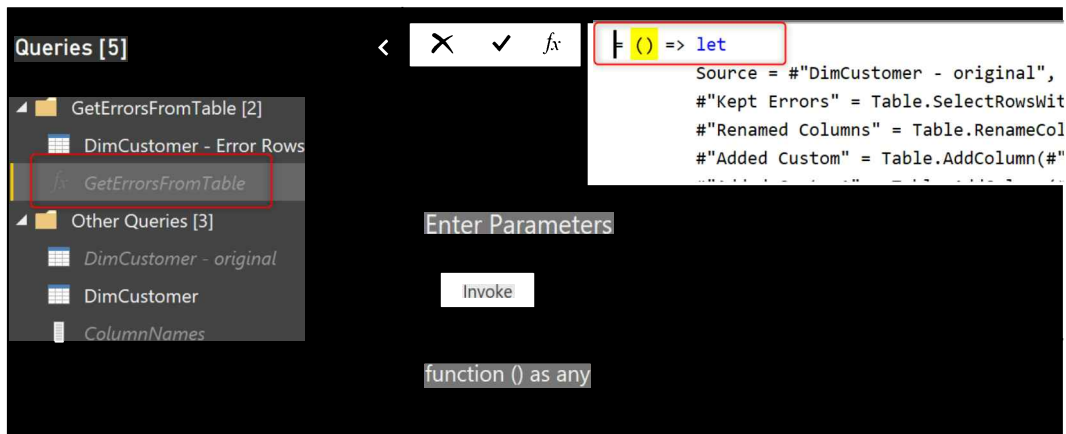
Let's call the function as GetErrorsFromTable (function name can have space too, but it would make our code much cleaner if we avoid it);



Add parameters to the function

Now that the function is created, click on the function on the left-hand side and expand the formula bar;

Part 6: Error handling, Exception reporting, and data profiling



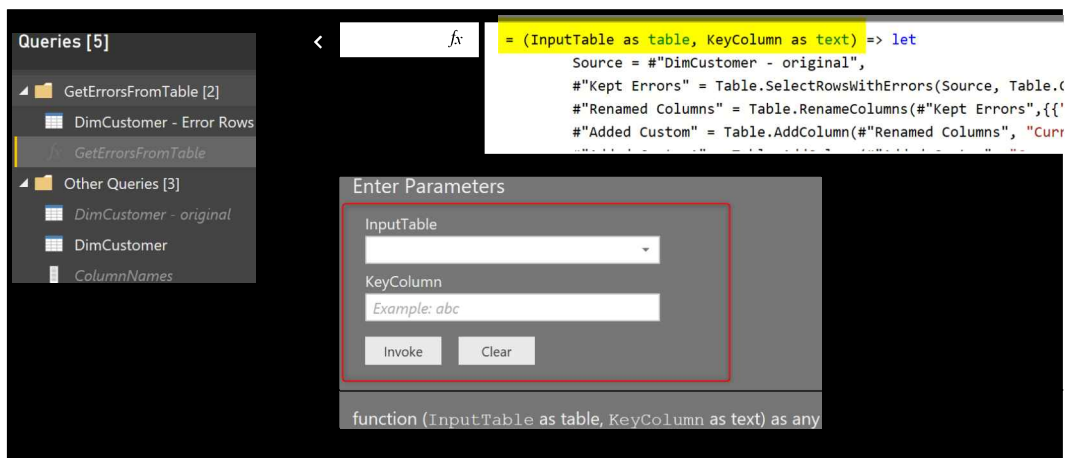
The very first line of the code in the formula bar and, more specifically, the section inside the parenthesis is where we should change. That is the place that parameters of a function are set.

To add parameters to the existing function, add them in this syntax:

(<parameter> as <data type>, <parameter> as <data type>, ...)

So the change in the first line would be:

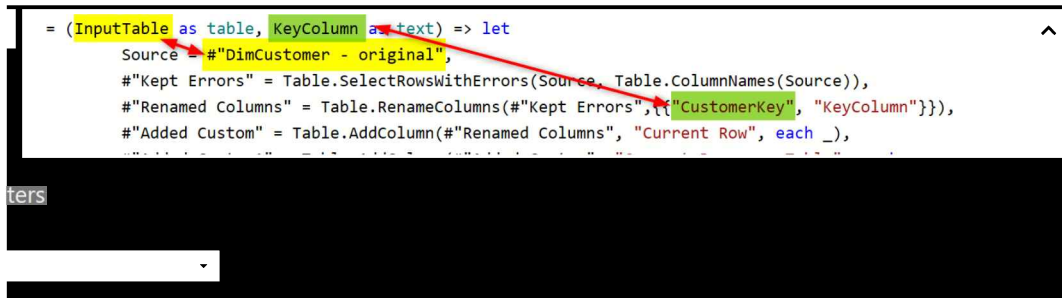
= (InputTable as table, KeyColumn as text) => let



After making that change, click on the checkbox beside the formula bar, and you get the function UI changing now with accepting two parameters.

Using the parameters in the code

You need to use both parameters in the code because otherwise, there is no point in having parameters. The screenshot below shows where in the code should change with their respective parameters names;



Here is the entire code of the function now:

```
= (InputTable as table, KeyColumn as text) => let
Source = InputTable,
#"Kept Errors" = Table.SelectRowsWithErrors(Source, Table.ColumnNames(Source)),
#"Renamed Columns" = Table.RenameColumns(#"Kept Errors",{{KeyColumn, "KeyColumn"}}),
#"Added Custom" = Table.AddColumn(#"Renamed Columns", "Current Row", each _),
#"Added Custom1" = Table.AddColumn(#"Added Custom", "Current Row as a Table", each
Record.ToTable([Current Row])),
#"Removed Other Columns" = Table.SelectColumns(#"Added Custom1",{"KeyColumn", "Current
Row as a Table"}),
#"Expanded Current Row as a Table" = Table.ExpandTableColumn(#"Removed Other Columns",
"Current Row as a Table", {"Name", "Value"}, {"Name", "Value"}),
#"Kept Errors1" = Table.SelectRowsWithErrors(#"Expanded Current Row as a Table", {"Value"}),
#"Added Custom2" = Table.AddColumn(#"Kept Errors1", "Custom", each try [Value]),
#"Expanded Custom" = Table.ExpandRecordColumn(#"Added Custom2", "Custom", {"Error"},
{"Error"}),
#"Expanded Error" = Table.ExpandRecordColumn(#"Expanded Custom", "Error", {"Reason",
"Message", "Detail"}, {"Error.Reason", "Error.Message", "Error.Detail"}),
#"Removed Columns" = Table.RemoveColumns(#"Expanded Error",{"Value"})
in
#"Removed Columns"
```

After making this change, you will see a message like below; confirm it by clicking on OK.

Edit Function

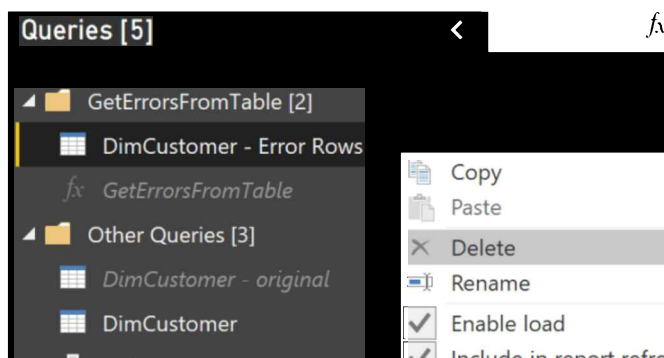
The definition of function 'GetErrorsFromTable' is updated whenever query 'DimCustomer - Error Rows' is updated. However, updates will stop if you directly modify function 'GetErrorsFromTable'. Are you sure you want to continue?

OK

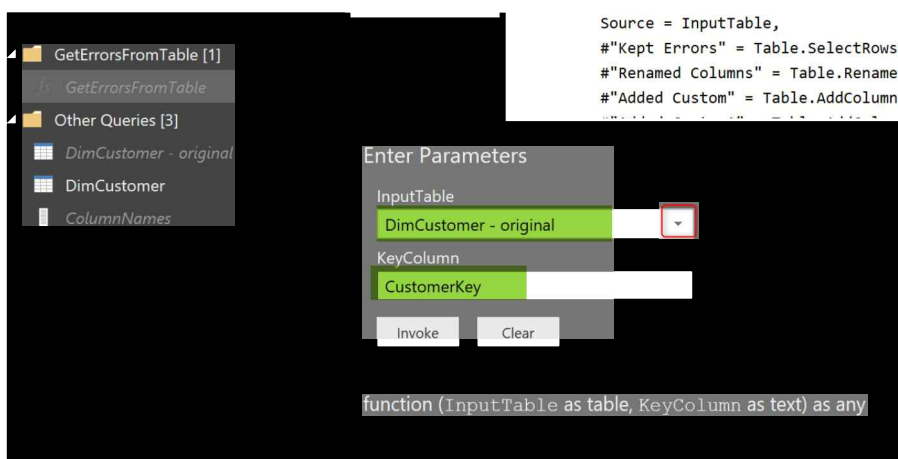
Cancel

Calling the custom function

Now that you have created the function, you can easily call it and get the error output. We won't need the DimCustomer – Error Rows anymore to be calculated separately. Having that in the solution means we will have two instances of the same code for detecting the error rows, which cause higher maintenance. Let's remove DimCustomer – Error Rows;



And we can create it this time by calling the GetErrorsFromTable function. Click on the function and set the parameters as below;



You have to select the table in the InputTable drop-down list, and the KeyColumn is a case-sensitive name. The output would be as below:

Queries [5]

- GetErrorsFromTable [1]
 - GetErrorsFromTable
- Other Queries [4]
 - DimCustomer - original
 - DimCustomer
 - ColumnNames
 - Invoked Function**

fx = GetErrorsFromTable("#DimCustomer - original", "CustomerKey")

| | KeyColumn | Name | Error.Reason | Error.Message |
|----|-----------|----------------------|------------------|-----------------------------|
| 1 | 12057 | DateFirstPurchase | DataFormat.Error | We couldn't parse the input |
| 2 | 12064 | BirthDate | DataFormat.Error | We couldn't parse the input |
| 3 | 12073 | NumberCarsOwned | DataFormat.Error | We couldn't convert to N |
| 4 | 12111 | DateFirstPurchase | DataFormat.Error | We couldn't parse the input |
| 5 | 12113 | BirthDate | DataFormat.Error | We couldn't parse the input |
| 6 | 12143 | BirthDate | DataFormat.Error | We couldn't parse the input |
| 7 | 12242 | NumberChildrenAtHome | DataFormat.Error | We couldn't convert to N |
| 8 | 12262 | BirthDate | DataFormat.Error | We couldn't parse the input |
| 9 | 12337 | BirthDate | DataFormat.Error | We couldn't convert to D |
| 10 | 12448 | BirthDate | DataFormat.Error | We couldn't parse the input |
| 11 | 12507 | BirthDate | DataFormat.Error | We couldn't parse the input |

By just calling this function, we get all error rows and error columns in the table. This Invoked Function name can now be renamed to DimCustomer – Error Rows.

- Queries [5]
 - GetErrorsFromTable [1]
 - GetErrorsFromTable*
 - Other Queries [4]
 - DimCustomer - original*
 - DimCustomer
 - ColumnNames*
 - DimCustomer - Error Rows

After doing the back-end work, you can now build a report page with some visualizations that serve as an exception report.

Note that in the table output below, the KeyColumn is a row identifier, and the Name is Column Name that caused an error.

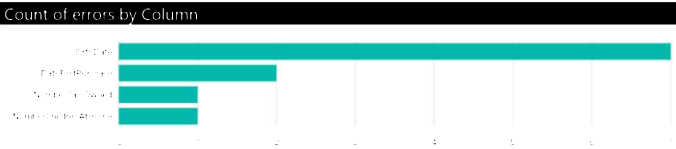
The screenshot displays the Power BI Desktop interface. In the top center, the Formula Bar contains the DAX query: `GetErrorsFromTable("DimCustomer - original", "CustomerKey")`. Below the Formula Bar, the Data View shows the results of this query as a table with 11 rows. The table has five columns: **KeyColumn**, **Name**, **ErrorReason**, **ErrorMessage**, and **ErrorDetail**. The data rows show various errors, primarily related to date parsing and data format issues. For example, the first row shows an error for 'DateFirstPurchase' with the message 'We couldn't parse the input provided as a Date value' and the error detail '30/08/2007'. The last row shows an error for 'BirthDate' with the message 'We couldn't parse the input provided as a Date value' and the error detail 'NA'.

| KeyColumn | Name | ErrorReason | ErrorMessage | ErrorDetail |
|-----------|----------------------------|-------------------|--|--------------|
| 1 | 12057 DateFirstPurchase | DataFormat: Error | We couldn't parse the input provided as a Date value | 30/08/2007 |
| 2 | 12064 BirthDate | DataFormat: Error | We couldn't parse the input provided as a Date value | 9/100/1944 |
| 3 | 12073 NumberCarsOwned | DataFormat: Error | We couldn't convert to Number. | two |
| 4 | 12111 DateFirstPurchase | DataFormat: Error | We couldn't parse the input provided as a Date value | No Date |
| 5 | 12113 BirthDate | DataFormat: Error | We couldn't parse the input provided as a Date value | d 23/08/1951 |
| 6 | 12143 BirthDate | DataFormat: Error | We couldn't parse the input provided as a Date value | 13/11/14965 |
| 7 | 12242 NumberChildrenAtHome | DataFormat: Error | We couldn't convert to Number. | zero |
| 8 | 12262 BirthDate | DataFormat: Error | We couldn't parse the input provided as a Date value | 2205/1960 |
| 9 | 12337 BirthDate | DataFormat: Error | We couldn't convert to Date. | |
| 10 | 12448 BirthDate | DataFormat: Error | We couldn't parse the input provided as a Date value | 20/505/1958 |
| 11 | 12507 BirthDate | DataFormat: Error | We couldn't parse the input provided as a Date value | NA |

I created a report with some visualizations that show the number of error rows by Columns and by some other factors;

Part 6: Error handling, Exception reporting, and data profiling

Exception Report: showing all errors happened across all columns



Count of Errors

11

| Column Name | Row Identifier | Error Reason | Error Message | Error Detail |
|----------------------|----------------|------------------|---|--------------|
| BirthDate | 12337 | DataFormat.Error | We couldn't convert to Date. | 9101975 |
| BirthDate | 12143 | DataFormat.Error | We couldn't parse the input provided as a Date value. | 13/11/14965 |
| BirthDate | 12448 | DataFormat.Error | We couldn't parse the input provided as a Date value. | 20/505/1958 |
| BirthDate | 12262 | DataFormat.Error | We couldn't parse the input provided as a Date value. | 2205/1960 |
| BirthDate | 12064 | DataFormat.Error | We couldn't parse the input provided as a Date value. | 9/100/1944 |
| BirthDate | 12113 | DataFormat.Error | We couldn't parse the input provided as a Date value. | d 23/08/1951 |
| BirthDate | 12507 | DataFormat.Error | We couldn't parse the input provided as a Date value. | NA |
| DateFirstPurchase | 12057 | DataFormat.Error | We couldn't parse the input provided as a Date value. | 30/808/2007 |
| DateFirstPurchase | 12111 | DataFormat.Error | We couldn't parse the input provided as a Date value. | No Date |
| NumberCarsOwned | 12073 | DataFormat.Error | We couldn't convert to Number. | two |
| NumberChildrenAtHome | 12242 | DataFormat.Error | We couldn't convert to Number. | zero |

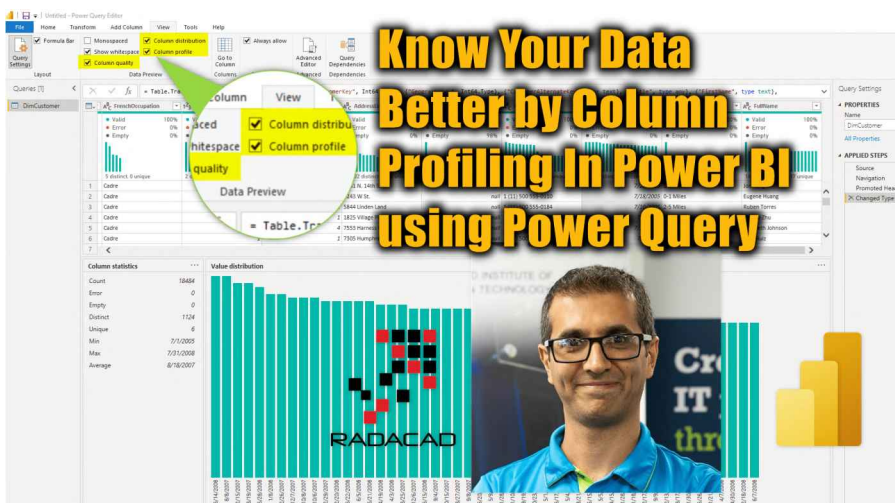


Summary

In this chapter, you learned how to achieve an exception report even if you don't know which column causes the error. This new method will search through all rows and all columns that cause the error. The method implemented in this chapter ended up with a function that you can use for any table to get the error rows output.



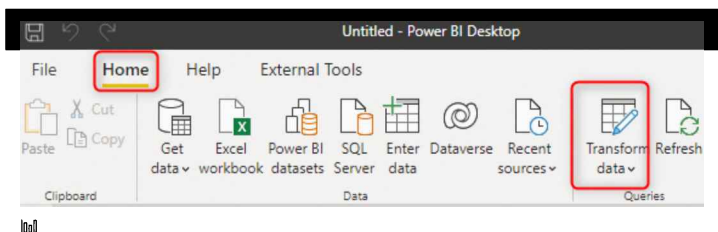
Chapter 27: Know Your Data Better by Column Profiling In Power BI using Power Query



An essential step towards a better analytics solution is to know your data. You have to know your data to be able to determine key columns, to learn what transformations need to be done, etc. Power Query provides a useful way of knowing your data in Power BI. Power BI has features for column profiling, column distribution, and column quality. All of these features are helpful to familiarize you with the data. In this chapter, we'll have a look at these options and how to use them.

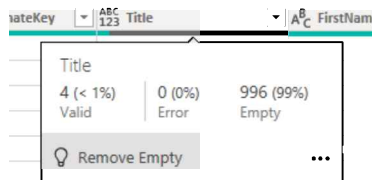
Power Query Editor

Column profiling, distribution, and quality are all Power Query features. You have to go to Power Query Editor window using the Transform Data in Power BI Desktop.



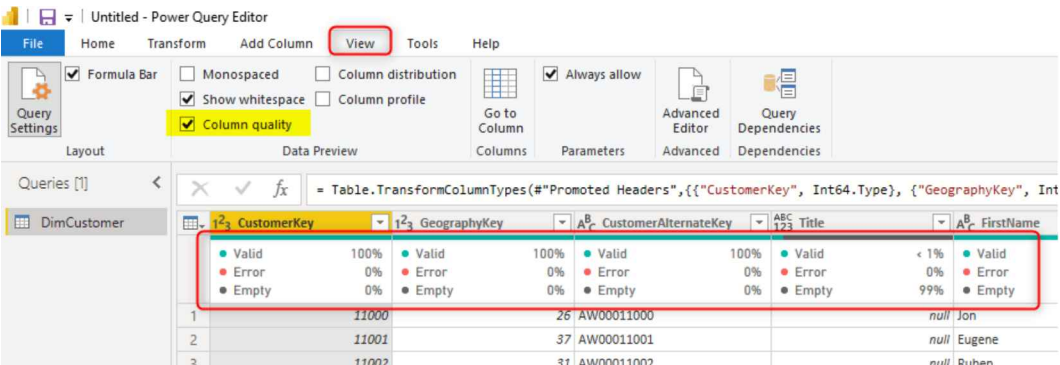
Column Quality

Often you have applied some transformations, and you want to know if you have any errors or empty values in your columns. There is a bar at the top of the column, right under the column header, which will give you some information about it.



Column quality

The information provided includes the number of Error, Empty, and Valid data cells in that column and their percentage. There is a simpler way to see this for all columns in the table. You can enable the Column Quality in the View tab in the Power Query Editor.

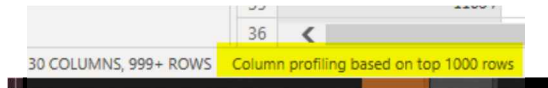


Enable Column Quality in Power Query Editor

This can be particularly helpful if you have applied a transformation and want to check if any rows return an error as a result.

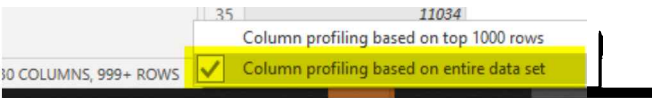
Column Profiling Based on the entire dataset

However, you have to be careful that the column profiling by default is based on the top 1000 rows. You can see this at the bottom of the Power Query Editor window.



column profiling based on 1000 rows

You can change this by clicking on it and choosing Column profiling based on the entire dataset.

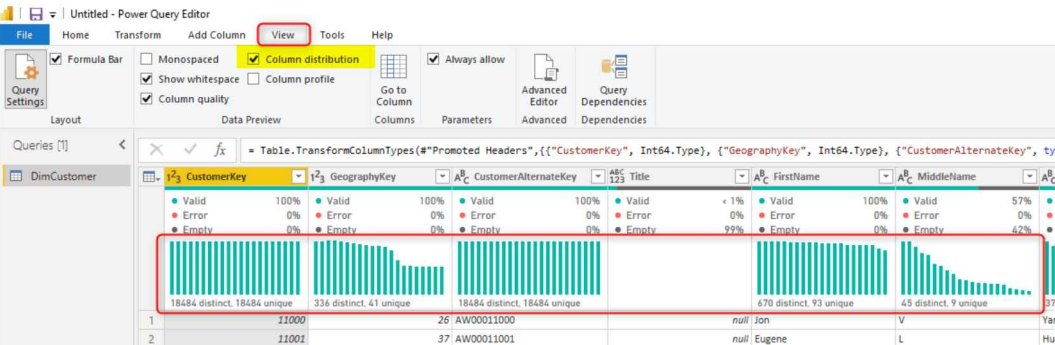


Column profiling based on the entire dataset

This might look like a very cool option to enable, but be careful that if your table size is big, this will slow down the Power Query Editor window. I suggest only enable it when needed and immediately change it back to the top 1000 rows.

Column Distribution

Another useful point to know about the data in a column is how the values are spread. Column distribution is helpful information for that, which can be enabled under the View tab in the Power Query Editor.



Column distribution

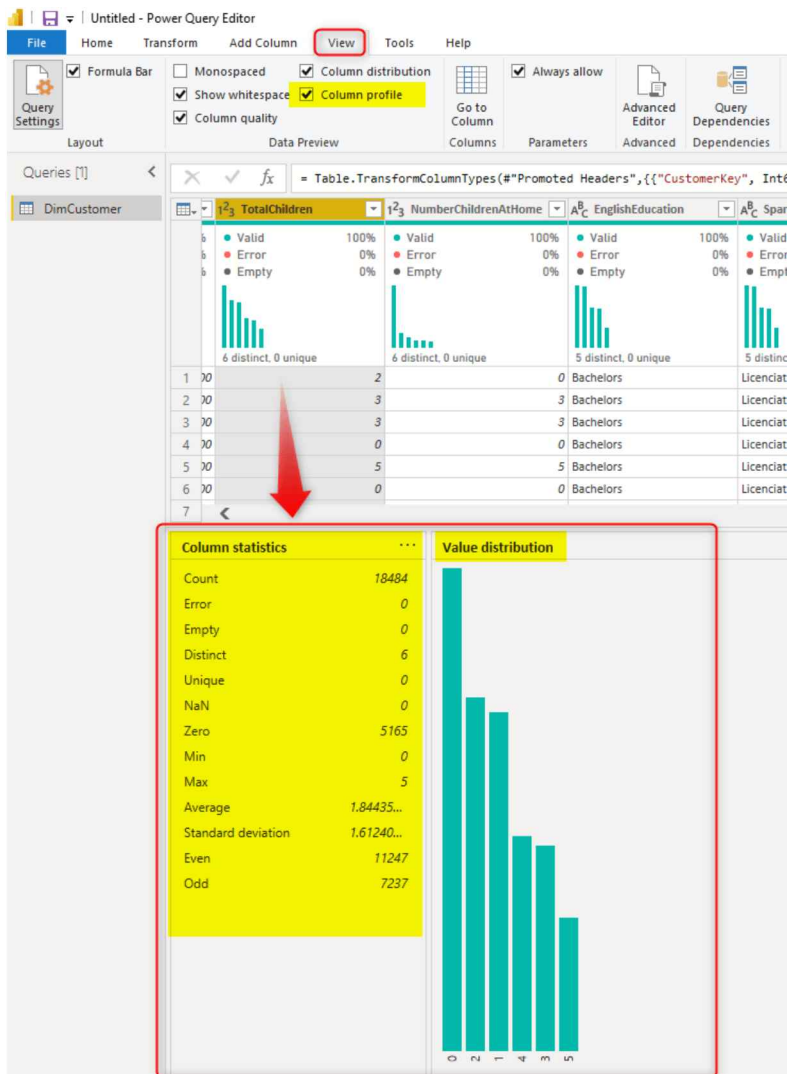
However, reading the values in detail in the small charts is a bit hard. This is why Column Profile can help.

Column Profile

Column profile includes the count of errors and empty values, and it also includes the distribution of values in the column. However, the column profile includes much more information too. Information such as the minimum, and the maximum values, count of unique values, and the distribution of values in detail. The column profile is a much more complete version of the other two options.

You can enable the column profile from the View tab in the Power Query Editor.

Part 6: Error handling, Exception reporting, and data profiling



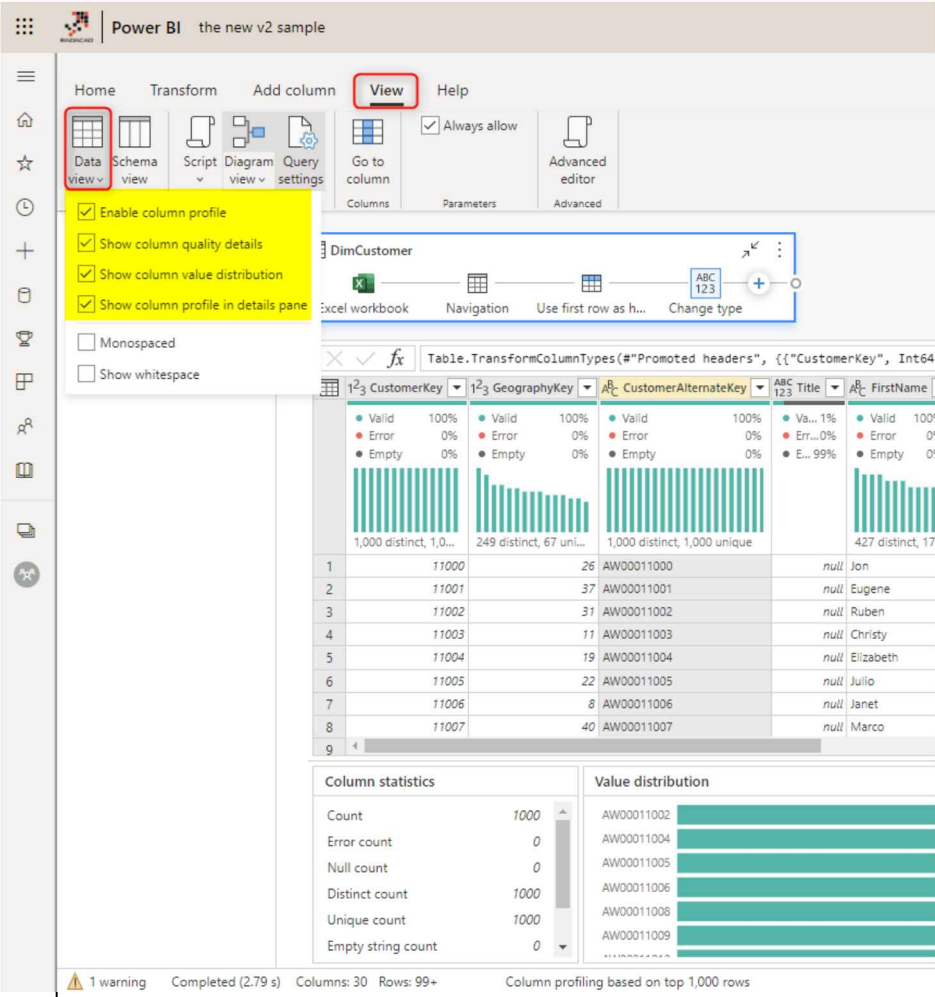
Column Profile in Power Query

In the column profile above, you can see the distribution of values in the TotalChildren column, and you can see more customers with no children and fewer with five children. You can also see the count of rows, errors, and empty values. As well as the min and max value in the column, the count of Even and Odd values, and much other information.

Knowing the information above is very helpful for a data engineer because you can apply the right transformation and target the right data values based on it.

Power Query Online

All of the profiling information above is also available in the Power Query online; you can find it in dataflow’s Power Query Editor



Column profiling in Power Query online

Performance Consideration

Data Profiling, quality, and distribution are great features to know your data better. However, having them enabled all the time is not recommended, especially if you have large tables. This will make your development experience inside the Power Query Editor much slower. I recommend enabling these features only to learn more about the data in the column and disable it soon after. These features will not impact the performance of the report, though, because they are developer-only features inside the Power Query Editor.

Data Profiling in Power BI reports

Data Profiling is in Power Query Editor, and it is not for the end-user. It is for the data engineer to be able to know the data. However, If you ever need to bring the data profiling information into a Power BI report and enhance the user experience by providing the profiling information, I have written about it in the next chapter.

Column profiling is an important part of knowing your data better, and the features above can help you with that.

Chapter 28: Create a Profiling Report in Power BI: Give the End User Information about the Data



Most of the time, you would like to know how the data quality looks like in your tables. Although this is a process that you mainly check at the time of data preparation, having such a report for end-users is also helpful because sometimes they see an odd outcome and wonder where this is coming from. A data profiling report tab in your Power BI report tells them more about your data and if there are any anomalies in it. In this chapter, I will show you an easy way to build a data profiling report in Power BI.

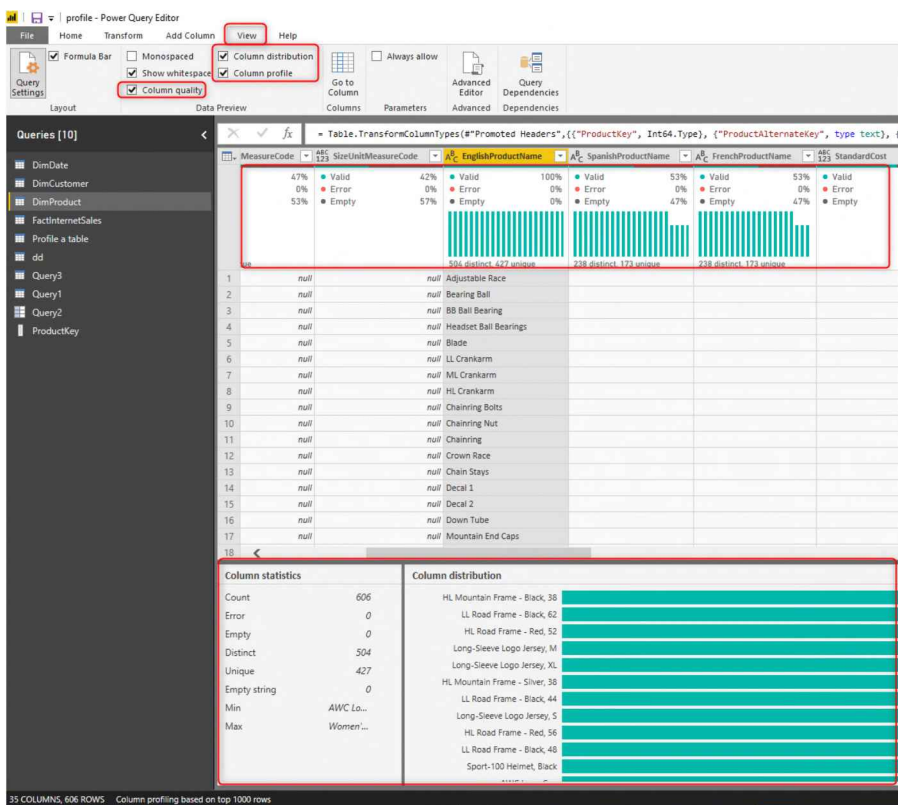
What is Data Profiling?

Data profiling is the process of understanding more about the data. Any information that can help to understand the data would be helpful. For example, you probably like to know how many unique values you have in the column, what is the minimum values, what the maximum is, the average, standard deviation, etc. This is what Data Profiling will provide for you.

Profiling in Power Query Editor

There is a data profiling section in Power Query Editor, which helps at the time of data preparation to understand the nature of the data. This can be done by selecting the Column Profiling and Quality and Distribution details in the View tab of the Power Query Editor of Power BI Desktop.

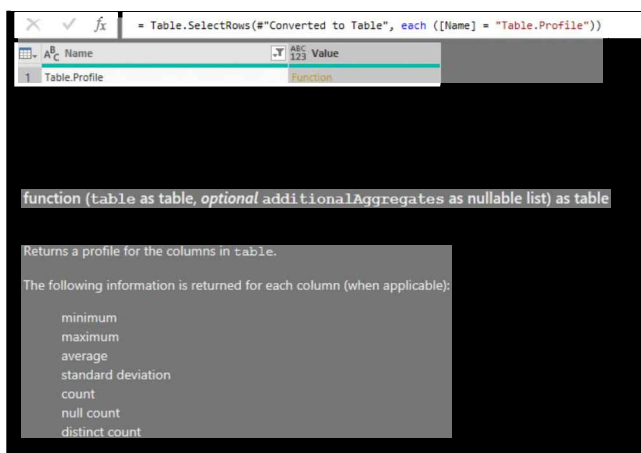
Part 6: Error handling, Exception reporting, and data profiling



The profiling data is great, but unfortunately, not available for end-users. I'm going to show you how to get that details as a report to the end-user.

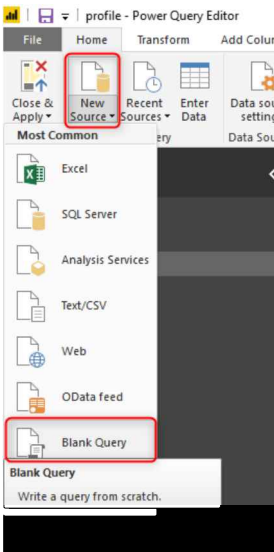
Power Query Function: Table.Profile

Table.Profile is a Power Query function that processes a table and gives you a table of all columns and data profiling information of those columns. Here is how Table.Profile function works;



Using Table.Profile

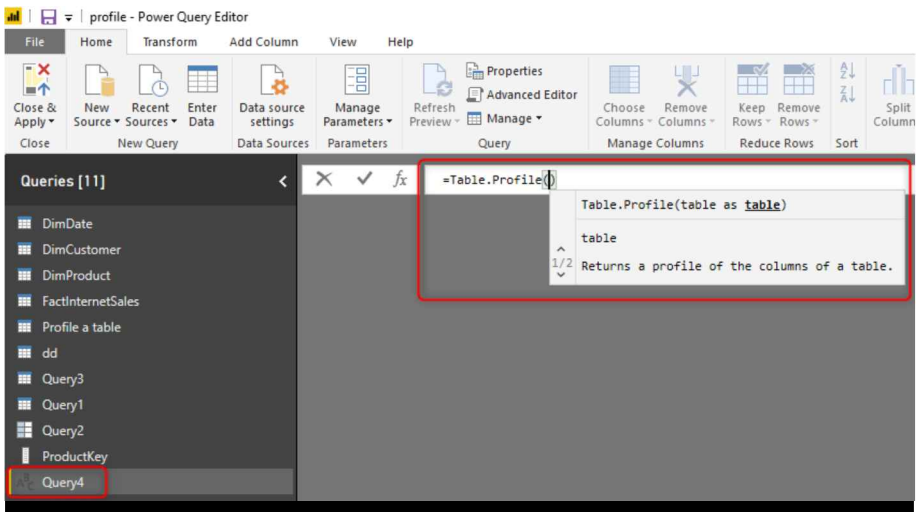
To see how the function works, you can start with a New Source in Power Query Editor and then Blank Query;



In the new blank query, in the formula bar (if you don't see the formula bar, check the formula bar option in the View tab of the Power Query Editor), type the below expression:

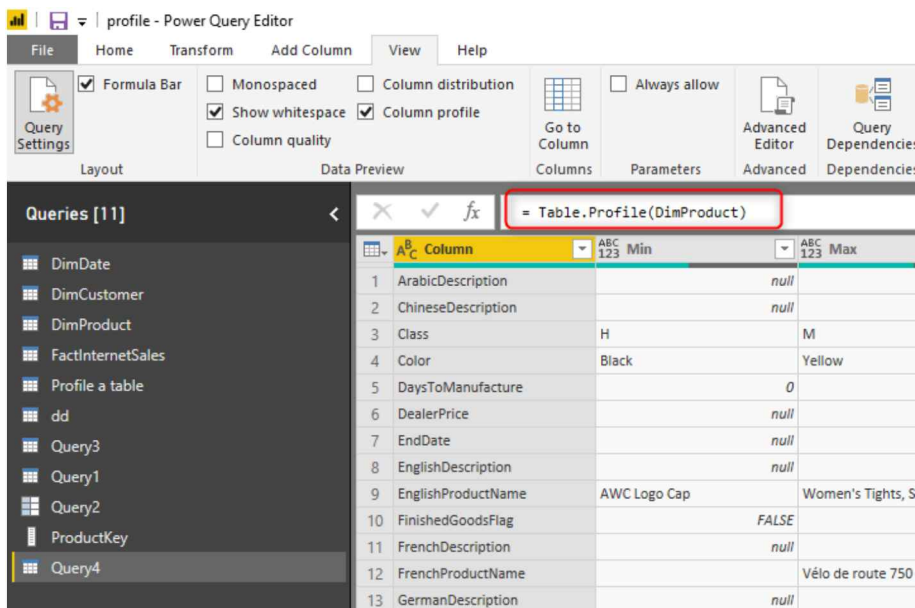
`=Table.Profile()`

Note that this code is not complete yet; we need to provide a table as the input of this function.

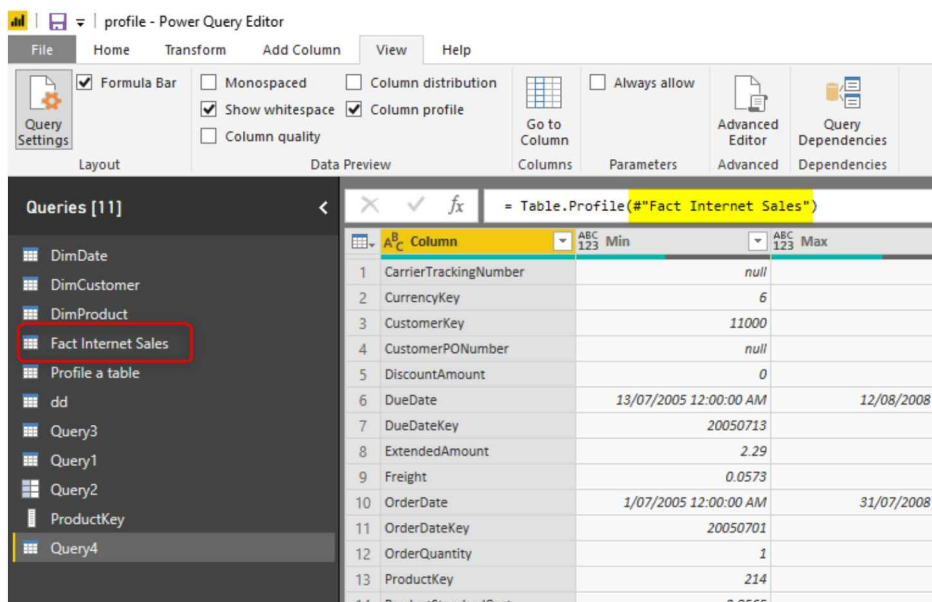


Part 6: Error handling, Exception reporting, and data profiling

Note that everything in Power Query is case-sensitive when you type something. Type the table name inside the bracket. Like below:



If the table name has space or any special characters in it, then you should follow the rule of variable naming for it, and put that inside double quote with a # at the beginning, for example, Fact Internet Sales should be written like below;



The good thing is that the intellisense is sometimes helpful, and you can just select the item from there.

The Profiling Data

The profiling data that you get from Table.Profile function is like below;

| Table.Profile(DimProduct) | | | | | | | | | |
|---------------------------|-----------------------|-----------------------------------|-----------------------|-------------------|-------------|-----------|---------------|--|------|
| Column | Min | Max | Average | StandardDeviation | Count | NullCount | DistinctCount | | |
| ArabicDescription | null | | null | null | null | 606 | 210 | | null |
| ChineseDescription | | | | null | null | 606 | 210 | | null |
| Class | H | M | | | null | 606 | 276 | | 4 |
| Color | Black | Yellow | | null | null | 606 | 0 | | 10 |
| DaysToManufacture | 0 | | 4 | 1.201320132 | 1.508893166 | 606 | 0 | | 4 |
| DealerPrice | null | | null | | | 606 | 211 | | null |
| EndDate | null | | null | | | 606 | 406 | | null |
| EnglishDescription | | | null | | | 606 | 210 | | null |
| EnglishProductName | AWC Logo Cap | Women's Tights, S | | null | | 606 | 0 | | 504 |
| FinishedGoodsFlag | FALSE | | TRUE | | | 606 | 0 | | 2 |
| FrenchDescription | null | | null | | | 606 | 210 | | null |
| FrenchProductName | | Vélo de route 750 noir, 58 | | null | | 606 | 0 | | 238 |
| GermanDescription | | | null | | | 606 | 210 | | null |
| HebrewDescription | null | | null | | | 606 | 210 | | null |
| JapaneseDescription | null | | null | | | 606 | 210 | | null |
| ListPrice | | | null | | | 606 | 211 | | null |
| ModelName | | | null | | | 606 | 209 | | null |
| ProductAlternateKey | AR-5381 | WB-H098 | | | | 606 | 0 | | 504 |
| ProductKey | 1 | | 606 | 303.5 | 175.0814096 | 606 | 0 | | 606 |
| ProductLine | | null | null | | | 606 | 226 | | null |
| ProductSubcategoryKey | | null | null | | | 606 | 209 | | null |
| ReorderPoint | 3 | 750 | | 371.4009901 | 273.0053357 | 606 | 0 | | 6 |
| SafetyStockLevel | 4 | 1000 | | 495.2013201 | 364.0071143 | 606 | 0 | | 6 |
| Size | null | | null | | | 606 | 307 | | null |
| SizeRange | 38-40 CM | XL | | null | | 606 | 0 | | 11 |
| SizeUnitMeasureCode | | null | null | | | 606 | 353 | | null |
| SpanishProductName | | Soporte multiusos para bicicletas | | null | | 606 | 0 | | 238 |
| StandardCost | | null | null | | | 606 | 211 | | null |
| StartDate | 1/06/1998 12:00:00 AM | 1/07/2007 12:00:00 AM | 20/11/2003 5:49:18 AM | | | 606 | 0 | | 4 |
| Status | Current | Current | | | | 606 | 200 | | 2 |
| Style | | null | null | | | 606 | 305 | | null |
| ThaiDescription | | null | null | | | 606 | 210 | | null |
| TurkishDescription | | null | null | | | 606 | 210 | | null |
| Weight | 2 | 1050 | | 56.11702128 | 158.0243589 | 606 | 324 | | 43 |
| WeightUnitMeasureCode | G | LB | | null | null | 606 | 324 | | 3 |

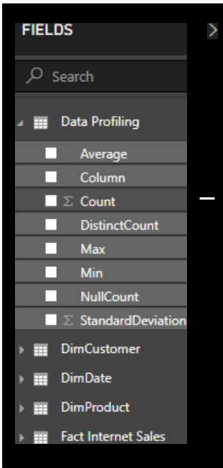
This will give me the list of all columns in the table, one at each row, and the information such as minimum value, maximum, average, standard deviation, count, null count, and distinct count. Great! Now we have this data as a table in Power Query, which means we can load it into Power BI. You can call this query Data Profiling or something like that, then Close and Apply the Power Query Editor window to load it into the Power BI.

Data Profiling Report

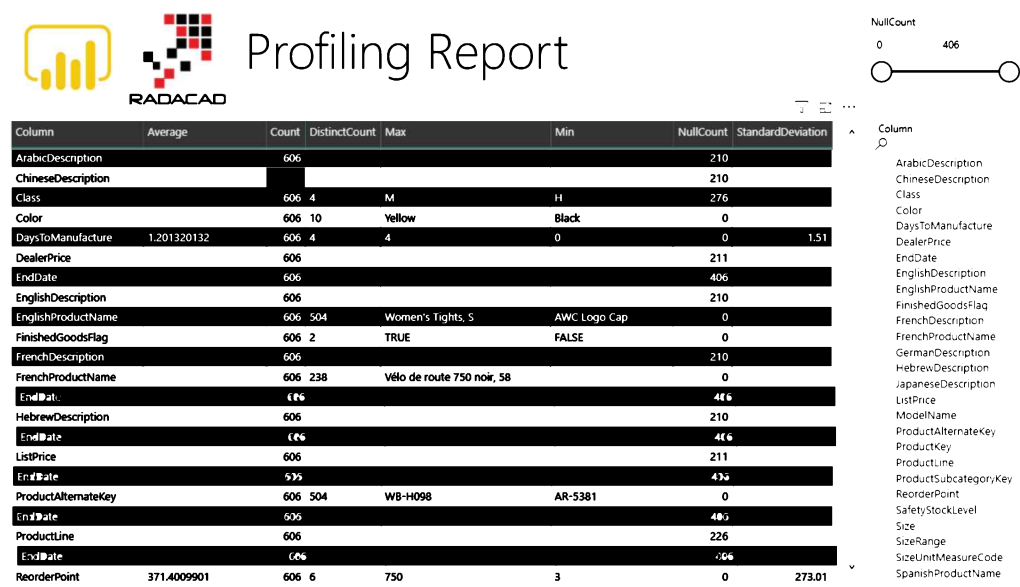
After loading the data into Power BI, you'll have the table with all columns, and it can be used in any visuals. I prefer to have a tab in my report (report page) named Profiling report with all details needed.



Part 6: Error handling, Exception reporting, and data profiling



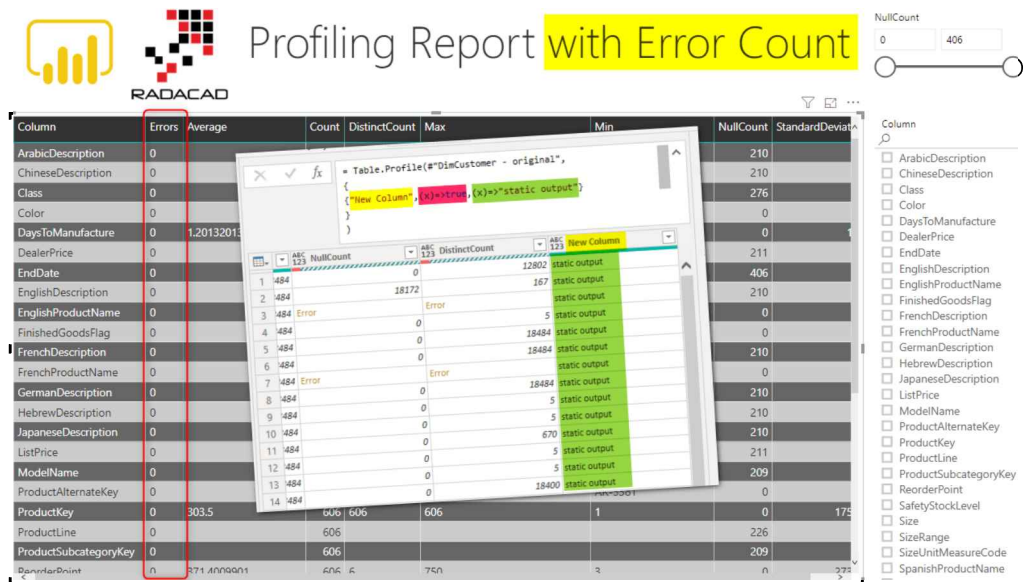
The rest of it is playing with visuals and slicers to give you the visualization of the data profiling in the report. Here is a simple example I built;



Summary

Information about the data in each column and each table is useful information some time to be shared with the end-user. In this chapter, you learned that using Table.Profile function, you can fetch this information, and then load it into Power BI, and visualize it in the report. For a real-world error-proof and comprehensive Power BI report, I do always recommend bringing the profiling report and the exception report in your implementation.

Chapter 29: Get The Error Count with the Profiling Data of Power BI Data Table using Power Query



I have explained previously how you can use the Table.Profile Power Query function to fetch the profiling data of a table, such as minimum value, maximum value, null count, etc. Use it on a report page to inform the user about the quality of the data. However, that function doesn't provide you with one important piece of information: The count of error values in each column. Although I have explained another method of catching error rows, It is also helpful if the profiling report can also have error information in it. So in this chapter, I am explaining how to add the count of errors to the Table.Profile function in Power Query.

Table.Profile Missing the Count of Errors

The default output of Table.Profile function in Power Query is missing the count of errors for a column. Something that you can see in the column statistics in the Power Query editor anyways!

fx = Table.TransformColumnTypes(#"Promoted Headers",{

| | NameStyle | BirthDate | MaritalStatus |
|----|-----------|------------|---------------|
| 1 | FALSE | 8/04/1966 | M |
| 2 | FALSE | 14/05/1965 | S |
| 3 | FALSE | 12/08/1965 | M |
| 4 | FALSE | 15/02/1968 | S |
| 5 | FALSE | 8/08/1968 | S |
| 6 | FALSE | 5/08/1965 | S |
| 7 | FALSE | 6/12/1965 | S |
| 8 | FALSE | 9/05/1964 | M |
| 9 | FALSE | 7/07/1964 | S |
| 10 | FALSE | 1/04/1964 | S |
| 11 | | | |

| Column statistics | Value distribution |
|-------------------|--------------------|
| Count 18484 | |
| Error 7 | |

We encountered an error while computing the distribution.

Now let me show you how you can fetch the count of errors with this function.

Using the second Parameter of Table.Profile to Fetch more information about the column

If Table.Profile function is used by default, only provides specific information, such as max, min, count, etc. If you want more information, you can get it, but you need to write your function. I have explained fully in details what is a custom function in Power BI, and also a sample of how to create it. Here is how the second parameter can be used:

*Table.Profile(**table** as table, optional **additionalAggregates** as nullable list)*

The additionalAggregates that you see in the above syntax definition is an optional parameter, a list data type. This is a list that is going to have a function in each item. Each function should have a name, a true or false logic in which the function returns a value, and the function itself. something like this:

- Function Name
- The true or false logic in which the function would return the result (if true) or returns null (if false)
- The function itself

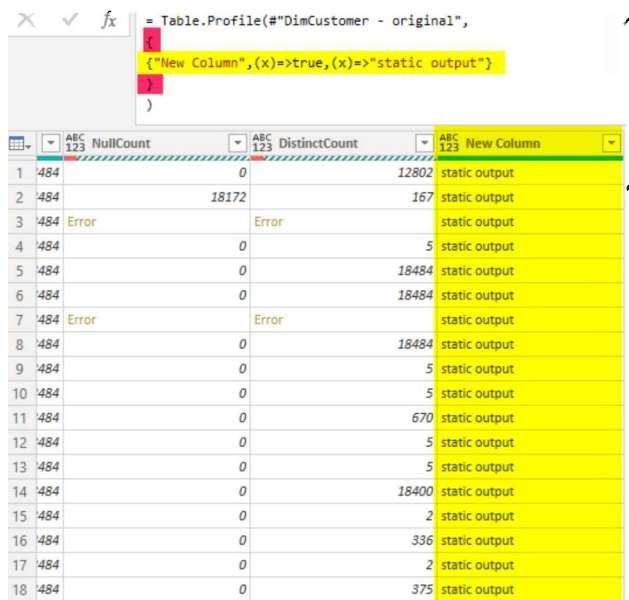
Chapter 29: Get The Error Count with the Profiling Data of Power BI Data Table using Power Query

The above three items should be in a list itself. So we are dealing with a list inside a list. I highly recommend you to read my chapters about the basics of Power Query scripting in later parts of this book.

To define a list, we use {}. and if you want to have a list under a list, it would be: { {}, {}, ... }

Each of the inside {} is representing a function and would return a column as the result of Table.Profile function. Here is a simple example:

```
= Table.Profile("#DimCustomer - original",  
{  
{"New Column",(x)=>true,(x)=>"static output"}  
}  
)
```



The screenshot shows the Power Query Editor. The formula bar contains the M code: `= Table.Profile("#DimCustomer - original", { {"New Column", (x) => true, (x) => "static output"} })`. The curly braces are highlighted in red and yellow. Below the formula bar, a table is displayed with three columns: 'NullCount', 'DistinctCount', and 'New Column'. The 'New Column' column contains the value 'static output' for all rows. The 'NullCount' and 'DistinctCount' columns show various values, including errors in some rows.

| | NullCount | DistinctCount | New Column |
|----|-----------|---------------|---------------|
| 1 | 484 | 0 | static output |
| 2 | 484 | 18172 | static output |
| 3 | 484 | Error | static output |
| 4 | 484 | 0 | static output |
| 5 | 484 | 0 | static output |
| 6 | 484 | 0 | static output |
| 7 | 484 | Error | static output |
| 8 | 484 | 0 | static output |
| 9 | 484 | 0 | static output |
| 10 | 484 | 0 | static output |
| 11 | 484 | 0 | static output |
| 12 | 484 | 0 | static output |
| 13 | 484 | 0 | static output |
| 14 | 484 | 0 | static output |
| 15 | 484 | 0 | static output |
| 16 | 484 | 0 | static output |
| 17 | 484 | 0 | static output |
| 18 | 484 | 0 | static output |

Let me explain what happened in a more detailed explanation. The Table.Profile function is getting the table as the first input, and then the second input is a list {} which is highlighted red in the above screenshot. Inside that list, we have another list {}, which is highlighted yellow. That list includes the three information I mentioned earlier: function name, the return logic, the function to return the value. The highlighted screenshot below shows how each part is related to building the output.

The screenshot shows a Power Query formula bar with the following code:

```

= Table.Profile("#DimCustomer - original",
{
  "New Column", (x)=>true, (x)=>"static output"
})

```

Below the formula bar is a table with the following columns: NullCount, DistinctCount, and New Column. The table contains 14 rows of data. The 'New Column' column contains the value 'static output' for all rows. The 'NullCount' and 'DistinctCount' columns show various values, including errors in some rows.

| | NullCount | DistinctCount | New Column |
|----|-----------|---------------|---------------|
| 1 | 484 | 0 | static output |
| 2 | 484 | 18172 | static output |
| 3 | 484 Error | Error | static output |
| 4 | 484 | 0 | static output |
| 5 | 484 | 0 | static output |
| 6 | 484 | 0 | static output |
| 7 | 484 Error | Error | static output |
| 8 | 484 | 0 | static output |
| 9 | 484 | 0 | static output |
| 10 | 484 | 0 | static output |
| 11 | 484 | 0 | static output |
| 12 | 484 | 0 | static output |
| 13 | 484 | 0 | static output |
| 14 | 484 | 0 | static output |

I have used the above simple example to show a static output. If you want to learn how an actual function is built-in Power Query, read the chapter about it earlier in this book. Because that is how the second and the third item in the list above are created.

The Function to Fetch Error Count

Now that you know how to add a column using a function to the Table.Profile column, all you need is a function that returns the count of errors.

The input of this function is always the current column that the Table.Profile is working on, and all the values of that column are a list. I just showed the input parameter itself, and you can see the result.

The screenshot shows a Power Query formula bar with the following code:

```

= Table.Profile("#DimCustomer - original",
{
  "New Column", (x)=>true, (x)=>x
})

```

Below the formula bar is a table with the following columns: NullCount, DistinctCount, and New Column. The table contains 14 rows of data. The 'New Column' column contains the value 'List' for all rows. The 'NullCount' and 'DistinctCount' columns show various values, including errors in some rows. A red arrow points from the 'List' value in the 'New Column' of row 3 to a detailed view of the list below the table.

| | NullCount | DistinctCount | New Column |
|---|-----------|---------------|------------|
| 1 | 484 | 0 | List |
| 2 | 484 | 18172 | List |
| 3 | 484 Error | Error | List |
| 4 | 484 | 0 | List |

The detailed view of the list for row 3 shows the following values:

| List |
|------------|
| 8/04/1966 |
| 14/05/1965 |
| 12/08/1965 |
| 15/02/1968 |
| 8/08/1968 |
| 5/08/1965 |
| 6/12/1965 |
| 9/05/1964 |
| 7/07/1964 |
| 1/04/1964 |
| 6/02/1964 |

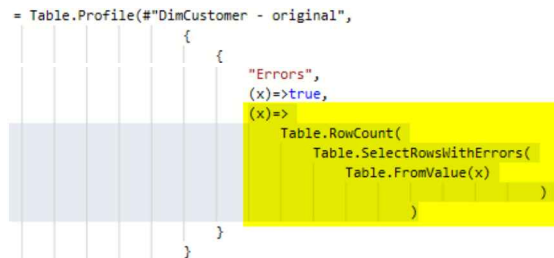
Chapter 29: Get The Error Count with the Profiling Data of Power BI Data Table using Power Query

Now, it is up to you how to write the function to fetch error values, here is one way of doing it:

```
(x)=>
Table.RowCount(
Table.SelectRowsWithErrors(
Table.FromValue(x)
)
)
)
```

The function above first converts the list to the table, then selects only error rows and then counts the number of rows for that as the value to return.

and this is where the function should be used:



```
= Table.Profile("#DimCustomer - original",
{
{
"Errors",
(x)=>true,
(x)=>
Table.RowCount(
Table.SelectRowsWithErrors(
Table.FromValue(x)
)
)
}
}
})
```

I have created a column named Errors, and it returns the count of errors as below. the entire M script for this query is as below:

let

```
Source = Table.Profile("#DimCustomer - original",
{
{
"Errors",
(x)=>true,
(x)=>
Table.RowCount(
Table.SelectRowsWithErrors(
Table.FromValue(x)
)
)
}
}
})
```

Part 6: Error handling, Exception reporting, and data profiling

```
}  
}  
)
```

in

Source

```
(x)=>  
Table.RowCount(  
  Table.SelectRowsWithErrors(  
    Table.FromValue(x)  
  )  
)
```

| | Min | Average | StandardDeviation | Count | NullCount | DistinctCount | Errors |
|------------------------|--------------------------|---------|-------------------|-------------|-----------|---------------|--------|
| Saint-Lazare | Zur Linderung 787 | | null | 18484 | 0 | 12802 | 0 |
| | Verkaufsabteilung | | null | 18484 | 18172 | 167 | 0 |
| | Error | Error | | 18484 | Error | Error | 7 |
| les | 5-10 Miles | | null | 18484 | 0 | 5 | 0 |
| 011000 | AW00029483 | | null | 18484 | 0 | 18484 | 0 |
| | 11000 | 29483 | 20241.5 | 5336.015523 | 18484 | 0 | 18484 |
| | Error | Error | | 18484 | Error | Error | 1 |
| 10@adventure-works.com | zoe9@adventure-works.com | | null | 18484 | 0 | 18484 | 0 |
| ors | Partial High School | | null | 18484 | 0 | 5 | 0 |
| it | Skilled Manual | | null | 18484 | 0 | 5 | 0 |
| | Zoe | | null | 18484 | 0 | 670 | 0 |
| 2 | Niveau bac | | null | 18484 | 0 | 5 | 0 |
| | Techniciens | | null | 18484 | 0 | 5 | 0 |
| Adams | Zoe Watson | | null | 18484 | 0 | 18400 | 0 |
| | M | | null | 18484 | 0 | 2 | 0 |
| 2 | | 654 | 257.9562865 | 196.5310625 | 18484 | 0 | 336 |
| 0 | | 1 | 0.676368751 | 0.467873815 | 18484 | 0 | 2 |
| | Zukowski | | null | 18484 | 0 | 375 | 0 |

Using the result of this query, now I can get my profiling report even better:

Profiling Report with Error Count

NullCount: 0 to 406

| Column | Errors | Average | Count | DistinctCount | Max | Min | NullCount | StandardDeviation |
|-----------------------|--------|-------------|-------|---------------|----------------------------|--------------|-----------|-------------------|
| ArabicDescription | 0 | | 606 | | | | 210 | |
| ChineseDescription | 0 | | 606 | | | | 210 | |
| Class | 0 | | 606 | 4 | M | H | 276 | |
| Color | 0 | | 606 | 10 | Yellow | Black | 0 | |
| DaysToManufacture | 0 | 1.201320132 | 606 | 4 | 4 | 0 | 0 | |
| DealerPrice | 0 | | 606 | | | | 211 | |
| EndDate | 0 | | 606 | | | | 406 | |
| EnglishDescription | 0 | | 606 | | | | 210 | |
| EnglishProductName | 0 | | 606 | 504 | Women's Tights, S | AWC Logo Cap | 0 | |
| FinishedGoodsFlag | 0 | | 606 | 2 | TRUE | FALSE | 0 | |
| FrenchDescription | 0 | | 606 | | | | 210 | |
| FrenchProductName | 0 | | 606 | 238 | Vélo de route 750 noir, 58 | | 0 | |
| GermanDescription | 0 | | 606 | | | | 210 | |
| HebrewDescription | 0 | | 606 | | | | 210 | |
| JapaneseDescription | 0 | | 606 | | | | 210 | |
| ListPrice | 0 | | 606 | | | | 211 | |
| ModelName | 0 | | 606 | | | | 209 | |
| ProductAlternateKey | 0 | | 606 | 504 | WB-H098 | AR-5381 | 0 | |
| ProductKey | 0 | 303.5 | 606 | 606 | 606 | 1 | 0 | 175 |
| ProductLine | 0 | | 606 | | | | 226 | |
| ProductSubcategoryKey | 0 | | 606 | | | | 209 | |
| SpanishDescription | 0 | | 606 | | | | 210 | |

- ☐ ArabicDescription
- ☐ ChineseDescription
- ☐ Class
- ☐ Color
- ☐ DaysToManufacture
- ☐ DealerPrice
- ☐ EndDate
- ☐ EnglishDescription
- ☐ FinishedGoodsFlag
- ☐ FrenchDescription
- ☐ FrenchProductName
- ☐ GermanDescription
- ☐ HebrewDescription
- ☐ JapaneseDescription
- ☐ ListPrice
- ☐ ModelName
- ☐ ProductAlternateKey
- ☐ ProductKey
- ☐ ProductLine
- ☐ ProductSubcategoryKey
- ☐ ReorderPoint
- ☐ SafetyStockLevel
- ☐ Size
- ☐ SizeRange
- ☐ SizeUnitMeasureCode
- ☐ SpanishProductName

Summary

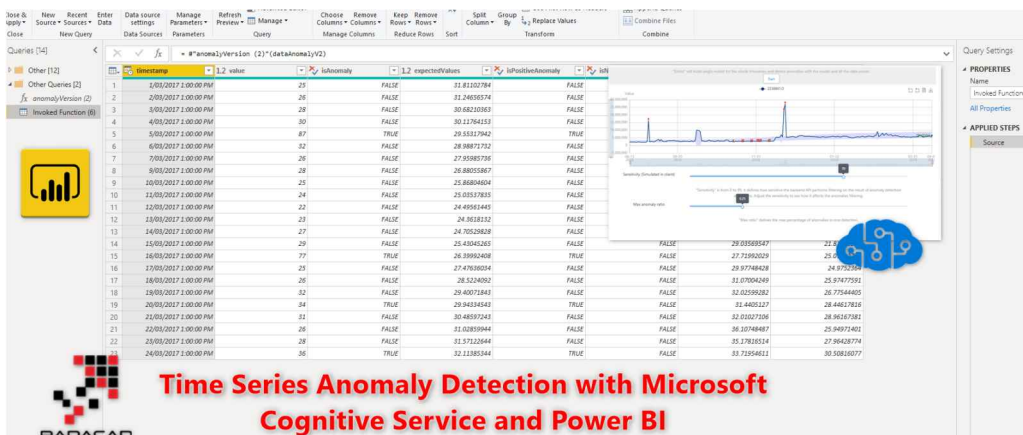
The second argument of Table.Profile function helps do any customization you want, even changing how min, max, or other aggregations are calculated. You just need to

Chapter 29: Get The Error Count with the Profiling Data of Power BI Data Table using Power Query

write your custom function for it. In this chapter, you have seen an example of a custom function to fetch the count of errors. Special thanks to Curt from the Power Query team for his help in pointing me in the right direction wherever needed.

Part 7: Advanced analytics using Power Query

Chapter 30: Time Series Anomaly Detection in Power BI using Cognitive Service and Power Query

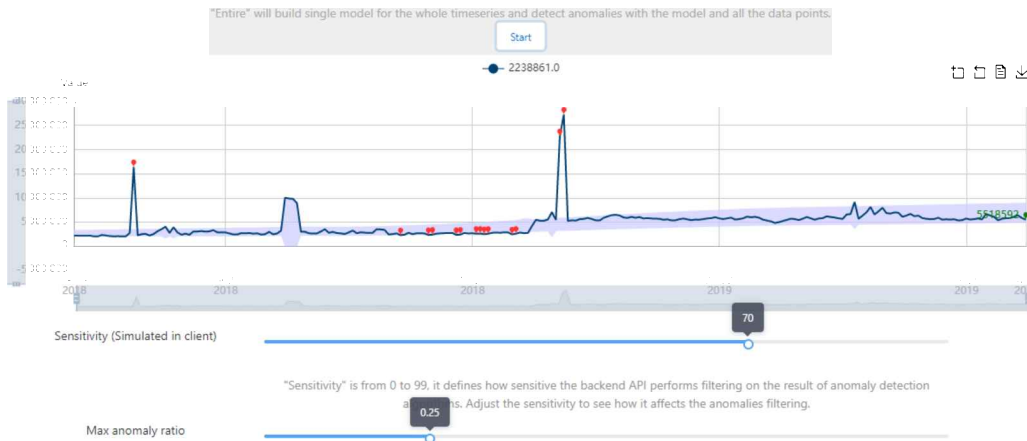


This chapter is based on the service in Cognitive Service name “Anomaly Detection,” which is now in Preview. I am going to talk about how to use it in Power BI. In this chapter, a brief introduction to anomaly detection will be presented. Then, how it can be used inside Power BI will be discussed.

Anomaly detection

Anomaly detection is an approach in machine learning that can detect the rare data points, events, or observations in the data that are different from the majority of the data. There are many algorithms that are able to be used for this purpose. Microsoft Cognitive Service mainly used [Spectral Residual \(SR\) and Convolutional Neural Network \(CNN\)](https://arxiv.org/abs/1906.03821). [https://arxiv.org/abs/1906.03821] The anomaly Detection in Microsoft is a time-series anomaly detection service. Check the Microsoft research article from [here](http://delivery.acm.org/10.1145/3340000/3330680/p3009-ren.pdf?ip=125.236.139.38&id=3330680&acc=OPENTOC&key=4D4702B0C3E38B35%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35%2E9F04A3A78F7D3B8D&_a_cm_=1575570542_44b0e3f41a38538312b0a64c6b2fd969) [http://delivery.acm.org/10.1145/3340000/3330680/p3009-ren.pdf?ip=125.236.139.38&id=3330680&acc=OPENTOC&key=4D4702B0C3E38B35%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35%2E9F04A3A78F7D3B8D&_a_cm_=1575570542_44b0e3f41a38538312b0a64c6b2fd969]

Part 7: Advanced analytics using Power Query

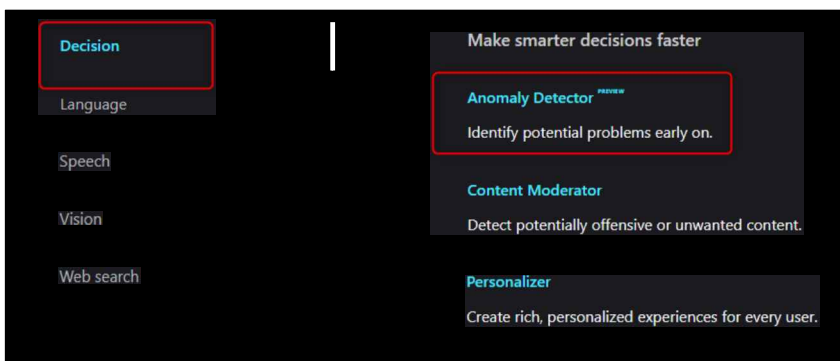


Time Series Anomaly detection in Microsoft Cognitive Service Cognitive Service

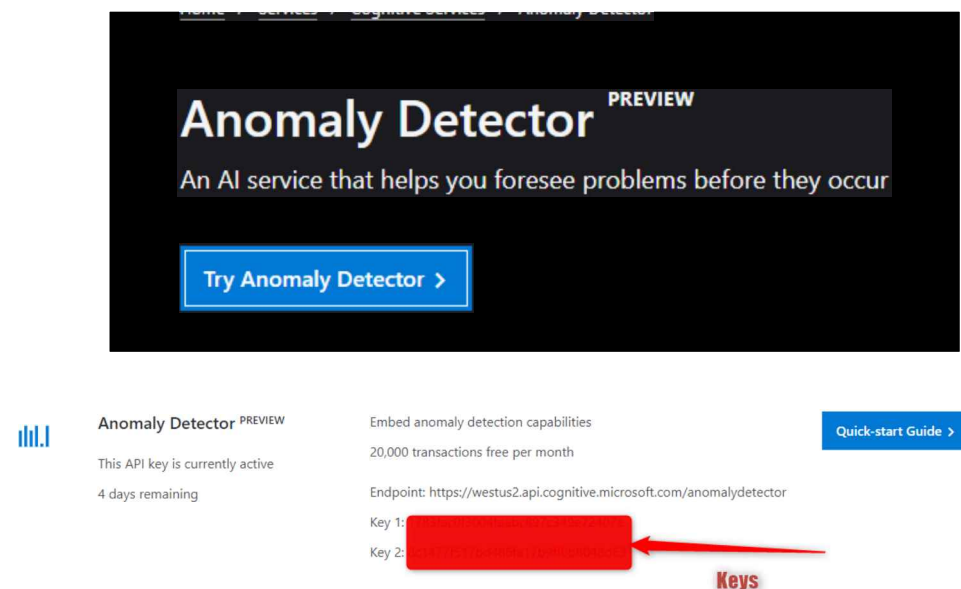
To have an overview of how it works, follow the procedure below;

1- Navigate to <https://azure.microsoft.com/en-us/services/cognitive-services/> [\[https://azure.microsoft.com/en-us/services/cognitive-services/\]](https://azure.microsoft.com/en-us/services/cognitive-services/)

2- under the decision, click on the Anomaly detection



3- Click on the Try Anomaly detector to get the **Key and endpoint**



On the [page\[https://docs.microsoft.com/en-nz/azure/cognitive-services/anomaly-detector/\]](https://docs.microsoft.com/en-nz/azure/cognitive-services/anomaly-detector/), there are some great articles on how to use Anomaly detection. To see a live demo of Anomaly detection and how it works, you can use the below link

[Live Demo\[https://algoevaluation.azurewebsites.net/\]](https://algoevaluation.azurewebsites.net/)

Power BI and Time Series Anomaly detection

There is a document in the Microsoft webpage about how to connect to the API of Anomaly detection, which was a big help, but I have to change some parts of the code to make it work as I thought the API of the anomaly detection change as it is in Preview, so I have to change the M code to make it work, but in future also this one may change as well if the API change to make sure it works need to check the input JSON data.

This is the original article that I change to do work with the new API;

[Link to the article\[https://docs.microsoft.com/en-nz/azure/cognitive-services/anomaly-detector/tutorials/batch-anomaly-detection-powerbi\]](https://docs.microsoft.com/en-nz/azure/cognitive-services/anomaly-detector/tutorials/batch-anomaly-detection-powerbi)

Data

There is a specific format of data that you can use. It should be two columns; one is for the date with the name “timestamp,” the other one is “value.”

Part 7: Advanced analytics using Power Query

| | A_C timestamp | A_C value |
|----|----------------------|-------------|
| 1 | 2017-03-01T00:00:00Z | 25 |
| 2 | 2017-03-02T00:00:00Z | 26 |
| 3 | 2017-03-03T00:00:00Z | 28 |
| 4 | 2017-03-04T00:00:00Z | 30 |
| 5 | 2017-03-05T00:00:00Z | 87 |
| 6 | 2017-03-06T00:00:00Z | 32 |
| 7 | 2017-03-07T00:00:00Z | 26 |
| 8 | 2017-03-09T00:00:00Z | 28 |
| 9 | 2017-03-10T00:00:00Z | 25 |
| 10 | 2017-03-11T00:00:00Z | 24 |
| 11 | 2017-03-12T00:00:00Z | 22 |
| 12 | 2017-03-13T00:00:00Z | 23 |
| 13 | 2017-03-14T00:00:00Z | 27 |
| 14 | 2017-03-15T00:00:00Z | 29 |
| 15 | 2017-03-16T00:00:00Z | 77 |
| 16 | 2017-03-17T00:00:00Z | 25 |
| 17 | 2017-03-18T00:00:00Z | 26 |
| 18 | 2017-03-19T00:00:00Z | 32 |
| 19 | 2017-03-20T00:00:00Z | 34 |
| 20 | 2017-03-21T00:00:00Z | 31 |
| 21 | 2017-03-22T00:00:00Z | 26 |
| 22 | 2017-03-23T00:00:00Z | 28 |
| 23 | 2017-03-24T00:00:00Z | 36 |

This is the specific format for the time. In my scenario, it was the daily data, so in the API, we have a field name `granularity`, so the form was `daily`.

| Fields | |
|----------|---|
| Daily | 3 |
| Hourly | 4 |
| Minutely | 5 |
| Monthly | 1 |
| Weekly | 2 |
| Yearly | 0 |

2017-03-01T00:00:00Z

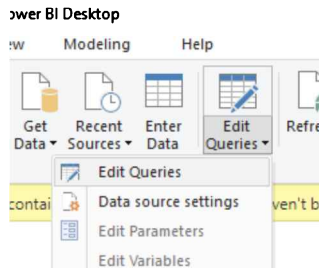
Use the dataset provided in the book's code files for your demo

Function for calling The API

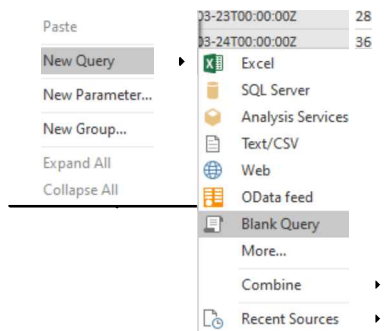
Now we need to create a function to call the API.

To start; go to the Power Query Editor

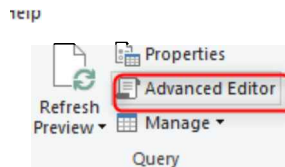
Chapter 30: Time Series Anomaly Detection in Power BI using Cognitive Service and Power Query



Then in the Query list, create a new Query and Blank Query



Then click on the Advanced Editor to paste the below code



The in the new Query Editor, remove everything and paste the code



Part 7: Advanced analytics using Power Query

the above code is here (check the image as the format may change while copying from here)

```
(table as the table) => let
    apikey = "Your API Code,"
    endpoint =
        "https://westus2.api.cognitive.microsoft.com/anomalydetector/v1.0/timeseries/entire/detect",
    inputTable = Table.TransformColumnTypes(table,{{"timestamp", type text},{ "value1", type
        number}}),
    jsonText = Text.FromBinary(Json.FromValue(inputTable)),
    jsonbody = anomaly,
    bytesbody = Text.ToBinary(jsonbody),
    headers = {#"Content-Type" = "application/json", #"Ocp-Apim-Subscription-Key" = apikey},
    bytesresp = Web.Contents(endpoint, [Headers=headers, Content=bytesbody]),
    jsonresp = Json.Document(bytesresp),

    respTable = Table.FromColumns({
        Table.Column(inputTable, "timestamp")
        ,Table.Column(inputTable, "value1")
        , Record.Field(jsonresp, "isAnomaly") as list
        , Record.Field(jsonresp, "expectedValues") as list
        , Record.Field(jsonresp, "upperMargins") as list
        , Record.Field(jsonresp, "lowerMargins") as list
        , Record.Field(jsonresp, "isPositiveAnomaly") as list
        , Record.Field(jsonresp, "isNegativeAnomaly") as list
    }, {"timestamp", "value1", "isAnomaly", "expectedValues", "upperMargin", "lowerMargin",
        "isPositiveAnomaly", "isNegativeAnomaly"}
    ),

    respTable1 = Table.AddColumn(respTable, "upperMargins", (row) => row[expectedValues] +
        row[upperMargin]),
    respTable2 = Table.AddColumn(respTable1, "lowerMargins", (row) => row[expectedValues] -
        row[lowerMargin]),
    respTable3 = Table.RemoveColumns(respTable2, "upperMargin"),
    respTable4 = Table.RemoveColumns(respTable3, "lowerMargin"),

    results = Table.TransformColumnTypes(
        respTable4,
        {{"timestamp", type datetime}, {"value1", type number}, {"isAnomaly", type logical},
        {"isPositiveAnomaly", type logical}, {"isNegativeAnomaly", type logical},
        {"expectedValues", type number}, {"upperMargins", type number}, {"lowerMargins", type
        number}}
    )
in results
```

This code is a bit different from Microsoft documents to support the API.

| | A ^B _C timestamp | A ^B _C value |
|----|---------------------------------------|-----------------------------------|
| 1 | 2017-03-01T00:00:00Z | 25 |
| 2 | 2017-03-02T00:00:00Z | 26 |
| 3 | 2017-03-03T00:00:00Z | 28 |
| 4 | 2017-03-04T00:00:00Z | 30 |
| 5 | 2017-03-05T00:00:00Z | 87 |
| 6 | 2017-03-06T00:00:00Z | 32 |
| 7 | 2017-03-07T00:00:00Z | 26 |
| 8 | 2017-03-09T00:00:00Z | 28 |
| 9 | 2017-03-10T00:00:00Z | 25 |
| 10 | 2017-03-11T00:00:00Z | 24 |
| 11 | 2017-03-12T00:00:00Z | 22 |
| 12 | 2017-03-13T00:00:00Z | 23 |
| 13 | 2017-03-14T00:00:00Z | 27 |
| 14 | 2017-03-15T00:00:00Z | 29 |
| 15 | 2017-03-16T00:00:00Z | 77 |
| 16 | 2017-03-17T00:00:00Z | 25 |
| 17 | 2017-03-18T00:00:00Z | 26 |
| 18 | 2017-03-19T00:00:00Z | 32 |
| 19 | 2017-03-20T00:00:00Z | 34 |
| 20 | 2017-03-21T00:00:00Z | 31 |
| 21 | 2017-03-22T00:00:00Z | 26 |
| 22 | 2017-03-23T00:00:00Z | 28 |
| 23 | 2017-03-24T00:00:00Z | 36 |

WeatherData (1)

fx Query1

WeatherData (2)

! Invoked Function

data1

! Invoked Function (2)

milk

! Invoked Function (3)

A^B_C anomaly

fx Query1 (2)

Invoked Function (4)

fx anomalyVersion1

Invoked Function (5)

A^B_C Query2

dataAnomalyV2

fx anomalyVersion (2)

Invoked Function (6)

Enter Parameter

table

WeatherData (1)

WeatherData (2)

data1

fu milk

Invoked Function (5)

dataAnomalyV2

Invoked Function (6)

Invoked Function (6)

My table

Now you should see the below result.

The original data in the first two columns, then the boolean column to show if anomaly happens or not, then expected value.

the [ISPositiveAnomaly](https://docs.microsoft.com/en-nz/dotnet/api/microsoft.azure.cognitiveservices.anomalydetector.models.entiredetectresponse.ispositiveanomaly?view=azure-dotnet-preview)[https://docs.microsoft.com/en-nz/dotnet/api/microsoft.azure.cognitiveservices.anomalydetector.models.entiredetectresponse.ispositiveanomaly?view=azure-dotnet-preview] and [IsNegativeAnomaly](https://docs.microsoft.com/en-nz/dotnet/api/microsoft.azure.cognitiveservices.anomalydetector.models.entiredetectresponse.isnegativeanomaly?view=azure-dotnet-preview)[https://docs.microsoft.com/en-nz/dotnet/api/microsoft.azure.cognitiveservices.anomalydetector.models.entiredetectresponse.isnegativeanomaly?view=azure-dotnet-preview]

Part 7: Advanced analytics using Power Query

[nz/dotnet/api/microsoft.azure.cognitiveservices.anomalydetector.models.entiredetectorresponse.isnegativeanomaly?view=azure-dotnet-preview\]](https://dotnet/api/microsoft.azure.cognitiveservices.anomalydetector.models.entiredetectorresponse.isnegativeanomaly?view=azure-dotnet-preview)

| | timestamp | 1.2 value | 1.2 isAnomaly | 1.2 expectedValues | 1.2 isPositiveAnomaly | 1.2 isNegativeAnomaly | 1.2 upperMargins | 1.2 lowerMargins |
|----|-----------------------|-----------|---------------|--------------------|-----------------------|-----------------------|------------------|------------------|
| 1 | 1/03/2017 1:00:00 PM | 25 | FALSE | 31.81102784 | FALSE | FALSE | 38.69016597 | 24.93188972 |
| 2 | 2/03/2017 1:00:00 PM | 26 | FALSE | 31.24656574 | FALSE | FALSE | 36.54559713 | 25.94753434 |
| 3 | 3/03/2017 1:00:00 PM | 28 | FALSE | 30.68210363 | FALSE | FALSE | 33.3910283 | 27.97317896 |
| 4 | 4/03/2017 1:00:00 PM | 30 | FALSE | 30.11764153 | FALSE | FALSE | 31.6235236 | 28.61175945 |
| 5 | 5/03/2017 1:00:00 PM | 87 | TRUE | 29.55317942 | TRUE | FALSE | 31.03083839 | 28.07552045 |
| 6 | 6/03/2017 1:00:00 PM | 32 | FALSE | 28.98871732 | FALSE | FALSE | 32.03011283 | 25.94732118 |
| 7 | 7/03/2017 1:00:00 PM | 26 | FALSE | 27.95985736 | FALSE | FALSE | 29.93931329 | 25.98040143 |
| 8 | 9/03/2017 1:00:00 PM | 28 | FALSE | 26.88055867 | FALSE | FALSE | 28.2245866 | 25.53633073 |
| 9 | 10/03/2017 1:00:00 PM | 25 | FALSE | 25.86804604 | FALSE | FALSE | 27.16144835 | 24.57464374 |
| 10 | 11/03/2017 1:00:00 PM | 24 | FALSE | 25.03537835 | FALSE | FALSE | 26.28714727 | 23.78509943 |
| 11 | 12/03/2017 1:00:00 PM | 22 | FALSE | 24.49561445 | FALSE | FALSE | 27.01618504 | 21.97504386 |
| 12 | 13/03/2017 1:00:00 PM | 23 | FALSE | 24.3618132 | FALSE | FALSE | 25.73724453 | 22.98638187 |
| 13 | 14/03/2017 1:00:00 PM | 27 | FALSE | 24.70529828 | FALSE | FALSE | 27.02294702 | 22.38749555 |
| 14 | 15/03/2017 1:00:00 PM | 29 | FALSE | 25.43045265 | FALSE | FALSE | 29.03569547 | 21.82520983 |
| 15 | 16/03/2017 1:00:00 PM | 77 | TRUE | 26.39992408 | TRUE | FALSE | 27.71992029 | 25.07992788 |
| 16 | 17/03/2017 1:00:00 PM | 25 | FALSE | 27.47636034 | FALSE | FALSE | 29.97748428 | 24.9752364 |
| 17 | 18/03/2017 1:00:00 PM | 26 | FALSE | 28.5224092 | FALSE | FALSE | 31.07004249 | 25.97477591 |
| 18 | 19/03/2017 1:00:00 PM | 32 | FALSE | 29.40071843 | FALSE | FALSE | 32.02599282 | 26.77544405 |
| 19 | 20/03/2017 1:00:00 PM | 34 | TRUE | 29.94334543 | TRUE | FALSE | 31.4405127 | 28.44617816 |
| 20 | 21/03/2017 1:00:00 PM | 31 | FALSE | 30.48597243 | FALSE | FALSE | 32.01027106 | 28.96167381 |
| 21 | 22/03/2017 1:00:00 PM | 26 | FALSE | 31.02859944 | FALSE | FALSE | 36.10748487 | 25.94971401 |
| 22 | 23/03/2017 1:00:00 PM | 28 | FALSE | 31.57122644 | FALSE | FALSE | 35.17816514 | 27.96428774 |
| 23 | 24/03/2017 1:00:00 PM | 36 | TRUE | 32.11385344 | TRUE | FALSE | 33.71954611 | 30.50816077 |

Attention

This is the Preview service so the API can change; what I recommend to make the above code updated is to check the JSON code on the Demo page

Anomaly detector Last API | **Anomaly detector Entire API** | Azure Anomaly Detector | Privacy

Sample 1 | Sample 2 | Sample 3 with seasonality | **Run the Anomaly for Sample data and for entire dataset** | Choose local file

"Entire" will build single model for the whole timeseries and detect anomalies with the model and all the data points.

Start

Value

2238861.0

08-13-2018 | 09-20-2018 | 11-21-2018 | 01-22-2019 | 03-25-2019 | 04-06-2019

Sensitivity (Simulated in client) | 70

"Sensitivity" is from 0 to 99. It defines how sensitive the backend API performs filtering on the result of anomaly detection

Max anomaly ratio | 0.25

"Max ratio" defines the max percentage of anomalies in one detection.

API key Get a key

Endpoint: <https://westus2.api.cognitive.microsoft.com/anomalydetector/v1.0/timeseries/entire/det>

Key: [Redacted]

Put your API Key here

Current request

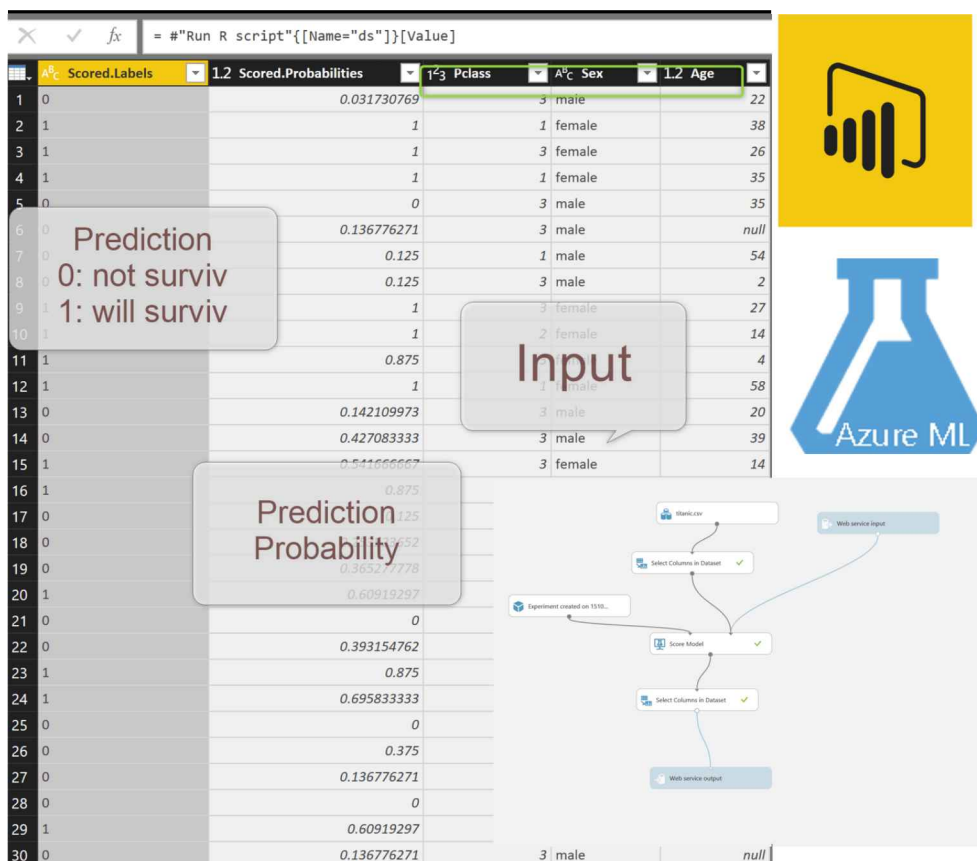
```
{
  "timeseries": [
    {
      "timestamp": "2018-08-13T00:00:00Z", "value": 22324746},
    {
      "timestamp": "2018-08-14T00:00:00Z", "value": 2238861},
    {
      "timestamp": "2018-08-15T00:00:00Z", "value": 2224746},
    {
      "timestamp": "2018-08-16T00:00:00Z", "value": 2238861},
    {
      "timestamp": "2018-08-17T00:00:00Z", "value": 2224746},
    {
      "timestamp": "2018-08-18T00:00:00Z", "value": 2083943},
    {
      "timestamp": "2018-08-19T00:00:00Z", "value": 2084364},
    {
      "timestamp": "2018-08-20T00:00:00Z", "value": 2345557}
    ]
  }
```

Current response

```
{
  "expectedValues": [
    2613398.6833355846,
    2622467.2449678,
    1273652017.24857,
    143268987.5713940154,
    26917031.272237,
    0.15424435,
    2731306.8764318395,
    2740376.7374392436,
    2749446.5984466476,
    2758516.459454052,
    2767586.320461456,
    2776656.1814688602,
    2785726.0424762648,
    2794795.903483669,
    2803865.764491073,
    2812935.6254984774,
    2822005.4865058814,
    2831075.3475132855,
    2840145.20852089,
    2849215.069528094,
    28582
  ]
}
```

Copy the code and compare with what you have in the JSONbody in M code

Chapter 31: Power BI and Azure ML make them work with Power Query



In this chapter, You will learn how to use the Azure ML web service in Power BI (Power Query).

First Step:

You need to create a model in Azure ML Studio and create a web service for it.

The traditional example in Predict a passenger on Titanic ship is going to survived or not?

we have a dataset about passengers like their age, gender, and passenger class, and then we are going to predict whether they are going to survive or not

Open Azure ML Studio and follow the steps to create a model for predicting this.

Navigate to [Azure ML Studio.\[http://studio.azureml.net/\]](http://studio.azureml.net/)

Part 7: Advanced analytics using Power Query

Then download the dataset for titanic from this book's code files.

click on the Database in Azure ML and upload the Titanic dataset from your local machine into the Azure ML.



Then import from your local PC.

×

Upload a new dataset

SELECT THE DATA TO UPLOAD:

E:\ella speak\Europe2018\OneDrive-2018-10

Browse...

☒ This is the new version of an existing dataset

EXISTING DATASET:

titanic.csv

✓

SELECT A TYPE FOR THE NEW DATASET:

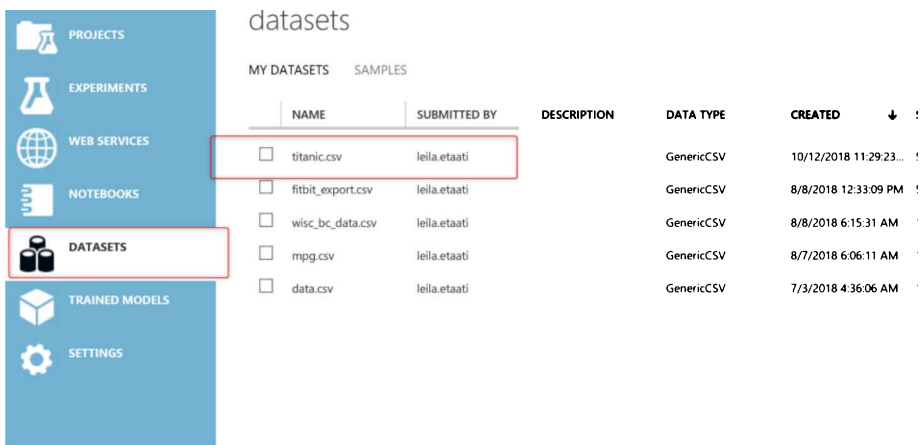
Generic CSV File with a header (.csv)

✓

PROVIDE AN OPTIONAL DESCRIPTION:

✓

Then, you can see the imported dataset in your Azure ML dataset list.

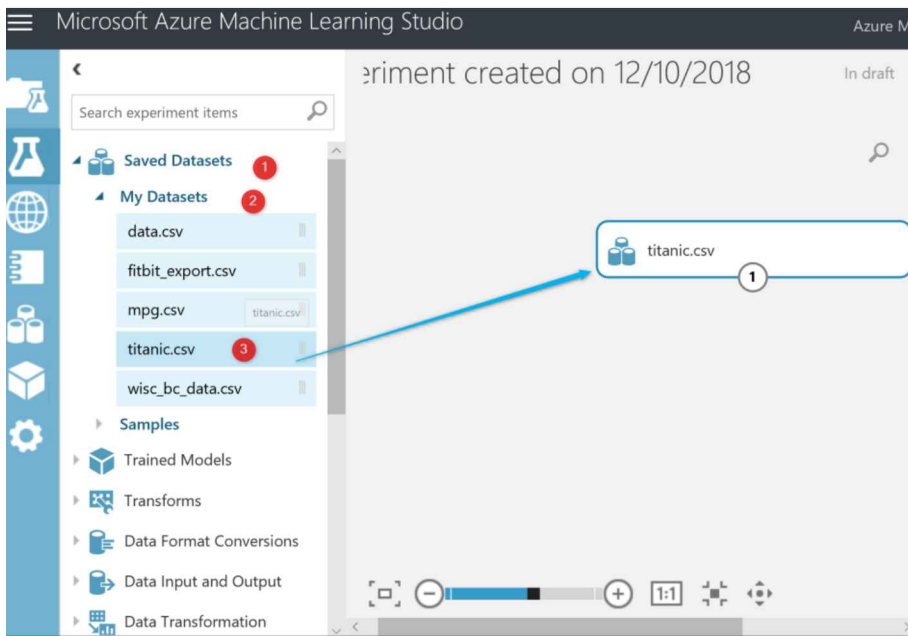


Next, we start to create an experiment in Azure ML by clicking on the Experiment on the left side of the window.



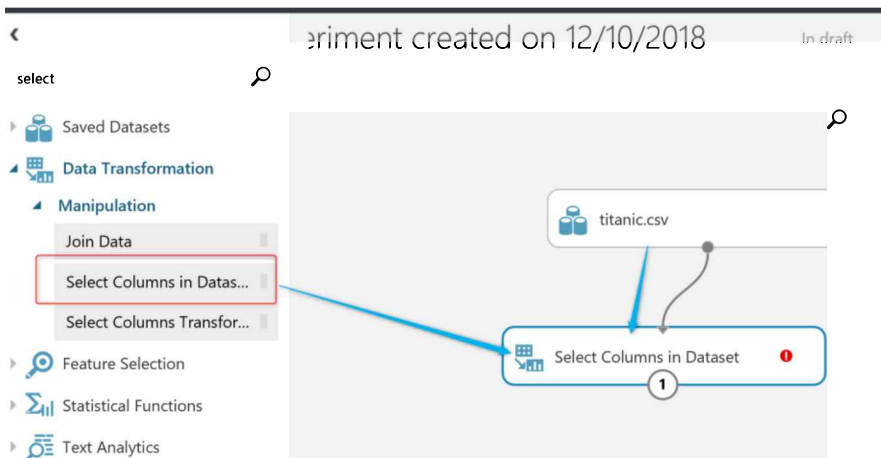
As has been explained before, the Azure ML is a drag and drop environment. in the blank experiment, click on the dataset and choose the Titanic, and drag and drop into the Experiment area.

Part 7: Advanced analytics using Power Query

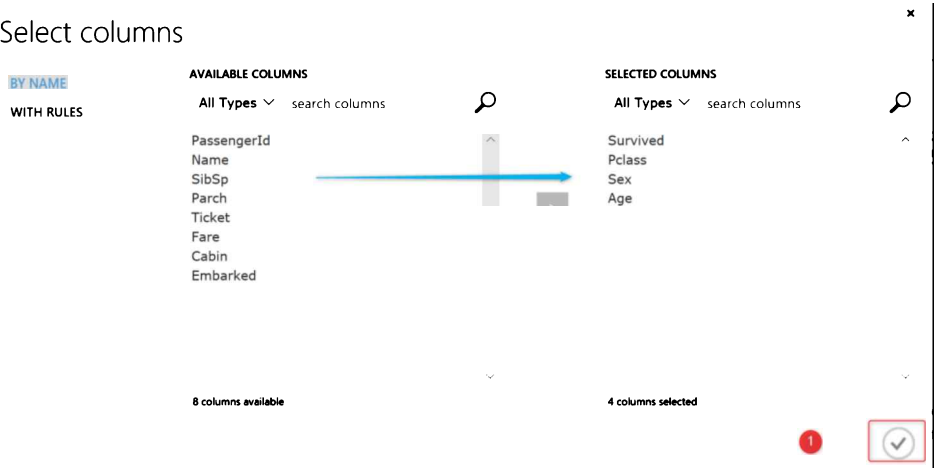


Now you have a dataset in the experiment, and you can create a model there. You can explore the dataset by clicking on the output node and choose the visualization option.

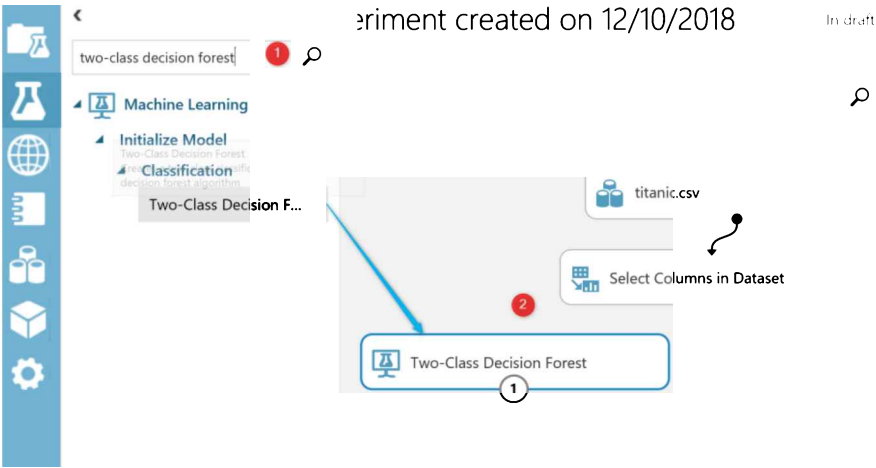
Next, we will select some of the columns, not all of them. We do not need the price, far, cabin, and some descriptive data. We only need Survived, Age, Sex, and Passenger class from all 11 columns. There is a module under the transformation tab on the left side.



Then click on the **Select Columns in Dataset**, and on the right side of the window, there is a property panel. Choose the four required columns.

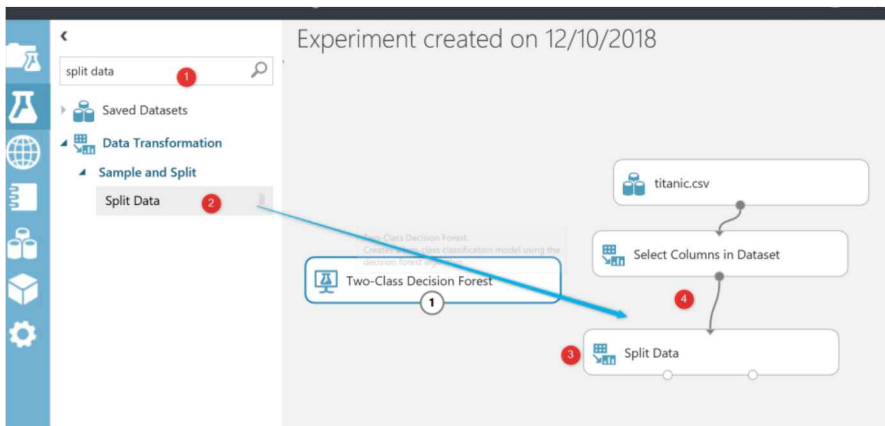


Then, we need to put a model for machine learning on. In this experiment, I choose a model name **two-class decision forest** which is an algorithm that helps me to predict whether it will be survived or not.

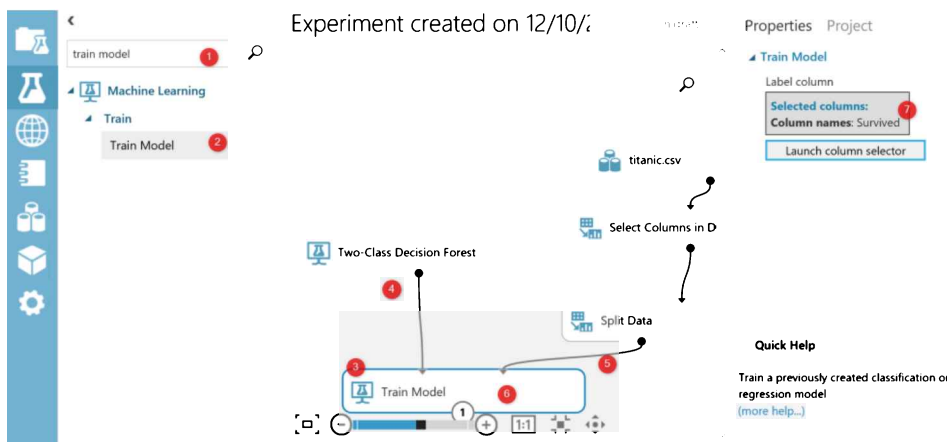


Then based on the machine learning process, we need to split the data into **train and test** datasets.

Part 7: Advanced analytics using Power Query

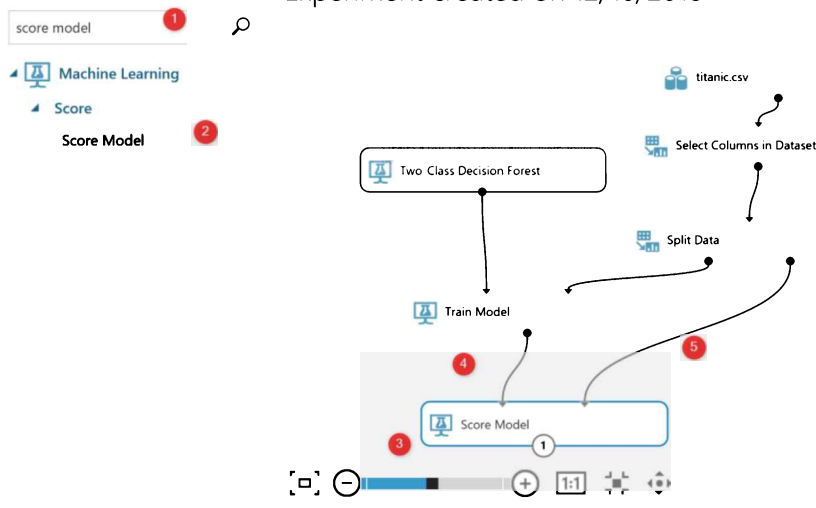


After splitting the data, then you need to train the mode using a module name *Train Model*. Drag and drop this module into the Azure ML experiment area, then connect it as shown in the below picture. Finally, click on the *Train Model* module, and the right side, choose the Select Column to choose the *Survived* column.



Then, we need to test the result using *Score Model*.

Experiment created on 12/10/2018



Now that our simple model is created, we need to run it first.

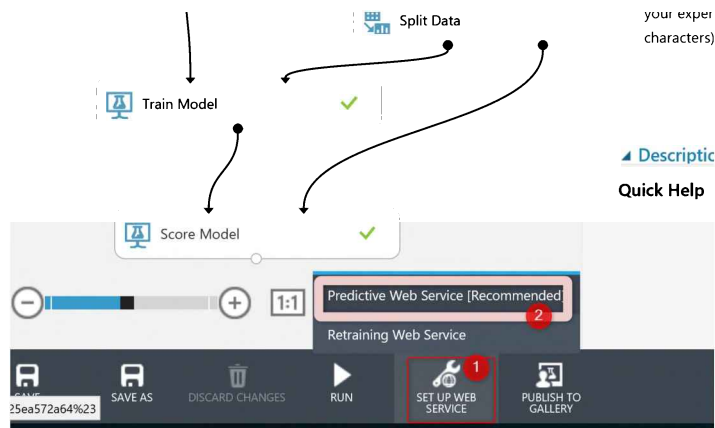


After running the code, we created a web service that we were then using in Power BI (Power Query).

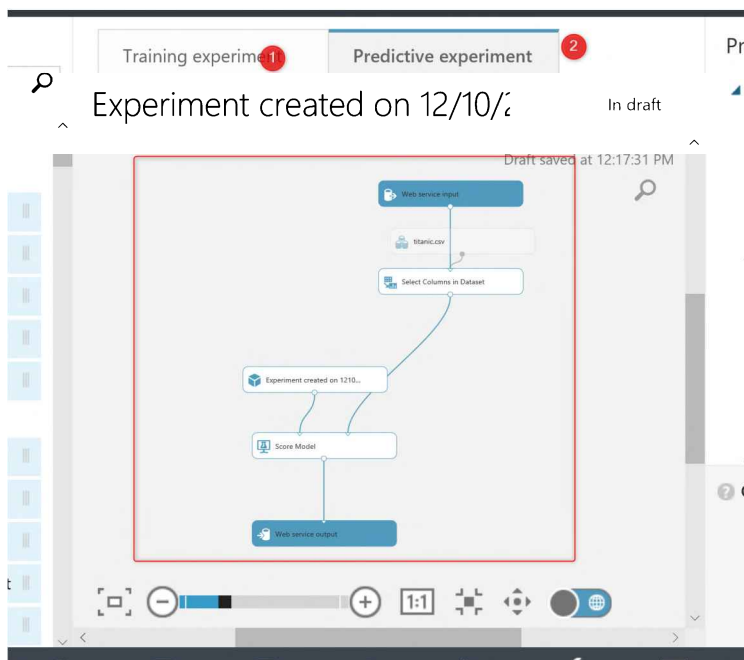
Create Web Service

Now we need to create a web service for this model by clicking on the *Setup Webservice*.

Part 7: Advanced analytics using Power Query



Next, the web service is created, and we have a node for input and another for output.



Then click on the Deploy Webservice option to deploy the web service to be used for prediction. However, before doing that, we need to set up better input and output.

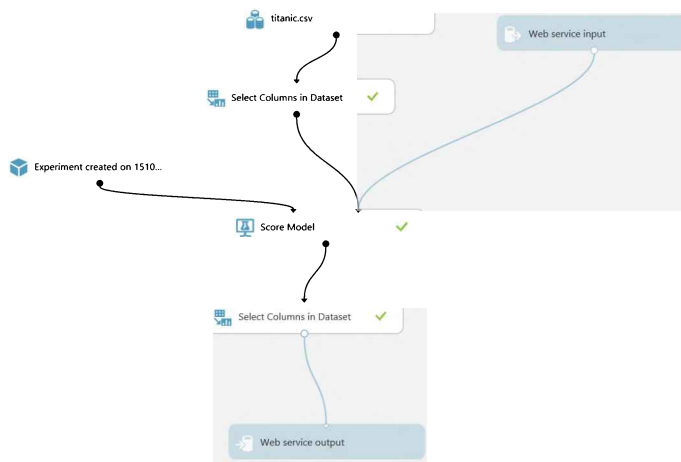
The input should be just;

age, Sex, and Pclass

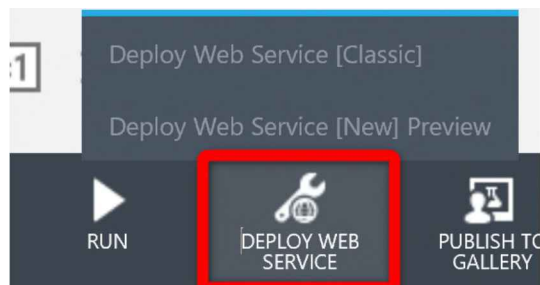
The output should be just: Score and the probability.

To do that, we need to change the experiment a bit by adding the *select column* for both the input and output nodes. To this aim, the final diagram for machine learning looks like the below picture.

Chapter 31: Power BI and Azure ML make them work with Power Query



After creating the diagram, there is a need to run the model again and push the Deploy web Service button.



Next, you have a web service for the titanic problem. Now, you will navigate to a new page that shows detailed information about the web service.

Part 7: Advanced analytics using Power Query

The screenshot shows the 'New Web Services Experience' dashboard in Azure ML. The left sidebar contains navigation icons for Projects, Experiments, Web Services, Notebooks, Datasets, Trained Models, and Settings. The main content area has tabs for 'DASHBOARD' and 'CONFIGURATION'. Under 'DASHBOARD', there's a 'General' section with 'Published experiment' (View snapshot, View latest), 'Description' (No description provided), and 'API key' (blurred). Below is the 'Default Endpoint' section with 'API HELP PAGE', 'REQUEST/RESPONSE', and 'BATCH EXECUTION' links. A 'TEST' button is visible. To the right, the 'APPS' table lists applications with their last updated times.

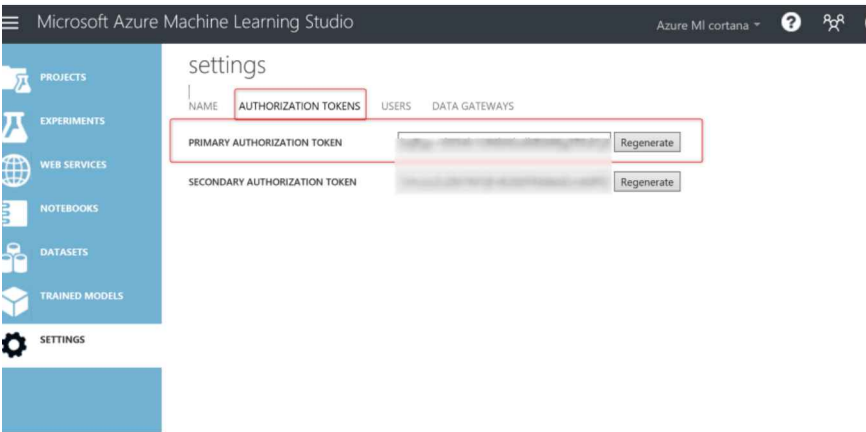
| APPS | LAST UPDATED |
|--|-----------------------|
| Excel 2013 or later Excel 2010 or earlier workbook | 10/15/2018 4:04 54 PM |
| Excel 2013 or later workbook | 10/15/2018 4:04 54 PM |

Now we have the API; the next step is to get some identification from the Azure ML environment.

The first parameter is *Workspace ID*

The screenshot shows the 'settings' page in Azure ML. The left sidebar has navigation icons for Projects, Experiments, Web Services, Notebooks, Datasets, Trained Models, and Settings. The main content area has tabs for 'NAME', 'AUTHORIZATION TOKENS', 'USERS', and 'DATA GATEWAYS'. The 'NAME' tab is active, showing 'WORKSPACE NAME' (Azure ML cortana), 'WORKSPACE DESCRIPTION' (empty text area), 'WORKSPACE TYPE' (Standard, with a 'Learn More' link), 'WORKSPACE ID' (blurred and highlighted with a red box), 'CREATION TIME' (18/10/2017 9:17:27 PM), 'OWNER'S EMAIL' (live.com#leila.etaati@gmail.com), and 'SUBSCRIPTION ID' (blurred).

Go to the Authentication



Then the name of the service.

Power BI

Now we created a model; besides, we have the workspace and authentication id as well.

We are going to apply this model to the Data.

First, import the data into Power BI, Powe Query.

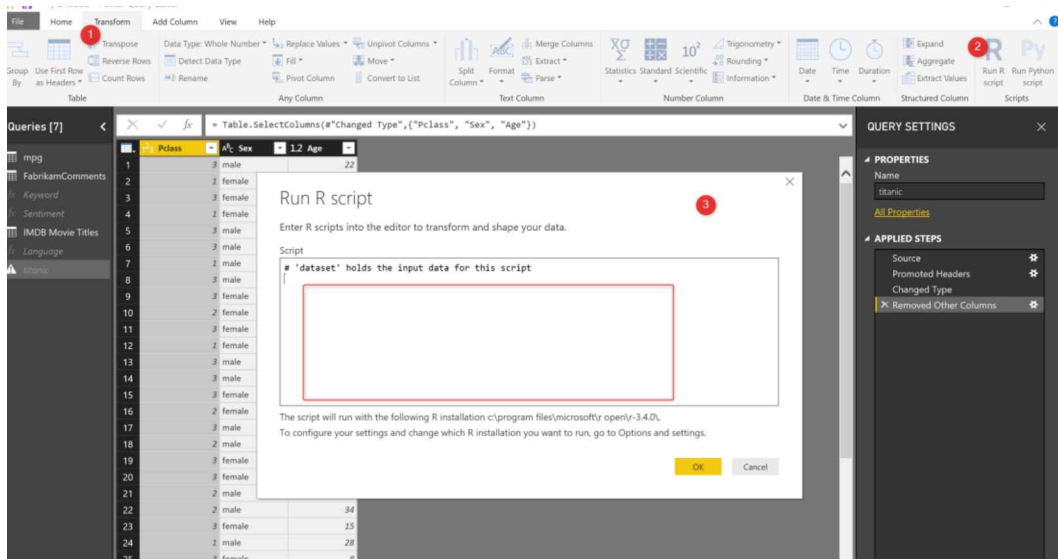
fx

= Table.SelectColumns(#"Changed Type",{"Pclass", "Sex", "Age"})

| Pclass | Sex | Age |
|--------|--------|------|
| 3 | male | 22 |
| 1 | female | 38 |
| 3 | female | 26 |
| 1 | female | 35 |
| 3 | male | 35 |
| 3 | male | null |
| 1 | male | 54 |
| 3 | male | 2 |
| 3 | female | 27 |
| 2 | female | 14 |
| 3 | female | 4 |
| 1 | female | 58 |
| 3 | male | 20 |
| 3 | male | 39 |
| 3 | female | 14 |
| 2 | female | 55 |
| 3 | male | 2 |
| 2 | male | null |
| 3 | female | 31 |
| 3 | female | null |
| 2 | male | 35 |
| 2 | male | 34 |
| 3 | female | 15 |
| 1 | male | 28 |
| 3 | female | 8 |
| 3 | female | 38 |
| 3 | male | null |
| 1 | male | 19 |
| 3 | female | null |
| 3 | male | null |

Part 7: Advanced analytics using Power Query

Now we need to navigate to *Run R Script*,



In the R editor, write the below. The first line is allocating the workspace id from Azure environment to a variable name *wsid*

Next, it is going to store the *Authentication* to the *auth*.

Finally, the web service name will store in the variable name *Service Name*

Next, using some R packages will be used, and the service will be called to connect to the Azure ML environment.

```
wsid = "<Workspace ID from Azure environment >"
```

```
auth = "<Authentication from Azure environment>"
```

```
serviceName = "Titanic Video API [Predictive Exp.]."
```

```
library("AzureML")
```

```
ws <- workspace(wsid,auth)
```

```
ds <- consume(services(ws, name = serviceName),dataset)
```

```
ds<-data.frame(ds,dataset)
```

```
#output<-data.frame(ds$Sentiment,ds$Score)
```

Next, hit the OK bottom. Now you can see the result.

fx = #Run R script"{{Name="ds"}}[Value]

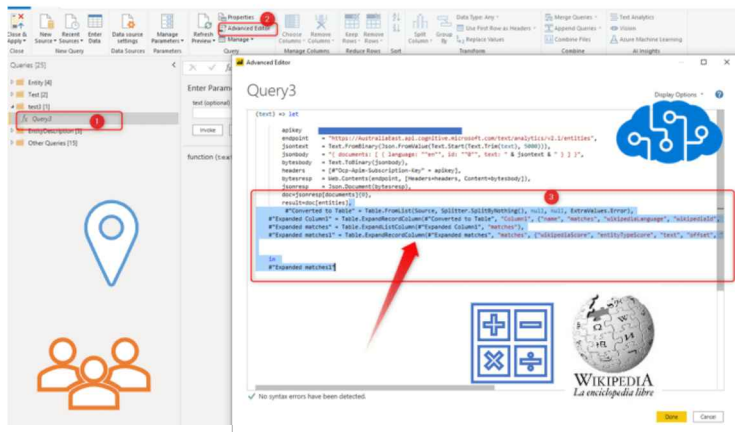
| | 1.0 Scored.Labels | 1.2 Scored.Probabilities | 1.3 Pclass | 1.0 Sex | 1.2 Age |
|----|-------------------|--------------------------|------------|---------|---------|
| 1 | 0 | 0.031730769 | 3 | male | 22 |
| 2 | 1 | 1 | 1 | female | 38 |
| 3 | 1 | 1 | 3 | female | 26 |
| 4 | 1 | 1 | 1 | female | 35 |
| 5 | 0 | 0 | 3 | male | 35 |
| 6 | 0 | 0.136776271 | 3 | male | null |
| 7 | 0 | 0.125 | 1 | male | 54 |
| 8 | 0 | 0.125 | 3 | male | 2 |
| 9 | 1 | 1 | 3 | female | 27 |
| 10 | 1 | 1 | 2 | female | 14 |
| 11 | 1 | 0.875 | 1 | male | 4 |
| 12 | 1 | 1 | 1 | male | 58 |
| 13 | 0 | 0.142109973 | 3 | male | 20 |
| 14 | 0 | 0.427083333 | 3 | male | 39 |
| 15 | 1 | 0.541666667 | 3 | female | 14 |
| 16 | 1 | 0.875 | 2 | female | 55 |
| 17 | 0 | 0.125 | 3 | male | 2 |
| 18 | 0 | 0.365277778 | 2 | male | null |
| 19 | 0 | 0.365277778 | 3 | female | 31 |
| 20 | 1 | 0.60919297 | 3 | female | null |
| 21 | 0 | 0 | 2 | male | 35 |
| 22 | 0 | 0.393154762 | 2 | male | 34 |
| 23 | 1 | 0.875 | 3 | female | 15 |
| 24 | 1 | 0.695833333 | 1 | male | 28 |
| 25 | 0 | 0 | 3 | female | 8 |
| 26 | 0 | 0.375 | 3 | female | 38 |
| 27 | 0 | 0.136776271 | 3 | male | null |
| 28 | 0 | 0 | 1 | male | 19 |
| 29 | 1 | 0.60919297 | 3 | female | null |
| 30 | 0 | 0.136776271 | 3 | male | null |

Summary

In this chapter, you learned how to create a machine learning web service using Azure ML and then applying the web service on a sample dataset using Power Query.

Chapter 32: Text Entity Extraction in Power BI, Text Analytics Service

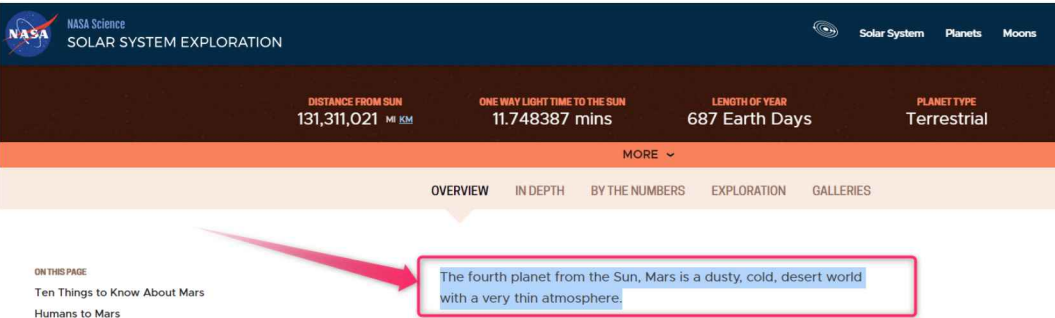
Text Entity Extraction in Power BI, Text Analytics Service



Another service to Microsoft Cognitive Text Analytics API is Text Analytics API, Which can be helpful for Entity Linking and Named Entity Recognition (NER).

Entity Linking: Entity Linking can identify the disambiguate the identity of an entity found in the text (for example, determining whether an occurrence of the word Mars refers to the planet or to the Roman god of war).[<https://docs.microsoft.com/en-us/azure/cognitive-services/text-analytics/how-to-text-analytics-how-to-entity-linking?tabs=version-3>]

Entity Linking refers to the Wikipedia website as the main knowledge base. For example:



The result of the Entity Linking extraction is the link to Wikipedia about Mars and Sun.

The screenshot shows a Power BI table with the following columns: name, wikipediaScore, entityTypeScore, text, wikipediaUrl, type, offset, and length. The table contains four rows of data, corresponding to the entities in the text snippet. A red box highlights the 'type' column, which shows 'Quantity' for 'fourth', 'Other' for 'the Sun', 'Other' for 'Mars', and 'Location' for 'Mars'. A red arrow points from the text snippet in the previous image to this table.

| | name | wikipediaScore | entityTypeScore | text | wikipediaUrl | type | offset | length |
|---|--------|----------------|-----------------|---------|------------------------------------|----------|--------|--------|
| 1 | fourth | 0.271091184 | 0.206359258 | fourth | https://en.wikipedia.org/wiki/Sun | Quantity | 4 | 23 |
| 2 | Sun | 0.271091184 | 0.206359258 | the Sun | https://en.wikipedia.org/wiki/Mars | Other | 32 | 32 |
| 3 | Mars | 0.206359258 | 0.206359258 | Mars | https://en.wikipedia.org/wiki/Mars | Other | 32 | 32 |
| 4 | Mars | 0.206359258 | 0.206359258 | Mars | https://en.wikipedia.org/wiki/Mars | Location | 32 | 32 |

The other service is;

Named Entity Recognition (NER):

This service is also included in the Entity Recognition in Text Analytics that can classify different entities in a sentence to prebuild class as :

Person, Location, Product and Organization, Quantity, and so on

in the above example, it can recognize three different Entity types as **Quantity, Location, and Other...**

How to use it on a Power BI desktop?

To extract the entity, you can use the same Text Analytics API in Microsoft Cognitive Service for Sentiment Analysis, Keyword Extraction, and language detection.

Part 7: Advanced analytics using Power Query

The screenshot displays the Microsoft Azure Cognitive Services interface. On the left, a text input box contains a paragraph about a dinner at Contoso Steakhouse. Below it is a blue 'Analyze' button. On the right, the 'Analyzed text' tab is selected, showing a list of 'NAMED ENTITIES' categorized by sentiment (POSITIVE, NEUTRAL, NEGATIVE). The entities include locations, dates, people, and contact information.

Identify and categorize important concepts

Extract key phrases in unstructured text

We can use the same API and Endpoint for Text Analytics. [<https://radacad.com/cognitive-services-in-power-bi>]

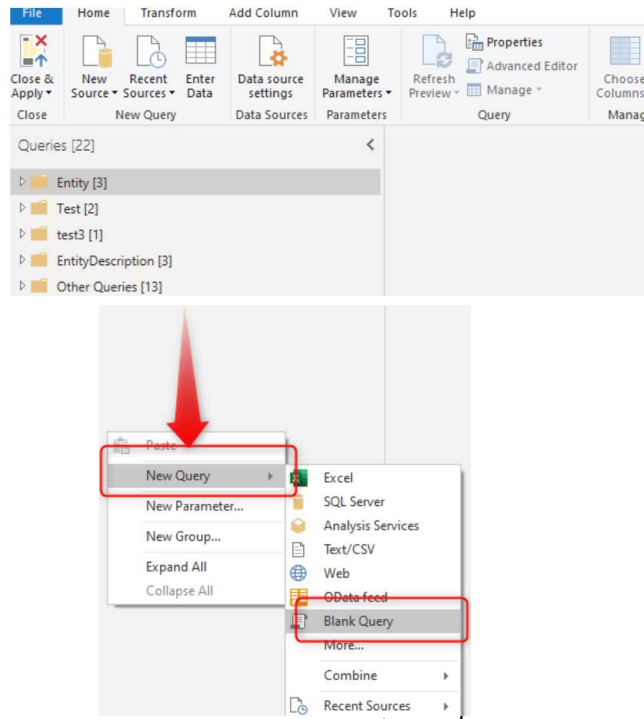
1- Open Power BI desktop.

2- Navigate to Transform Data to Power Query.



3- In the Power Query, click here to create a new blank Query.

Chapter 32: Text Entity Extraction in Power BI, Text Analytics Service

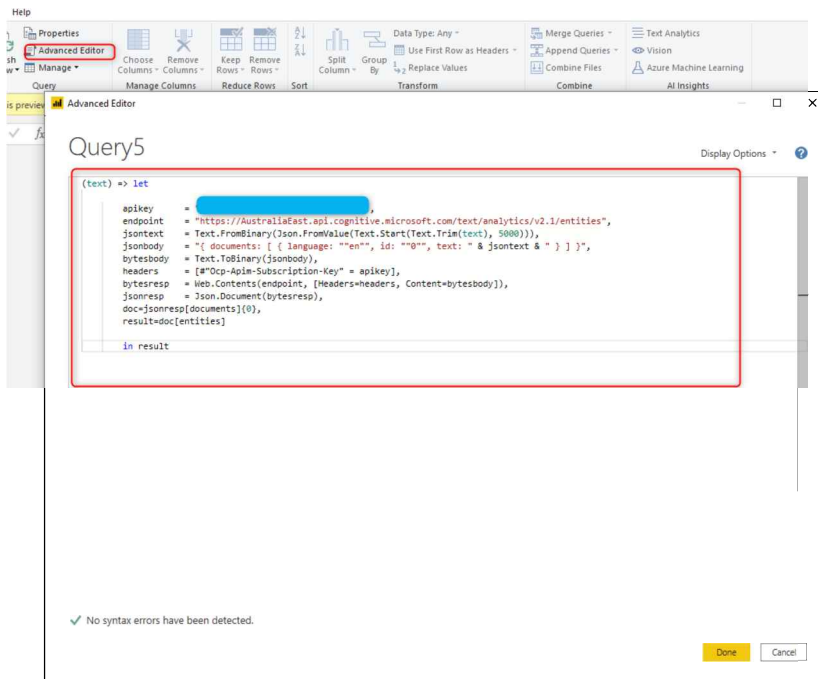


4- in the new Query, click on the Advance Editor, paste the below code.

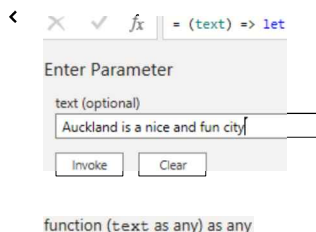
```
(text) => let
    apikey      = "API Key",
    endpoint    =
        "https://AustraliaEast.api.cognitive.microsoft.com/text/analytics/v2.1/entities",
    jsontext    =
        Text.FromBinary(Json.FromValue(Text.Start(Text.Trim(text), 5000))),
    jsonbody    = "{ documents: [ { Language: ""en"", id: ""0"", text: "
    & jsontext & " } ] }",
    bytesbody   = Text.ToBinary(jsonbody),
    headers     = [{"Ocp-Apim-Subscription-Key" = apikey}],
    bytesresp   = Web.Contents(endpoint, [Headers=headers,
    Content=bytesbody]),
    jsonresp    = Json.Document(bytesresp),
    doc=jsonresp[documents]{0},
    result=doc[entities]
```

Part 7: Advanced analytics using Power Query

in result

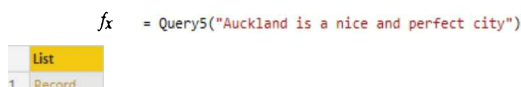


It is a function we use for Text Analytics to connect to the Microsoft Cognitive Service, Text Analytics Service.



Just write, "Auckland is a nice and fun city."

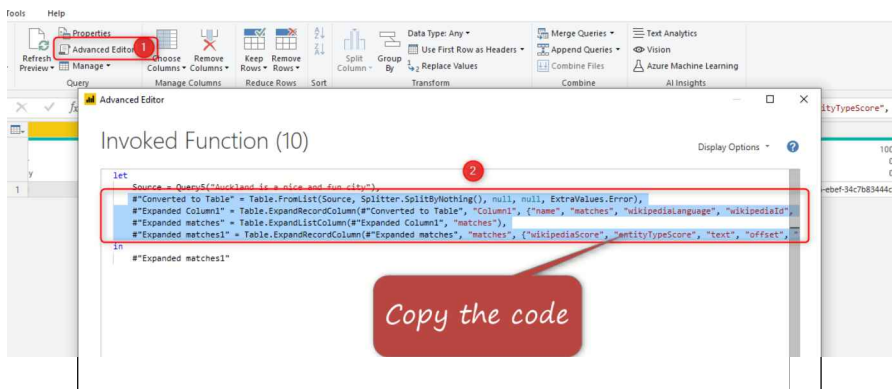
It will return a list as a record.



To make it functional, I need to expand the record and drill down to the small granularity. Click on the Convert to the table, and then expand it to the columns, and then also expand the column named “matches” until you get to the results table.

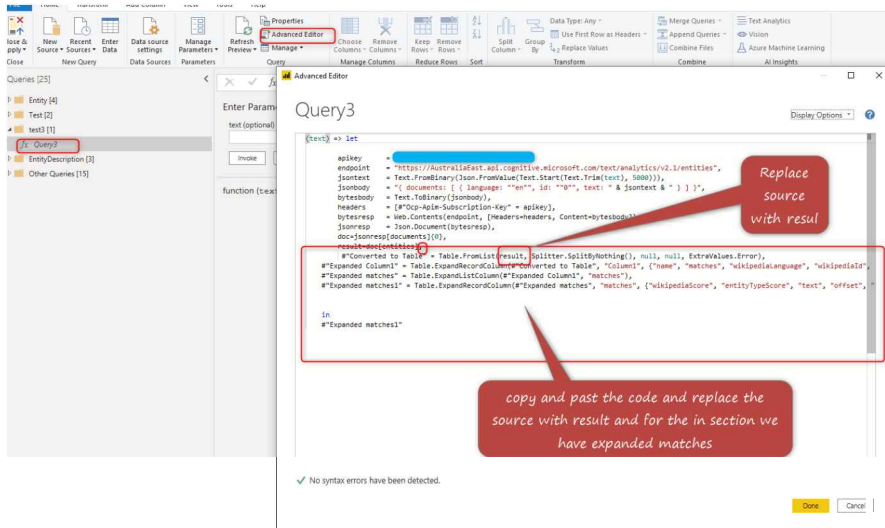
Next, click on Advanced Analytics, and copy the transformation part.

```
,
#"Converted to Table" = Table.FromList(result,
Splitter.SplitByNothing(), null, null, ExtraValues.Error),
#"Expanded Column1" = Table.ExpandRecordColumn("#Converted to Table",
"Column1", {"name", "matches", "wikipediaLanguage", "wikipediaId",
"wikipediaUrl", "bingId", "type"}, {"name", "matches",
"wikipediaLanguage", "wikipediaId", "wikipediaUrl", "bingId", "type"}),
#"Expanded matches" = Table.ExpandListColumn("#Expanded Column1",
"matches"),
#"Expanded matches1" = Table.ExpandRecordColumn("#Expanded matches",
"matches", {"wikipediaScore", "entityTypeScore", "text", "offset",
"length"}, {"wikipediaScore", "entityTypeScore", "text", "offset",
"length"})
in
#"Expanded matches1"
```



Then in the query, click on the Advanced Editor, and replace the code as below

Part 7: Advanced analytics using Power Query



Then just in the editor, test it as below;

“Auckland and Sydney are big cities.”

| Table.ReorderColumns(Source, {\"name\", \"wikipediaScore\", \"entityTypeScore\", \"text\", \"type\", \"wikipediaId\", \"offset\", \"length\", \"wikipediaLanguage\", \"wikipediaId\", \"single\"}) | | | | | | | | | | | |
|--|----------------|-----------------|----------|----------|--------------------------------------|--------|--------|-------------------|-------------|---------------------------------|--|
| name | wikipediaScore | entityTypeScore | text | type | wikipediaId | offset | length | wikipediaLanguage | wikipediaId | single | |
| Auckland | 0.176588426 | 0.86432132 | Auckland | Location | https://en.wikipedia.org/wiki/Sydney | 22 | 11 | en | Sydney | Records-2993-4247-7299-69332632 | |
| Sydney | | | | | | | | | | | |

The result sends a link to Wikipedia and identifies the type of entity as location.

Part 8: Power Query Formula Language: M

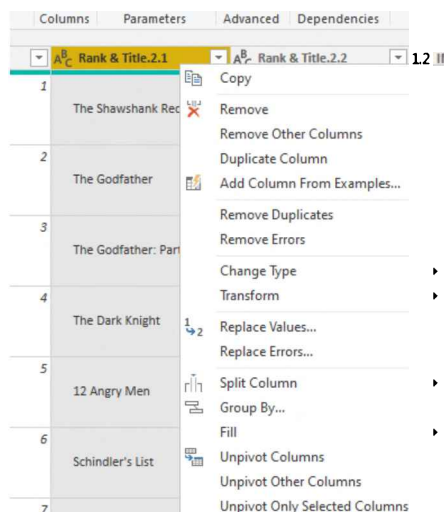
Chapter 33: Power BI Quick Tip: the Formula Bar in Power Query



Often you do the transformation in Power Query using the graphical interface, but having the formula bar visible, makes it much easier to understand or change the transformations. In this short chapter, I'll explain that.

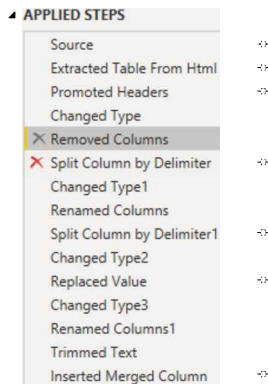
Power Query transformations

In Power BI, we use Power Query a lot for transformations, but mostly using the graphical interface;



the graphical user interface for Power Query

To view or change the transformations, you can always use the Applied Steps. Some steps have a great icon beside them, which makes it easier to apply a change.



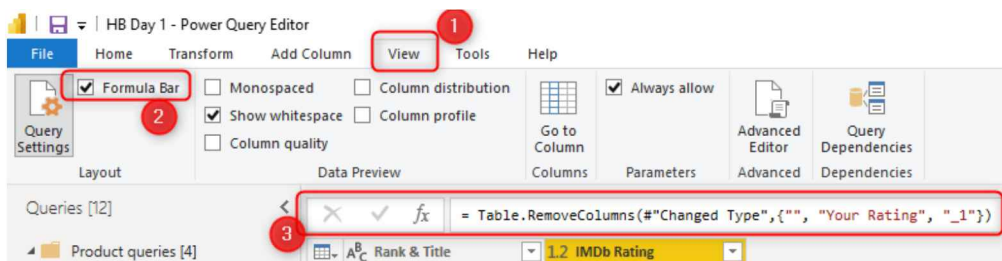
Applied steps in Power Query editor

However, some steps don't have the gear icon, making it hard to understand them or even change the task done there. For example, in the above screenshot, it is not easy to find out what columns have been removed (unless you go to the previous steps and compare what columns are missing in this step manually), let aside to change the configuration.

Power Query Formula Bar

Power Query formula bar is a very useful place to understand the action applied in each step and also change it if it is needed.

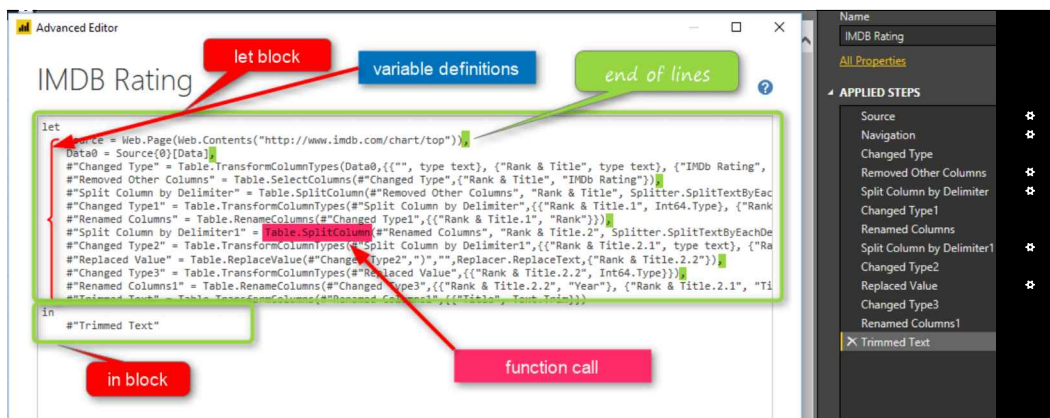
To make the formula bar visible, go to the View tab, and check the Formula Bar;



making the formula bar of the Power Query editor visible

When the formula bar is visible, each task's action is visible there. You don't have to be an M deep-dive developer to understand that. You might need some help in some areas if you are modifying things, but even the formula bar itself there would help you to understand the transformation better.

Chapter 34: Basics of M: Power Query Formula Language



M is the powerful language behind the scene of Power Query. Any transformation you apply will be written in M language. For many, M looks like a scary language. In this chapter, I like to explain a bit of the basic of M., Not mentioning any functions. Mainly I want to explain to you how the M language syntax is structured. Once you know the syntax, then everything becomes simple. M is a language that you can learn its syntax easily. As a Power Query developer, I highly recommend you to spend time on M because there are MANY operations that you can do with M, but you might not be able to do it simply with the graphical interface.

What is M?

M is the informal name of Power Query Formula Language. The formal name is so long that no one uses that. Everyone calls it M! M stands for Data Mashup. Some say it stands for Data Modeling. M is a functional language, and it is important to know its functions it. However, each language has a structure and syntax, the beginner level of learning that language. In this chapter, I will explain the syntax of M. Before learning M; I would like you to read this sentence loud;

M is much more powerful than the graphical interface of Power Query

Yes, you read it correctly! The graphical interface of Power Query is changing every month. Every month new functionality comes to this graphical interface. But the fact is all of these functionalities have been in the language for many years! If you knew the

language, you could easily use them instead of waiting for the graphical interface option. There are heaps of examples for it. One very small example is [here\[https://radacad.com/grouping-in-power-query-getting-the-last-item-in-each-group\]](https://radacad.com/grouping-in-power-query-getting-the-last-item-in-each-group): you can extend your Grouping ability in Power Query with a minor change in M script.

Syntax of M

The syntax of this language is simple. It always has two blocks of programming: LET expression block and IN expression block. Here is the most simple M syntax;

let

x=1

in

x

Let and **in** are reserved words. before going even one step further, the first and foremost thing you need to know;

M (Power Query Formula Language) is Case Sensitive. There is a difference between x and X.

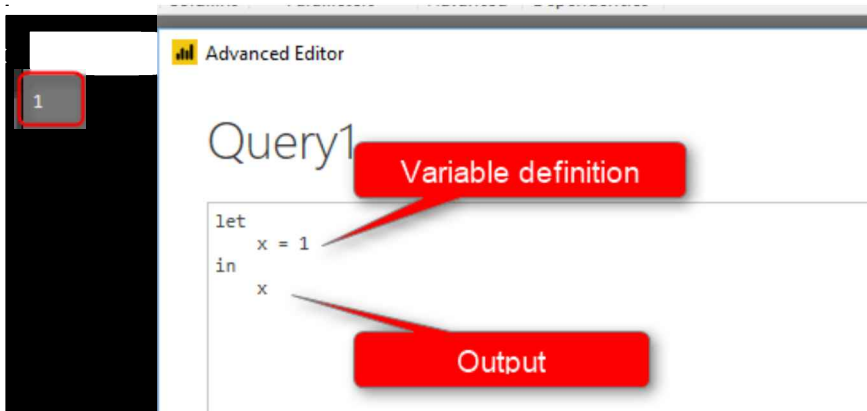
what are these two programming blocks:

let: definition of all variables

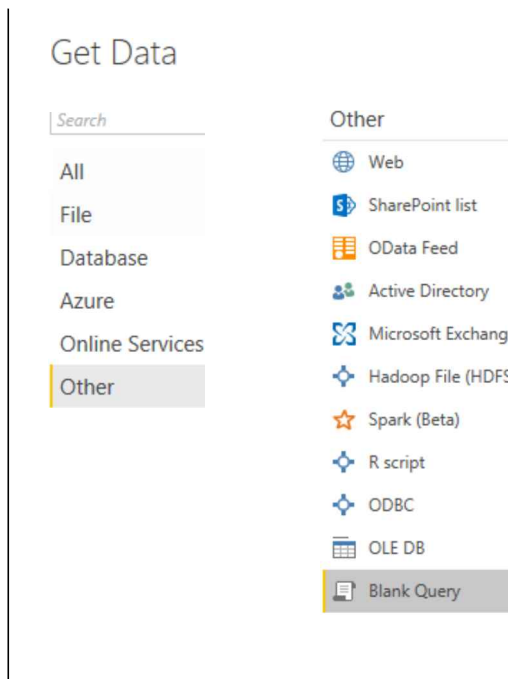
In: output! Yes, it means out! Just named as in. everything you put in this block will be the output of your query.

So basically, the query below means defining a variable named as x, assigning the value 1 to it, and showing it as the result set. So the query will return 1.

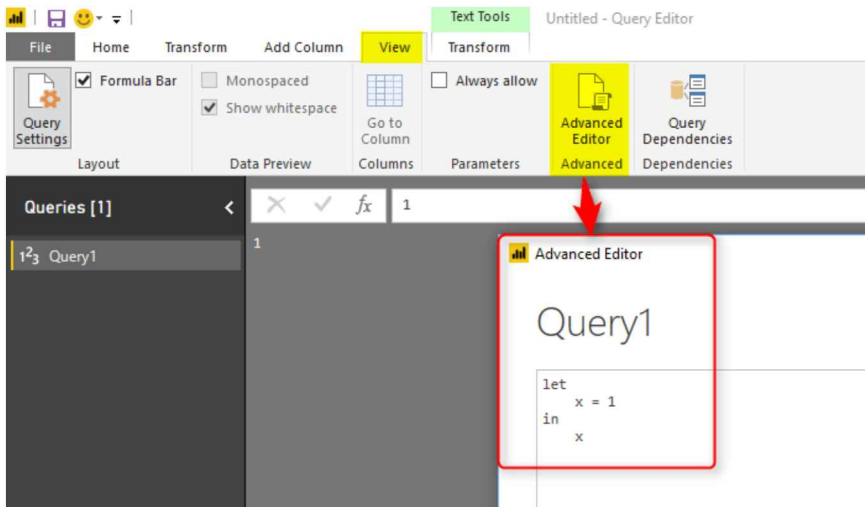
Part 8: Power Query Formula Language: M



To run this example, you need to Open the Power BI Desktop. Go to getting Data, start with New Blank Query.



then in the View tab, select advanced Editor;



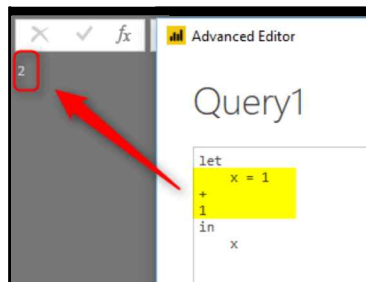
Make sure when you write the script that you put reserved words such as `let` and `in` all lowercase. Also, your variable name should be the same case in both `let` and `section`.

As you can see, there is no need to define data types for a variable. It will be automatically assigned when the first assignment occurs.

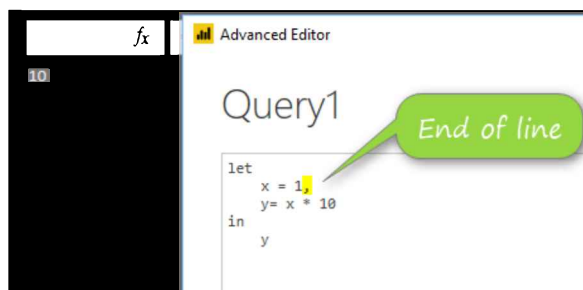
If you specify a text value, then the variable would be a text data type automatically.

End of the Line

Lines of codes in M continue if you don't put the end of the line character.



As you can see in the above example, the line continues, and `x` will be equal to `x=1+1`. If you want to put an end to a line, use comma(`,`). Example here:

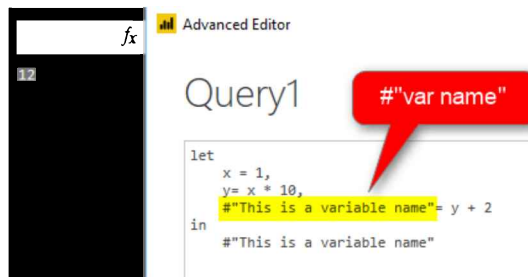


Every line needs a comma(,) to finish. Except for the last line before **in**.

Variable Names

The name of variables can be all one word, like Source. It can have spaces in it. In case that you have some characters, such as space, then you need to put the name inside a double quote (") and put a hashtag at the beginning of it(#). something similar to:

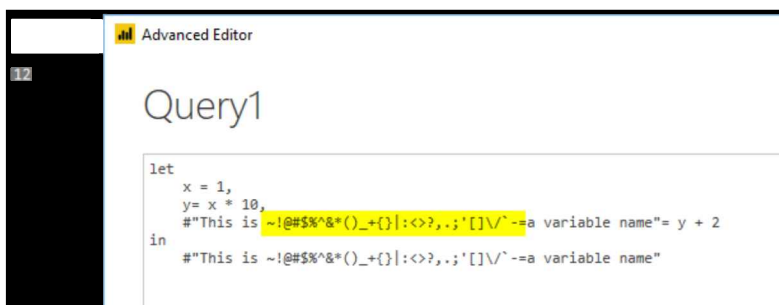
```
#"This is a variable name."
```



The variable name can contain special characters; here is an example:

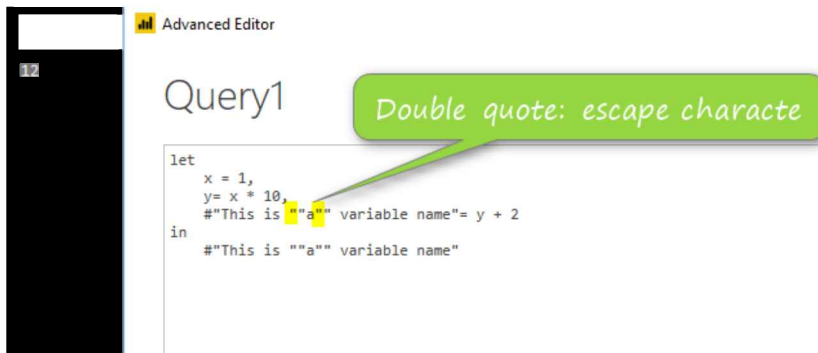
Special characters

Variable names can have a special character; as you can see below, a variable has all types of characters in it and still runs well.



Escape character

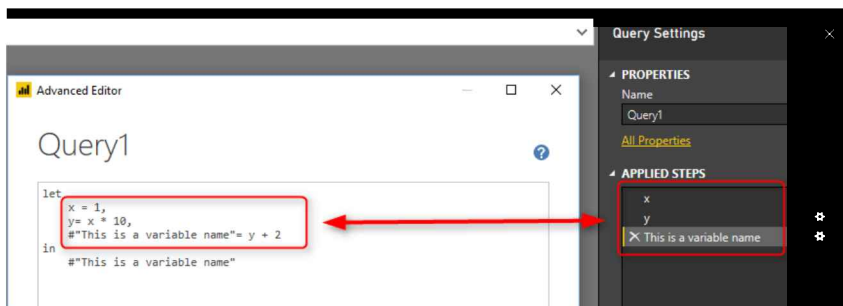
Double quote (") is escape character. You can use it to define variables with names that have another double quote in them. Here is an example:



The first double quote (highlighted) above must be before the second double quote (which is part of the variable name).

Step by Step Coding

Power Query is a step-by-step transformation. Every transformation usually happens in a step. While you are writing the code, you can also notice that on the right-hand side, you will see every variable form a step.



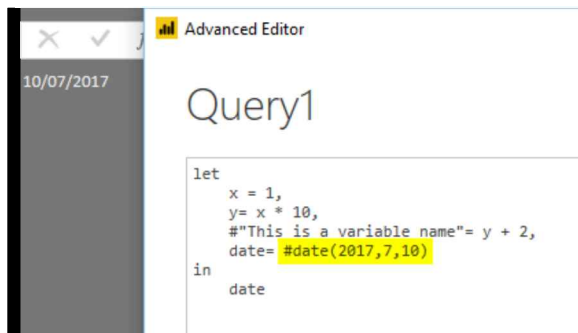
In the screenshot above, you can see every variable is determined as a step. And if the variable has space in the name, it will show it with spaces in the list of applied steps.

The last variable is always specified in the **in** section.

Literals

There are different ways of defining every literal in Power Query. For example, if you want to define a date variable, here is how to do it;

Part 8: Power Query Formula Language: M

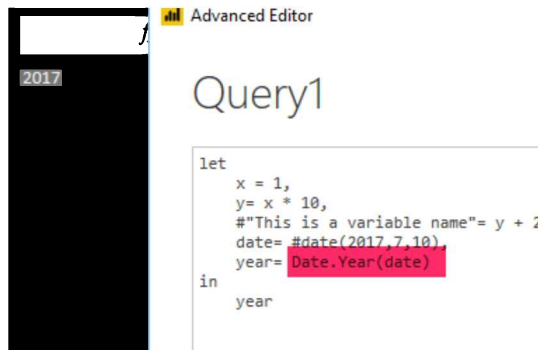


for defining all other types of literals, here is the reference table to use:

| Kind | Literal |
|---------------------|---|
| <i>Null</i> | null |
| <i>Logical</i> | true false |
| <i>Number</i> | 0 1 -1 1.5 2.3e-5 |
| <i>Time</i> | #time(09,15,00) |
| <i>Date</i> | #date(2013,02,26) |
| <i>DateTime</i> | #datetime(2013,02,26, 09,15,00) |
| <i>DateTimeZone</i> | #datetimezone(2013,02,26, 09,15,00, 09,00) |
| <i>Duration</i> | #duration(0,1,30,0) |
| <i>Text</i> | "hello" |
| <i>Binary</i> | #binary("AQID") |
| <i>List</i> | {1, 2, 3} |
| <i>Record</i> | [A = 1, B = 2] |
| <i>Table</i> | #table({"X","Y"},{{0,1},{1,0}}) |
| <i>Function</i> | (x) => x + 1 |
| <i>Type</i> | type { number } type table [A = any, B = text] |

Function Call

M is a functional language, and for doing almost everything, you need to call a function for it. Functions can be easily called with the name of the function and specifying parameters for it.



The screenshot above uses Date.Year function, which fetches the year part of a date. Functions names always start with capital letters: **Date.Year()**

Comments

like any programming language, you can put some comments in your code. it can be in two forms;

Single line commentary with a double slash (//)

```
let
// this is a comment line and will not be executed
  x = 1,
  y = x * 10,
  #\"This is a variable name\" = y + 2,
  date = #date(2017, 7, 10),
  year = Date.Year(date)
in
  year
```

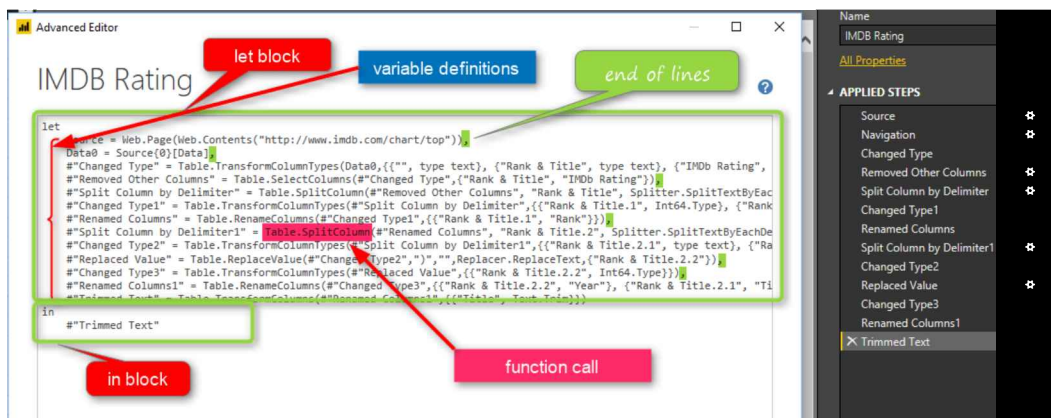
Multi-line commentary between slash and stars (/* comments */)

```
let
/* this is a
multi
lin
comment and will not be executed
*/
  x = 1,
  y = x * 10,
  #\"This is a variable name\" = y + 2,
  date = #date(2017, 7, 10),
  year = Date.Year(date)
in
  year
```

A real-world example

Now that you know some basics, let's look at an existing query in advanced editor mode and understand it.

Part 8: Power Query Formula Language: M



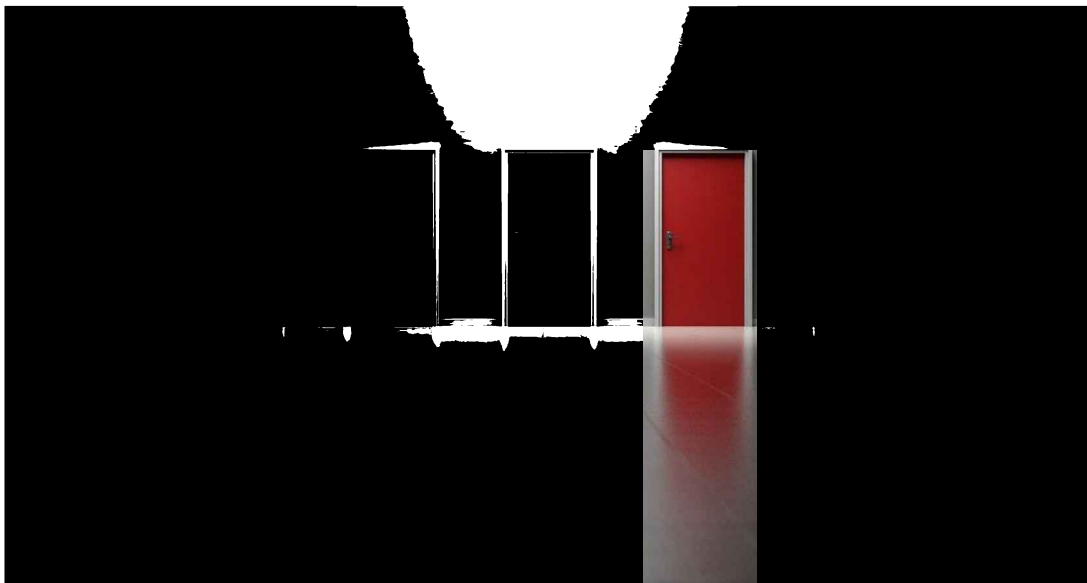
in the screenshot above, you can see all basics mentioned so far:

1. let and in block
2. variable names matching steps applied in the query
3. some variable names with a hashtag and double quote: `"#var name."`
4. end of the line characters: comma
5. calling many functions

There are still many parts of this query that you might not understand, especially when using functions. You need to learn what functions are doing to understand the code fully. I have written a chapter that explains how to use the `#shared` keyword to get documentation of all functions in Power Query.

In the next chapters, I'll explain other levels of structures in M.

Chapter 35: M or DAX? That is the Question!



“What is the main difference between M and DAX? Why can we do a calculated column in two different places? What are the pros and cons of each? Which one should I use for creating a profit column? Why two different languages?! Why structure of these two are so different?” If any of these are your questions, then you need to read this chapter. In this chapter, I’ll go through the differences between these two languages. I will explain why, when, where.

What is M?

M is the scripting language behind the scene for Power Query. M is the informal name of this language. The formal name is Power Query Formula Language! This is a long name, and even Microsoft refers to it as M. M stands for many things, but one of its most common words is Mashup. This means this language is capable of data mashup and transformation. M is a functional language. The structure of the M script can be similar to this:

```
let
    FirstAndLastDayOfTheMonth = (date) =>
        let
            dated=Date.FromText(date),
            year=Date.Year(dated),
            month=Date.Month(dated),
            FirstDateText=Text.From(year)&"-"&Text.From(month)&"-01",
            FirstDate=Date.FromText(FirstDateText),
            daysInMonth=Date.DaysInMonth(dated),
            LastDateText=Text.From(year)&"-"&Text.From(month)&"-"&Text.From(daysInMonth),
            LastDate=Date.FromText(LastDateText),
            record=Record.AddField([], "First Date of Month", FirstDate),
            resultset=Record.AddField(record, "Last Date of Month", LastDate)
        in
            resultset
in
    FirstAndLastDayOfTheMonth("30/07/2015")
```

M is a step-by-step language structure. Usually (Not always), every line in the M script is a data transformation step. And the step after that will use the result of the previous step. It is usually easy to follow the structure of the M language for a programmer. Because it is understandable with programming blocks of Let and In and some other programming language features alike.

What is DAX?

DAX is Data Analysis eXpression language. This is the common language between SQL Server Analysis Services Tabular, Power BI, and Power Pivot in Excel. DAX is an expression language, and unlike M, it is very similar to Excel functions. DAX has many functions in common with Excel. However, DAX is much more potent than the Excel formula in many ways. Here is an example DAX expression:

```
Sales Rolling 12 Months =
CALCULATE(
    SUM(FactInternetSales[SalesAmount]),
    DATESBETWEEN(
        DimDate[FullDateAlternateKey],
        NEXTDAY(SAMEPERIODLASTYEAR(LASTDATE(DimDate[FullDateAlternateKey]))),
        LASTDATE(DimDate[FullDateAlternateKey])
    ),
    ALL(DimDate)
)
```

DAX calculations are built in a way that makes sense mainly for Excel users. Usually, Excel users are very comfortable with this language. Everything goes through functions. DAX doesn't have programming blocks in it and combines function uses, filters, and expressions.

Example Usage of M

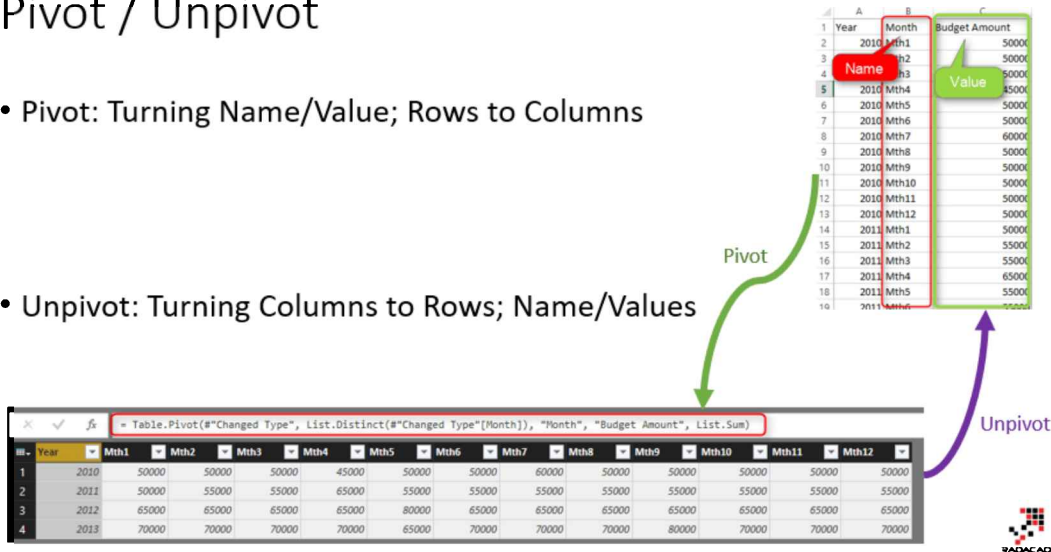
M can be used in many data transformation scenarios. For example, it can be used to Pivot or Unpivot Data, To [Group](https://radacad.com/grouping-in-power-query-getting-314)<https://radacad.com/grouping-in-power-query-getting-314>

[the-last-item-in-each-group](#) is based on some columns. Here is how a [Pivot/Unpivot](https://radacad.com/pivot-and-unpivot-with-power-bi)[\[https://radacad.com/pivot-and-unpivot-with-power-bi\]](https://radacad.com/pivot-and-unpivot-with-power-bi) can work in Power Query;

Pivot / Unpivot

- Pivot: Turning Name/Value; Rows to Columns

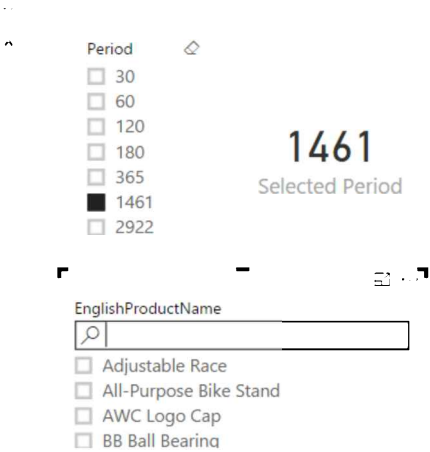
- Unpivot: Turning Columns to Rows; Name/Values



Example Usage of DAX?

DAX can be used for many calculations for analyzing data. For example, calculating Year To Date, Calculating [Rolling 12 Months Average](https://radacad.com/secret-of-time-intelligence-functions-in-power-bi)[\[https://radacad.com/secret-of-time-intelligence-functions-in-power-bi\]](https://radacad.com/secret-of-time-intelligence-functions-in-power-bi), or anything like that. Here is an example which based on selection criteria in the report and few simple DAX expressions, we can do a [customer retention](https://radacad.com/lost-customers-dax-calculation-for-power-bi)[\[https://radacad.com/lost-customers-dax-calculation-for-power-bi\]](https://radacad.com/lost-customers-dax-calculation-for-power-bi) case with DAX;

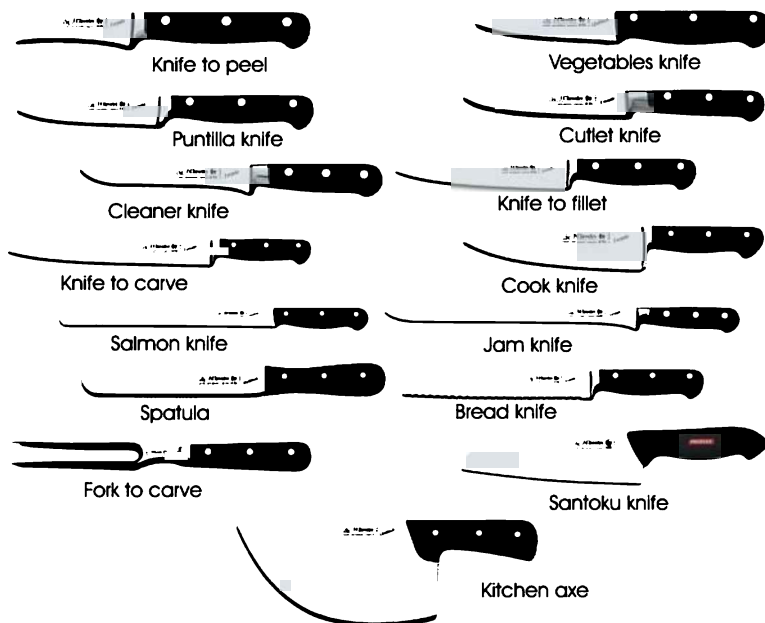
| FullName | Total Revenue | Last Period Revenue | Lost Customers | New Customers |
|-----------------|---------------|---------------------|----------------|---------------|
| Aaron Adams | \$118 | \$118 | 0 | 1 |
| Aaron Alexander | \$70 | \$70 | 0 | 1 |
| Aaron Allen | \$3,400 | | 1 | 0 |
| Aaron Baker | \$1,751 | \$1,751 | 0 | 1 |
| Aaron Bryant | \$134 | \$134 | 0 | 1 |
| Aaron Butler | \$15 | \$15 | 0 | 1 |
| Aaron Campbell | \$1,155 | \$1,155 | 0 | 1 |
| Aaron Carter | \$40 | \$40 | 0 | 1 |
| Aaron Chen | \$40 | \$40 | 0 | 1 |
| Aaron Coleman | \$62 | \$62 | 0 | 1 |
| Aaron Collins | \$6,047 | \$2,469 | 0 | 0 |
| Aaron Diaz | \$6,030 | \$2,451 | 0 | 0 |
| Aaron Edwards | \$94 | \$94 | 0 | 1 |
| Aaron Evans | \$2,433 | \$2,433 | 0 | 1 |



Calculated Column Dilemma

In my opinion, the main question of choosing between DAX and M comes from the calculated column dilemma. You can create many calculated columns in both M or DAX, and it is confusing where is the best place to do it or why there are two different places to do it?! For example, you can create a full name concatenated of the FirstName and LastName columns. You can do that in M and also in DAX. So this question comes up that: Why two different places? Which one is best to use? can we always use one language?

To answer this question, I would like to use another example; There are many types of knives, and you can use almost all of them to cut cheese!



reference: [here\[http://forkitchen.blogspot.co.nz/2008/10/what-are-different-types-of-kitchen.html\]](http://forkitchen.blogspot.co.nz/2008/10/what-are-different-types-of-kitchen.html)

Almost every knife in the above picture can be used for cutting cheese except one of them! So why are there so many knives for cutting cheese?! The answer is that; these are not knives for cutting the cheese! Each knife is good for doing one particular case. For cutting bread, a bread knife gives you the best result. For cutting a fillet, you usually need another type of knife. But as you agree, you can use many of these knives for some cases (such as cutting the cheese!). Let's now go back to the original question;

Why can I create the same calculated column in DAX or M?

These two languages are built independently. They are made in a way that they can handle most business-related solutions. So, as a result, there are some use cases that both languages can do it. For example, both of these languages can easily create a concatenated column of two other columns.

Which one is best?

The quick answer is It depends! It depends on the type of usage. If you want to create a concatenated column, Power Query (M) is a better option in my view because that usually is like the ETL part of your BI solution. You can build your model and data sets in the way you like them to be. But if you want to create something like Year To Date, you can do that in Power Query or M, but it will be lots of code, and you have to consider many combinations of possibilities to create a correct result.

In contrast, in DAX, you can create that with the usage of the TotalYTD function. So the answer is; there is no best language between these two. The type of usage identifies which one is best. Usually, any changes for data preparation are best to be done in M. Any analysis calculation on top of the model is best to be done in DAX.

Two Languages for Two different Purposes

There are many programming languages in the world. Each language has its pros and cons. JavaScript is a language of web scripting, which is very different from ASP.NET or PHP. The same thing happens here. When M was born, it meant to be a language for data transformation, and it is still that language. DAX was created to answer business analysis questions.

What Questions Can DAX Answer?

DAX is the analytical engine in Power BI. It is the best language to answer analytical questions which their responses will be different based on the selection criteria in the report. For example; You might want to calculate the Rolling 12 Months Average of Sales. It is tough to calculate that in M because you have to consider all different types of possibilities; Rolling 12 months for each product, every customer, every combination, etc. However, if you use a DAX calculation for it, the analytical engine of DAX takes care of all different combinations selected through Filter Context in the report.

What Questions Can M Answer?

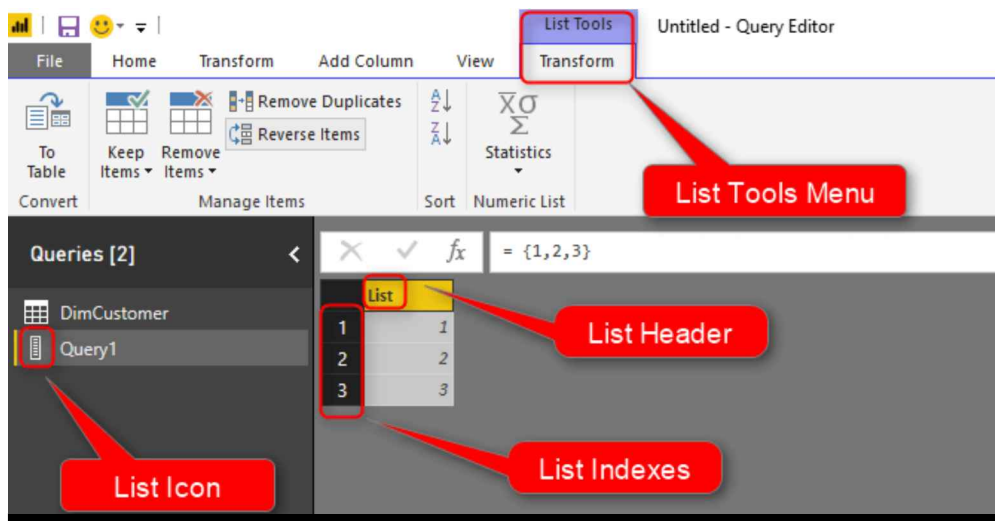
M is the language of data transformation in Power BI. You can use M for doing any data preparation and data transformation before loading that into your model. Instead of bringing three tables of DimProduct, DimProductSubcategory, and DimProductCategory, you can merge them all in Power Query and create a single DimProduct including all columns from these tables, and load that into the model. If you decide to Load all of

these into the model and use DAX to relate these, it will consume extra memory for something that is not required to be in the model. M can combine those three tables, and based on the “Step Based” operational structure of M, they can be used to create a final data set.

As a Power BI Developer, Which Language Is Important to Learn?

Both! With no hesitation! M is your ETL language, and DAX is the analytical language. You cannot live with only one. If you want to be an expert in Power BI, you should be an expert in both of these languages. You will need to understand both languages to understand which one is best for which purpose and easily use it in real-world scenarios.

Chapter 36: Basics of Value Structures in M – Power Query Formula Language



In this chapter, I'm going to the next step and will explain few other structure definitions in this language. In this chapter, you will learn about Tables, Records, Lists, and how to navigate structures. These structures are the main value structures in Power Query and M. Every data value in Power Query is in one of these value structures, and you must have the ability to work with these structures.

Prerequisite

To understand parts of code from this chapter, you might need first to read earlier chapters of this part of the book.

Five Main Value Structures in Power Query

Power Query has five structures for values. Data is either in one of these five structure types. By structure type, I don't mean the data type. I mean the way that data is stored. Sometimes data is stored as a simple value like text, date, or number. Sometimes it is a complex value like a table, record, list, or function.

Primitive Value

Any single part data type is considered as a primitive value. examples:

12 – number value

"text sample" – text value

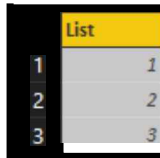
2017/09/21 – date value

Part 8: Power Query Formula Language: M

null – null value

List

A list is a structure that has only one column but multiple rows. Each row is identified with an index. Example of a list in Power Query window;



| | List |
|---|------|
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |

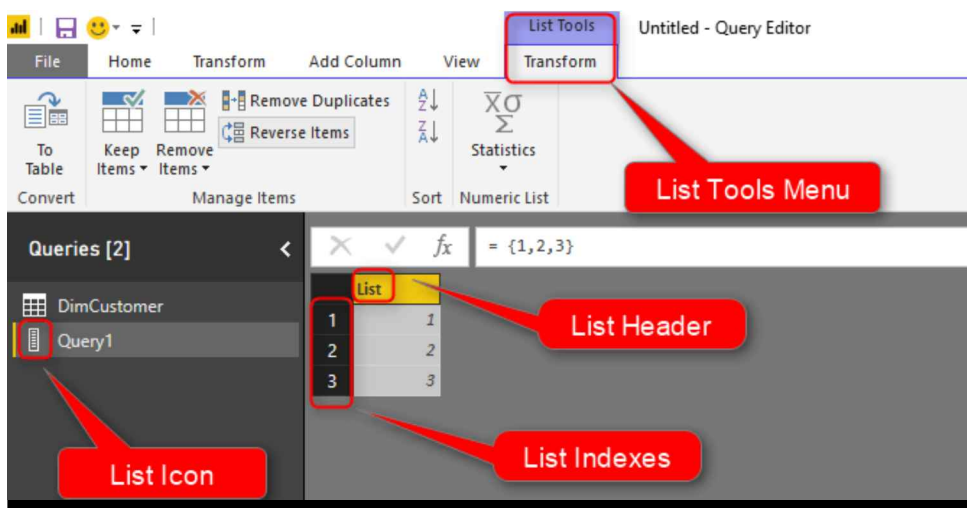
The M script to define a list is as below;

```
Source = {1,2,3}
```

List Definition is always started with { and ends with }, items placed in between with a comma separator;

```
List = {<item 1>,<item 2>,<item 3>}
```

There are some ways to understand if a structure is a list or not. In the screenshot below, all mentioned;



- List Icon: There will be a specific icon for the list in the Queries pane.
- List Tools: When you select a list, you will see the List Tools menu. This menu gives you some options later on for changing and transforming the list.
- List Header: At the top of the list column, you will see the “List” name.

- List Indexes: Every row in the list should have a numeric index that starts from zero.

A list can have items that are different in data types. Here is an example;

The screenshot shows the Power Query formula bar with the formula `= {1, "text value", #date(2017,9,21)}`. Below the formula bar, a preview of the list is shown as a vertical table with three rows. The first row has a yellow header 'list' and a value '1'. The second row has a value 'text value'. The third row has a value '21/09/2017'.

| list |
|------------|
| 1 |
| text value |
| 21/09/2017 |

Here is the line definition for this list;

Source = {1,"text value", #date(2017,9,21)}

Record

A record is a structure with a single row but multiple columns. However, the way that record is showed in Query editor is vertical! The main reason is that scrolling to the right is always harder than scrolling down. So, Record is only visualized similarly to the list. However, it is a different structure. This is how a record looks like in Power Query;

The screenshot shows the Power Query formula bar with the formula `= [Column 1=1,Column 2=2]`. Below the formula bar, a preview of the record is shown as a vertical table with two columns. The first column has a yellow header 'Column 1' and a value '1'. The second column has a header 'Column 2' and a value '2'.

| Column 1 | Column 2 |
|----------|----------|
| 1 | 2 |

As you can see, the record showed vertically, but every column header is visible there.

Here is the script for defining the record;

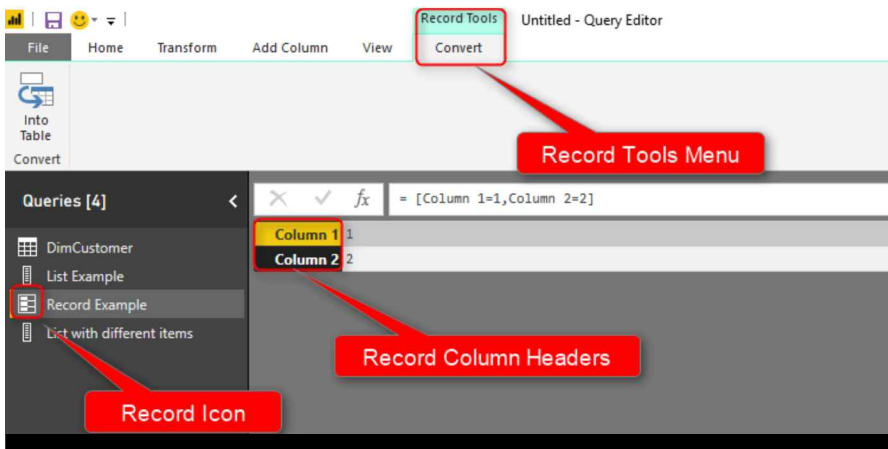
Source = [Column 1=1,Column 2=2]

Record definition always starts with [and ends with]. for every column. You will have the column name before = sign and the value of that after the = sign.

Record = [Column 1 = <value>, Column 2 = <value>]

The screenshot below shows how you can identify that the structure is a record;

Part 8: Power Query Formula Language: M



- Record Icon: There is a specific Icon that determines the object is a Record.
- Record Column Headers: You can see column headers in the record. In a list, you can see only numbers, but in a record, you see column names.
- Record Tools Menu: Every time you select a record object, you will see Record Tools which gives you the option to convert it to a table.

A record also can have different types of items. It is in different columns.

| | |
|---------------|------------|
| Column Text | text value |
| Column Number | 12 |

Here is the definition of the record above;

Source = [Column Text="text value", Column Number=12]

Table

A table is a structure that is most well known among others. A table simply is a combination of multiple rows and multiple columns. Here is a table sample;

| | Column A | Column B |
|---|----------|----------|
| 1 | 1 | 10 |
| 2 | 2 | 20 |

To create a table through M script, you can run a script such as below;

```
Source = #table(  
    {"Column A", "Column B"},  
    {
```



```
{1,10},
{2,20}
}
)
```

Table definition always starts with #table, then inside the bracket, you have to set of brackets; one set for defining headers, and the other set for all row values. Here is how this works;

```
Source = #table(
  {"Column A","Column B"}, // all column headers
  { // start of row values
    {1,10}, // row one
    {2,20} // row two
  } // end of row values
)
```

If you see a table, then you can recognize it immediately because the table is the only structure that has multiple rows and multiple columns in it.

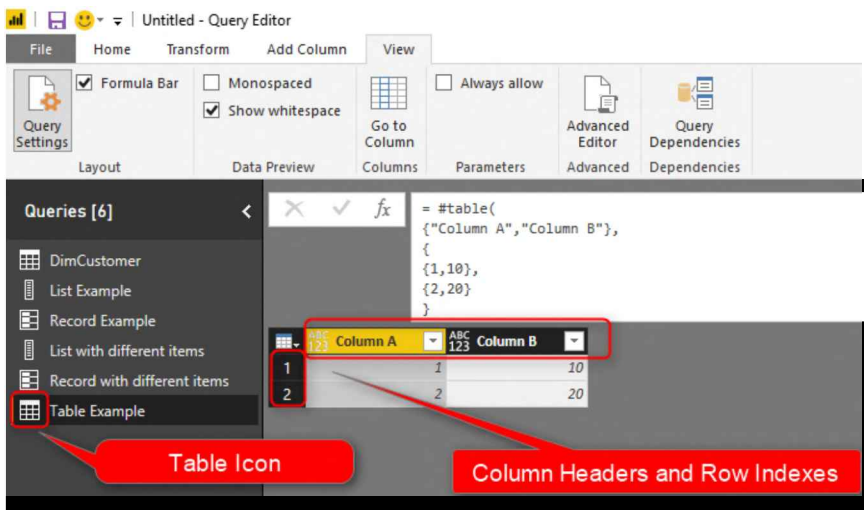
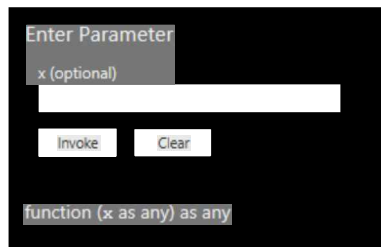


Table Icons shows that this is a table, and multiple columns and rows are only possible in a table.

You can also have a table with different items in each cell.

Function

A function is a data type that performs an operation and gives you a result. Here is an example of how a function looks like in the query editor window;



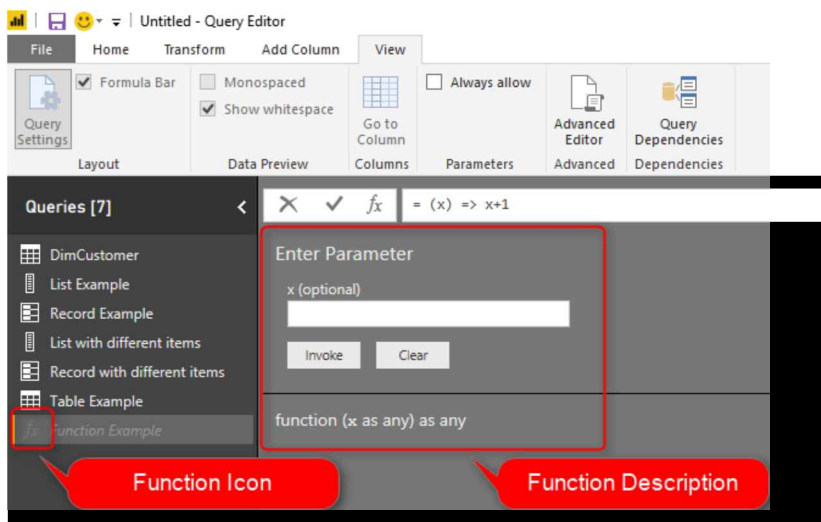
Function can be defined based on Lambda language as below;

Source = (x) => x+1

Function Definition has many details in it. I'll talk about it separately in another chapter. For now, just knowing this is enough that function has an input and output. Input and output are separated from each other with => signs.

Source = (<input of function>) => <output of function>

You can easily understand a function based on its specific icon and also the function call dialog box.



A function is one of the most powerful features in Power Query and can't be explained only in few paragraphs. Stay tuned, and I'll write more details about functions in other chapters separately.

Navigating Through List

Now that you know what a list is and how to define it let's look at how you can navigate through the list. Each list item has a row index. You can easily navigate to that item using that index.

let

Source = {1,2,3},

Source1 = *Source*{1}

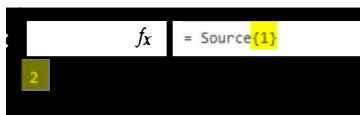
in

Source1

For navigating through a list, simply use a bracket and put the item's index in the bracket. Now here is the tricky part:

Index starts from zero

When you look at the list in the query editor window, the index starts from 1. However, the actual index starts from zero. So if you want to drill down to a specific item in the list, you have to use a zero-based index to get to that. The example above will navigate to the **second item** in the list.



So this is simply the syntax to navigate through the list;

List{<item index starting from zero>}

List Functions

There are also many functions that work on a list. For example, you can get the count of items in a list with a `List.Count()` function.



I will write more about List functions in the future separately.

Navigating Through Record

To navigate through a record, you need to use the column name for that record.

Part 8: Power Query Formula Language: M

let

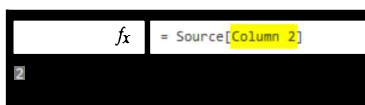
```
Source = [Column 1=1,Column 2=2],
```

```
#"Column 2" = Source[Column 2]
```

in

```
#"Column 2"
```

You can simply put the column name inside bracket [and], and as a result, you will have the item;



Record Functions

The Record also has a lot of functions. For example, you can use Record.FieldCount() to get the count of columns in a record.



Navigating Through Table

Navigating through a table happens a lot in real-world scenarios when you want to drill down into a specific area of the table. For the table, you can use different methods to navigate;

Navigate with Record Index: Drill Down to Record

You can navigate to any records in the table simply with putting the record's index (zero-based) inside { and }. Here is an example;

let

```
Source = #table(
```

```
{"Column A","Column B"},
```

```
{
```

```
{1,10},
```

```
{2,20}
```

```
}
```

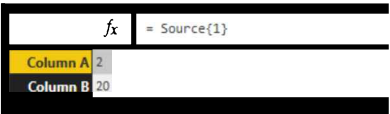
```
),
```

```
record=Source{1}
```

in

record

This would bring the second row of the table as a Record structure;



The screenshot shows the Power Query formula bar with the formula `= Source{1}`. Below the formula bar, a preview of the data is shown as a table with two columns: 'Column A' and 'Column B'. The second row is highlighted, showing the values 2 and 20 respectively.

| Column A | Column B |
|----------|----------|
| 2 | 20 |

Navigate with Column Name: Drill Down to List

You can also fetch every column of the table by referring to that column name inside [and]. The result would be a list.

let

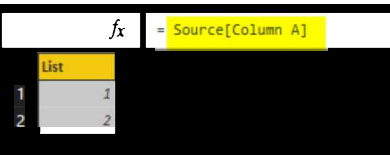
```
Source = #table(  
{"Column A","Column B"},  
{  
{1,10},  
{2,20}  
}  
,  
),
```

```
#"Column A" = Source[Column A]
```

in

```
#"Column A"
```

Navigating to a column will result in a list;



The screenshot shows the Power Query formula bar with the formula `= Source[Column A]`. Below the formula bar, a preview of the data is shown as a table with two columns: 'List' and 'Column A'. The first column 'List' contains the values 1 and 2, and the second column 'Column A' contains the values 1 and 2.

| List | Column A |
|------|----------|
| 1 | 1 |
| 2 | 2 |

Navigate with Row Index and Column Name: Drill Down to individual Value

Sometimes you want to drill down to a specific cell; you need row index (zero-based) and column name both.

let

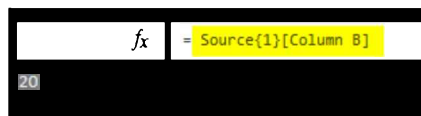
```
Source = #table(  

```

Part 8: Power Query Formula Language: M

```
{"Column A","Column B"},  
{  
  {1,10},  
  {2,20}  
}  
,  
#"Column B" = Source{1}[Column B]  
in  
#"Column B"
```

The script above navigates to the second row (Index 1 belongs to the second row), and column B. Result would be a primitive value in this case;



You can also do this navigation the other way around. Navigate to column first and then to row. An example is below;



Navigate with Filter Criteria

For the majority of the cases, you want to navigate based on criteria. For example, you want to get the Column B value of the row that Column A value for that row is something specific. Like writing a SQL Statement clause. In the table below. For example, we want to navigate to column B, where column A is 1. You cannot do that search based on an index. However, you can search based on criteria.

| | ABC 123 Column A | ABC 123 Column B |
|---|------------------------|------------------------|
| 1 | 1 | 10 |
| 2 | 2 | 20 |

In the table below. For example, we want to navigate to column B, where column A is 1. You cannot do that search based on an index. However, you can search based on criteria.

Here is how you can search;

let

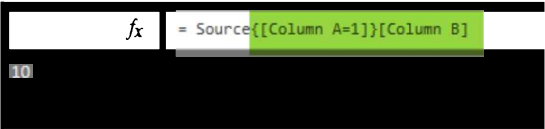
328



<https://radacad.com>

```
Source = #table(  
  {"Column A", "Column B"},  
  {  
    {1,10},  
    {2,20}  
  }  
,  
  filtered=Source{[Column A=1]}[Column B]  
in  
  filtered
```

The result would be 10 as expected;



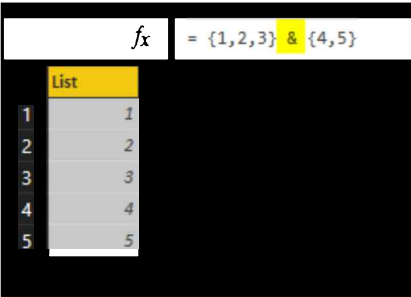
The structure simply is as below;

```
= <Table Name>{[Column Name="criteria 1", Column Name="criteria 2"]}[Output Column]
```

This is similar to SQL Where and Select Clause.

Concatenate List or Records

You can concatenate lists or records together with ampersand (&) character. Here is an example for list concatenation;



and here is an example for record concatenation;

| | | |
|----------|---|--|
| fx | | = [Column 1=1,Column 2=2] & [Column 3=3] |
| Column 1 | 1 | |
| Column 2 | 2 | |
| Column 3 | 3 | |

For concatenating tables, you need to use either Append or Merge. I explained fully in detail about Append and Merged earlier in this book

Summary

In this chapter, you learned some basics about Power BI's different value structures: Primitive values, list, record, table, and function. You learned how to navigate through lists, records, and tables with different methods with M script. In future chapters, I'll explain more details about each value structure and functions related to that structure.

Chapter 37: Writing Custom Functions in Power Query M



One of the most powerful features of M is that you can write custom functions to re-use part of your code. Custom functions can be single or multiple liners; they will be written in Lambda-style syntax. In this chapter, you will learn how to write functions and invoke them in M.

In the earlier parts of this book, you learned how to create a custom function in Power Query using the graphical interface of the Power Query Editor. However, it is helpful to understand the M structure of a custom function. This comes in handy when you want to do something which is not supported through the UI.

Basic Syntax

Functions in M can be written in this format:

$(x) \Rightarrow x+1$

This is lambda syntax (that was previously used in LINQ if you come from a .NET development background). the line above is equal to the pseudo-code below:

Function anonymous (x)

```
{
return x+1
}
```

As you see, lambda made it much simpler to define the function with just that single line. So the function above gets a parameter from input and adds 1 to it. Please note

Part 8: Power Query Formula Language: M

that the datatype of the parameter is not defined, so that means if a text is an input, then an error would occur, so you would require to do error handling as well.

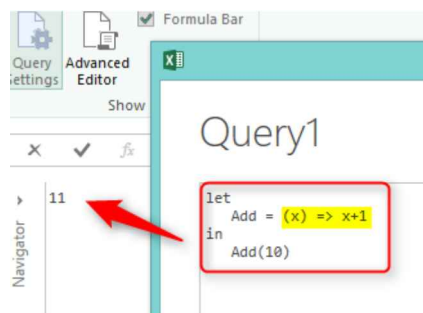
So let's see how this function works and how we can call this function. script below shows how to define the function and invoke it with a parameter:

```
let
    Add = (x) => x+1
in
    Add(10)
```

Creating inline function

To create this script, open Power Query Editor, create a blank query, go to Advanced Editor, and replace the script with the script above.

The result of the above expression is: 11



In the example above, we've named the function "Add," and then we simply call that with Add(inputparam).

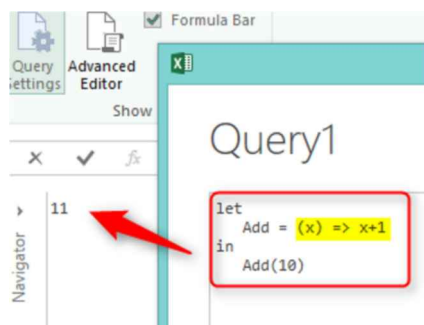
The screenshot below shows how the lambda expression works in defining function:

Lambda Expression

$(x) \Rightarrow x+1$

Input Parameters \Rightarrow Function Body

$(x,y) \Rightarrow x+y$



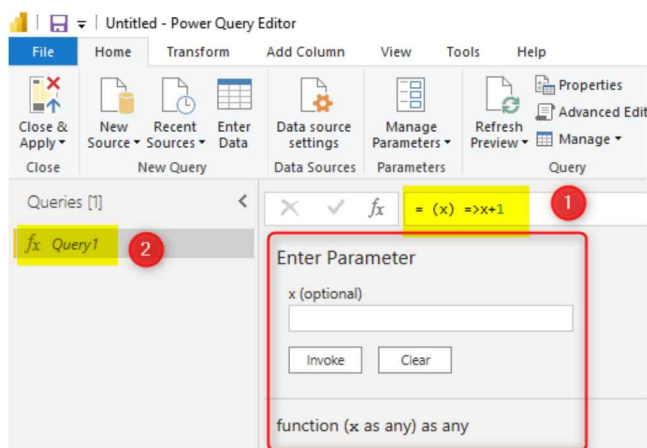
Creating Standalone function

If you write a script as mentioned in the above example, your whole query won't be a function. It would be a query with an inline function in it. What if you want the Add function to be its function so that you can re-use it in your Power Query Editor from other queries. Here is how you can do it.

Create a new blank Query, and then in the formula bar type in the body of the function only as below;

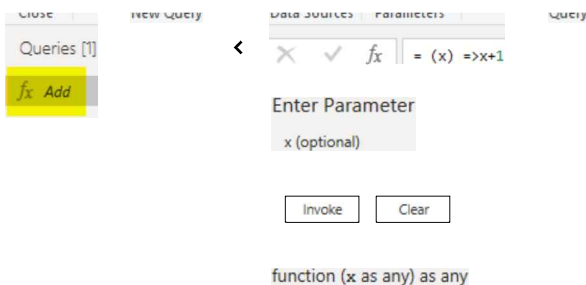
`= (x)=>x+1`

Note that the first equal sign is needed; otherwise, your script will be treated as a text instead of a function. The result will be this:



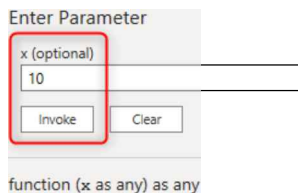
As you can see, the query type changed to be a function. You can see the *fx* beside it, and you can see the definition of the function in the graphical interface.

You can now rename this function to whatever you want, such as Add.



Part 8: Power Query Formula Language: M

To invoke this function, you can either create a new blank query and use Add(x), which x is a number, or simply use the UI and then click on Invoke, which will do the same thing.



The Invoke will create a new blank query with the script to invoke the function;



As you can see, there is not much difference between defining inline functions or standalone functions. The main body of the function is the same. Through the rest of this chapter, a combination of them is used, but you will see mostly inline functions through the examples.

Parameters

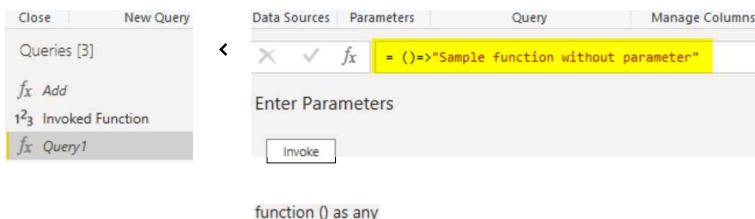
If you want to define a function with more parameters, then simply add parameters as below:

$(x,y,z) \Rightarrow x+y+z$

you can also define a function without parameters, as below:

$= () \Rightarrow \text{"sample function without parameter"}$

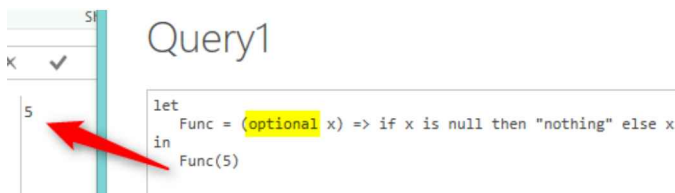
This function gets no parameters and returns the text "sample function without parameter."



Optional Parameters

parameters are defined as Required by default. This means you should specify the parameter at the time of invoking the function. So if you want to define an optional parameter, use the Optional keyword as below:

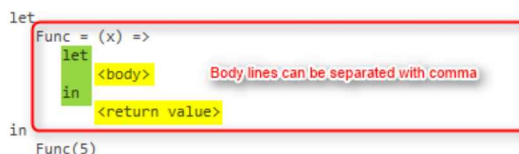
```
let
    Func = (optional x) => if x is null then "nothing" else x
in
    Func(5)
```



Variables in Function

The samples above showed how to write a single line expression function, but in most of the cases, you would require to write a multiple line function that contains variables inside the function. In that case, you can define the function within the LET / IN structure as below:

```
let
    Func = (x) =>
        let
            <body>
        in
            <return value>
in
    Func(5)
```



As you see in the above script, another set of Let/In is added inside the function. You can write the body of the function in the LET clause. If you want to add multiple lines there, or if you want to define variables, do this as usual with a single comma at the end of each line. Finally, you can return the value in the IN clause.

Example

The following example shows how we can use the structure above to create a function that returns the number of days passed till date from the start of the year.

let

DayPassedInYear = (x) =>

let

MonthList=List.Numbers(1,Date.Month(DateTime.FromText(x))-1),

Year=Date.Year(DateTime.FromText(x)),

DaysInMonthList=List.Transform(MonthList,each

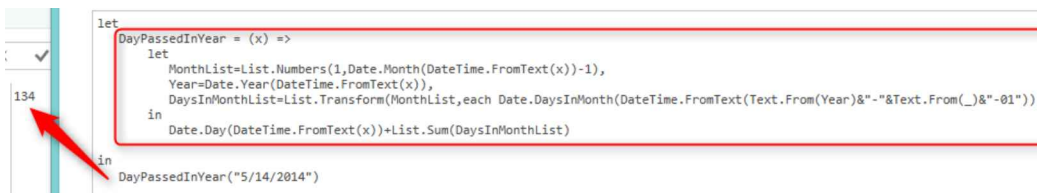
Date.DaysInMonth(DateTime.FromText(Text.From(Year)&"-"&Text.From(_)&"-01")))

in

Date.Day(DateTime.FromText(x))+List.Sum(DaysInMonthList)

in

DayPassedInYear("5/14/2014")



Line by Line Description

Let's go through the script line by line;

DayPassedInYear = (x) =>

In the above line, we define the function name as "DayPassedInYear," this function requires a single parameter; the definition of the body would come in the next lines.

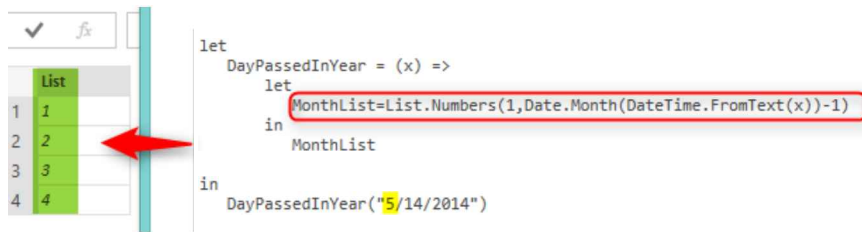
let

MonthList=List.Numbers(1,Date.Month(DateTime.FromText(x))-1),

We used Let to define a multi-line body. The first line of the body defines a variable that creates a list of Month numbers from the first month (1) to the previous month of the input data. For calculating the previous month of the input data, we used this expression: **Date.Month(DateTime.FromText(x))-1** .

This part generates a list based on numbers from a beginning number (1) to an ending number (the result of the previous month function): **List.Numbers(1,...)**

So, as a result, the MonthList would be a variable that contains a list of months from the first month up to the prior month in the current year. Let's see how that single line would return the result:



The next line in the script calculates the year of the input date and store that into a variable named "Year":

```
Year=Date.Year(DateTime.FromText(x)),
```

Consider that you should end each line with a single comma (if you don't want to logic of two-line to be parsed together).

The final line of the body combined from multiple expressions, I'll describe the one by one;

```
DateTime.FromText(Text.From(Year)&"-"&Text.From(_)&"-01")
```

The above expression will return on the first day of a month. Please note that there is a single underscore in the expression. That underscore would be used in EACH expression. EACH is a single parameter function. We used EACH in this example to apply a transformation to every item in the list. We want to replace each month's number in the list with the number of days in that month. So we use EACH single liner function to fetch the number of days in that month. When you use EACH, you can use underscore as the parameter marker. In simpler words, if you have a list as below:

ListA

1

2

3

And if you transform that list with List.Transform function as below:

```
List.Transform(ListA,each_*10)
```

The result would be:

ListA

10

20

So the result of This line below:

```
DaysInMonthList=List.Transform(MonthList,each
Date.DaysInMonth(DateTime.FromText(Text.From(Year)&"-"&Text.From(_)&"-01"))))
```

Would be:

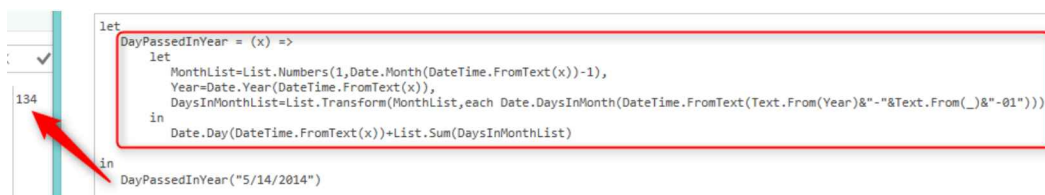


As you see, each list item that had the month number earlier, now replaced (transformed) with the number of days in that month.

In the final step of the function, we return the result;

```
in
    Date.Day(DateTime.FromText(x))+List.Sum(DaysInMonthList)
```

IN clause used to return the result, and we use Date.Day function to return the day part of the input date. We add that with the sum of all values from the list. As a result, we would see the number of days passed from the first year to the input date.



Summary

In this chapter, you've learned how to define functions and how to invoke them. You've learned that you can define optional or required parameters. You can also define multi-line functions that contain variables inside the body. You've seen a sample function that

shows how useful functions are in the real world. You've also learned about EACH keyword that can be used as a single parameter function (especially in lists and tables).

Chapter 38: Day Number of Year, Power Query Custom Function

```

let
    DayNumberOfYear = (date) =>
        let
            dated = try DateTime.FromText(date),
            month = Date.Month(dated[Value]),
            MonthList = List.Numbers(1, month - 1),
            year = Date.Year(dated[Value]),
            TransformedMonthList = List.Transform(
                MonthList,
                each Text.From(year) & "-" & Text.From(_) & "-1",
            ),
            DateList = List.Transform(
                TransformedMonthList,
                each DateTime.FromText(_),
            ),
            DaysList = List.Transform(
                DateList,
                each Date.DaysInMonth(_)
            )
        in
            if dated[HasError]
            then dated[Error]
            else List.Sum(DaysList) + Date.Day(dated[Value])
in
    DayNumberOfYear("07/28/2015")

```

There are many Date and DateTime built-in functions in Power Query, which are helpful. There is also a function for DayNumberOfYear. However, I've thought it would be a good example to write a function that uses Generators, Each singleton function, and error handling all inside a custom function. Through this chapter, you will also learn;

- how to create Custom Function
- how to use Generators as a loop structure
- and how to use Error Handling.

Let's consider this date as today's date: 28th of July of 2015.

There might be many methods to calculate the day number of years for this date (209). I just use one of them here. This choice is because it involves parsing a list and transforming it to learn how that process works. The steps are as below;

- fetch the number of days for each month from January of this year (of the date above) to the previous month (of the date above).
- Calculate the sum of the values above will give me the number of days in all months before this month.
- Add the day number of the date to the calculated sum above.

Some of the calculations can be helped through with Power Query Date functions. So let's start;

1 – Create a function in Power Query called DayNumberOfYear as below

If you don't know where to write the below code:

- Open Excel, Go to Power Query Tab, Click On Get Data from Source, Blank Query, In the Query Editor window, go to View tab, and click on Advanced Editor.
- Open Power BI, click on Edit Queries, In the Query Editor window go to View tab, and click on Advanced Editor.

```
let                                     //start of the code
    DayNumberOfYear= (date) =>         //Function name, and input parameter
        let                             //start of function body
            dated=DateTime.FromText(date) //date conversion from text
        in                             //start of function output
            dated                       //function body output
in                                     //start of output lines generated
    DayNumberOfYear("07/28/2015")      //call function by a value
```

I've put some comments in the above script to help you understand each line. In general, DayNumberOfYear is the name of the function. It accepts an input parameter "date." and converts the parameter from text value to DateTime. The last line of the code calls the function with a specific date ("07/28/2015").

Note that the Date Conversion function is locale-dependent. So if the date-time of your system is no MM/DD/YYYY, you have to enter the date as it is formatted in your system (look below the clock on the right-hand side bottom of your monitor to check the format).

the result of the above script will be:

07/28/2015 12:00:00 a.m.

2 – Fetch the Month Number and generate a list of all prior months.

Fetching month number is easily possible with Date.Month function. The remaining part is looping through months from January of this year to the previous month (of the given date). Unfortunately, there is no loop structure in the Power Query M language yet, but fortunately, we can use Generator functions. A generator function is a function that

Part 8: Power Query Formula Language: M

produces/generates a list based on some parameters. For example, you can generate a list of dates from a start date based on the given occurrence of a period of time. Or you can generate a list of numbers. For this example, we want to generate a list of numbers, starting from 1 (month January) to the current number minus 1 (previous month).

Here is the code:

let

DayNumberOfYear= (date) =>

let

dated=DateTime.FromText(date),

month=Date.Month(dated),//month number

MonthList=List.Numbers(1,month-1) // generate list of months from Jan to previous month

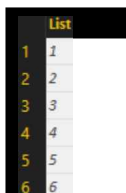
in

MonthList

in

DayNumberOfYear("07/28/2015")

The result is a list of month numbers as below:



| | List |
|---|------|
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |
| 4 | 4 |
| 5 | 5 |
| 6 | 6 |

The generator Function used in the above code is List.Numbers. This function generates a list of numbers starting from a value.

3 – Transform list to full date list

We have to calculate the number of days for each month in the function. The number of days in each month can be fetched by Date.DaysInMonth function. However, this function accepts a full DateTime data type, and the value that we have in our list members is text. So we have to produce a DateTime value from it. For generating a full date, we need the year portion as well; we use Date.Year function to fetch that.

Here is the code to transform the list:

let

DayNumberOfYear= (date) =>

```

let
    dated=DateTime.FromText(date),
    month=Date.Month(dated),
    MonthList=List.Numbers(1,month-1),
    year=Date.Year(dated), // fetch year
    TransformedMonthList=List.Transform // transform list
        (MonthList,
        each Text.From(year)&"-"&Text.From(_)&"-1") // generate full text value
in
    TransformedMonthList
in
    DayNumberOfYear("07/28/2015")

```

The result is:

| | List |
|---|----------|
| 1 | 2015-1-1 |
| 2 | 2015-2-1 |
| 3 | 2015-3-1 |
| 4 | 2015-4-1 |
| 5 | 2015-5-1 |
| 6 | 2015-6-1 |

As you see, the above result is not still of DateTime data type, and we've only generated the full date as a text value. Now we can convert values to DateTime data type

Here is the code:

```

let
    DayNumberOfYear= (date) =>
        let
            dated=DateTime.FromText(date),
            month=Date.Month(dated),
            MonthList=List.Numbers(1,month-1),
            year=Date.Year(dated),
            TransformedMonthList=List.Transform
                (MonthList,
                each Text.From(year)&"-"&Text.From(_)&"-1"),

```

Part 8: Power Query Formula Language: M

```
DateList=List.Transform(  
    TransformedMonthList,  
    each DateTime.FromText(_)//transform to DateTime value  
in  
    DateList  
in  
    DayNumberOfYear("07/28/2015")
```

and the result:

| | List |
|---|----------------------|
| 1 | 1/1/2015 12:00:00 AM |
| 2 | 2/1/2015 12:00:00 AM |
| 3 | 3/1/2015 12:00:00 AM |
| 4 | 4/1/2015 12:00:00 AM |
| 5 | 5/1/2015 12:00:00 AM |
| 6 | 6/1/2015 12:00:00 AM |

4 – Transform the list to List of DaysNumberOfMonths

We use the DaysInMonth function to fetch the number of days in each month from the list. here is the code:

```
let  
    DayNumberOfYear= (date) =>  
        let  
            dated=DateTime.FromText(date),  
            month=Date.Month(dated),  
            MonthList=List.Numbers(1,month-1),  
            year=Date.Year(dated),  
            TransformedMonthList=List.Transform  
                (MonthList,  
                each Text.From(year)&"-"&Text.From(_)&"-1"),  
            DateList=List.Transform(  
                TransformedMonthList,  
                each DateTime.FromText(_)),  
            DaysList=List.Transform(  
                DateList,  
                each List.Transform(  
                    List.Numbers(1,DaysInMonth(DateTime.FromText(_))),  
                    each DayNumberOfYear(DateTime.FromText(_&"-"&Text.From(_)&"-1"))
```

```

        DateList,
        each Date.DaysInMonth(_)
    in
        DaysList
in
    DayNumberOfYear("07/28/2015")

```

and the result:

| | List |
|---|------|
| 1 | 31 |
| 2 | 28 |
| 3 | 31 |
| 4 | 30 |
| 5 | 31 |
| 6 | 30 |

5 – Calculate Sum of Dates and Add the day number of this month to it

The list is ready to use, and we need to sum it up only. And then add the current day Date.Day from the given date to it.

Here is the code:

```

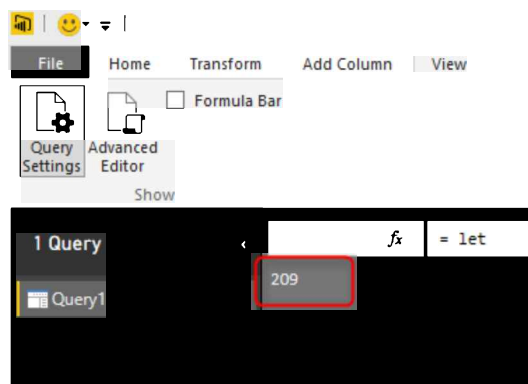
let
    DayNumberOfYear= (date) =>
        let
            dated=DateTime.FromText(date),
            month=Date.Month(dated),
            MonthList=List.Numbers(1,month-1),
            year=Date.Year(dated),
            TransformedMonthList=List.Transform
                (MonthList,
                each Text.From(year)&"-"&Text.From(_)&"-1"),
            DateList=List.Transform(
                TransformedMonthList,
                each DateTime.FromText(_)),
            DaysList=List.Transform(

```

Part 8: Power Query Formula Language: M

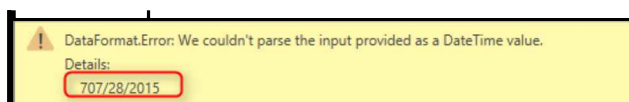
```
        DateList,
        each Date.DaysInMonth(_))
in
    List.Sum(DaysList)//sum of values in the list
    +Date.Day(dated)//current date's day number
in
    DayNumberOfYear("07/28/2015")
```

and the result is:



6- Error Handling

The function is working, but if the given date format is wrong, we will face an error such as below:



So let's add few lines of error handling to the code. We can simply use the Try clause to the code as below:

```
let
    DayNumberOfYear= (date) =>
    let
        dated= try DateTime.FromText(date),
        month=Date.Month(dated[Value]),
        MonthList=List.Numbers(1,month-1),
        year=Date.Year(dated[Value]),
```

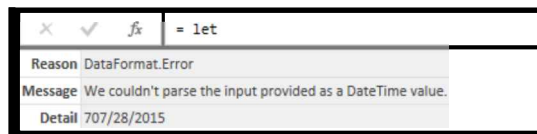


```

TransformedMonthList=List.Transform
    (MonthList,
    each Text.From(year)&"-"&Text.From(_)&"-1"),
DateList=List.Transform(
    TransformedMonthList,
    each DateTime.FromText(_)),
DaysList=List.Transform(
    DateList,
    each Date.DaysInMonth(_))
in
    if dated[HasError]
    then dated[Error]
    else List.Sum(DaysList)+Date.Day(dated[Value])
in
    DayNumberOfYear("707/28/2015")

```

The result for a bad formatted given date is :



Here is the full code of the script:

```

let
    DayNumberOfYear= (date) =>
        let
            dated= try DateTime.FromText(date),
            month=Date.Month(dated[Value]),
            MonthList=List.Numbers(1,month-1),
            year=Date.Year(dated[Value]),
            TransformedMonthList=List.Transform
                (MonthList,
                each Text.From(year)&"-"&Text.From(_)&"-1"),
            DateList=List.Transform(

```

Part 8: Power Query Formula Language: M

```
        TransformedMonthList,  
        each DateTime.FromText(_)),  
    DaysList=List.Transform(  
        DateList,  
        each Date.DaysInMonth(_))  
in  
    if dated[HasError]  
    then dated[Error]  
    else List.Sum(DaysList)+Date.Day(dated[Value])  
in  
    DayNumberOfYear("707/28/2015")
```

Summary

In this chapter, you've learned:

- A Function that calculates Day Number of Year for a given date
- Creating Custom Function
- using Generators as Loop structure
- Error Handling

Chapter 39: Power Query Function that Returns Multiple Values

```

let
    FirstAndLastDayOfTheMonth = (date) =>
        let
            dated=Date.FromText(date),
            year=Date.Year(dated),
            month=Date.Month(dated),
            FirstDateText=Text.From(year)&"-"&Text.From(month)&"-01",
            FirstDate=Date.FromText(FirstDateText),
            daysInMonth=Date.DaysInMonth(dated),
            LastDateText=Text.From(year)&"-"&Text.From(month)&"-"&Text.From(daysInMonth),
            LastDate=Date.FromText(LastDateText),
            record=Record.AddField([], "First Date of Month", FirstDate),
            resultset=Record.AddField(record, "Last Date of Month", LastDate)
        in
            resultset
in
    FirstAndLastDayOfTheMonth("30/07/2015")

```

Power Query custom functions return only one value as the result set. Can you return multiple values? Yes, In this chapter, I'll show you an example of how to return multiple results from a Power Query function.

As you probably know, the Power Query function returns a single value by default, which is the value result of the operation in the "in" clause of the function. Now how to return multiple values? Simply by returning different types of objects. The trick is that the Power Query custom function can return any single object. That **object** can be a simple structure object such as Date, Text, Number. Or it can be multiple value objects such as Record, List, and Table.

To understand the difference between Record, List, and Table:

- Record: Is a single record structure with one or more fields. Each field has a field name and a field value.
- List: This is a single-column structure with one or more rows. Each row contains a value.
- Table: Multiple rows and columns data structure (as you probably all know)

Above objects can hold multiple values. So the only thing you need to do in return is one of the above objects based on your requirement. In the example below, I've returned a Record; as a result, set, but you can do it with the other two data types.

Return First and Last Dates of Month

For example, I would like to write a function that fetches both the first and last date of a month; the input parameter of this function is a date value with text data types, such as "30/07/2015".

Note that the Date Conversion function is locale-dependent. So if the date-time of your system is no DD/MM/YYYY, you have to enter the date as it is formatted in your system (look below the clock on the right-hand side bottom of your monitor to check the format).

Let's start by calculation of the First Date of the Month

First Date of the Month

We need to fetch the year and the month, build a date string for the first day (day 1) of that month and year and finally convert it to Date datatype.

Here is the script:

let

FirstAndLastDayOfTheMonth = (date) => //function definition

let

dated=Date.FromText(date),//convert input text to Date

year=Date.Year(dated),//fetch year

month=Date.Month(dated),//fetch month

FirstDateText=Text.From(year)&"-"&Text.From(month)&"-01",//generate text value of the first date

FirstDate=Date.FromText(FirstDateText)//convert text value to date

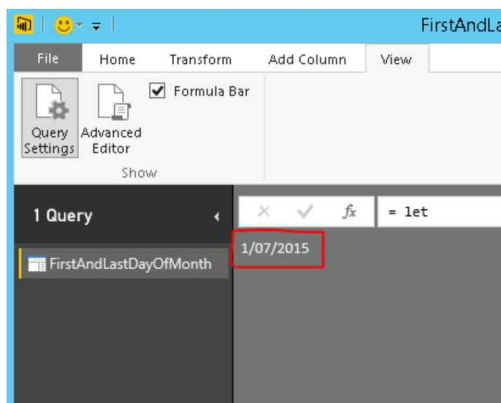
in

FirstDate//return result of the function

in

FirstAndLastDayOfTheMonth("30/07/2015")//function call

and the Result:



Last Date of the Month

For fetching the last date of the month, we use the same method of the first date, except for one change. The day part of the calculation should be the number of days in the month, which comes from the `Date.DaysInMonth` function.

Here is the script:

let

FirstAndLastDayOfTheMonth = (date) =>

let

dated=Date.FromText(date),

year=Date.Year(dated),

month=Date.Month(dated),

FirstDateText=Text.From(year)&"-"&Text.From(month)&"-01",

FirstDate=Date.FromText(FirstDateText),

daysInMonth=Date.DaysInMonth(dated),//fetch number of days in month

LastDateText=Text.From(year)&"-"&Text.From(month)&"-"&Text.From(daysInMonth),

LastDate=Date.FromText>LastDateText)

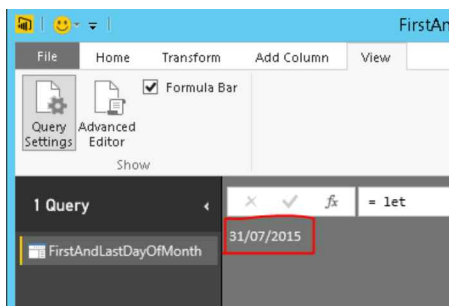
in

LastDate

in

FirstAndLastDayOfTheMonth("30/07/2015")

and the result:



Combining both values into a Record and returning Record as a Result

Now we have both values, and we want to return them both. I'll create an empty record first. The empty record can be created simply with this : `[]`.

Then I used `Record.AddField` function to add fields one by one. `Record.AddField` gets three parameters: the record that field will be added to it, the name of the new field, and the value of the new field.

Here is the script:

let

FirstAndLastDayOfTheMonth = (date) =>

let

dated=Date.FromText(date),

year=Date.Year(dated),

month=Date.Month(dated),

FirstDateText=Text.From(year)&"-"&Text.From(month)&"-01",

FirstDate=Date.FromText(FirstDateText),

daysInMonth=Date.DaysInMonth(dated),

LastDateText=Text.From(year)&"-"&Text.From(month)&"-"&Text.From(daysInMonth),

LastDate=Date.FromText>LastDateText),

record=Record.AddField([], "First Date of Month",FirstDate),

resultset=Record.AddField(record, "Last Date of Month",LastDate)

in

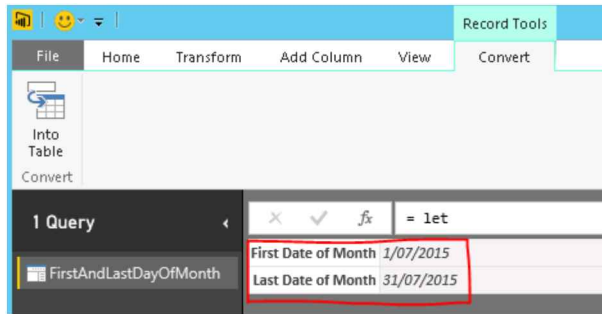
resultset

in

FirstAndLastDayOfTheMonth("30/07/2015")

and the result:

Chapter 39: Power Query Function that Returns Multiple Values

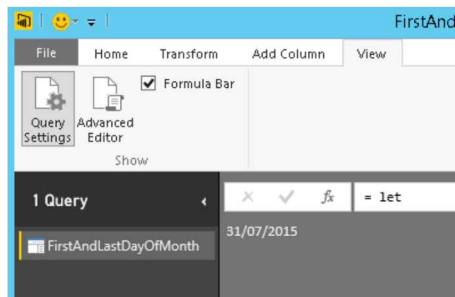


How to access this Record's values

As you see above, the function returns a record with two fields. now we can access fields by name with a script like this:

```
FirstAndLastDayOfTheMonth("30/07/2015")[Last Date of Month]
```

and the result would be a single value

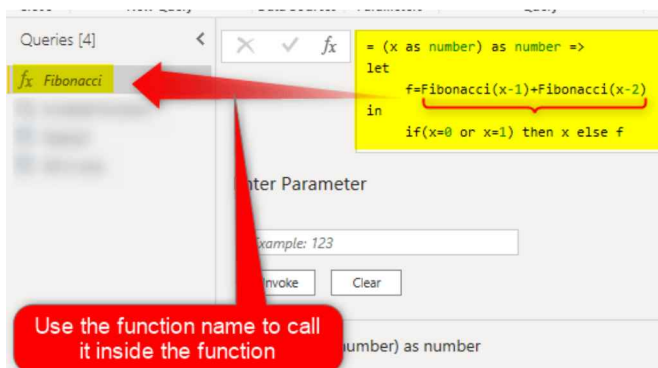


Here is the full code of example if you want to try it yourself.

Summary

You can have another record in a field's value, you can have a list in a field's value, and you can have a table in a field's value. So you can create any data structure that you want as the result set of your function.

Chapter 40: Fibonacci Sequence: Understanding the Power Query Recursive Function for Power BI



Recursive functions in Power Query are not very popular but are sometimes very helpful when in need. This chapter will explain what a recursive function is, how it works, and explain it through a famous recursive example of the Fibonacci Sequence.

Fibonacci Sequence

Fibonacci sequence is one of the fundamental recursive operations in math; below are a few numbers from this sequence:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34...

As you can see, the numbers above don't follow a normal order. Each number is the sum of the two numbers before it:

34= 21+13

21= 13+8

13= 8+5

...

Although I love math, I am not that advanced to explain the benefits of this sequence. [Here\[https://en.wikipedia.org/wiki/Fibonacci_number\]](https://en.wikipedia.org/wiki/Fibonacci_number) is the Wikipedia page with more info about the Fibonacci Sequence if you wish to read more. My goal here is not to explain to you what this sequence is but to teach you the Power Query recursive function.

Recursive Function

A recursive function is a function that calls itself to produce a result. In the above definition of the Fibonacci Sequence, the numbers are called F_x like below:

$$F(0)=0$$

$$F(1)=1$$

$$F(2)=1$$

$$F(3)=2$$

$$F(4)=3$$

$$F(5)=5$$

$$F(6)=8$$

$$F(7)=13$$

$$F(8)=21$$

$$F(9)=34$$

...

When you look at the calculation, each F_x is the sum up of the two F_x before, so the calculation is:

$$F(x)=F(x-1)+F(x-2)$$

This means that if I want to calculate the value of $F(9)$, I first need to calculate the value of $F(8)$ and $F(7)$, but then for those, I need to calculate $F(6)$, and $F(5)$ too, etc.

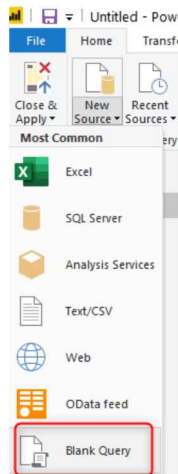
The function call with the number X , will cause calling the function again with $X-1$, and $X-2$, and that will cause it again for the rest of the loop until it hits 1 or 0. The $F(1)=1$, and the $F(0)=0$.

Power Query Function

Writing a Power Query recursive function is very simple. In fact, you don't even need to do anything except the fact that you need to create a function and use the function inside itself, like below;

Start with a Blank Query;

Part 8: Power Query Formula Language: M



Rename the Query to Fibonacci.

And then write the function code below;

```
= (x as number) as number =>
```

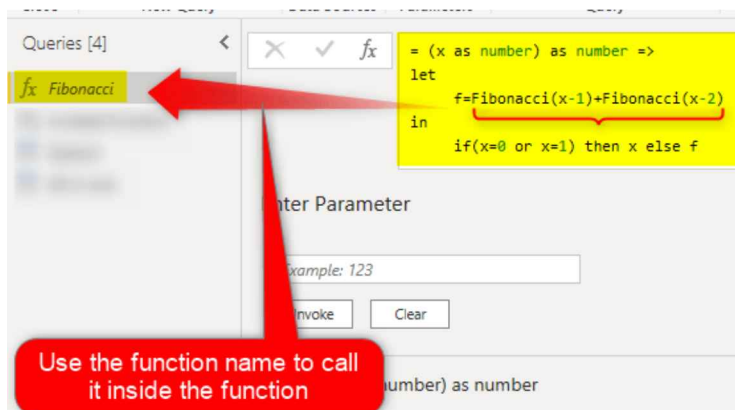
```
let
```

```
    f=Fibonacci(x-1)+Fibonacci(x-2)
```

```
in
```

```
    if(x=0 or x=1) then x else f
```

As you can see, I have used the function name inside the function itself, saying that if the input parameter is not one or zero, then return the same function with x-1 and x-2;

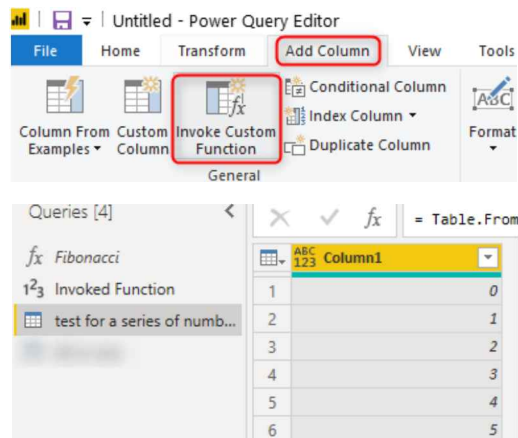


To understand the code above, you first need to understand how custom functions work in Power BI. I explained that concept in a chapter earlier in this part of the book with full details about input parameters and other blocks of the function's structure.

As you can see, the only thing I needed in the above expression was to use the function name (or, let’s say, the Power Query query name) inside the function itself. This only works if your function is defined as a query itself. For Inline functions, you need to use another approach, which I’ll explain later in this chapter.

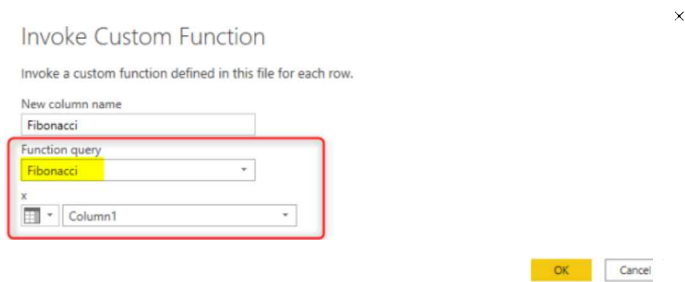
Using the Function

Now, if I want to test the function to see how it works, I can call it for a series of numbers, using the Add Column, Invoke Custom Function;



Invoking custom function

Now the custom function query would be the name of the function, and the input would be the column that includes the numbers;



The result simply is the Fibonacci Sequence value for each number:

| 1 ² ₃ Column1 | 1 ² ₃ Fibonacci |
|-------------------------------------|---------------------------------------|
| 0 | 0 |
| 1 | 1 |
| 2 | 1 |
| 3 | 2 |
| 4 | 3 |
| 5 | 5 |
| 6 | 8 |
| 7 | 13 |
| 8 | 21 |
| 9 | 34 |
| 10 | 55 |
| 11 | 89 |
| 12 | 144 |
| 13 | 233 |
| 14 | 377 |
| 15 | 610 |
| 16 | 987 |
| 17 | 1597 |
| 18 | 2584 |
| 19 | 4181 |

Fibonacci Sequence created by the function

Consider Exit for the Function

Writing a custom function that works recursively is simple; you just use the function name, and that’s all, but having Exit criteria for that function is necessary. Without the exit criteria, your function falls into an endless loop. here is an example of such a function (don’t invoke the function below, it might cause you to close your Power BI Desktop)

= (x as number) as number =>

let

f=Fibonacci(x-1)+Fibonacci(x-2)

in

f

The above function falls into an endless loop; for calculating each number, you calculate the two numbers before, and this never ends!

If you compare the above with the Fibonacci function we have created previously, you’ll see that there is an IF statement that defines when the recursive loop has to finish:

```
= (x as number) as number =>
let
    f=Fibonacci(x-1)+Fibonacci(x-2)
in
    if(x=0 or x=1) then x else f
```

Function with Exit

A recursive function without Exit causes an endless loop and cannot be used.

Inline Power Query Recursive Function

If your function is defined as a query, then creating a recursive structure means just calling the function's name. However, if the function is not a separate query itself, then the process is different.

Below is an example of a function that created only inside the below query. This function does not exist as a separate query itself.

let

Source = List.Numbers(0,20),

#'Converted to Table' = Table.FromList(Source, Splitter.SplitByNothing(), null, null, ExtraValues.Error),

F= (x as number) as number =>

let

f=F(x-1)+F(x-2)

in

if(x=0 or x=1) then x else f,

#'Invoked Custom Function' = Table.AddColumn(#'Converted to Table', 'Fibonacci', each F([Column1]))

in

#'Invoked Custom Function'

The function above is highlighted inline the main query;

```
let
    Source = List.Numbers(0,20),
    #"Converted to Table" = Table.FromList(Source, Splitter.SplitByNothing(), null, null, ExtraValues.Error),
    F= (x as number) as number =>
        let
            f=F(x-1)+F(x-2)
        in
            if(x=0 or x=1) then x else f,
    #'Invoked Custom Function' = Table.AddColumn(#"Converted to Table", "Fibonacci", each F([Column1]))
in
    #'Invoked Custom Function'
```

inline Power Query function



Part 8: Power Query Formula Language: M

The function is defined inside the query, and it is called using the function name, which is F.

The result, however, is error;

| Column1 | Fibonacci |
|---------|-----------|
| 1 | 0 |
| 2 | 1 |
| 3 | Error |
| 4 | Error |
| 5 | Error |
| 6 | Error |
| 7 | Error |
| 8 | Error |
| 9 | Error |
| 10 | Error |
| 11 | Error |
| 12 | Error |
| 13 | Error |
| 14 | Error |
| 15 | Error |
| 16 | Error |
| 17 | Error |
| 18 | Error |
| 19 | Error |
| 20 | Error |

Expression.Error: The import F matches no exports. Did you miss a module reference?

Expression.Error: The import F matches no exports. Did you miss a module reference?

Because there is no query called F, the expression above fails. For the inline Power Query functions, in order to call the function itself, you have to use the character @ before the function name. This character act as the function reference. Below is a code that works correctly;

let

Source = List.Numbers(0,20),

#'Converted to Table' = Table.FromList(Source, Splitter.SplitByNothing(), null, null, ExtraValues.Error),

F= (x as number) as number =>

let

f=@F(x-1)+@F(x-2)

in

if(x=0 or x=1) then x else f,

#'Invoked Custom Function' = Table.AddColumn(#'Converted to Table', 'Fibonacci', each F([Column1]))

in

#'Invoked Custom Function'

The two function calls inside the inline function are prefixed with @. the function call outside is normal without @.

```
let
Source = List.Numbers(0,20),
#"Converted to Table" = Table.FromList(Source, Splitter.SplitByNothing(), null, null, ExtraValues.Error),
F= (x as number) as number =>
    let
        f=@F(x-1)+@F(x-2)
    in
        if(x=0 or x=1) then x else f,
#"Invoked Custom Function" = Table.AddColumn(#"Converted to Table", "Fibonacci", each F([Column1]))
in
    #"Invoked Custom Function"
```

call the function recursively from the inside of the inline function using character @

Using the inclusive identifier reference

The character @ is called the inclusive-identifier-reference. This can be used for accessing the environment that includes the identifier being initialized.

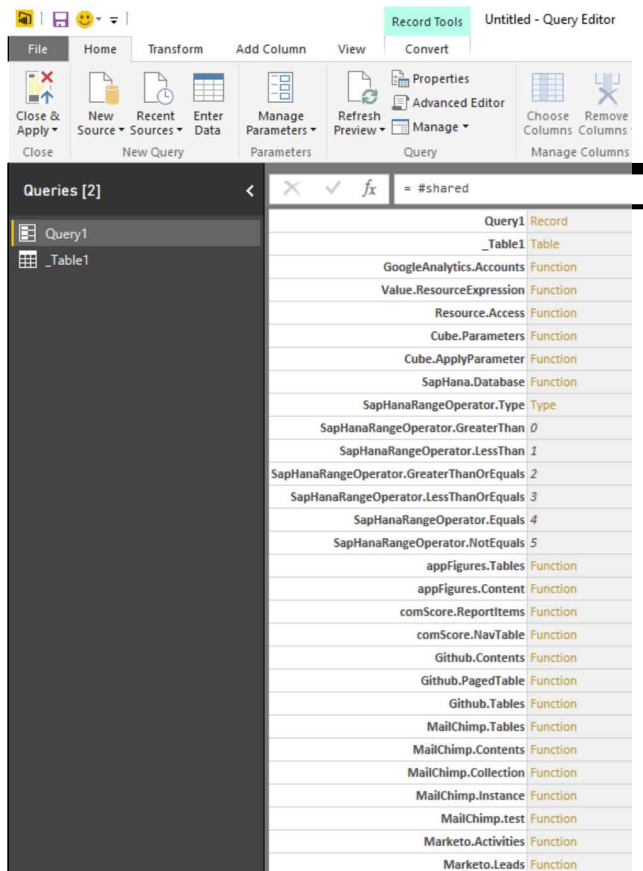
The result using the above code is working correctly as below;

| Column1 | Fibonacci |
|---------|-----------|
| 0 | 0 |
| 1 | 1 |
| 2 | 1 |
| 3 | 2 |
| 4 | 3 |
| 5 | 5 |
| 6 | 8 |
| 7 | 13 |
| 8 | 21 |
| 9 | 34 |
| 10 | 55 |
| 11 | 89 |
| 12 | 144 |
| 13 | 233 |
| 14 | 377 |
| 15 | 610 |
| 16 | 987 |
| 17 | 1597 |
| 18 | 2584 |
| 19 | 4181 |

Summary

Recursive functions, although not very common, but are useful in some instances in Power Query and Power BI. This chapter taught that you could create a recursive function by calling the function name (if it is not an inline function) or using the inclusive identifier reference (character @) if the function is inline. In both cases, ensure that your function always has Exit criteria and doesn't fall into an endless loop.

Chapter 41: Power Query Library of Functions; Shared Keyword

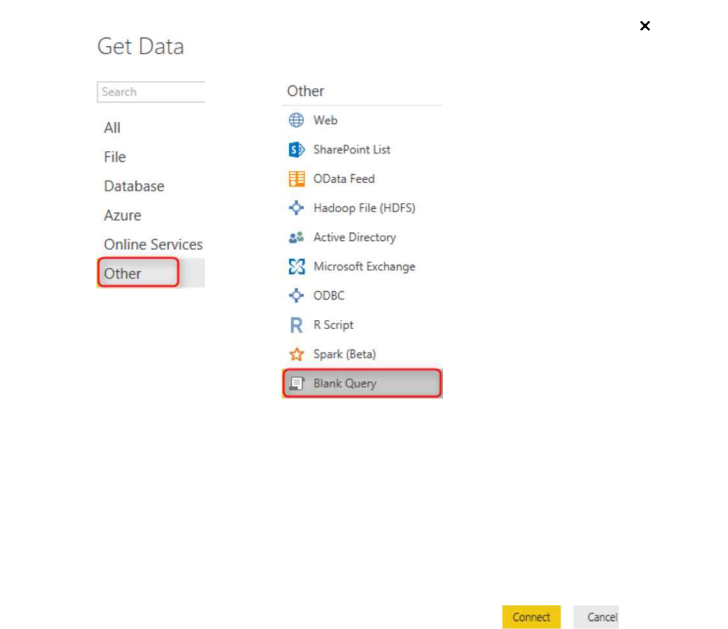


As I mentioned earlier in this book, Power Query is a functional language. Knowing functions is your best helper when you work with a functional language. Fortunately, Power Query in Excel and Power BI can use shared keywords to reveal a document library of all functions. With the method in this chapter, you can easily find any function you want in Power Query, and you won't need an internet connection to search for functions.

#shared Keyword

Shared is a keyword that loads all functions and enumerators in the result set. You can simply create a blank query in Power BI (or Excel)

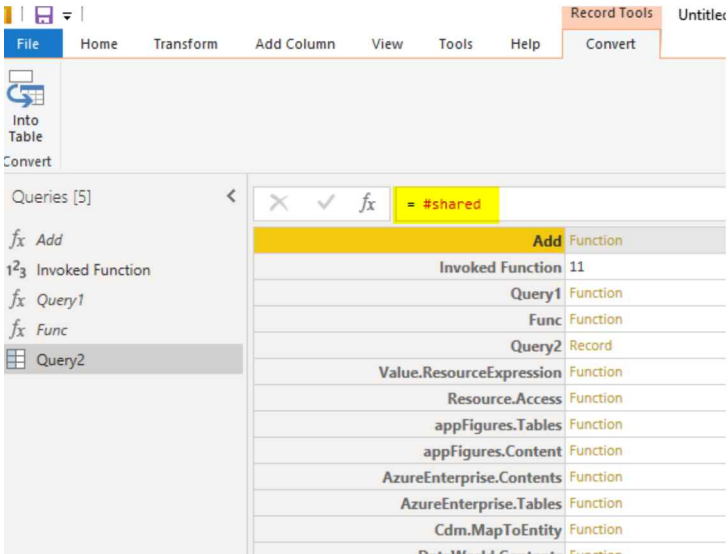
Chapter 41: Power Query Library of Functions; Shared Keyword



In the formula bar, type in

`=#shared`

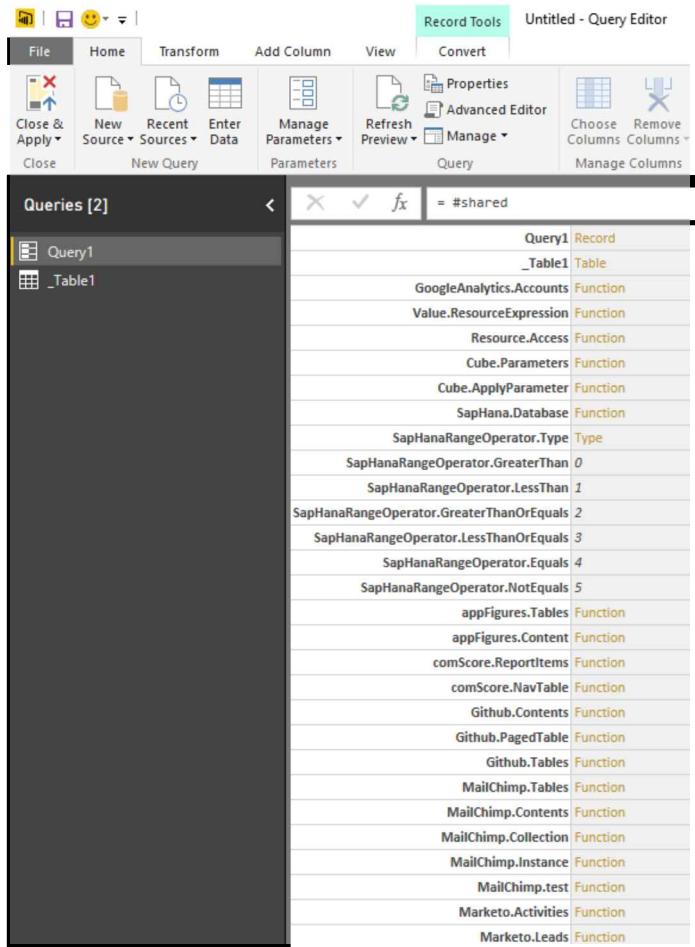
Don't forget the equal sign at the beginning.



After clicking on Done, you will see all functions and enumerators in the power query. You will also see other queries in your Power BI solution or workbook plus other custom functions.

Part 8: Power Query Formula Language: M

You may see a warning about data privacy again if you have queries from different sources; please continue and ignore the privacy levels.

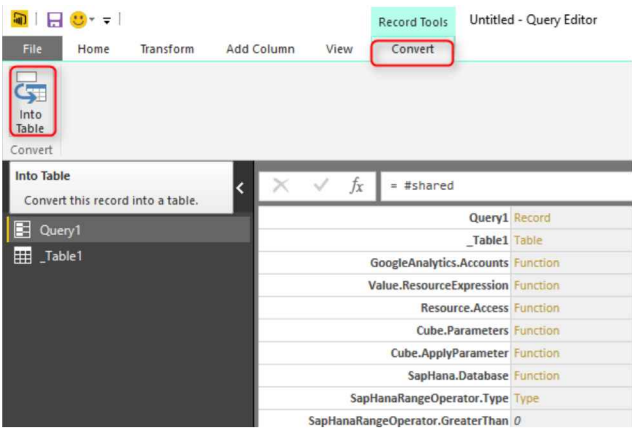


Result set here is a Record structure that has functions, enumerators, and queries in each item of the record. Now let's explore the record more in detail.

Use the Result Set as a Table

The result above is loaded in Power Query, and that is the greatest feature of Power Query itself that can turn this result into a table.

Chapter 41: Power Query Library of Functions; Shared Keyword



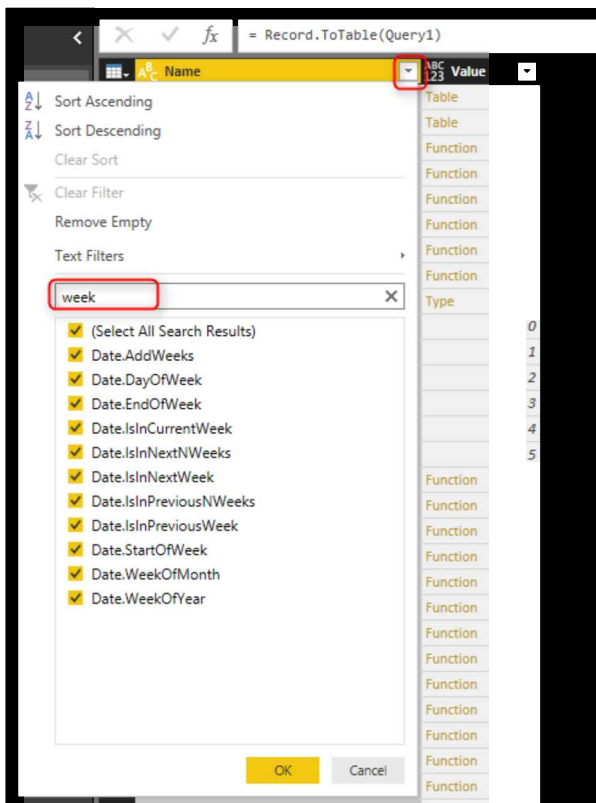
This will convert the result set the record to a table. And the table is really easy to search in as you know.

The screenshot shows the Power Query function list after conversion. The list is displayed in a table format with columns for 'Name' and 'Value'. The functions are listed in a table with 30 rows. The first two rows are 'Query1' (Table) and '_Table1' (Table). The remaining rows are functions and types, including 'GoogleAnalytics.Accounts' (Function), 'Value.ResourceExpression' (Function), 'Resource.Access' (Function), 'Cube.Parameters' (Function), 'Cube.ApplyParameter' (Function), 'SapHana.Database' (Function), 'SapHanaRangeOperator.Type' (Type), 'SapHanaRangeOperator.GreaterThan' (0), 'SapHanaRangeOperator.LessThan' (1), 'SapHanaRangeOperator.GreaterThanOrEquals' (2), 'SapHanaRangeOperator.LessThanOrEquals' (3), 'SapHanaRangeOperator.Equals' (4), 'SapHanaRangeOperator.NotEquals' (5), 'appFigures.Tables' (Function), 'appFigures.Content' (Function), 'comScore.ReportItems' (Function), 'comScore.NavTable' (Function), 'Github.Contents' (Function), 'Github.PagedTable' (Function), 'Github.Tables' (Function), 'MailChimp.Tables' (Function), 'MailChimp.Contents' (Function), 'MailChimp.Collection' (Function), 'MailChimp.Instance' (Function), 'MailChimp.test' (Function), 'Marketo.Activities' (Function), 'Marketo.Leads' (Function), and 'Marketo.Tables' (Function).

| Name | Value |
|--|----------|
| Query1 | Table |
| _Table1 | Table |
| GoogleAnalytics.Accounts | Function |
| Value.ResourceExpression | Function |
| Resource.Access | Function |
| Cube.Parameters | Function |
| Cube.ApplyParameter | Function |
| SapHana.Database | Function |
| SapHanaRangeOperator.Type | Type |
| SapHanaRangeOperator.GreaterThan | 0 |
| SapHanaRangeOperator.LessThan | 1 |
| SapHanaRangeOperator.GreaterThanOrEquals | 2 |
| SapHanaRangeOperator.LessThanOrEquals | 3 |
| SapHanaRangeOperator.Equals | 4 |
| SapHanaRangeOperator.NotEquals | 5 |
| appFigures.Tables | Function |
| appFigures.Content | Function |
| comScore.ReportItems | Function |
| comScore.NavTable | Function |
| Github.Contents | Function |
| Github.PagedTable | Function |
| Github.Tables | Function |
| MailChimp.Tables | Function |
| MailChimp.Contents | Function |
| MailChimp.Collection | Function |
| MailChimp.Instance | Function |
| MailChimp.test | Function |
| Marketo.Activities | Function |
| Marketo.Leads | Function |
| Marketo.Tables | Function |

I can now simply search in the function list. Let's see, for example, what function I can find for working with WEEK;

Part 8: Power Query Formula Language: M



This will filter the table to a sub-set that I can see only functions with WEEK in the name of it.

Table.SelectRows("#Converted to Table", each ([Name] = "Date.AddWeeks" or [Name] = "Date.DayOfWeek" or [Name] = "Date.EndOfWeek" or [Name] = "Date.IsInCurrentWeek" or [Name] = "Date.IsInNextNWeeks" or [Name] = "Date.IsInNextWeek" or [Name] = "Date.IsInPreviousNWeeks" or [Name] = "Date.IsInPreviousWeek" or [Name] = "Date.StartOfWeek" or [Name] = "Date.WeekOfMonth" or [Name] = "Date.WeekOfYear"))

| ABC 123 | Name | Value |
|---------|-------------------------|----------|
| 1 | Date.IsInPreviousWeek | Function |
| 2 | Date.IsInPreviousNWeeks | Function |
| 3 | Date.IsInCurrentWeek | Function |
| 4 | Date.IsInNextWeek | Function |
| 5 | Date.IsInNextNWeeks | Function |
| 6 | Date.AddWeeks | Function |
| 7 | Date.StartOfWeek | Function |
| 8 | Date.EndOfWeek | Function |
| 9 | Date.DayOfWeek | Function |
| 10 | Date.WeekOfMonth | Function |
| 11 | Date.WeekOfYear | Function |

function (dateTime as any, numberOfWeeks as number) as any

Returns the date, datetime, or datetimezone result from adding numberOfWeeks weeks to the datetime value dateTime.

dateTime: The date, datetime, or datetimezone value to which weeks are being added.
numberOfWeeks: The number of weeks to add.

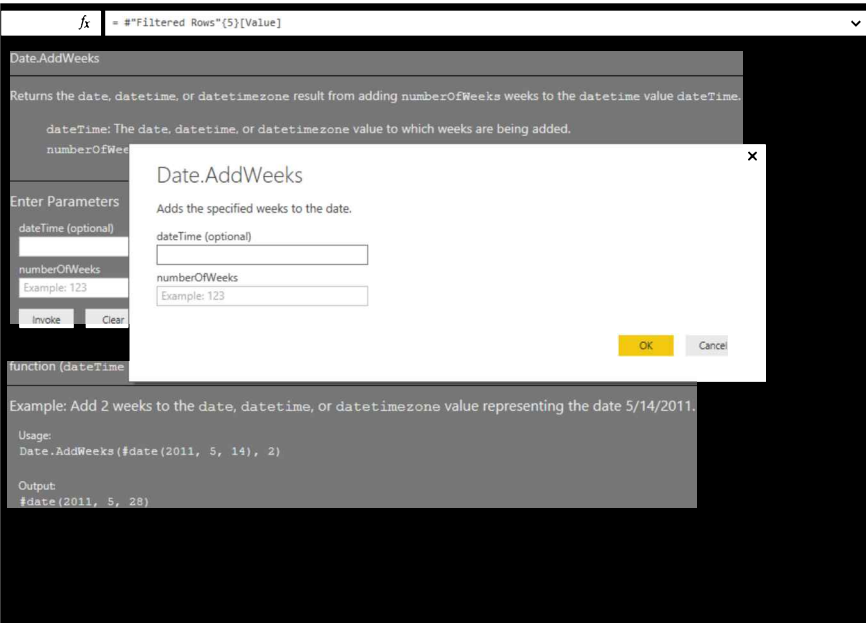
Example: Add 2 weeks to the date, datetime, or datetimezone value representing the date 5/14/2011.

Usage:
Date.AddWeeks (#date(2011, 5, 14), 2)

Output:
#date(2011, 5, 28)

Documentation of Function

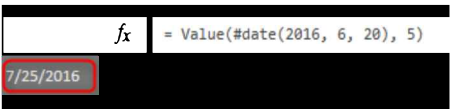
Now, for example, if I want to see the Date.AddWeeks function I can click on the “function” link of it in the value column, and this will redirect me to the documentation of this function and bring a dialog box to invoke the function!



You can see the documentation in grey that also includes examples of how to call this function. For invoking the function, I can simply provide parameters and click OK or Invoke;



And it calls the function and shows me the result as below;



Enumerators

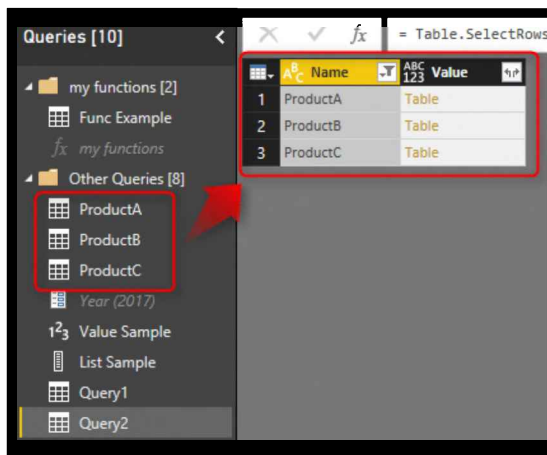
Finding enumerators is also easy with the help of the #shared keyword. Here I can see enumerators for JoinKind and JoinAlgorithm;

| | Name | Value |
|----|----------------------------|----------|
| 1 | JoinKind.Type | Type |
| 2 | JoinKind.Inner | 0 |
| 3 | JoinKind.LeftOuter | 1 |
| 4 | JoinKind.RightOuter | 2 |
| 5 | JoinKind.FullOuter | 3 |
| 6 | JoinKind.LeftAnti | 4 |
| 7 | JoinKind.RightAnti | 5 |
| 8 | Table.Join | Function |
| 9 | Table.AddJoinColumn | Function |
| 10 | Table.NestedJoin | Function |
| 11 | JoinAlgorithm.Type | Type |
| 12 | JoinAlgorithm.Dynamic | 0 |
| 13 | JoinAlgorithm.PairwiseHash | 1 |
| 14 | JoinAlgorithm.SortMerge | 2 |
| 15 | JoinAlgorithm.LeftHash | 3 |
| 16 | JoinAlgorithm.RightHash | 4 |
| 17 | JoinAlgorithm.LeftIndex | 5 |
| 18 | JoinAlgorithm.RightIndex | 6 |

Summary

You've learned how `#shared` keyword can help get a list of all functions and enumerators in Power Query. You learned you could convert the result set into a table and filter that to find a particular function you are looking for. This is superb, especially for people like myself who can't remember things well. Do you know how I look for functions? This chapter explained my method! I use the `#shared` keyword to find the function I want and start working on that. For me, `#shared` is the keyword that I use more than any other queries in the Power query side of Power BI.

Chapter 42: Get a List of Queries in Power BI

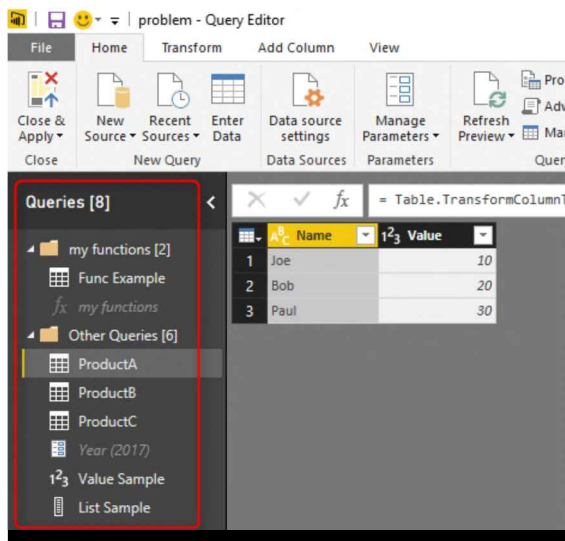


Power Query is the component of getting data in Power BI. But have you used Power Query to get metadata of the Power BI queries themselves? In this chapter, I'll show you a quick and simple way of using Power Query to get metadata (name of queries and the data in queries) from all queries in Power BI. I have previously explained how to use the #shared keyword to list all functions in Power Query. This chapter shows how to use #shared or #sections to get all queries (and parameters, functions, and lists...) from Power BI.

Question: How to Fetch Name of All Queries into One Query?

Consider below the Power BI file that has functions, parameters, and queries. Queries also return tables, values, and lists;

Part 8: Power Query Formula Language: M



The question is, how can I get a list of all these queries and their values as a new query? Let's see the answer

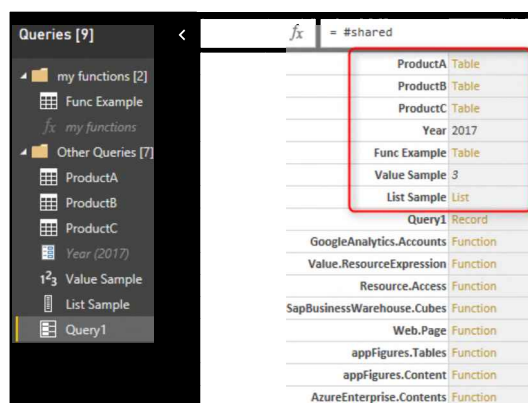
Answer: Using #shared or #sections Keywords

#shared to Get Current File's Queries Plus All Functions

I have previously explained what the #shared keyword does; it is a keyword that returns a list of all functions and enumerators in Power Query. It can be used as a document library in the Power Query itself. It will also fetch all queries in the existing Power BI file. Here is how you can use it:

Create a New Source from Blank Query. Then in the formula bar, type in the below code;

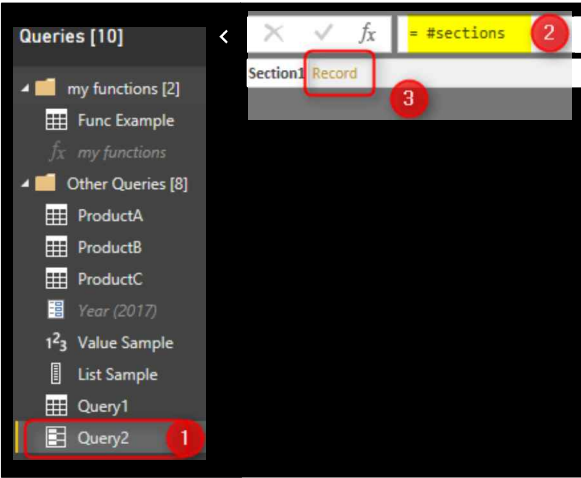
`=#shared`



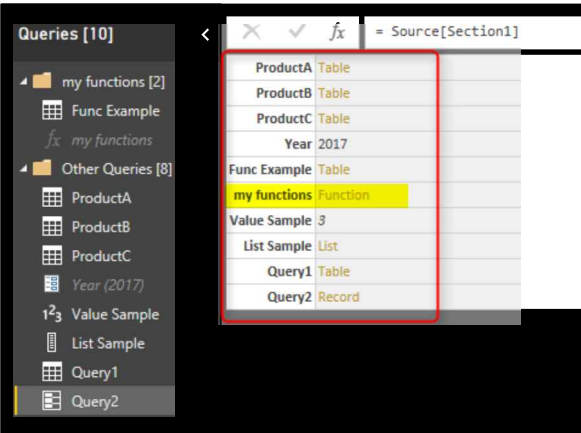
You can see in the result that #shared fetch all existing queries, plus all built-in functions in Power Query. the section marked in the above result set where you can find queries from the current file. Note that the Query1 itself (which includes the #shared keyword) is listed there. The limitation of this method is that it won't return your custom function, "my function," in this example. The next method, however, would pick that as well.

#sections To Get Current File's Queries

The other way of fetching a list of queries is using the #sections keyword. #sections keyword will give you a list of all sections in Power Query (This chapter isn't the right place to explain what sections are, but for now, consider every query here as a section). so same method, this time using #sections, will return the result below;



The result is a record that you can simply click on the Record to see what are columns in there. Columns are queries in the current file:



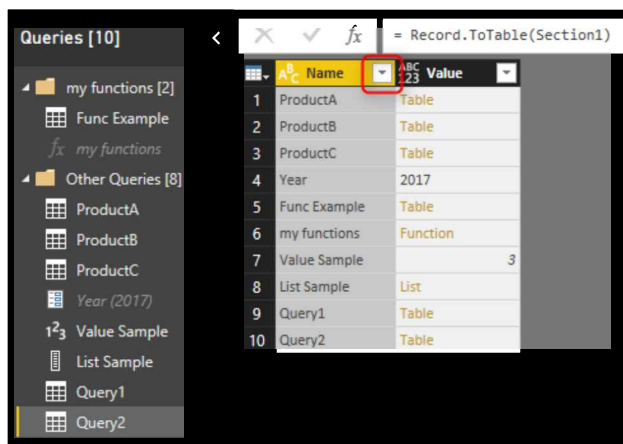
Part 8: Power Query Formula Language: M

This method also returns functions in the current file, which the previous method with `#shared` didn't. So this is a better method if you are interested in fetching function names as well.

The result is a record that can be converted to a table (this gives you better filtering options in the GUI). You can find the Convert Into Table under the Record Tools Convert Section menu.

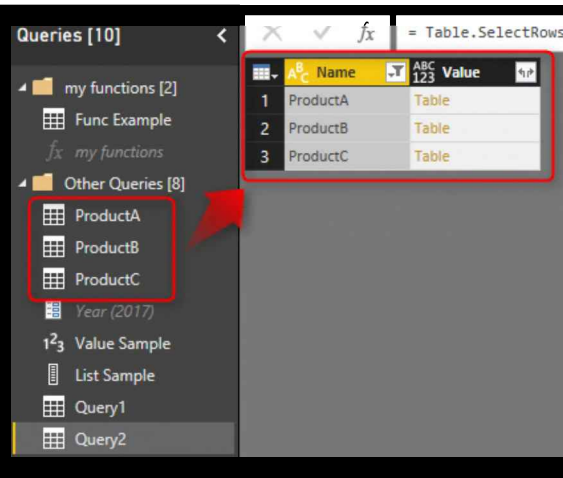


After having this as a table, then you can apply any filters you want;

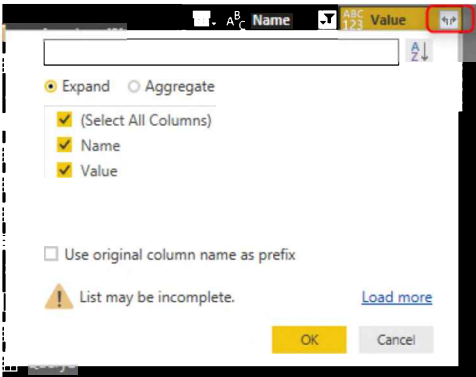


Sample Scenario of Usage

There are many usages of getting the name of queries in another query. One sample usage of that can be getting different queries coming from different places, and the only way to identify the source is the query name. In this case, instead of manually adding a column to each query and then combining them, you can use this method to get the query name dynamically. In the screenshot below, ProductA, ProductB, and ProductC are coming as source queries, and you can do filtering to get them all with their product names.



And you can expand it to tables underneath if you want to (this would work if they all have the same data structure);



and as a result, you have combined the result of all queries, with the query name as another column;

| | A ^B _C Name | ABC 123 Name.1 | ABC 123 Value.1 |
|---|----------------------------------|----------------|-----------------|
| 1 | ProductA | Joe | 10 |
| 2 | ProductA | Bob | 20 |
| 3 | ProductA | Paul | 30 |
| 4 | ProductB | Joe | 30 |
| 5 | ProductB | Bob | 60 |
| 6 | ProductB | Paul | 90 |
| 7 | ProductC | Joe | 1 |
| 8 | ProductC | Bob | 2 |
| 9 | ProductC | Paul | 3 |

Please note that the methods mentioned here work only inside the Power Query Editor, not when you load data into Power BI or Excel.

Part 9: Table and List functions

Chapter 43: Pre Concatenate List of Values in Power BI using Power Query



You can concatenate a list of values in Power BI in two ways; Using Power BI DAX functions such as ConcatenateX, or through Power Query. For some scenarios, there is an advantage in using Power Query to do the concatenation. In this chapter, I'll explain how this works.

Why doing concatenation in Power Query rather than DAX

Doing concatenation in DAX is good and useful. However, if the concatenation is coming from a large number of items in the list, and there is no need to do the concatenation dynamically using a measure. Power Query would work faster from the end-user point of view, and the main reason is that you do the concatenation in Power Query BEFORE loading the data as a pre-calculated value. There are, of course, pros and cons of doing concatenation in each place (DAX vs. Power Query). This chapter, however, is not about where you should apply the concatenation. In this chapter, I explain how you can do it if you decided to do the concatenation in Power Query.

Sample dataset

The sample dataset I use for this example is the AdventureWorksDW2012 Excel file, which you can download for this book's files. The two tables that I have in my datasets are DimProductCategory and DimProductSubcategory. I removed unnecessary columns, and here is the structure of the tables:

DimProductCategory:

Part 9: Table and List functions

| 1^2_3 ProductCategoryKey | A^B_C EnglishProductCategoryName |
|----------------------------|------------------------------------|
| 1 | Bikes |
| 2 | Components |
| 3 | Clothing |
| 4 | Accessories |

DimProductSubcategory:

| 1^2_3 ProductSubcategoryKey | A^B_C EnglishProductSubcategoryName | 1^2_3 ProductCategoryKey |
|-------------------------------|---------------------------------------|----------------------------|
| 1 | Mountain Bikes | 1 |
| 2 | Road Bikes | 1 |
| 3 | Touring Bikes | 1 |
| 4 | Handlebars | 2 |
| 5 | Bottom Brackets | 2 |
| 6 | Brakes | 2 |
| 7 | Chains | 2 |
| 8 | Cranksets | 2 |
| 9 | Derailleurs | 2 |
| 10 | Forks | 2 |
| 11 | Headsets | 2 |
| 12 | Mountain Frames | 2 |
| 13 | Pedals | 2 |
| 14 | Road Frames | 2 |

List of Values

Many different transformations can lead to having a list of values. For this example, I used Merge transformation to create a structure like this: (merged the two tables based on ProductCategoryKey)

Queries [2]

DimProductCategory

DimProductSubcategory

1^2_3 ProductCategoryKey

A^B_C EnglishProductCategoryName

DimProductSubcategory

| | | |
|---|---------------|-------|
| 1 | 1 Bikes | Table |
| 2 | 2 Components | Table |
| 3 | 3 Clothing | Table |
| 4 | 4 Accessories | Table |

Every cell in the DimProductSubcategory column in the above screenshot represents a sub-table of DimProductSubcategory for a specific category of products.

Chapter 43: Pre Concatenate List of Values in Power BI using Power Query

| ProductCategoryKey | EnglishProductCategoryName | DimProductSubcategory |
|--------------------|----------------------------|-----------------------|
| 1 | Bikes | Table |
| 2 | Components | Table |
| 3 | Clothing | Table |
| 4 | Accessories | Table |

| ProductSubcategoryKey | EnglishProductSubcategoryName | ProductCategoryKey |
|-----------------------|-------------------------------|--------------------|
| 4 | Handlebars | 2 |
| 5 | Bottom Brackets | 2 |
| 6 | Brakes | 2 |
| 7 | Chains | 2 |
| 8 | Cranksets | 2 |
| 9 | Derailleurs | 2 |
| 10 | Forks | 2 |
| 11 | Headsets | 2 |
| 12 | Mountain Frames | 2 |
| 13 | Pedals | 2 |
| 14 | Road Frames | 2 |
| 15 | Saddles | 2 |
| 16 | Touring Frames | 2 |
| 17 | Wheels | 2 |

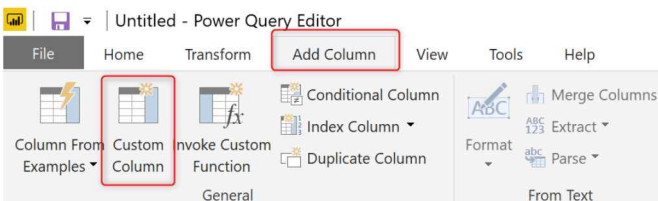
The goal here is to create the list of EnglishProductSubcategoryName as a concatenated list of values, such as below:

| ProductCategoryKey | EnglishProductCategoryName | Concatenated List of subcategories |
|--------------------|----------------------------|---|
| 1 | Bikes | Mountain Bikes, Road Bikes, Touring Bikes |
| 2 | Components | Handlebars, Bottom Brackets, Brakes, Chains, Cranksets, Derailleurs, Forks, Headsets, Mountain F... |
| 3 | Clothing | Bib-Shorts, Caps, Gloves, Jerseys, Shorts, Socks, Tights, Vests |
| 4 | Accessories | Bike Racks, Bike Stands, Bottles and Cages, Cleaners, Fenders, Helmets, Hydration Packs, Lights, L... |

I will show you a method that you can achieve this using Power Query. but before that, let's see how from a sub-table, you can get only one column of it:

Get only one column from the sub-table

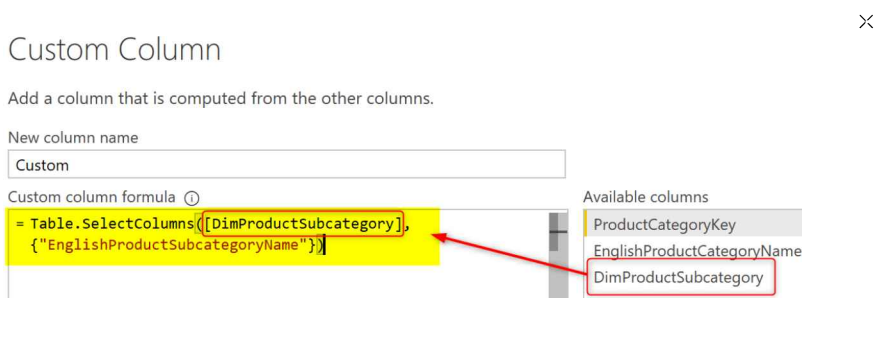
The sub-table of subcategories that you see in the above screenshot has two extra columns. All we need is just a list of names. You can use Table.SelectColumns Power Query function to achieve only the columns you want. To use this function, you can add a new custom column first;



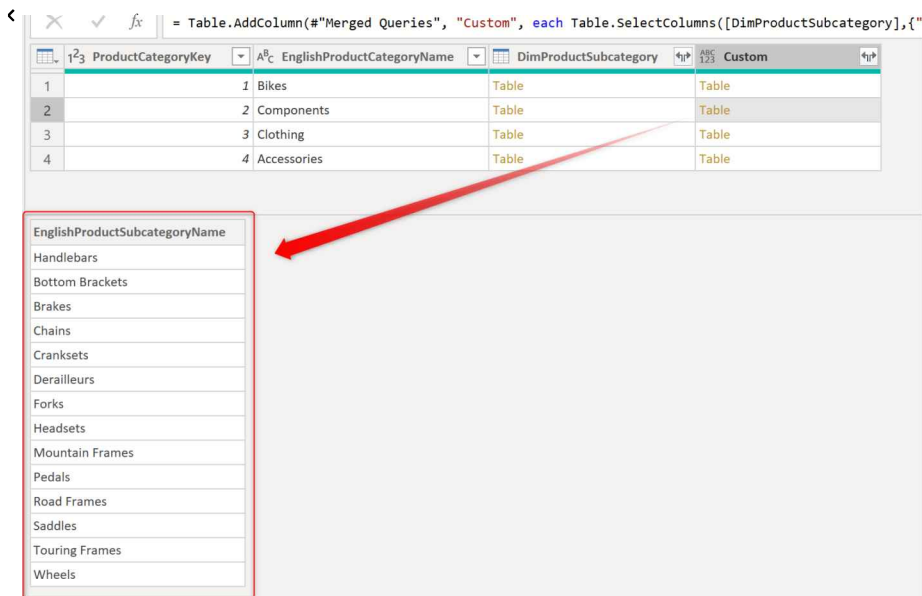
Part 9: Table and List functions

Then you can use the `Table.SelectColumns` function to only select the `EnglishProductSubcategoryName`.

`Table.SelectColumns([DimProductSubcategory],{"EnglishProductSubcategoryName"})`



One input of this function is the table column (coming from the previous step's result, the column with a table in every cell), and the other input is the list of columns we want as the output. This column would now produce a table in every cell, but this new table would have only one column: the `EnglishProductSubcategoryName`.



Convert the Table to a List

You can now convert the sub-table of subcategories to a list so that it can be then concatenated. Here is how you can do it using Add a Custom Column again and using the `Table.ToList` function:

`Table.ToList([Single Column Product Subcategory])`

Please note that I had a column name change before this step; I renamed the column from the previous step to “Single Column Product Subcategory,” which I am using for the Table.ToList Power Query function.

Custom Column

Add a column that is computed from the other columns.

New column name

List of Product subcategories

Custom column formula

= Table.ToList([Single Column Product Subcategory])

Available columns

ProductCategoryKey

EnglishProductCategoryName

DimProductSubcategory

Single Column Product Subcateg...

Now I have a list in every cell;

| ProductCategoryKey | EnglishProductCategoryName | DimProductSubcategory | Single Column Product Subcategory | List of Product subcategories |
|--------------------|----------------------------|-----------------------|-----------------------------------|-------------------------------|
| 1 | 1 Bikes | Table | Table | List |
| 2 | 2 Components | Table | Table | List |
| 3 | 3 Clothing | Table | Table | List |
| 4 | 4 Accessories | Table | Table | List |

Expand List using Extract Values

Now the final step is to expand the list. However, when you want to expand the list, there are two items: Expand to new rows, Expand to New Rows, and Extract Values.

ABC 123 List of Product subcategories

List

Expand to New Rows

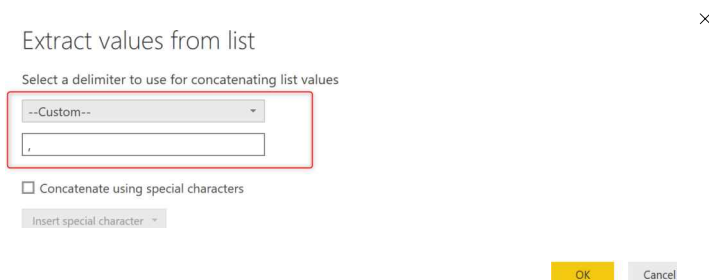
Extract Values...

List

List

Part 9: Table and List functions

If you use Expand to New Rows, you will get one product subcategory in each row, which we do not want. However, if you use the Extract Values, you can choose how the concatenated values would be delimited. If you want to have a command and then space between values, then you can choose custom and enter the characters that way;



Note that there is a space after comma character in the above screenshot, which you can't see.

Now here is the final result: (I have removed extra columns)

| ProductCategoryKey | EnglishProductCategoryName | List of Product subcategories |
|--------------------|----------------------------|--|
| 1 | Bikes | Mountain Bikes, Road Bikes, Touring Bikes |
| 2 | Components | Handlebars, Bottom Brackets, Brakes, Chains, Cranksets, Derailleurs, Forks, Headsets, Mou... |
| 3 | Clothing | Bib-Shorts, Caps, Gloves, Jerseys, Shorts, Socks, Tights, Vests |
| 4 | Accessories | Bike Racks, Bike Stands, Bottles and Cages, Cleaners, Fenders, Helmets, Hydration Packs, Li... |

Here is the whole M expression of this query if you are interested:

let

```
Source = Excel.Workbook(File.Contents("C:\Users\RezaRad\OneDrive\00 Power BI from Rookie to Rock Star - Module 1 Power BI for Data Analysts\1-Power BI Essentials\Data Sources\AdventureWorksDW2012.xlsx"), null, true),
```

```
DimProductCategory_Sheet = Source[Item="DimProductCategory",Kind="Sheet"][Data],
```

```
#"Promoted Headers" = Table.PromoteHeaders(DimProductCategory_Sheet,  
[PromoteAllScalars=true]),
```

```
#"Changed Type" = Table.TransformColumnTypes(#"Promoted  
Headers",{{"ProductCategoryKey", Int64.Type}, {"ProductCategoryAlternateKey", Int64.Type},  
{"EnglishProductCategoryName", type text}, {"SpanishProductCategoryName", type text},  
{"FrenchProductCategoryName", type text}}),
```

```
#"Removed Columns" = Table.RemoveColumns(#"Changed  
Type",{ProductCategoryAlternateKey", "SpanishProductCategoryName",  
"FrenchProductCategoryName"}),
```

```
#"Merged Queries" = Table.NestedJoin(#"Removed Columns", {"ProductCategoryKey"},  
DimProductSubcategory, {"ProductCategoryKey"}, "DimProductSubcategory", JoinKind.LeftOuter),
```

```
#"Added Custom" = Table.AddColumn(#"Merged Queries", "Custom", each
Table.SelectColumns([DimProductSubcategory],{"EnglishProductSubcategoryName"})),

#"Renamed Columns1" = Table.RenameColumns(#"Added Custom",{{"Custom", "Single Column
Product Subcategory"}}),

#"Added Custom1" = Table.AddColumn(#"Renamed Columns1", "List of Product subcategories",
each Table.ToList([Single Column Product Subcategory])),

#"Extracted Values" = Table.TransformColumns(#"Added Custom1", {"List of Product
subcategories", each Text.Combine(List.Transform(_, Text.From), ", ", type text)},

#"Removed Columns1" = Table.RemoveColumns(#"Extracted
Values",{"DimProductSubcategory", "Single Column Product Subcategory"})

in

#"Removed Columns1"
```

The generated concatenated list can be used as a normal column in a table visual in Power BI:

| Category | List of Product subcategories |
|-------------|---|
| Accessories | Bike Racks, Bike Stands, Bottles and Cages, Cleaners, Fenders, Helmets, Hydration Packs, Lights, Locks, Panniers, Pumps, Tires and Tubes |
| Bikes | Mountain Bikes, Road Bikes, Touring Bikes |
| Clothing | Bib-Shorts, Caps, Gloves, Jerseys, Shorts, Socks, Tights, Vests |
| Components | Handlebars, Bottom Brackets, Brakes, Chains, Cranksets, Derailleurs, Forks, Headsets, Mountain Frames, Pedals, Road Frames, Saddles, Touring Frames, Wheels |

Summary

When you have a list of values in a column and want to concatenate them together, there are two ways: Doing in DAX using functions such as ConcatenateX, or doing it in Power Query. For some scenarios, doing it in Power Query speed up the process, especially if the concatenation doesn't need to be dynamic and can be pre-calculated. Using the steps above, you have seen how you can only select one column from a table, then convert it to a list, and finally, extract values from it. There are other methods to achieve the same thing in Power Query too.

Chapter 44: Search for a Column in the Entire Database with Table.ColumnNames in Power Query and Power BI

</

Sometimes for tables with too many columns, and also for databases with too many tables, you do need a bit of help to explore the data. As an example, you know that you are looking for a column named “account status,” but the column does not exist in the accounts table. You need to search through all database tables for that column and find out which table has that value in it. Power Query has a great function that can help in such a scenario. Table.ColumnNames is a function that we are going to check out in this chapter.

The Problem

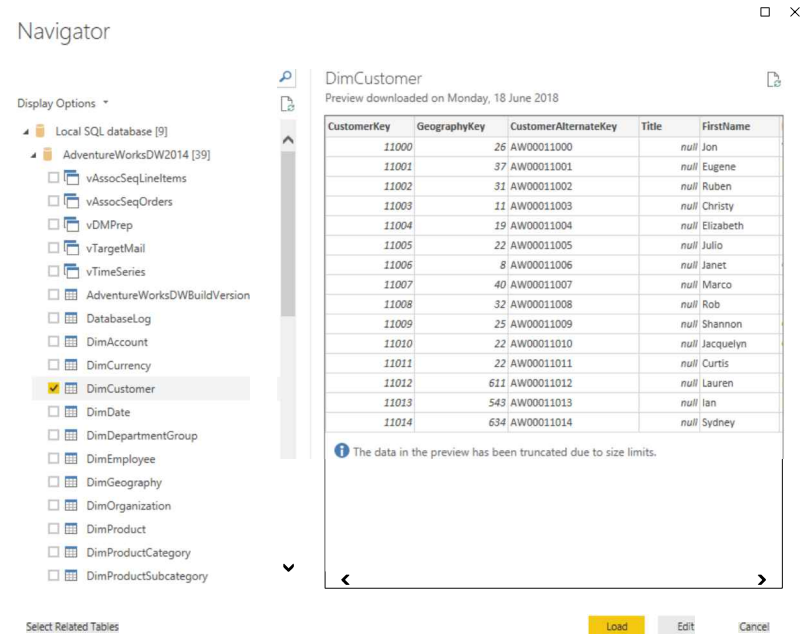
You have a database with hundreds of tables or even more, and each table has many columns. You are looking for a specific column in the database and want to find out tables that such column exists in. Power Query has a function named Table.ColumnNames, which gives you a list of all columns in a table as the output. Let’s see how this function works.

Sample Dataset

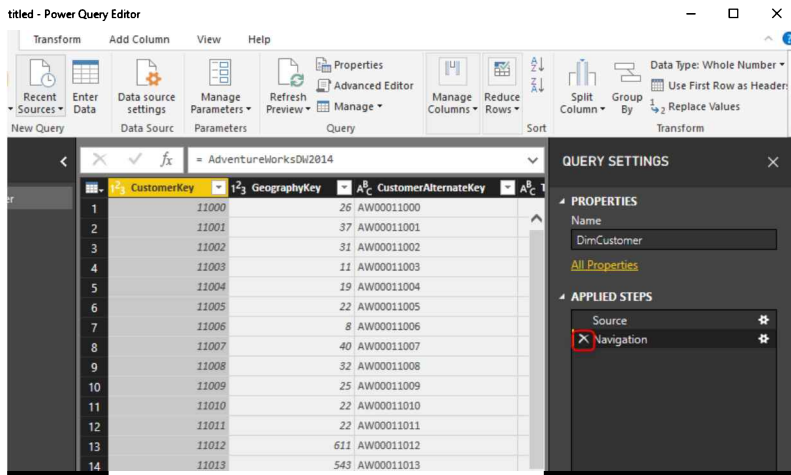
For this example, you can connect to the [AdventureWorksDW database](https://github.com/Microsoft/sql-server-database)[\[https://github.com/Microsoft/sql-server-](https://github.com/Microsoft/sql-server-database)

Chapter 44: Search for a Column in the Entire Database with Table.ColumnNames in Power Query and Power BI

[samples/releases/download/adventureworks/AdventureWorks2012.bak](#). Then select any of the tables to import data from the source database.

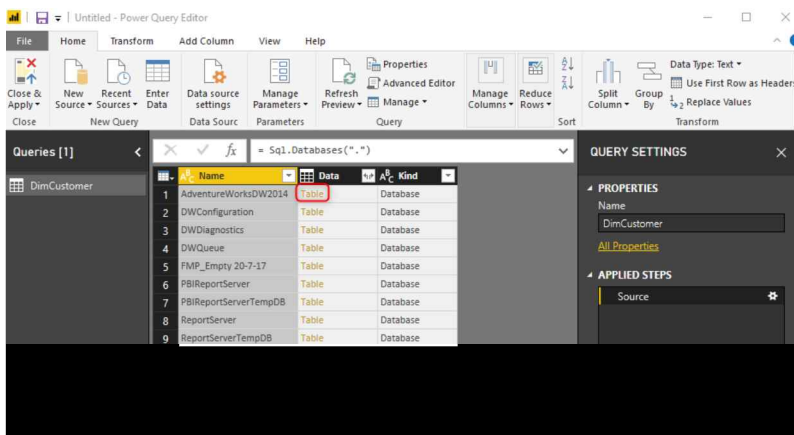


In the list of steps, you will find one step named Navigation. This is the step that we have navigated to the table. Remove this step.

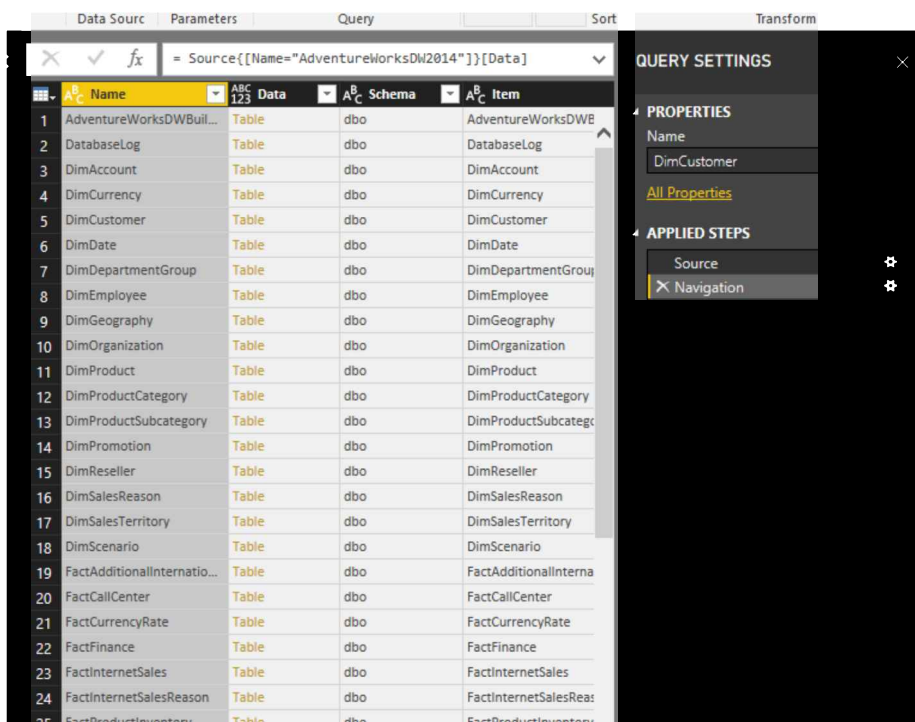


Then you will most probably see the list of all databases that you have access to (if you haven't chosen a specific database at the connection to SQL Server section). Click on the "Table" in the Data row of the database of AdventureWorks2014 (or any other databases that you want to explore columns in it).

Part 9: Table and List functions



This action will give you the list of all tables under that database.

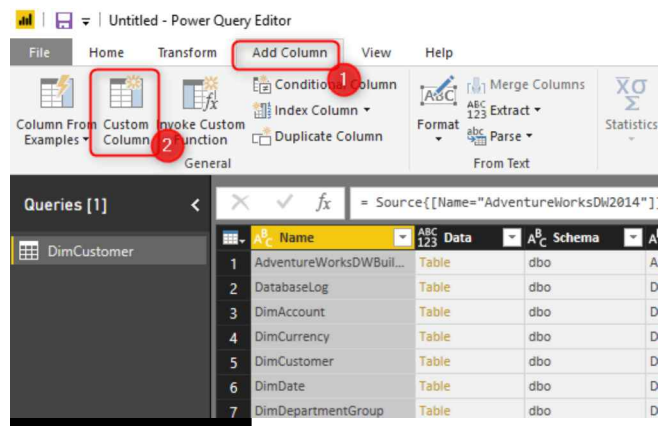


Now let's assume we want to search for the Account column in the entire database. Let's see how this is possible.

Table.ColumnNames

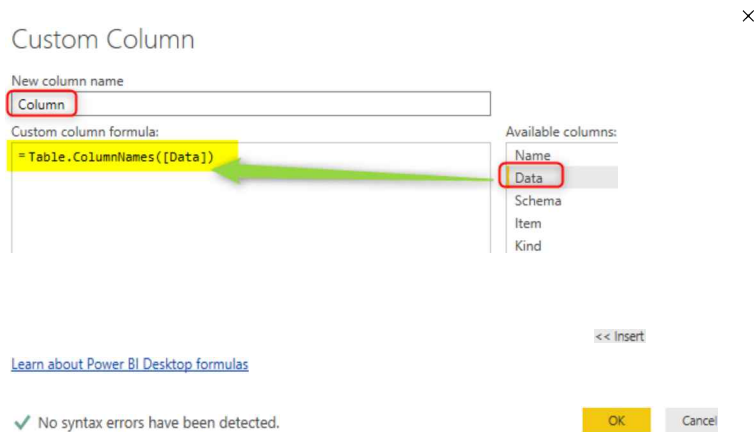
The Table.ColumnNames function in Power Query will give you the list of all the columns in any given table. You need to call this function by passing the table as the input, and you will get a list of column names as the output. To use it in the existing example, click on Add Column, and then Add Custom Column.

Chapter 44: Search for a Column in the Entire Database with Table.ColumnNames in Power Query and Power BI



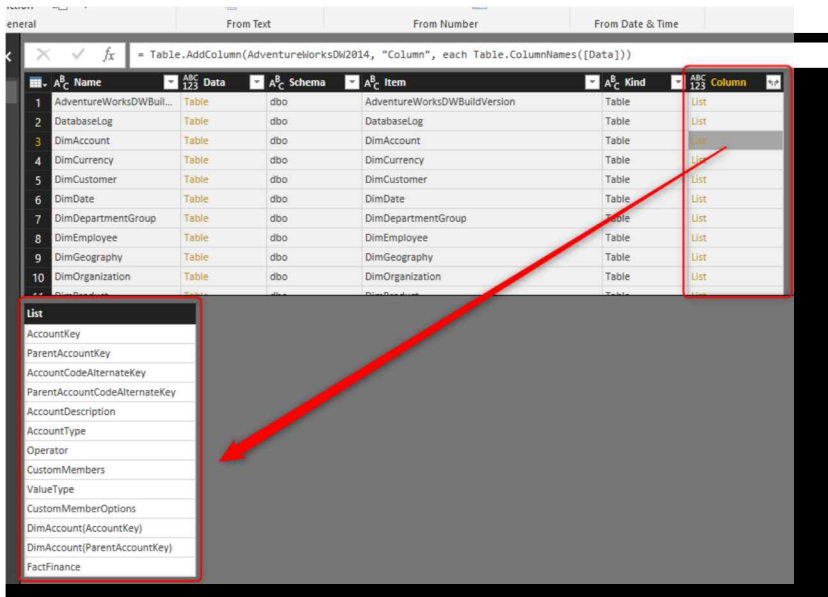
In the Custom Column expression section, you can write the Table.ColumnNames function with the input parameter of Data (Data is the column that includes the data table). Please note that Power Query is case sensitive and Table.ColumnNames should be written exactly as mentioned here in this chapter.

`=Table.ColumnNames([Data])`

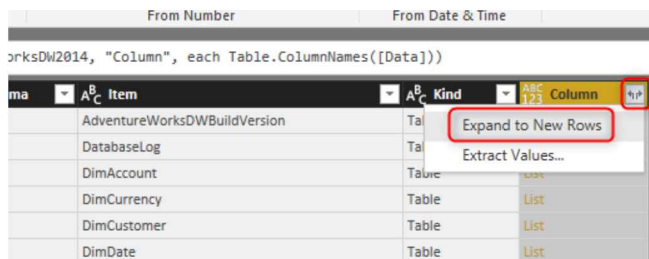


This action will give you a new column with a list in every cell. This list is the list of column names for every table.

Part 9: Table and List functions



Now to get the list of all columns in all tables, you just need to expand this column;



After expanding, you will have all columns in all tables listed under this "Column" field. You can convert it to Text data type.

Chapter 44: Search for a Column in the Entire Database with Table.ColumnNames in Power Query and Power BI

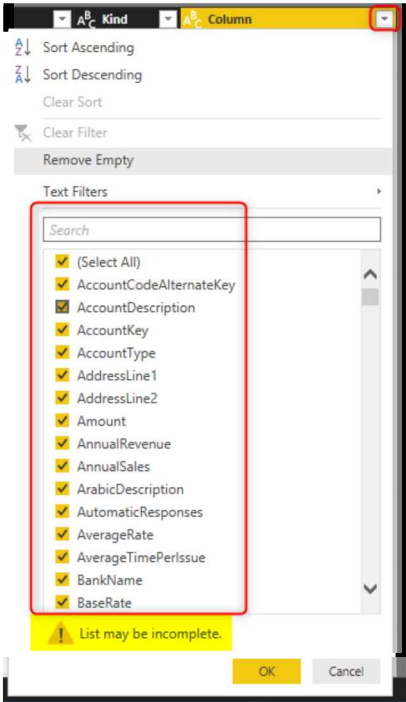
Table.TransformColumnTypes("#Expanded Column",{{"Column", type text}})

| ABC | ABC | ABC | ABC | ABC | ABC |
|--------------------------|-------|--------|------------------------------|-------|-------------------------------|
| Name | Data | Schema | Item | Kind | Column |
| AdventureWorksDWBuild... | Table | dbo | AdventureWorksDWBuildVersion | Table | dboVersion |
| AdventureWorksDWBuild... | Table | dbo | AdventureWorksDWBuildVersion | Table | VersionDate |
| DatabaseLog | Table | dbo | DatabaseLog | Table | DatabaseLogID |
| DatabaseLog | Table | dbo | DatabaseLog | Table | PostTime |
| DatabaseLog | Table | dbo | DatabaseLog | Table | DatabaseUser |
| DatabaseLog | Table | dbo | DatabaseLog | Table | Event |
| DatabaseLog | Table | dbo | DatabaseLog | Table | Schema |
| DatabaseLog | Table | dbo | DatabaseLog | Table | Object |
| DatabaseLog | Table | dbo | DatabaseLog | Table | TSQL |
| DatabaseLog | Table | dbo | DatabaseLog | Table | XmlEvent |
| DimAccount | Table | dbo | DimAccount | Table | AccountKey |
| DimAccount | Table | dbo | DimAccount | Table | ParentAccountKey |
| DimAccount | Table | dbo | DimAccount | Table | AccountCodeAlternateKey |
| DimAccount | Table | dbo | DimAccount | Table | ParentAccountCodeAlternateKey |
| DimAccount | Table | dbo | DimAccount | Table | AccountDescription |
| DimAccount | Table | dbo | DimAccount | Table | AccountType |
| DimAccount | Table | dbo | DimAccount | Table | Operator |
| DimAccount | Table | dbo | DimAccount | Table | CustomMembers |
| DimAccount | Table | dbo | DimAccount | Table | ValueType |
| DimAccount | Table | dbo | DimAccount | Table | CustomMemberOptions |
| DimAccount | Table | dbo | DimAccount | Table | DimAccount(AccountKey) |
| DimAccount | Table | dbo | DimAccount | Table | DimAccount(ParentAccountKey) |
| DimAccount | Table | dbo | DimAccount | Table | FactFinance |
| DimCurrency | Table | dbo | DimCurrency | Table | CurrencyKey |
| DimCurrency | Table | dbo | DimCurrency | Table | CurrencyAlternateKey |
| DimCurrency | Table | dbo | DimCurrency | Table | CurrencyName |

List of all columns in every table or view

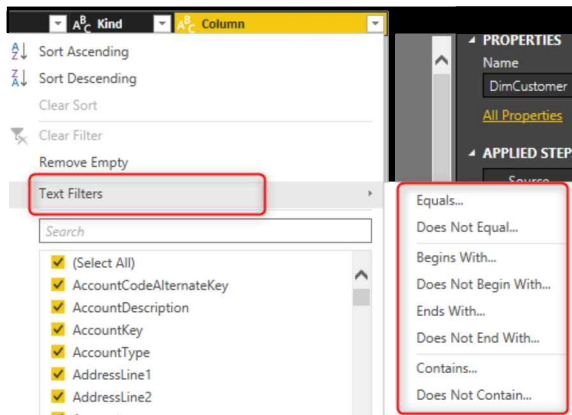
Search through Columns

Now that we got the list of all column names, then searching through it is very simple. You can use the basic search, but remember if you have more than 1000 columns in your data source, this will show you a limited list.

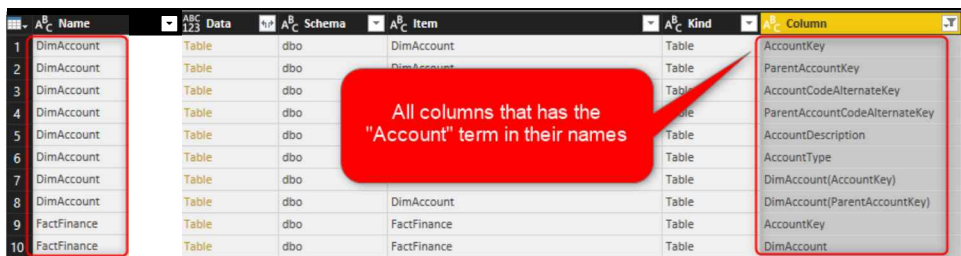


Part 9: Table and List functions

The best way to search into this list is using Advanced Search options. You can choose criteria such as Contains, Equals, Begins With, Ends With, etc.

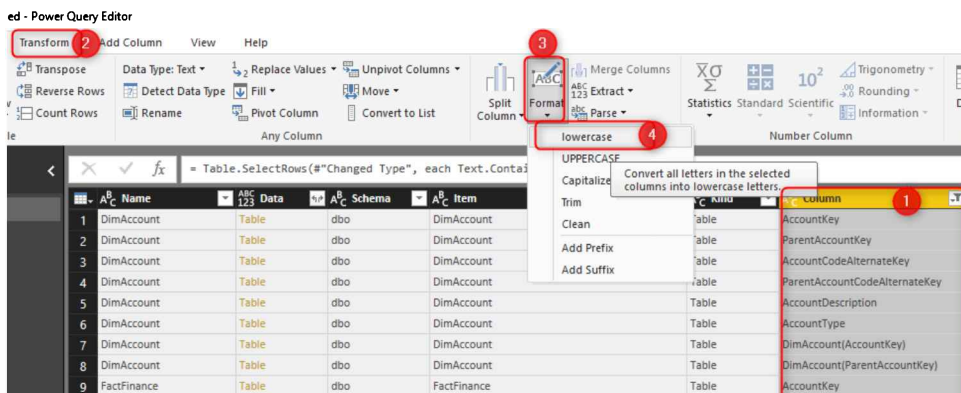


For example, a search for Contains... “Account” will end up with the below result:



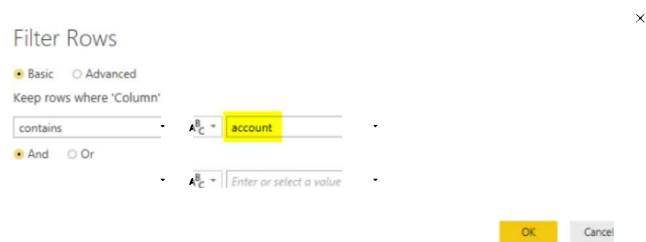
Be Careful of Case Sensitivity

Power Query is a case-sensitive language. There is a difference between “Account” as a text and “account” as a text. One is using capital A, and the other one; lowercase a. To make sure you can always search for an item, regardless of the case sensitivity, you can first convert the column names all to lower case or upper case. To do that, select the “Column” field, and from the Transform tab, select transform to Lower.



Chapter 44: Search for a Column in the Entire Database with Table.ColumnNames in Power Query and Power BI

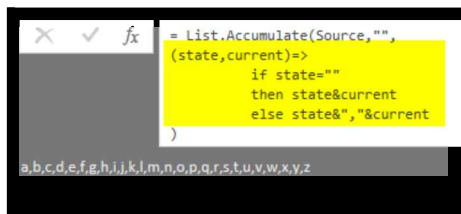
now the whole column names list will be lowercase, and you can search through it only with lowercase values;



Summary

Sometimes, simple transformations such as getting the list of columns from a table can help data exploration. In this chapter, you’ve seen how this can be helpful to search through all columns in a database. This approach can be used for any data source, regardless of SQL Server databases or anything else. As long as the data source has a table structure, you can get the list of all columns from that table. This approach is particularly useful when you connect to databases with thousands of tables. Each table has hundreds of columns; CRM or Dynamics data sources are examples.

Chapter 45: List.Accumulate Hidden Gem of Power Query List Functions in Power BI

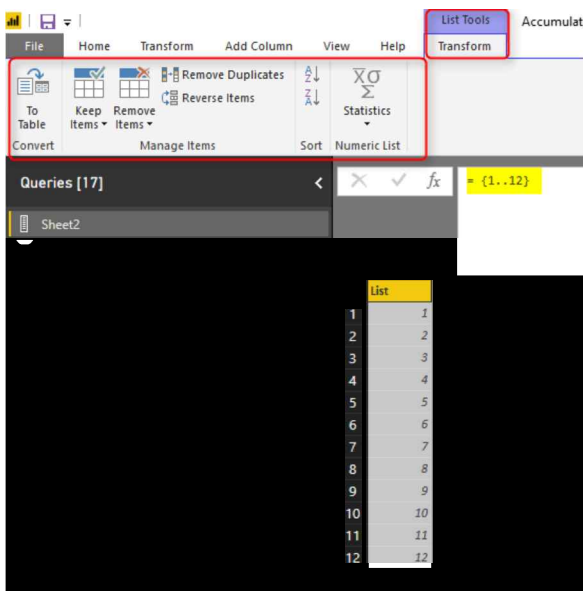


There are many List transformations available in Power Query's graphical interface. However, the number of functions in the graphical interface is very limited. In this chapter, I will explain a function that is powerful and is not yet listed in the graphical interface. List.Accumulate is a function that loops through items of a list and applies a transformation. This function at the time of writing this chapter is only available through Power Query M scripting.

List Transformations in Graphical Interface of Power Query

If you have a list, you can see what transformations to apply to it using the graphical interface. you can create a list with a simple M script as below;

= {1..12}



This code generates a list of numbers from 1 to 12. In the top menu under List Tools, you can see many transformations available for List. These items are very few options such as; convert to a table, keeping items, removing items, etc. Altogether considering all options in every element, this list is not more than 20 functions. However, let’s look at the number of List functions in the M script.

List Functions in M; Power Query Formula Language

I have written previously about using the #shared key to finding all functions available in M. you can then filter it to only “List.” Functions and then you will end up with a bit list of functions; 69 functions! More than three times what you see in the graphical interface.

| | A8C Name | AAC 123 Value |
|----|--------------------|---------------|
| 1 | List.Accumulate | Function |
| 2 | List.AllTrue | Function |
| 3 | List.Alternate | Function |
| 4 | List.AnyTrue | Function |
| 5 | List.Average | Function |
| 6 | List.Buffer | Function |
| 7 | List.Combine | Function |
| 8 | List.Contains | Function |
| 9 | List.ContainsAll | Function |
| 10 | List.ContainsAny | Function |
| 11 | List.Count | Function |
| 12 | List.Covariance | Function |
| 13 | List.DateTimeZones | Function |
| 14 | List.DatesTimes | Function |
| 15 | List.Dates | Function |
| 16 | List.Difference | Function |
| 17 | List.Distinct | Function |
| 18 | List.Durations | Function |
| 19 | List.FindText | Function |
| 20 | List.First | Function |
| 21 | List.FirstN | Function |
| 22 | List.Generate | Function |
| 23 | List.InsertRange | Function |
| 24 | List.Intersect | Function |
| 25 | List.IsDistinct | Function |
| 26 | List.IsEmpty | Function |
| 27 | List.Last | Function |
| 28 | List.LastN | Function |
| 29 | List.MatchesAll | Function |

Some of these functions Some of the functions in this list are very useful and powerful. An example of those functions is List.Dates, List.Numbers, List.Generate, List.Accumulate and many others. We cannot go through all functions in one chapter. In this chapter, I’ll be covering List.Accumulate.

List.Accumulate Function

List.Accumulate is a function that can easily save many steps in your Power Query transformations. Instead of applying multiple steps, you can simply use List.Accumulate to overcome what you want. List.Accumulate function loops through the list and accumulate a value as a result. This function usually needs three parameters; the list itself, seed, and accumulator. Here are parameters explained in detail;

- list; the list that we want to apply the transformation to it.
- Seed; is the initial value.
- Accumulator; is a function. This function determines what accumulation calculation happens on items on the list. The way that this function is defined is exactly the way that you write a function in Power Query M script using Lambda expressions.

The best way to learn about seed and accumulator is through some examples. Let's apply some transformations with List.Accumulate and see how these two parameters are working.

Accumulate to Calculate Sum

The sum is a function that accumulates every two values in the list until the end of the list. If you want to write Sum with List.Accumulate, you can do it with this expression:

```
= List.Accumulate(Source,0,(state,current)=>state+current)
```

The function part of this expression is: *(state, current)=>state+current*

state is the value accumulated in the calculation. Current is the current item in the list.
seed is the initial value of the state

Let's see how the calculation works. To clarify it more in detail, I explained the value of the state, current, and accumulator in every step;

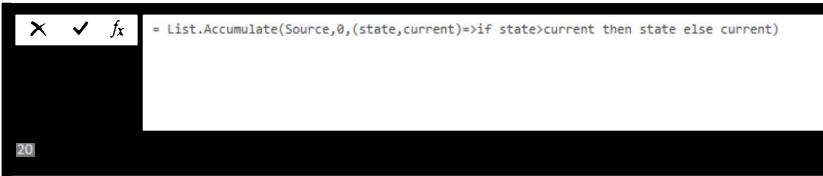
| List | state | current | (state,current)=>state+current |
|------|-------|---------|--------------------------------|
| 1 | 0 | 1 | 1 |
| 2 | 1 | 2 | 3 |
| 3 | 3 | 3 | 6 |
| 4 | 6 | 4 | 10 |
| 5 | 10 | 5 | 15 |
| 6 | 15 | 6 | 21 |
| 7 | 21 | 7 | 28 |
| 8 | 28 | 8 | 36 |
| 9 | 36 | 9 | 45 |
| 10 | 45 | 10 | 55 |
| 11 | 55 | 11 | 66 |
| 12 | 66 | 12 | 78 |
| 13 | 78 | 13 | 91 |
| 14 | 91 | 14 | 105 |
| 15 | 105 | 15 | 120 |
| 16 | 120 | 16 | 136 |
| 17 | 136 | 17 | 153 |
| 18 | 153 | 18 | 171 |
| 19 | 171 | 19 | 190 |
| 20 | 190 | 20 | 210 |

List.Accumulate loops through every item in the list and run the accumulator function. The very first time, the state value is equal to seed, which in this case is zero. Current is the current value in the list. For the very first item, that value is 1. so the accumulator result is state+current=0+1=1. This value then will be the state of the next item on the list. For the next item, the state is 1 (calculated from the previous row), and the current is 2. state+current becomes 1+2=3. This process continues through the whole list, so the final state value for a list from 1 to 20 becomes 210, equal to the sum of those values. In every row of the list, we added that to the previous row's result.

Accumulate to Calculate Max

learning how the accumulate function can cover basic tasks helps you to understand how the accumulator function works. For applying Max, you need to compare every two items in the list and pick the bigger one. Here is the script;

= List.Accumulate(Source,0,(state,current)=>if state>current then state else current)



Remember the same thing about the state, current, and seed. This is how the calculation works;

Part 9: Table and List functions

| List | state | current | (state,current)=>if state>current then state else current |
|------|-------|---------|---|
| 1 | 0 | 1 | 1 |
| 2 | 1 | 2 | 2 |
| 3 | 2 | 3 | 3 |
| 4 | 3 | 4 | 4 |
| 5 | 4 | 5 | 5 |
| 6 | 5 | 6 | 6 |
| 7 | 6 | 7 | 7 |
| 8 | 7 | 8 | 8 |
| 9 | 8 | 9 | 9 |
| 10 | 9 | 10 | 10 |
| 11 | 10 | 11 | 11 |
| 12 | 11 | 12 | 12 |
| 13 | 12 | 13 | 13 |
| 14 | 13 | 14 | 14 |
| 15 | 14 | 15 | 15 |
| 16 | 15 | 16 | 16 |
| 17 | 16 | 17 | 17 |
| 18 | 17 | 18 | 18 |
| 19 | 18 | 19 | 19 |
| 20 | 19 | 20 | 20 |

The function **(state, current)=>if state>current then state else current** will run on every row and give you the result. Remember that seed value should be a value less than any other value for this case.

Accumulate as Product or Divide

you can use the same logic with a different accumulator to calculate Product or Divide; here is the calculation for Product;

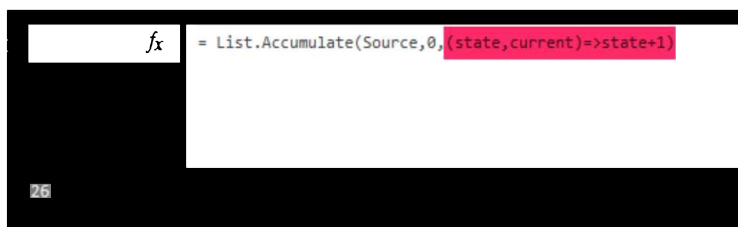
| | |
|---------------------|---|
| f_x | = List.Accumulate(Source, 1, (state, current) => state * current) |
| 2432902008176640000 | |

For Product or Divide, you need to set the seed as one because if it is zero, then divide or multiply, considering zero will end up zero always. Here is the calculation for Divide;

| | |
|-------------------------|---|
| f_x | = List.Accumulate(Source, 1, (state, current) => state / current) |
| 1.0715102881254669E-158 | |

Accumulate as Count

The count of a list is the number of items in a list. This can be achieved with no need of the current item, just using seed and state as below;

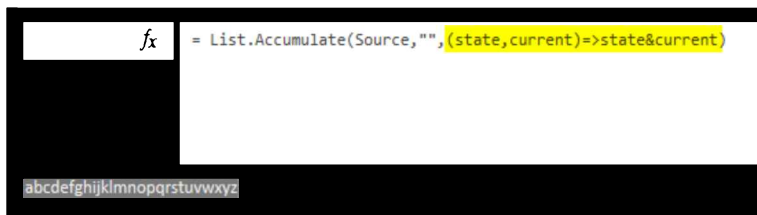


The screenshot shows the Power Query formula bar with the following M code: `= List.Accumulate(Source, 0, (state, current) => state + 1)`. The formula bar is highlighted in blue, and the result of the calculation, 26, is shown in a small box at the bottom left.

In every row, the value would be plus one of the state values from the previous row's calculation. As a result, you get the count.

Accumulate as Concatenate (with delimiter or without)

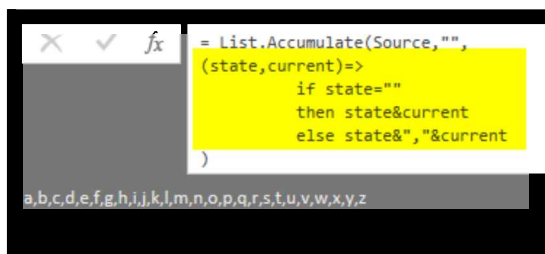
So far, we applied to accumulate on lists that had a number in every item. Now, let's apply it to a list of text items. To concatenate two items, you just need to add them one after each other with a concatenation character, the ampersand (&).



The screenshot shows the Power Query formula bar with the following M code: `= List.Accumulate(Source, "", (state, current) => state & current)`. The formula bar is highlighted in blue, and the result of the calculation, a string of all lowercase letters from 'a' to 'z', is shown in a small box at the bottom left.

This worked on a list of items which are text values {"a".. "z"}, and the result is the concatenation of all items in the list. Please note that the seed, in this case, is an empty text.

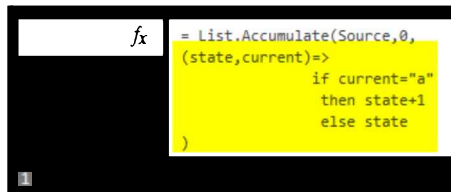
If you want to add a delimiter between items, you can add the delimiter plus a condition to check if this is the very first item or not.



The screenshot shows the Power Query formula bar with the following M code: `= List.Accumulate(Source, "", (state, current) => if state="" then state & current else state & ", " & current)`. The formula bar is highlighted in blue, and the result of the calculation, a string of all lowercase letters from 'a' to 'z' separated by commas, is shown in a small box at the bottom left.

Accumulate as Count Token Exact Match

now that you've learned the logic of the accumulator, you can apply it to do any expressions. For example, if you want to calculate the count of items in the list whose value matches exactly with a value, you can write this expression;

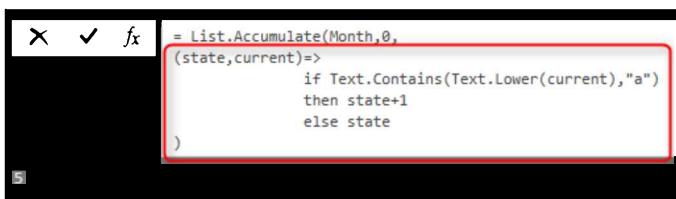


```
= List.Accumulate(Source,0,  
(state,current)=>  
    if current="a"  
    then state+1  
    else state  
)
```

The logic is simple, if the item matches with "a" (which in this case is our token), then count it, otherwise don't.

Accumulate as Count Token Partial Match

Similar to the previous calculation. However, this time we want to count the item, even it partially matches the text. This can be done with the help of Text.Contains function. Because of Text.Contains might not find the lower case or upper case matches; we convert it to lower case beforehand.



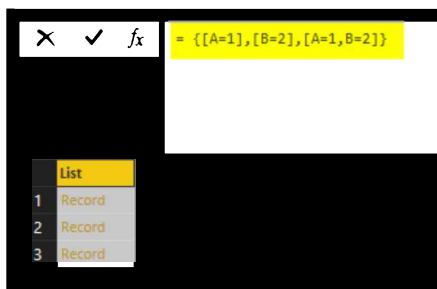
```
= List.Accumulate(Month,0,  
(state,current)=>  
    if Text.Contains(Text.Lower(current),"a")  
    then state+1  
    else state  
)
```

Accumulate as Conditions on Records

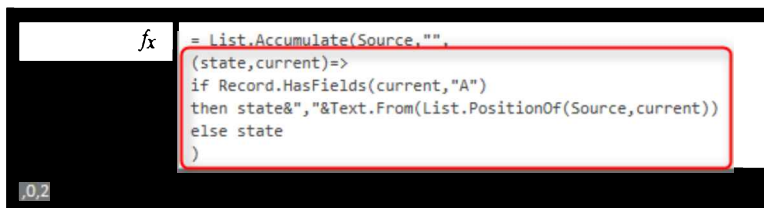
So far, we went through a lot of use cases and examples of List.Accumulate function. You understand that this function can be so powerful and useful in many scenarios. However, the main use cases of List.Accumulate is to apply to scenarios which other functions cannot resolve easily. List.Sum might be better used than List.Accumulate, which only calculates the sum. However, there are many scenarios where steps are needed to get the result you want with normal functions. In those cases, List.Accumulate is your friend.

For example, consider that you have a list, and this list is a list of records! Every record might have a different set of fields, and you want to fetch only records that have a specific field on their list and get their position as a concatenated result. This process, using other list functions, might take many steps, however, with List.Accumulate that is easy.

Here is the sample input list;



As you can see, the list includes records, and to find out what each record has, you need to expand it. A list of records cannot be expanded because it will add the number of columns, and the list can have only one column. So you have to convert it to a table and then expand it. As you feel now, there are many steps required to get the result we wanted. However, List.Accumulate can be a big help in this scenario. Here is the calculation with List.Accumulate;



```
= List.Accumulate(Source, "",
(state,current)=>
if Record.HasFields(current,"A")
```

```
then state & ", " & Text.From(List.PositionOf(Source, current))  
else state  
)
```

Record.HasFields is used to determine if a record contains a field ("A" in this example).

List.PositionOf is used to get the position of that record that satisfies the criteria above.

Summary

This chapter explained how List.Accumulate works. List.Accumulate is a very powerful function that can easily save many steps in your Power Query transformations. In this chapter, you've learned the basics of this function using it for simple operations such as Sum, Divide, Product, Max, Count, etc. You also learned that the main power of this function is when basic functions cannot operate easily. You learned that the accumulator function gives you full power to write exactly what you want.

Chapter 46: Remove Columns with Specific Patterns Name in Power BI using Power Query

Remove Columns with Specific Patterns Name

| Type_1 | Country | Year | Type_2 | Code | Type_3 |
|--------|---------|------|--------|------|--------|
| | | | | | |
| | | | | | |
| | | | | | |

MDG_Export_20200909_050438692 (2)

```

let
    Source = Csv.Document(File.Contents("C:\Users\leila\Downloads\MDG_Export_20200909_050438692\MDG_Export_20200909_050438692.csv"),[Delim
    #"Changed Type" = Table.TransformColumnTypes(Source,{{"Column1", type text}, {"Column2", type text}, {"Column3", type text}, {"Column4
    #"Promoted Headers" = Table.PromoteHeaders(#"Changed Type", [PromoteAllScalars=true]),
    #"Changed Type1" = Table.TransformColumnTypes(#"Promoted Headers",{"Country", type text}, {"SeriesCode",
    Table.RemoveColumns(table as table, columns as any),
    table
    #"Converted to Table" = Table.From
    #"Filtered Rows" = Table.SelectRow
    RemoveColumns=Table.RemoveColumns()
    in
    Columns
    
```

I was in the middle of data transformation to create a clean dataset for a Charticulator article, that I come across a dataset as attached in the code files of this book.

This dataset is for **Carbon dioxide emissions** in different countries around the world.

The only columns I need from this dataset are the country name and the number of emissions for each year (the year column). I need to remove other columns.

There are two types of columns:

Columns with specific patterns name: In the above example, Type, Type_2, Type_3, and so forth and also, Footnotes_1, Footnotes_2, Footnotes_3, and so forth...

Columns without specific patterns:

Part 9: Table and List functions

Such as country code, series codes, and MDG.

In this example, I need to remove all the above columns for the columns without a pattern; in this example, there are not many, so I decided to remove them manually, but for the column with a specific pattern in their name, I wrote some codes in Power Query

| Country | 1.2 1990 | Footnotes | 1.2 1991 | Footnotes_1 | Type_2 |
|-----------------------|----------|-----------|----------|-------------|--------|
| 1 Afghanistan | 2676.9 | | 2493.6 | | |
| 2 Albania | 7488 | | 3971 | | |
| 3 Algeria | 78924.8 | | 8132.4 | | |
| 4 Andorra | null | | null | | |
| 5 Angola | 4429.7 | | 4367.4 | | |
| 6 Anguilla | null | | null | | |
| 7 Antigua and Barbuda | 300.7 | | 289.7 | | |
| 8 Argentina | 112613.6 | | 117021.3 | | |
| 9 Armenia | null | | null | | |
| 10 Aruba | 1840.8 | | 1628.8 | | |
| 11 Australia | 263848 | | 261574.4 | | |
| 12 Austria | 57722.2 | | 61658.6 | | |
| 13 Azerbaijan | null | | null | | |
| 14 Bahamas | 1550.8 | | 1782.2 | | |
| 15 Bahrain | 12665.8 | | 12335.8 | | |

Pattern filtering

As a result, I start to check the available column's name. To do that, I click on the Advanced Editor

```
let
    Source = Csv.Document(File.Contents("C:\Users\leila\Downloads\MDG_Export_20200909_050438692\MDG_Export_20200909_050438692.csv"),[Delimiter=";", Quote="\"", Encoding=65001]),
    #"Changed Type" = Table.TransformColumnTypes(Source,{{"Column1", type text}, {"Column2", type text}, {"Column3", type text}, {"Column4", type text}}),
    #"Promoted Headers" = Table.PromoteHeaders(#"Changed Type", [PromoteAllScalars=true]),
    #"Changed Type1" = Table.TransformColumnTypes(#"Promoted Headers",{{"CountryCode", Int64.Type}, {"Country", type text}, {"SeriesCode", type text}, {"Series", type text}}),
    #"Removed Columns" = Table.RemoveColumns(#"Changed Type1",{"CountryCode", "SeriesCode", "MDG", "Series"}),
    columnname=Table.ColumnNames(#"Removed Columns" );
in
    columnname
```

Replace the last three line with below

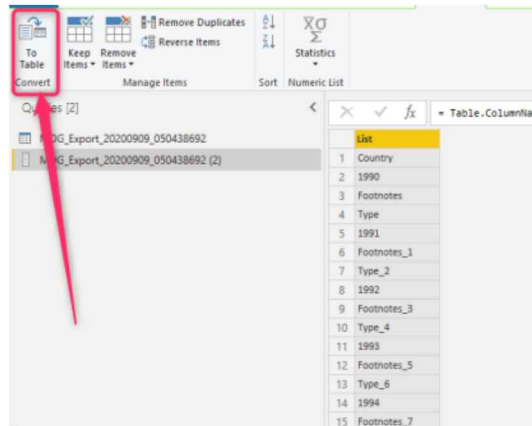
`columnname=Table.ColumnNames(#"Removed Columns")`

`in`

`columnname`

In this code, we will list the name of the columns as below, and then we will convert the list to a Table.

Chapter 46: Remove Columns with Specific Patterns Name in Power BI using Power Query

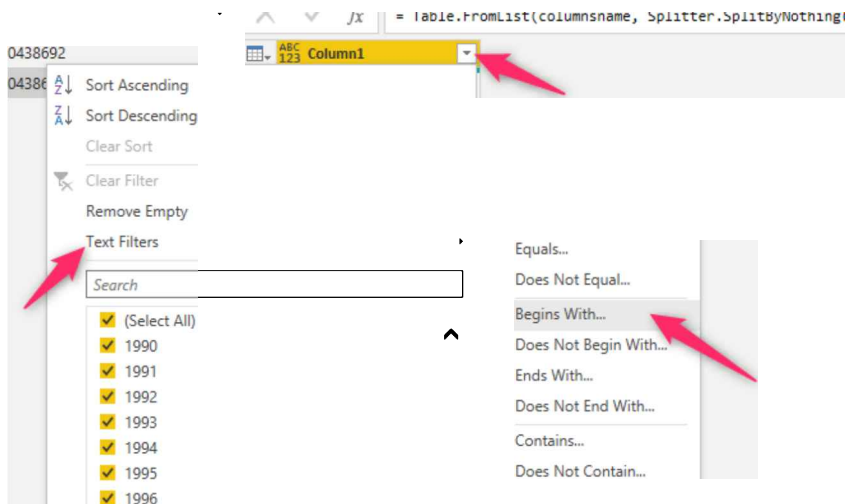


Now, I want to find and filter columns that have a specific pattern

1- all columns start with **Type**

2- All columns Start with **Footnote**

So in the new column, I am going to use the text filter



Next, in the new window, select the “Begin With” option. Power query is Case Sensitive, so make sure to write it down as it is.

Part 9: Table and List functions

Filter Rows

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

☒ Basic ☐ Advanced

Keep rows where 'Column1'

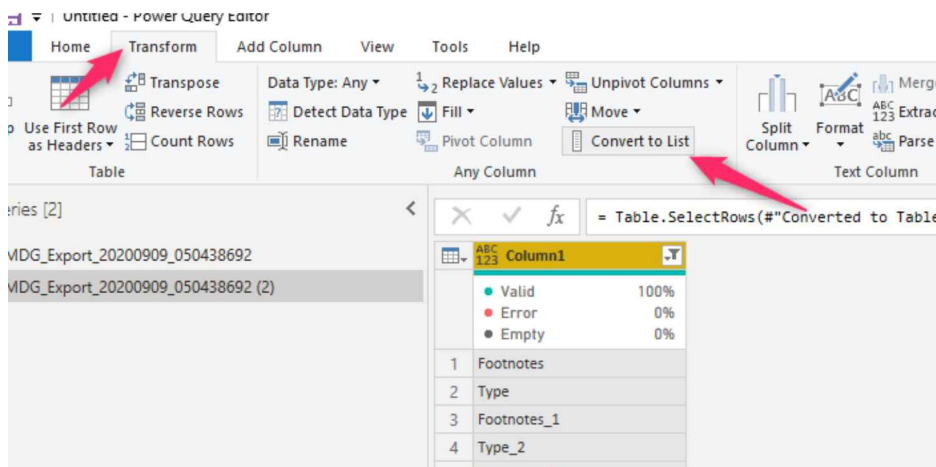
begins with

☐ And ☒ Or

begins with

OK Cancel

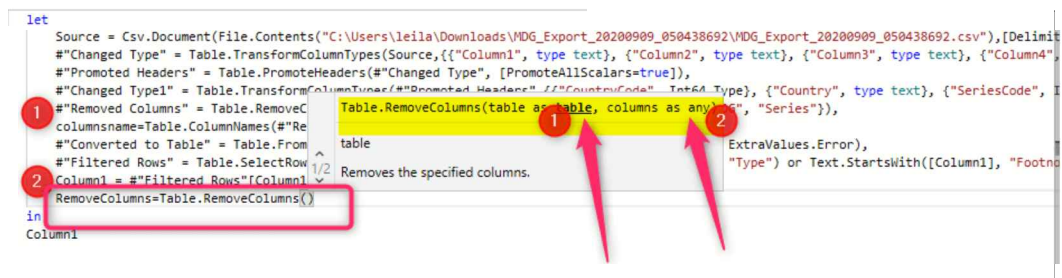
We are going to convert the columns to list again using the menu in Powe Query Editor; click on Transform, then Convert to list



Remove the filtered columns

Then back to Advance Editor to write some codes. We have the list of columns that need to be removed on the Column1. Also, we have the list of all columns as well

MDG_Export_20200909_050438692 (2)



I will add some code at the end as below, that we use the function, RemoveColumns, that accepted two parameters: The whole table and the name of columns need to be

Chapter 46: Remove Columns with Specific Patterns Name in Power BI using Power Query

removed. I have the list of the whole table on #” Removed Columns” and the list of columns that need to be removed from Column1

```
Column1 = #"Filtered Rows"[Column1],
RemoveColumns=Table.RemoveColumns( #"Removed Columns",Column1)
in
RemoveColumns
```

So the result is :

Table.RemoveColumns(#"Removed Columns",Column1)

| Country | 1.2.1990 | 1.2.1991 | 1.2.1992 | 1.2.1993 | 1.2.1994 | 1.2.1995 | 1.2.1996 | 1.2.1997 |
|-----------------------|----------|----------|----------|----------|----------|----------|----------|----------|
| 1 Afghanistan | 2676.9 | 2493.6 | 1426.5 | 1375.1 | 1220.1 | 1268.6 | 1199.1 | |
| 2 Albania | 7488 | 8971.4 | 2387.2 | 2342.2 | 1828.8 | 2086.5 | 2016.9 | |
| 3 Algeria | 78924.8 | 81220.4 | 81915.4 | 82294.8 | 86500.9 | 95445.7 | 97111.5 | |
| 4 Andorra | null | null | null | null | null | 407 | 425.4 | |
| 5 Angola | 4429.7 | 4367.4 | 4418.7 | 5801.2 | 5890.7 | 11012 | 10481.3 | |
| 6 Anguilla | null | null | null | null | null | null | null | |
| 7 Antigua and Barbuda | 300.7 | 289.7 | 289.7 | 304.4 | 311.7 | 322.7 | 322.7 | |
| 8 Argentina | 112613.6 | 117021.3 | 121447.4 | 118609.1 | 123310.5 | 122547.5 | 129217.7 | |
| 9 Armenia | null | null | 4052 | 2094.9 | 2066.6 | 1491 | 2607.2 | |
| 10 Aruba | 1840.8 | 1938.8 | 1721.5 | 1771.2 | 1761.8 | 1782.2 | 1800.5 | |
| 11 Australia | 261848 | 261574.4 | 268068.7 | 277478.2 | 278204.3 | 281941 | 302341.5 | |
| 12 Austria | 57722.2 | 61638.6 | 56888.2 | 57135.5 | 57095.2 | 59827.1 | 63226.4 | |
| 13 Azerbaijan | null | null | 19709.2 | 46365.2 | 42972.9 | 33479.7 | 21510.5 | |
| 14 Bahamas | 1950.8 | 1782.2 | 1792.2 | 1701.5 | 1686.6 | 1683.2 | 1668.5 | |
| 15 Bahrain | 12685.8 | 12355.8 | 11455.7 | 15702.1 | 15989.8 | 16021.1 | 15621.4 | |

Query Settings

PROPERTIES

Name
MDS_Export_20200609_010438662 (2)
All Properties

APPLIED STEPS

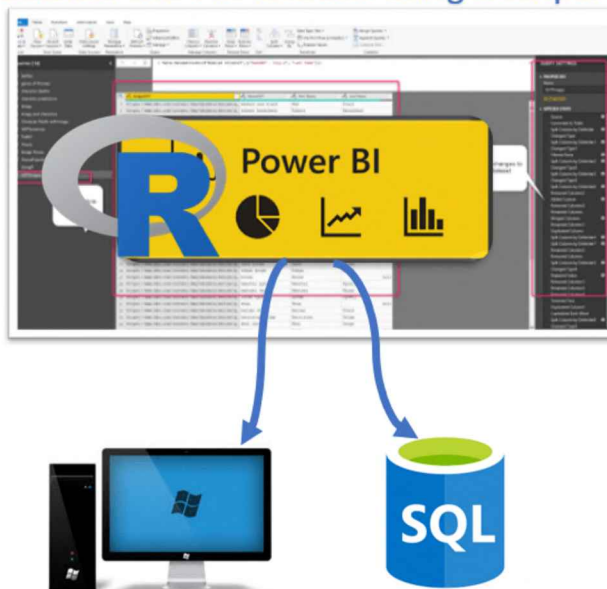
Source
Changed Type
Promoted Headers
Changed Type1
Removed Columns
columnname
Converted to Table
Filtered Rows
Column1
RemoveColumns

As you can see, all columns that start with Type or Footnote have been removed.



Chapter 47: Export data from Power Query to Local Machine or SQL Server using R scripts

Exporting Data from Power Query to SQL Database and File location using R scripts



A common question is how to store back the data from Power BI to local computers or SQL Server Databases. This short chapter will show how to do it by writing R scripts inside Power Query.

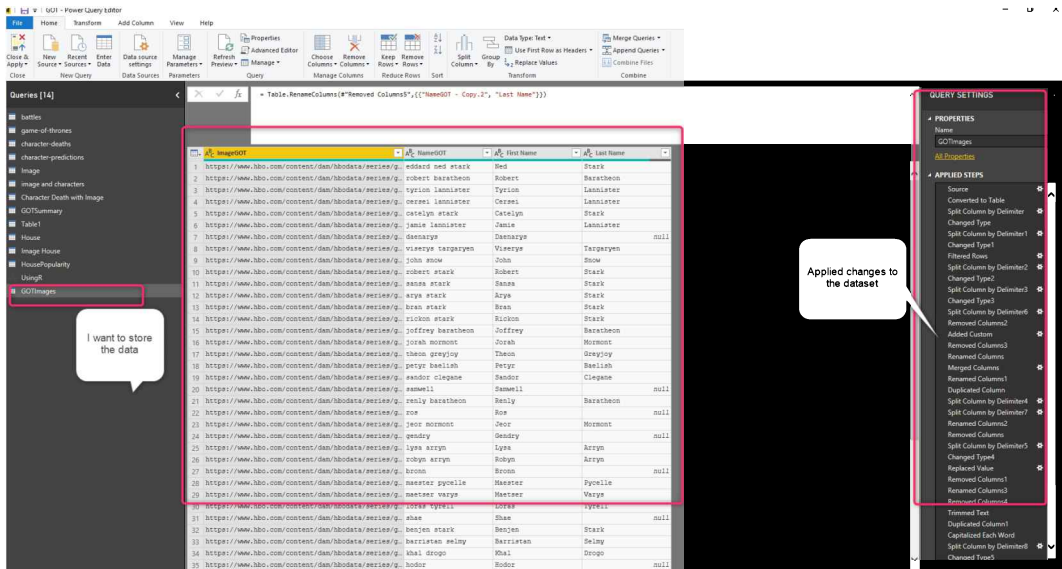
Export as CSV Files

To store data back to a file location in your PC, we can write one line of code in R.

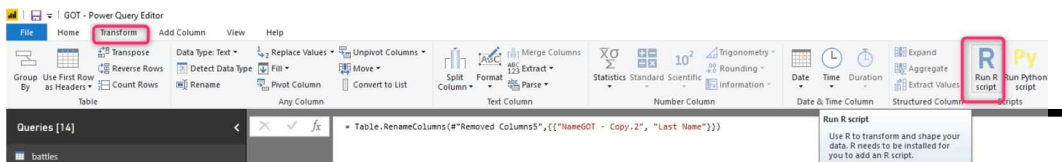
In my scenario, first, I get some data from <https://www.hbo.com/game-of-thrones/cast-and-crew>[\[https://www.hbo.com/game-of-thrones/cast-and-crew\]](https://www.hbo.com/game-of-thrones/cast-and-crew)

then I applied some changes to the data using Power Query. As you have seen in below picture, I applied more than 20 steps to fetch the images

Chapter 47: Export data from Power Query to Local Machine or SQL Server using R scripts



Now I want to store it back on my PC. To do that, you need to navigate to **Power Query Editor**, then click on **Transform**, then **Run R Scripts**

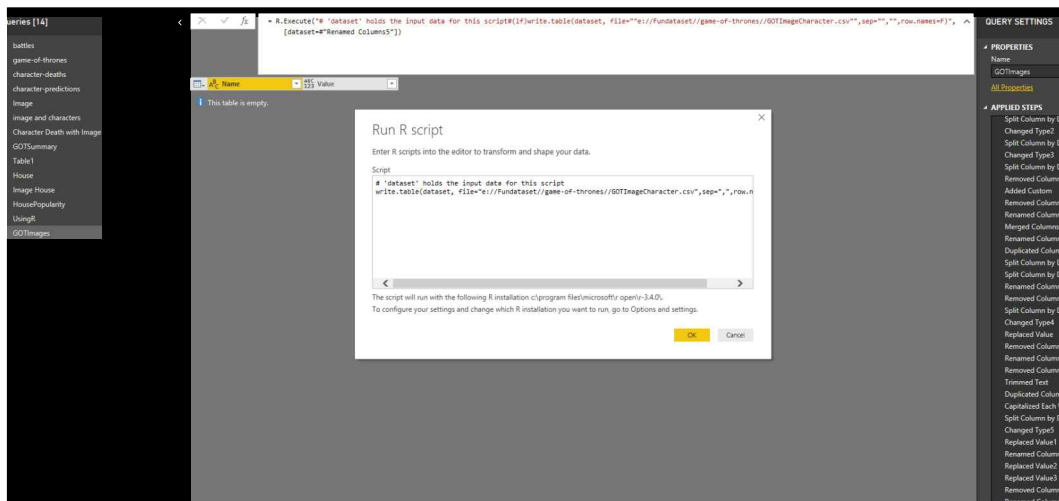


Then in the new R script, Editor writes the below code, the dataset variable holds all queries. In R scripts, we can use a function name **write.table** you need to pass the dataset and file location, then the separation of CSV file that is in my scenario comma,

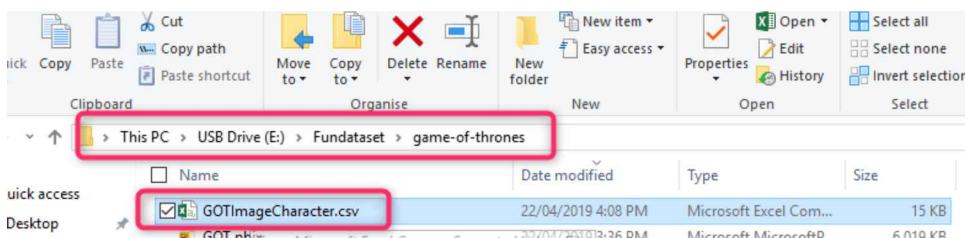
'dataset' holds the input data for this script

write.table(dataset, file="e://Fundataset//game-of-thrones//GOTImageCharacter.csv", sep=",", row.names=F)

Part 9: Table and List functions



Then push the OK bottom, then just check the location of the e://Fundataset//game-of-thrones//GOTImageCharacter.csv



You should see the file there; it was not a big dataset. I applied the same code for 60000 rows of data from the AdventureWorks Fact internet sales table; it also took 10 seconds.

Export to SQL Server database table

In this example, I will store the data in SQL Server, the same file, into SQL Server 2017. to do that, you need to access the specific package: **RODBC**

You need to install this package in R Studio.

In the below codes, write the codes, access the RODBC package, then you need to use a function **odbcDriverConnect**, you need to specify the driver, server, database name, and so forth.

Then you need to use **sqlSave**; you need to use **SQL connection**, **dataset**, and name of the table.

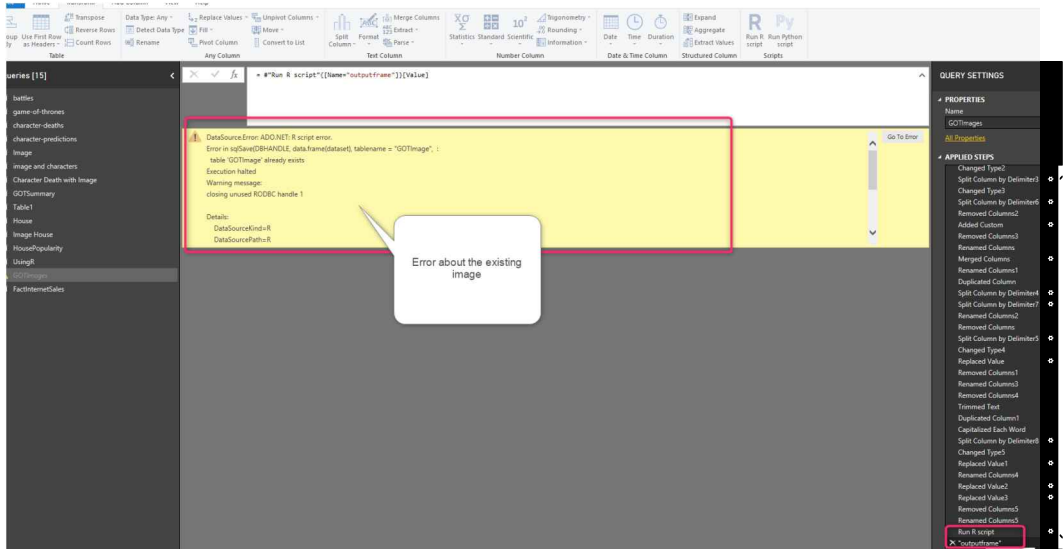
```
library(RODBC)
```

Chapter 47: Export data from Power Query to Local Machine or SQL Server using R scripts

```
DBHANDLE<-odbcDriverConnect('driver={SQL  
Server};server=.\ML2017;database=Ai;trusted_connection=true')
```

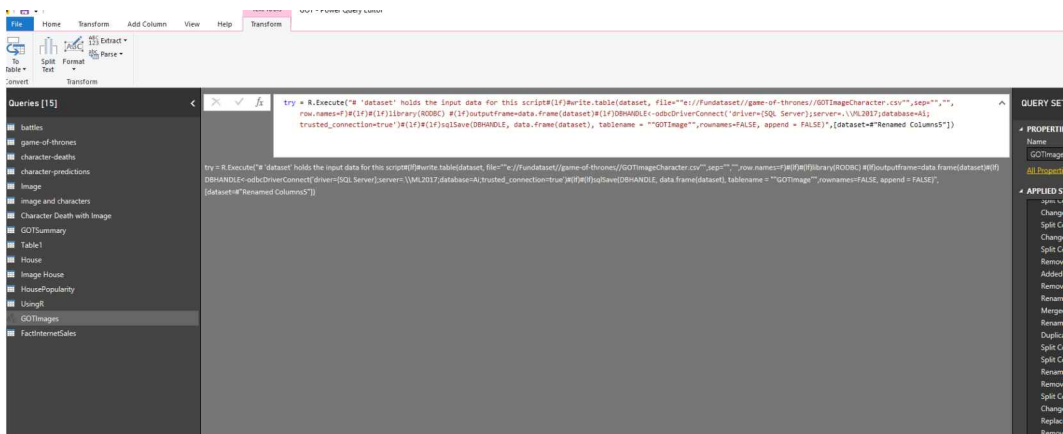
```
sqlSave(DBHANDLE, data.frame(dataset), tablename = "GOT",rownames=FALSE, append = FALSE)
```

When you run it, you may receive the below error!



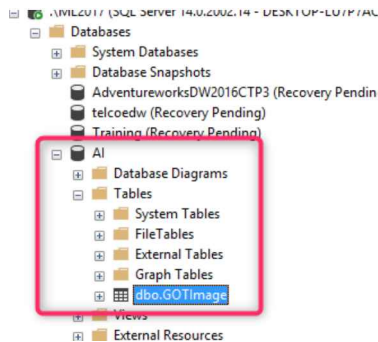
R will run twice, storing the file twice in the computer; you will get an error that the file already exists.

To solve that, we need to put a **try** keyword in front of the Power Query, as you can see below.



Check the SQL Server table.

Part 9: Table and List functions



To check if there is a possibility to check the dataset if exist not write the code there

```
TBExist="GOTImage" %in% sqlTables(DBHANDLE)$TABLE_NAME
if (!TBExist)
```

This will help, to check the existence of a table

```
# 'dataset' holds the input data for this script
#write.table(dataset, file="e://Fundataset//game-of-
thrones//GOTImageCharacter.csv",sep=";",row.names=F)
```

```
library(RODBC)
outputframe=data.frame(dataset)
DBHANDLE<-odbcDriverConnect('driver={SQL
Server};server=.\ML2017;database=AI;trusted_connection=true')
```

```
TBExist="GOTImage" %in% sqlTables(DBHANDLE)$TABLE_NAME
if (!TBExist)
sqlSave(DBHANDLE, data.frame(dataset), tablename = "GOTImage",rownames=FALSE, append =
FALSE)
```

By putting this code, you did not get any error regarding the existing database.

Chapter 48: Generate Random List of Numbers in Power BI Dataset Using Power Query

I've been asked by one of my friends who were trying to create a random dataset for his presentation. There is a great and quick way to create a random list of numbers in Power BI using Power Query. In this short chapter, I'll explain how to create this random list. You can use the generated output for building a sample dataset to work on with Power BI Desktop.

List.Random: Power Query Function

Some functions help with creating a random value. **List.Random** is one of those functions, which is very helpful. You just specify the number of random numbers that you want to generate:

List.Random(<how many times>)

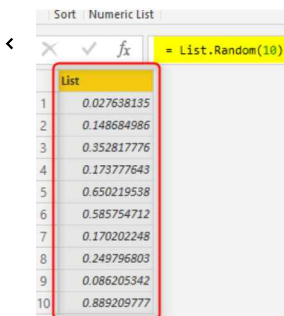
For example, if I want to create a list of ten random values, it would be `List.Random(10)`. Using this function to create a sample list is always easier to start from a blank query.

Get Data from Blank Query

Start with creating a blank query. If you don't see the formula bar in Power Query Editor, this is where you can enable it: In the View tab, check the Formula Bar:

In the Formula Bar, use the expression below;

`= List.Random(10)`



Part 9: Table and List functions

You can even use this for generating a list of 10 million numbers! However, be careful when you load all that data into Power BI! It might consume all the memory.

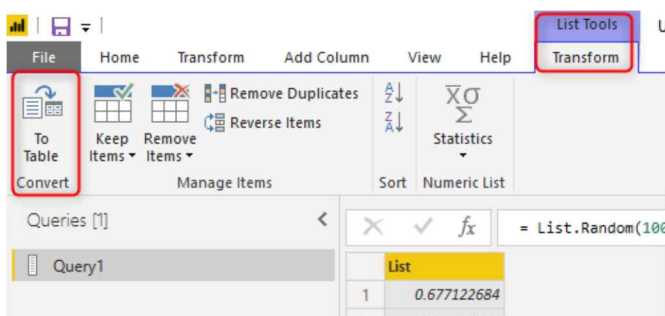
Sort | Numeric List

< f_x = List.Random(10000000)

| | List |
|----|-------------|
| 1 | 0.930521354 |
| 2 | 0.667182006 |
| 3 | 0.304979023 |
| 4 | 0.236976473 |
| 5 | 0.811715731 |
| 6 | 0.54033663 |
| 7 | 0.932071234 |
| 8 | 0.401194955 |
| 9 | 0.089684948 |
| 10 | 0.239069829 |
| 11 | 0.28844063 |
| 12 | 0.892515608 |
| 13 | 0.866428785 |
| 14 | 0.73368804 |
| 15 | 0.97890103 |
| 16 | 0.960265266 |
| 17 | 0.668177339 |
| 18 | 0.644807638 |
| 19 | 0.021970306 |
| 20 | 0.532634985 |
| 21 | 0.151134163 |
| 22 | 0.754016278 |
| 23 | 0.543531896 |
| 24 | 0.526632223 |
| 25 | 0.45393997 |
| 26 | 0.550825923 |
| 27 | 0.850245692 |
| 28 | 0.125483504 |
| 29 | 0.792491333 |
| 30 | 0.510012345 |
| 31 | 0.300955731 |
| 32 | 0.503599767 |
| 33 | 0.865171833 |
| 34 | 0.410393735 |
| 35 | 0.960329419 |
| 36 | 0.314766533 |
| 37 | 0.021452303 |

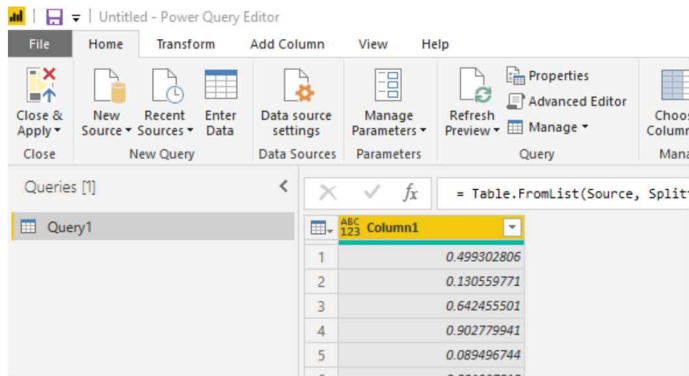
Convert to Table

You can convert this list now to a table:



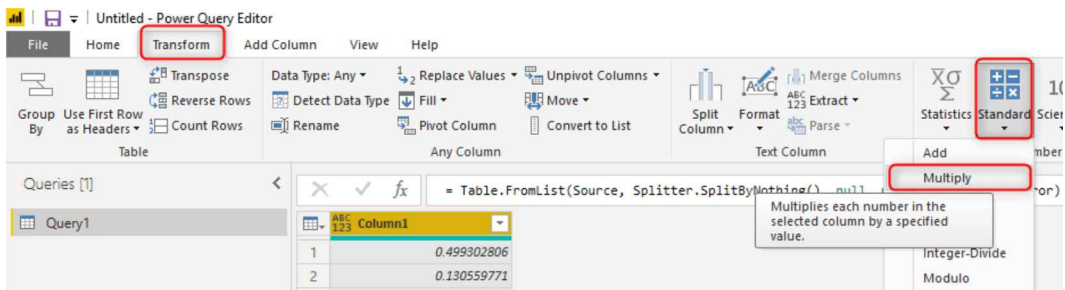
Chapter 48: Generate Random List of Numbers in Power BI Dataset Using Power Query

When you convert a list to a table, you can split it based on a delimiter, which is not what we want here, so just click OK on the “To Table” dialog box. the table output now should have one column:

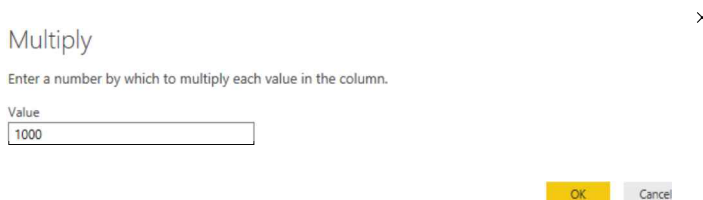


Changing the Range

As you see, the random number generated is between zero and one. If you want this to be on a higher value, you can always multiply it with something.

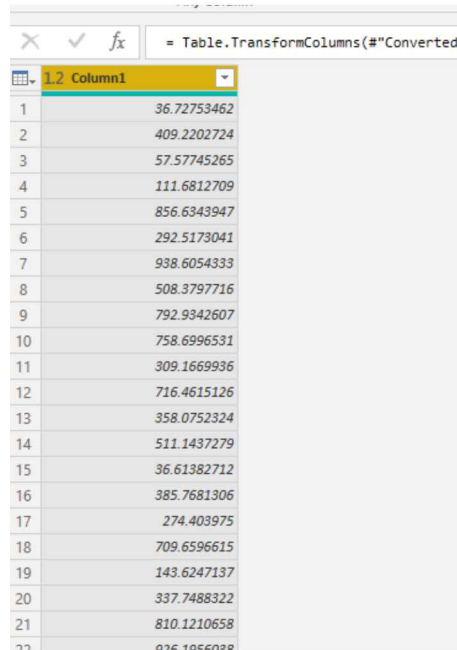


Let’s say, for example, I want this to be a random number up to 1000. I can multiply it by 1000 then.



Now, we have a table with values up to 1000;

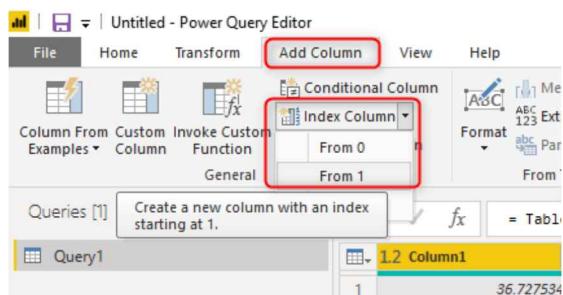
Part 9: Table and List functions



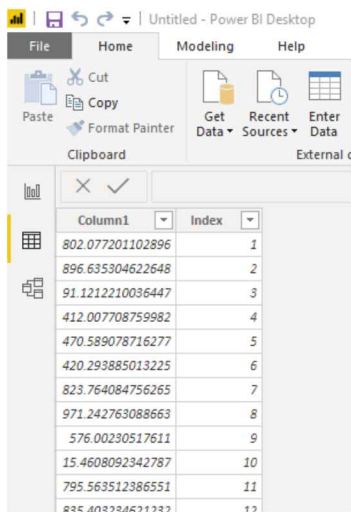
| | Column1 |
|----|-------------|
| 1 | 36.72753462 |
| 2 | 409.2202724 |
| 3 | 57.57745265 |
| 4 | 111.6812709 |
| 5 | 856.6343947 |
| 6 | 292.5173041 |
| 7 | 938.6054333 |
| 8 | 508.3797716 |
| 9 | 792.9342607 |
| 10 | 758.6996531 |
| 11 | 309.1669936 |
| 12 | 716.4615126 |
| 13 | 358.0752324 |
| 14 | 511.1437279 |
| 15 | 36.61382712 |
| 16 | 385.7681306 |
| 17 | 274.403975 |
| 18 | 709.6596615 |
| 19 | 143.6247137 |
| 20 | 337.7488322 |
| 21 | 810.1210658 |

Adding Row Number

You might also want to add a row number to the table; here is how you can do it: Under Add Column -> Index Column



And here is the dataset loaded into Power BI:



| Column1 | Index |
|------------------|-------|
| 802.077201102896 | 1 |
| 896.635304622648 | 2 |
| 91.1212210036447 | 3 |
| 412.007708759982 | 4 |
| 470.589078716277 | 5 |
| 420.293885013225 | 6 |
| 823.764084756265 | 7 |
| 971.242763088663 | 8 |
| 576.00230517611 | 9 |
| 15.4608092342787 | 10 |
| 795.563512386551 | 11 |
| 835.403294621232 | 12 |

Changing the Random List on Every Function Call

By default, when you use `List.Random` with one parameter, it will recreate the list of random numbers after each function call. It means for every step; you will see the random list changing!

If you don't like this to happen, and you want the list to be generated randomly only once and doesn't change with every function call, then you can specify the seed parameter:

`List.Random(<how many times to generate>,<seed>)`

< f_x = List.Random(100,1)

| | List |
|----|-------------|
| 1 | 0.248668584 |
| 2 | 0.110743977 |
| 3 | 0.46701068 |
| 4 | 0.771604122 |
| 5 | 0.657518894 |
| 6 | 0.432782601 |
| 7 | 0.354083764 |
| 8 | 0.943862276 |
| 9 | 0.101266454 |
| 10 | 0.642455555 |

This won't change with every call, then.

Other Random Functions in Power Query

There are two other random functions in Power

Query: **Number.Random** and **Number.RandomBetween**, they both give you a random number. However, you have to be careful when you use them on a list because

most of the time, query folding might happen, and then you get the same number for every row in the list after loading it into Power BI. I might write about tricks to use those later on.

Summary

This was a very quick chapter about how to use **List.Random** to generate a random dataset of numbers. Do you want to generate a random dataset and can't get it working with Power Query? let me know down below in the comments.

Book wrap up

Congratulations on finishing the book. We hope you enjoyed reading and this book shed some lights and guidelines toward data preparation and transformation using Power Query. Remember that Power Query is not just for building a BI system using Power BI. You can use whatever you learned in your day-to-day work at Excel and some other tools and services too.

Power Query and data preparation is just the start of the journey toward an analytics solution using Power BI. If you wish to master data analytics with Power BI, read our other books, such as Basics of Power BI Modeling, Power BI DAX Simplified, Pro Power BI Architecture, and Row-Level Security in Power BI. The link to these books is on the next page.

To leverage the learnings from this book, I encourage you to start applying the learning right away in your Power BI implementations. If you feel concerned or have a question about a particular scenario, feel free to reach out to us directly using the [RADACAD website\[https://radacad.com/\]](https://radacad.com/), We'd be more than happy to look into your question.

Wishing you the best

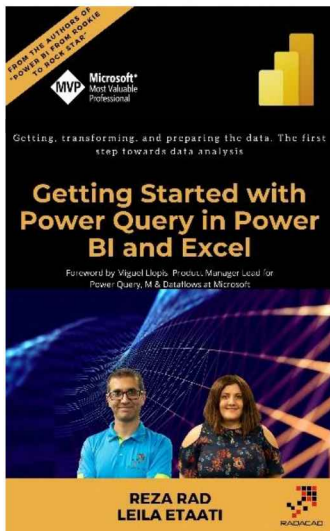
Reza Rad & Dr. Leila Etaati

August 2021

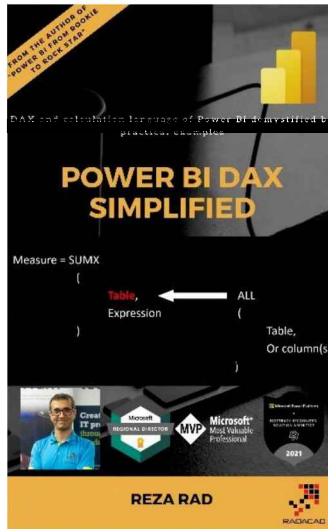
Other books from Reza Rad and Leila Etaati

Power BI from Rookie to Rock Star

This is a series of four books, over 1200 pages, available for free to download from [here](https://radacad.com/online-book-power-bi-from-rookie-to-rockstar)[\[https://radacad.com/online-book-power-bi-from-rookie-to-rockstar\]](https://radacad.com/online-book-power-bi-from-rookie-to-rockstar)

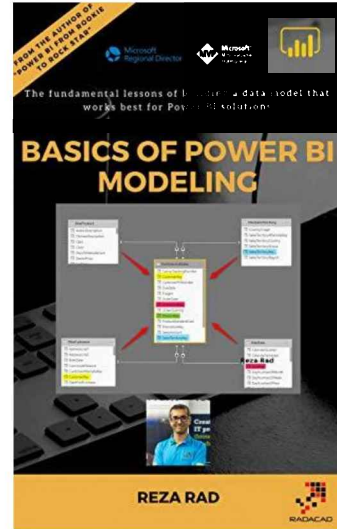


Getting Started with Query in Power BI and Excel



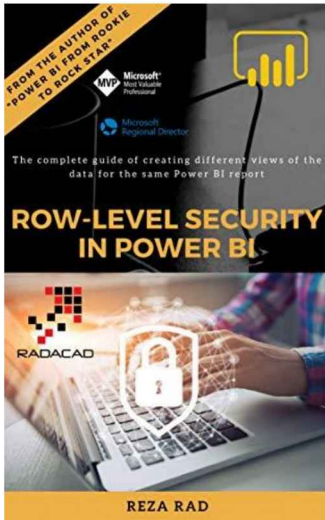
Power BI DAX Simplified

[Here](https://www.amazon.com/dp/B099SBN1XP)[\[https://www.amazon.com/dp/B099SBN1XP\]](https://www.amazon.com/dp/B099SBN1XP)



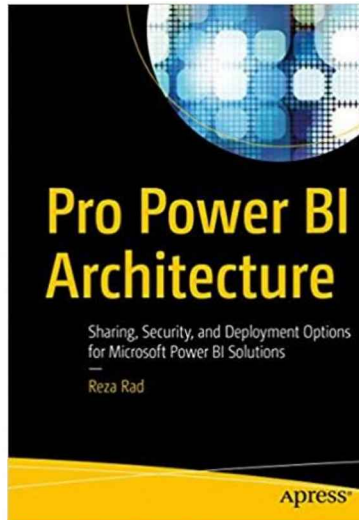
Basics of Power BI Modeling

[Here](https://www.amazon.com/gp/product/B08HWNZ7GC)[\[https://www.amazon.com/gp/product/B08HWNZ7GC\]](https://www.amazon.com/gp/product/B08HWNZ7GC)



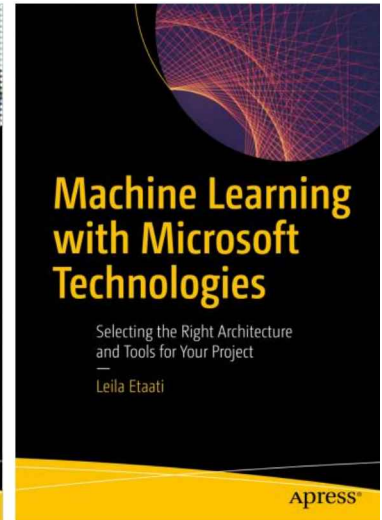
Row-Level Security in Power BI

[Here\[https://www.amazon.com/dp/B082SFR2J4\]](https://www.amazon.com/dp/B082SFR2J4)



Pro Power BI Architecture

[Here\[https://www.apress.com/gp/book/9781484240144\]](https://www.apress.com/gp/book/9781484240144)



Machine Learning with Microsoft Technologies

[Here\[https://www.apress.com/gp/book/9781484236574\]](https://www.apress.com/gp/book/9781484236574)

The complete list of our books, including the free books, are available [here\[https://radacad.com/ourbooks\]](https://radacad.com/ourbooks).