CocoTech SOLUTIONS

PowerBI Training

Table of Contents

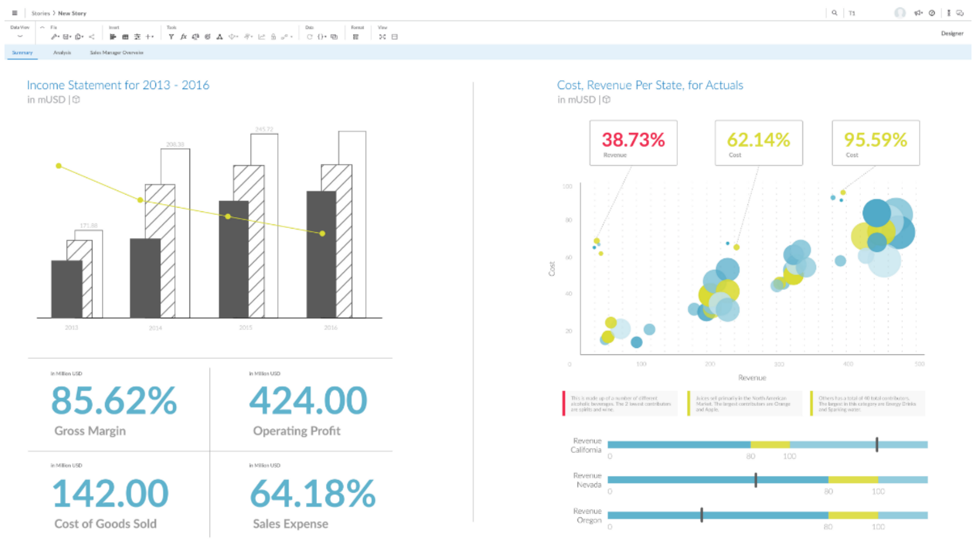
Chapter 1: Overview

What is Business Intelligence?

Is a software that ingests business data and presents it in user-friendly views such as reports, dashboards, charts and graphs. BI offers a way for people to examine data to understand trends and derive insights.1

Business intelligence refers to the processes and tools used to analyze business data, turn it into actionable insights, and help everyone in an organization make better-informed decisions. Also known as a decision support system (DSS), a BI system analyzes current and historical data and presents findings in easy-to-digest reports, dashboards, graphs, charts, and maps that can be shared across the company.

BI is sometimes called "descriptive analytics" because it describes how a business is performing today and how it performed in the past. It answers questions like "What happened?" and "What needs to change?" – but it does not get into why something happened or what might happen next.2

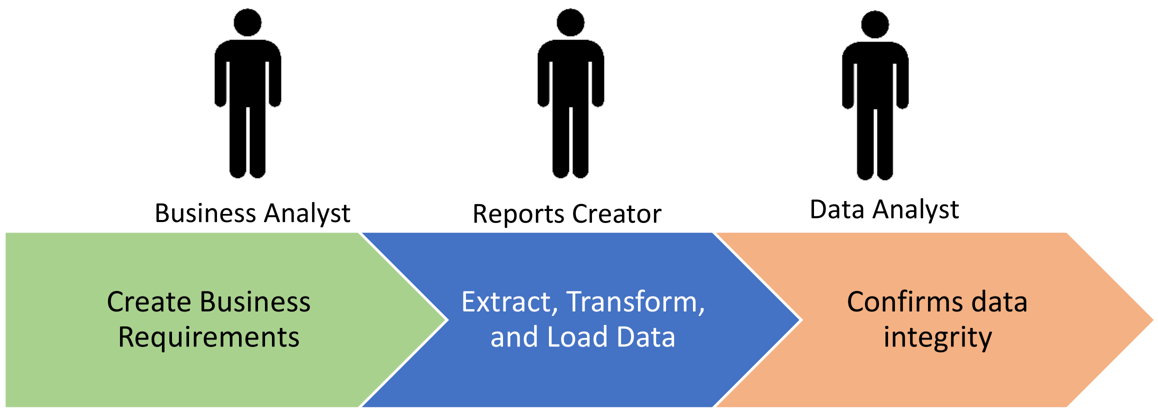


How Business Intelligence was done before

Business Intelligence was an expensive and extensive activity before in a business. This will require multiple resource in a company.

1 https://www.ibm.com/topics/business-intelligence

2 https://www.sap.com/products/technology-platform/cloud-analytics/what-is-business-intelligence.html

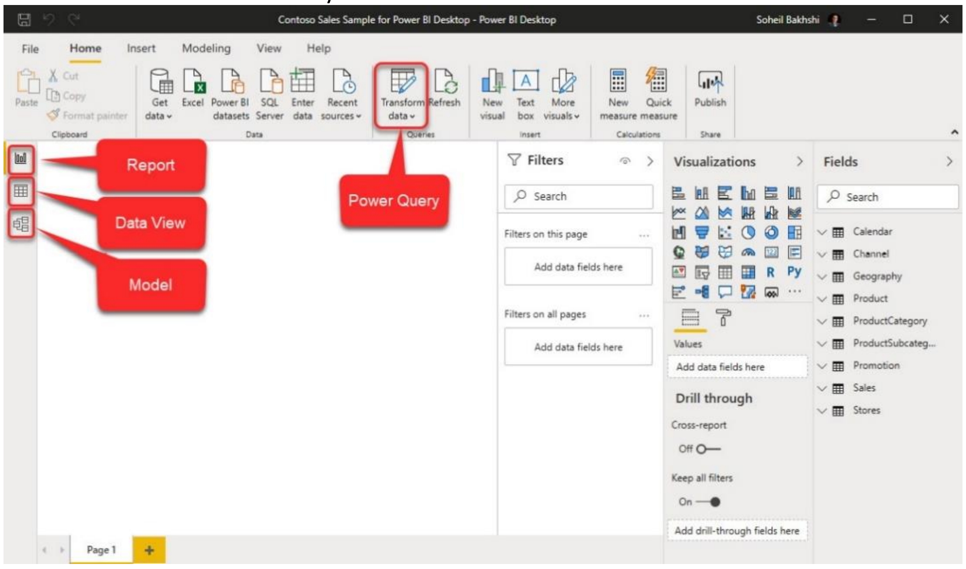


PowerBI is a self-service BI tool that allows you to generate visualizations, and at the same time process data from different sources. This allows users to perform ETL (Extract – Transform – Load) process using a single application.

Layers of PowerBI

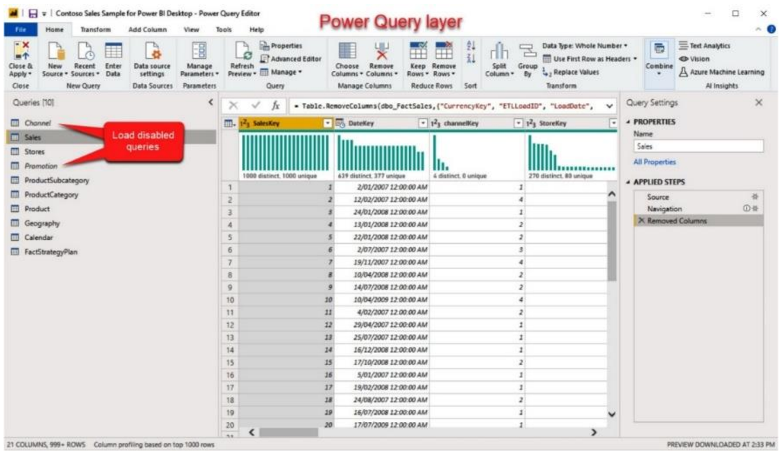
PowerBI have different layers that distinguish different parts and functionality of Business Intelligence. These layers are:

- Power Query Layer
- Data Model Layer
- Data Visualization Layers



Power Query Layer

In this layer you make your data preparation. You get data from various data sources and transform it to make it available for other layers.

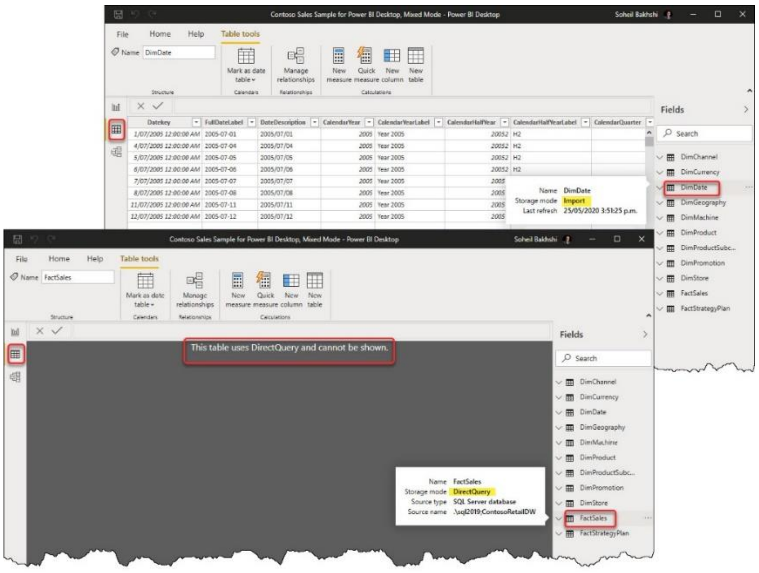


Data Model Layer

This layer has two views, the Data View and Model View.

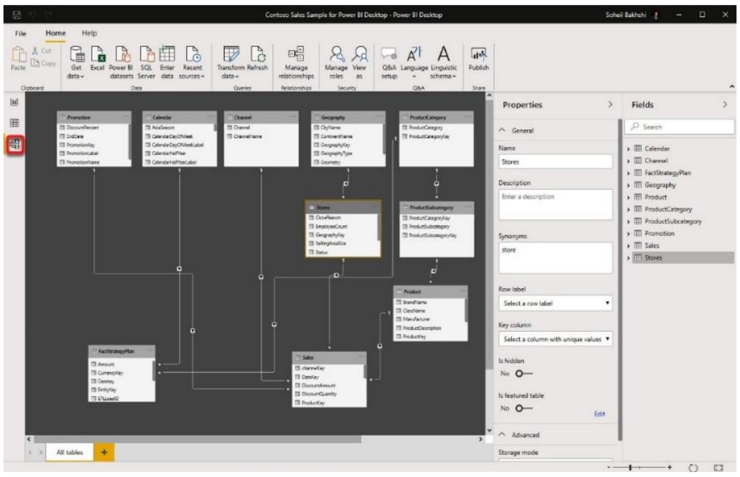
Data View

The Data View allows you to see the data in a tabular format. Table format gives users more familiarity on how their data looks like in their query.


Name DimDate
Storage mode Import
Last refresh 25/05/2020 3:51:25 p.m.
This table uses DirectQuery and cannot be shown.
Name FactSales
Storage mode DirectQuery
Source type SQL Server database
Source name .\sql2019;ContosoRetailDW
Fields

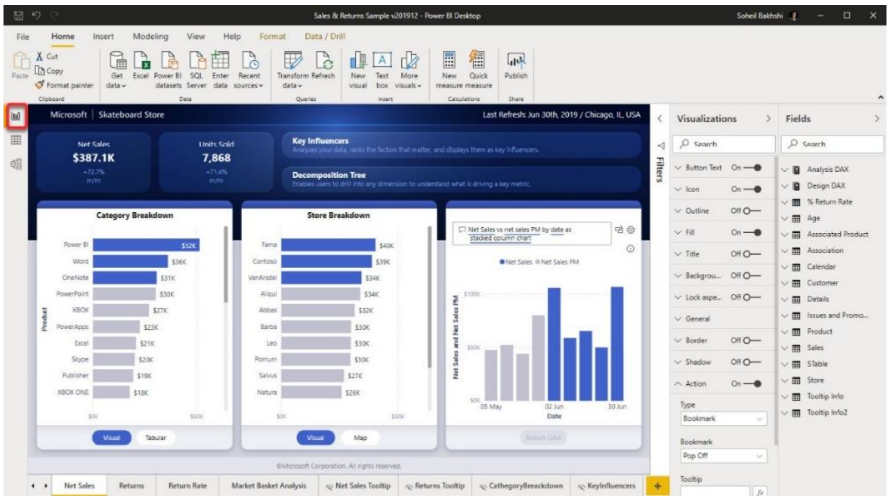
Model View

The Model View gives users a glimpse on how queries or tables are related to each other. This also allow users to create relationship between tables.



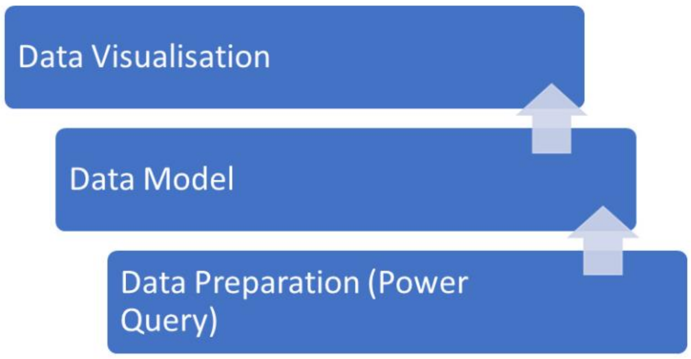
Data Visualization Layer

This layer is the Report View, which is the default view of PowerBI. This layer gives you access to different visualization tools. These tools allow you to show your data in a more appealing visual manner. This allows you to understand your data a lot easier.



How data flows in PowerBI

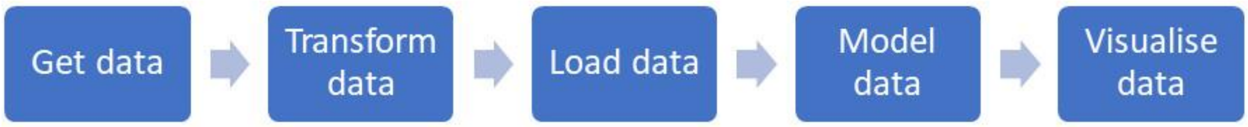
Each layer of PowerBI is responsible in the flow of data. It is important to have a good understanding of each layer for easier troubleshooting. For example, if you have an error in your line graph that uses a measure that is dependent on a calculated column you know that a measure or calculated column uses DAX Expressions, and it is not in the Power Query Layer.



Chapter 2: Data Model Layer

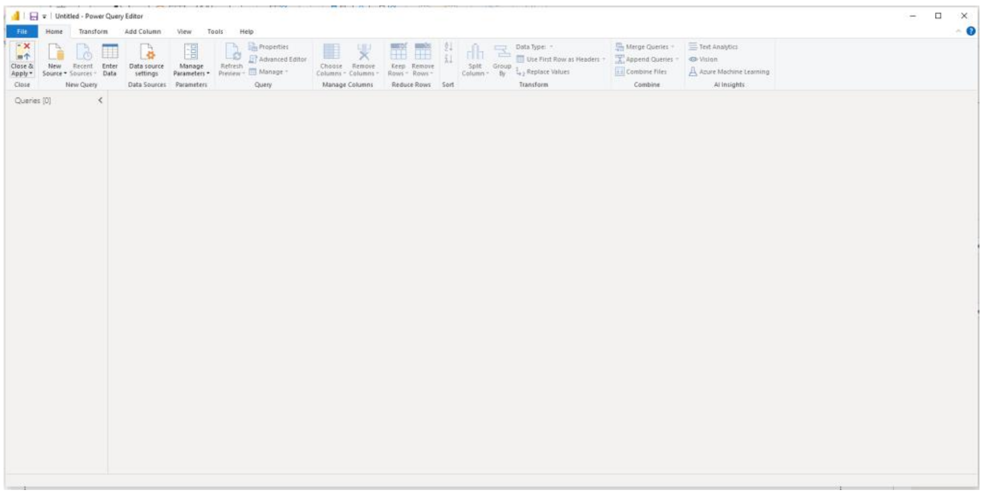
Data Modeling

Data modeling is undoubtedly one of the most important parts of Power BI development. The purpose of data modeling in Power BI is different from data models in transactional systems. In a transactional system, the goal is to have a model that is optimized for recording transactional data. Nevertheless, a well-designed data model in Power BI must be optimized for querying the data and reducing the dataset size by aggregating that data.



When modeling data in Power BI, you need to build a data model based on the business logic. Having said that, you may need to join different tables and aggregate your data to a certain level that answers all business-driven questions. It can get worse if you have data from various data sources of different grains representing the same logic.

Data modeling in PowerBI is done in the Query Editor. To access the query editor, you can select the Transform.



What is an efficient data model?

Efficiency in your data model is important to obtain sustainability. Your data model should be ***Easy to Understand, and Easy to maintain.***

A data model should be able to do the following:

- **Perform well (quickly)** – It should be able to deliver the needed information to the user in a quick manner.
- **Be business-driven** – It should answer business questions, and come up with insights for business decision.
- **Decrease the level of complexity (be easy to understand)** – Lets users understand the data easily.
- **Be maintainable with low costs** – Build the dashboard and maintain them in a cost-efficient manner

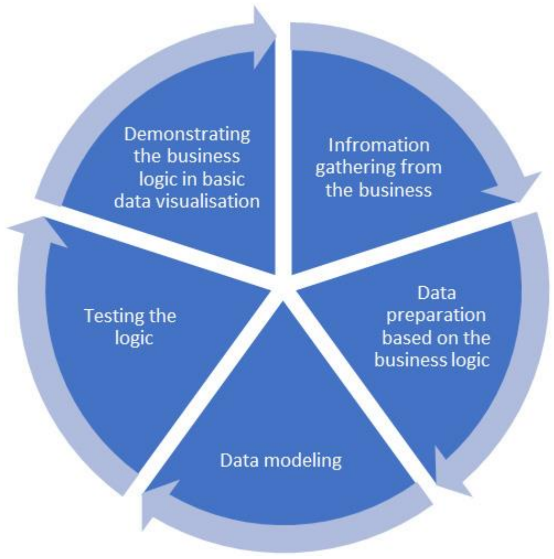
"You need to talk to the business and ask questions before starting the job"

- We need to ask questions of the business to avoid any confusions and potential reworks in the future.
- We need to understand the technology limitations and come up with solutions.
- We have to have a good understanding of data modeling, so we can look for common data patterns to prevent overlaps.

Data Modeling using Agile approach

Agile and iterative approach is the best methodology in developing your data model. It keeps all parties involved which allows faster development and clearer goals.

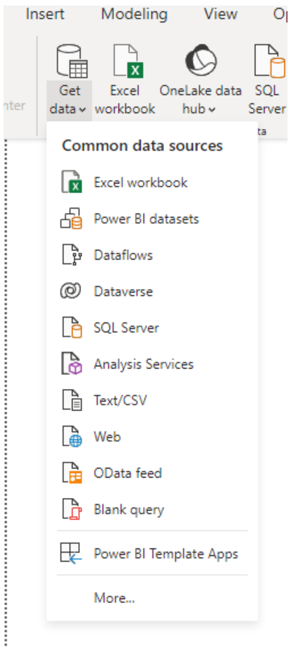
1. Information gathering from the business
2. Data preparation based on the business logic
3. Data modeling
4. Testing the logic
5. Demonstrating the business logic in a basic data visualization

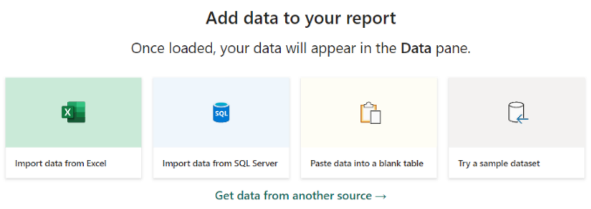

Infromation gathering from the business
Data preparation based on the business logic
Data modeling
Testing the logic
Demonstrating the business logic in basic data visualisation

Preparing Your Data

Get and Transform Data

The first step in creating your dashboard begins in Power BI Desktop where you connect to and transform your data in preparation for data modeling. Data can come from different sources. These data sources can be a folder path, shared drive, files, databases, API (Application Program Interface) and other data repositories.




Add data to your report
Once loaded, your data will appear in the Data pane.
Import data from Excel
Import data from SQL Server
Paste data into a blank table
Try a sample dataset
Get data from another source →

Connection to a Folder

PowerBI allows you to connect to a local folder, or a shared folder, given that you have the windows explorer link. This pulls all the supported files inside the folder. During transformation, you can filter the file types you will need in your query.

Connecting to files

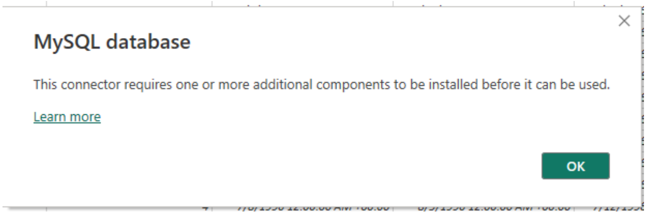
PowerBI supports connection to different types of files, like CSV, XLSX, HTML. When connecting to an MS Excel file (xlsx). It will show you all available sheets and tables inside the workbook.

HTML connection only supports <table> tags. Data inside a <div> tag will not be pulled by PowerBI.

Connecting to MySQL Database

To connect to a database, you will need a connector. For a MySQL database you will need the MySQL connector/Net (https://downloads.mysql.com/archives/c-net/). Currently, it only supports the MySQL connection version 8.0.26.

You will get an error when you try to connect to a MySQL server host without a connector.



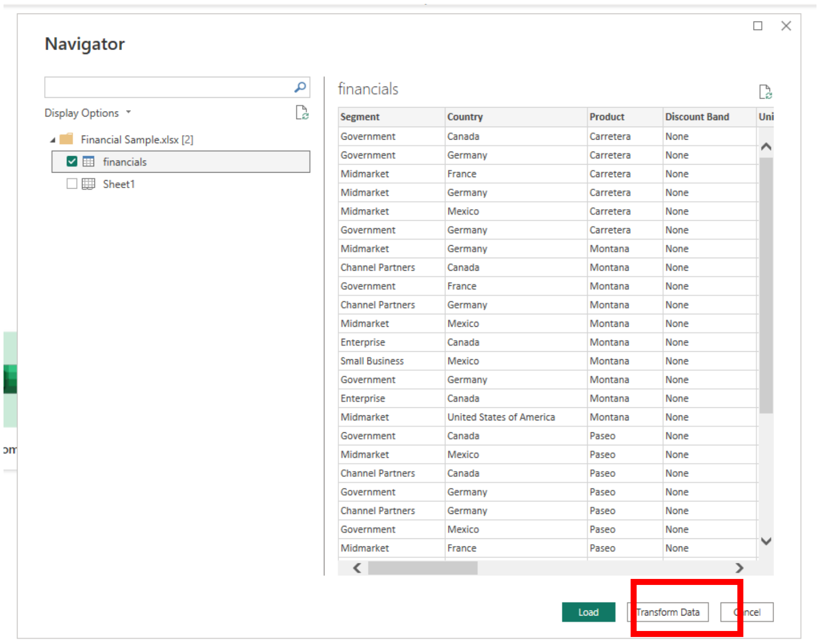
MySQL database

This connector requires one or more additional components to be installed before it can be used.

Transforming Data

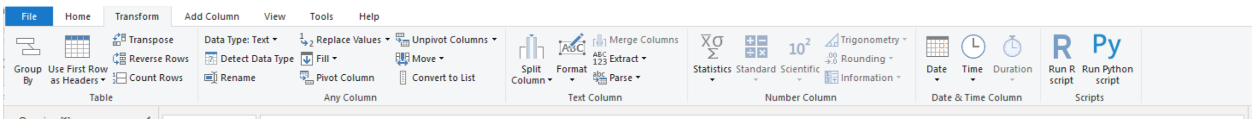
After selecting and connecting to a data source you have an option to transform your data. Data transformation is the process of converting raw data into a structured and usable format, crucial for effective Business Intelligence (BI) implementation. It involves cleansing, aggregating, and integrating data from multiple sources to create a unified view. Data transformation is vital as it enhances data quality, consistency, and accuracy, enabling better-informed decision-making.

By providing actionable insights and uncovering patterns, PowerBI empowers organizations to identify opportunities, optimize processes, and gain a competitive edge. In essence, data transformation is the backbone of BI, ensuring businesses extract meaningful value from their data assets, leading to improved efficiency and strategic growth.



Segment	Country	Product	Discount Band	Uni
Government	Canada	Carretera	None	
Government	Germany	Carretera	None	
Midmarket	France	Carretera	None	
Midmarket	Germany	Carretera	None	
Midmarket	Mexico	Carretera	None	
Government	Germany	Carretera	None	
Midmarket	Germany	Montana	None	
Channel Partners	Canada	Montana	None	
Government	France	Montana	None	
Channel Partners	Germany	Montana	None	
Midmarket	Mexico	Montana	None	
Enterprise	Canada	Montana	None	
Small Business	Mexico	Montana	None	
Government	Germany	Montana	None	
Enterprise	Canada	Montana	None	
Midmarket	United States of America	Montana	None	
Government	Canada	Paseo	None	
Midmarket	Mexico	Paseo	None	
Channel Partners	Canada	Paseo	None	
Government	Germany	Paseo	None	
Channel Partners	Germany	Paseo	None	
Government	Mexico	Paseo	None	
Midmarket	France	Paseo	None	

As a self-service BI tool, PowerBI have built-in features that will allow you to transform data easily. There is not need for a programming background to use these features.

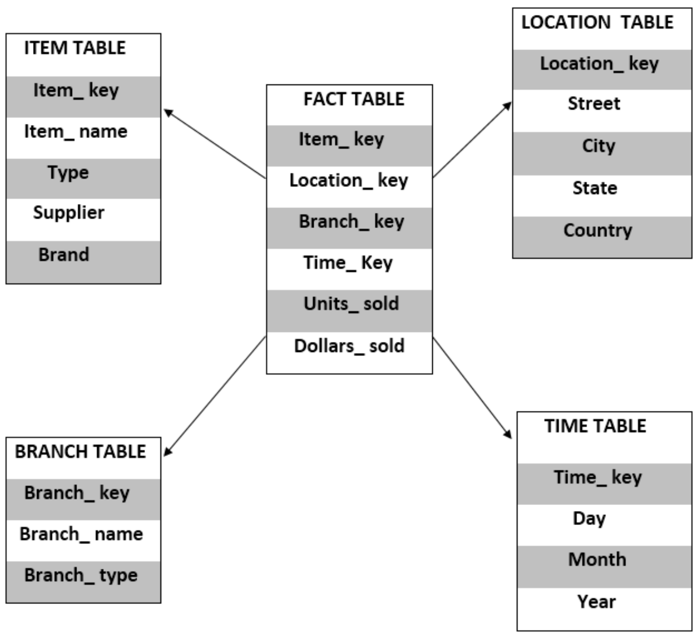


Modeling Schema

Transactional vs Star schema Model

Transactional and star schema models are two distinct data modeling approaches in Business Intelligence (BI). The **transactional model** follows a normalized structure, storing data in separate tables to reduce redundancy and maintain data integrity. While it's efficient for data storage and updates, it may require complex joins for analysis.

On the other hand, the **star schema model** denormalizes data into a central **fact** table connected to **dimension** tables, simplifying queries and enhancing performance for analytical tasks. While it consumes more storage space, the star schema accelerates BI reporting and facilitates easier data exploration, making it popular for decision-making purposes. The choice between these models depends on the specific BI requirements and trade-offs between storage efficiency and analytical speed.

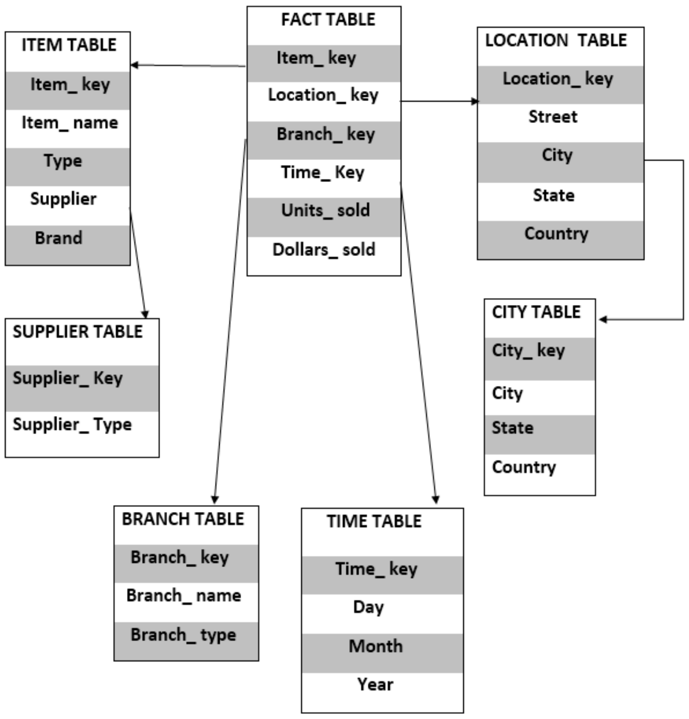


Fact tables contain numerical values, often called measures or metrics, and foreign keys that link to the associated dimension tables. These dimension tables provide descriptive information or attributes related to the measures in the fact table.

Snowflakes Model

It is an extension of the star schema model, where dimension tables are further normalized into multiple related tables. In the snowflake model, dimension tables are broken down into sub-dimensions, reducing data redundancy and improving data integrity.

The benefit of the snowflake model lies in its ability to save storage space by reducing redundancy and maintaining data consistency. However, it can lead to more complex queries due to the need for additional joins across the normalized tables. The choice between the star schema and the snowflake model depends on the specific requirements of the data warehouse and the trade-offs between query performance and storage efficiency.



In this schema, the single dimension table of the item has been normalized and split, resulting in the creation of a new supplier table that includes information on the type of supplier. Likewise, the dimension table of location has been normalized, and its data has been split into a new city table that contains details of each specific city

Star Schema vs Snowflake Schema

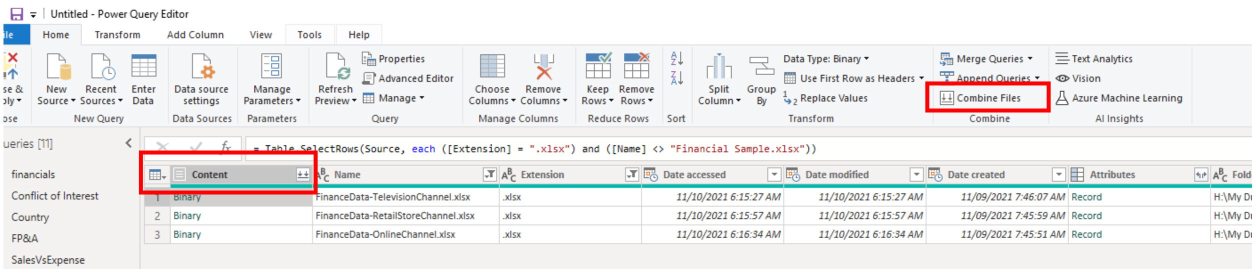
Star Schema	Snowflake Schema
Maintenance/Change	
It has more redundant data, and hence it is more difficult to change or maintain	This schema is easier to change and maintain due to less redundancy
Understandability	
The complexity of the query is less and hence easy to understand	Queries applied are more complex and hence difficult to understand
Query Execution Time	

If has fewer foreign keys, and hence the query execution is faster and takes lesser time	Due to more foreign keys, the query execution time is more, or the query executes slowly.
Type of Data Warehouse	
Better for datamarts having single relationship, 1:1 or 1:M.	Better for complex relationships: M:M relationship
Number of Joins	
It has a greater number of joins	It has lesser number of joins
Dimension Table	
It has only one dimension table for each dimension	It has one or more dimension table for a single dimension.
Usability	
Preference to the star schema when the dimension table has a smaller size	Good to use when the size of the dimension table is bigger
Normalization and Denormalization	
Both the fact table and dimension tables are denormalized	A fact table is denormalized, while dimension table is normalized.
Data Model	
If follows a top-down approach	It follows a bottom-up approach.

3

Combining Files

PowerBI allows you to combine files with the same schema into a single logical table. Combining files is usually done for data source coming from a folder.



```
= Table.SelectRows(Source, each ([Extension] = ".xlsx") and ([Name] <> "Financial Sample.xlsx"))
```

	Content	Name	Extension	Date accessed	Date modified	Date created	Attributes	Fold
1	Binary	FinanceData-TelevisionChannel.xlsx	.xlsx	11/10/2021 6:15:27 AM	11/10/2021 6:15:27 AM	11/09/2021 7:46:07 AM	Record	H:\My Di
2	Binary	FinanceData-RetailStoreChannel.xlsx	.xlsx	11/10/2021 6:15:57 AM	11/10/2021 6:15:57 AM	11/09/2021 7:45:59 AM	Record	H:\My Di
3	Binary	FinanceData-OnlineChannel.xlsx	.xlsx	11/10/2021 6:16:34 AM	11/10/2021 6:16:34 AM	11/09/2021 7:45:51 AM	Record	H:\My Di

Things to remember when combining files

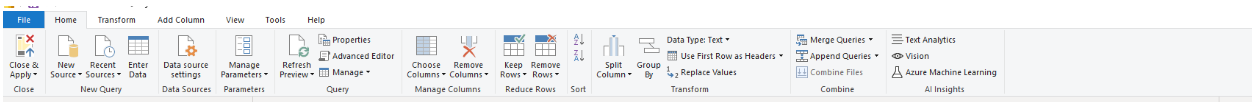
- Can be used on files in the same folder
- Same file type and structure

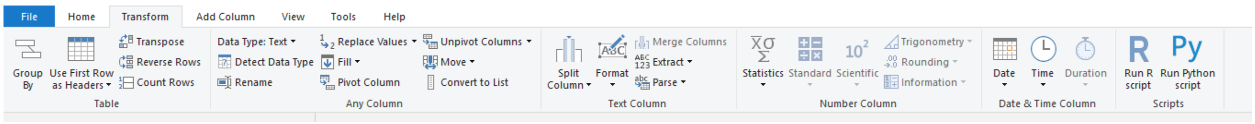
3 https://www.educba.com/star-schema-vs-snowflake-schema/

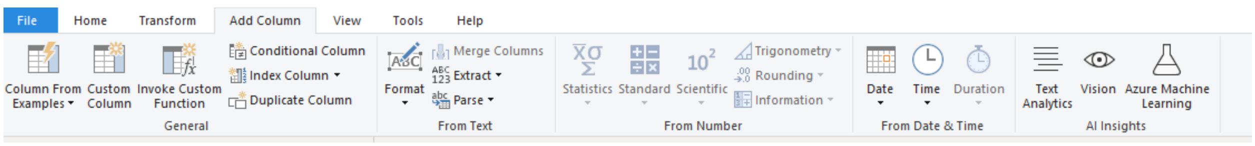
- Same column names (case sensitive)
- Same number of columns
- Any changes to the exemplar query are automatically generated in the linked function query.

Transforming Your Data

Transforming data in BI involves converting raw, disparate data into a unified, structured format for analysis. It includes cleansing, aggregating, and integrating data from various sources, enabling meaningful insights, informed decision-making, and a competitive edge for businesses.

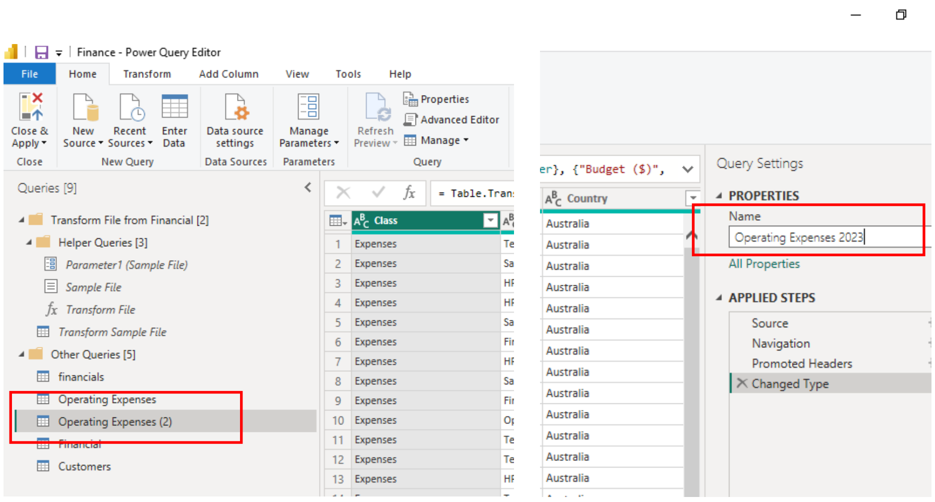






Renaming Queries

You can rename the query name by changing it in the Query Settings, Properties field.



Queries [9]
- Transform File from Financial [2]
  - Helper Queries [3]
    - Parameter1 (Sample File)
    - Sample File
    - Transform File
    - Transform Sample File
- Other Queries [5]
  - financials
  - Operating Expenses
  - Operating Expenses (2)
  - Financial
  - Customers

= Table.Tran

er}, {"Budget ($)",

	Class
1	Expenses
2	Expenses
3	Expenses
4	Expenses
5	Expenses
6	Expenses
7	Expenses
8	Expenses
9	Expenses
10	Expenses
11	Expenses
12	Expenses
13	Expenses

Country
Australia
Australia
Australia
Australia
Australia
Australia
Australia
Australia
Australia
Australia
Australia
Australia
Australia

Query Settings

PROPERTIES

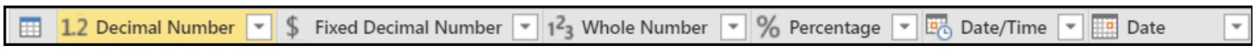
Name: Operating Expenses 2023

All Properties

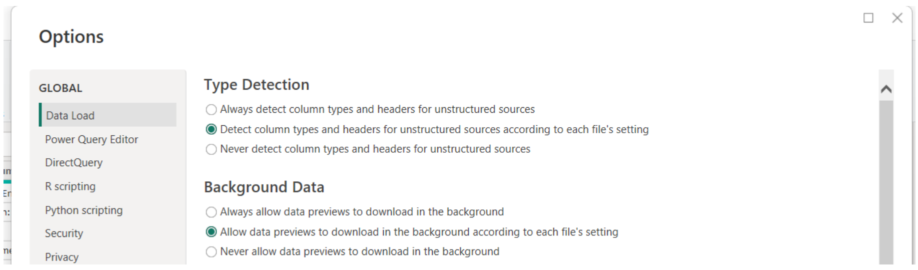
APPLIED STEPS
- Source
- Navigation
- Promoted Headers
- Changed Type

Data Types

PowerBI can automatically convert the possible data type of the column after you selected Transform data.



You can disable Automatic detection of data type from the **File Tab > Option and Settings > Options > Data Load.**

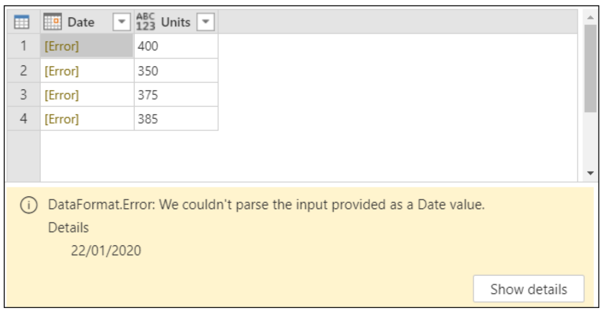

Options

GLOBAL
- Data Load
- Power Query Editor
- DirectQuery
- R scripting
- Python scripting
- Security
- Privacy

Type Detection
- ( ) Always detect column types and headers for unstructured sources
- (●) Detect column types and headers for unstructured sources according to each file's setting
- ( ) Never detect column types and headers for unstructured sources

Background Data
- ( ) Always allow data previews to download in the background
- (●) Allow data previews to download in the background according to each file's setting
- ( ) Never allow data previews to download in the background

Having the proper data type assigned to a column can improve performance of the query. Having the wrong value converted to a certain data type will give you an Error in your row (i.e. "07/01/2023" convert to Duration or Boolean).



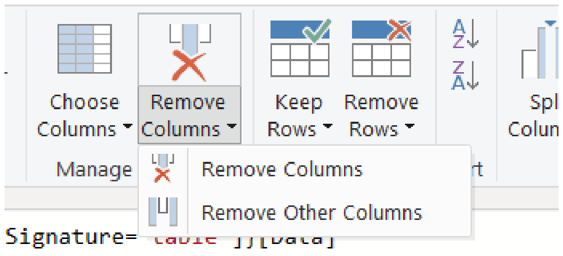
	Date	Units
1	[Error]	400
2	[Error]	350
3	[Error]	375
4	[Error]	385

DataFormat.Error: We couldn't parse the input provided as a Date value.
Details
22/01/2020

Show details

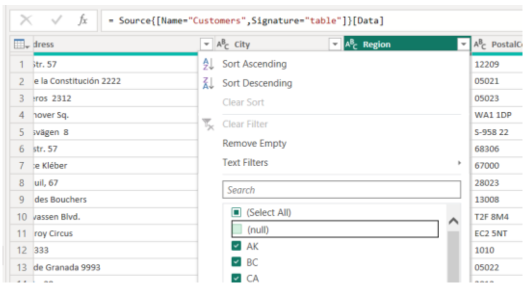
Removing columns and rows

You can remove columns in Power Query. There are 2 options of removing columns, you can either remove the selected column or remove the other columns aside from those selected.



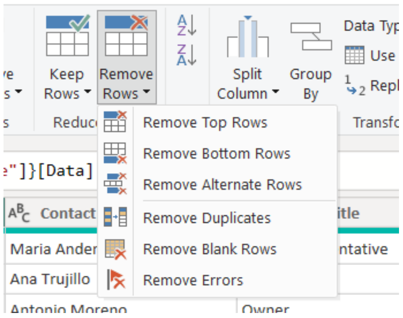
Removing Records

Removing the records basically is just hiding or deselecting the rows. These rows will not be included when the query is loaded.


```
= Source{[Name="Customers",Signature="table"]}[Data]
```

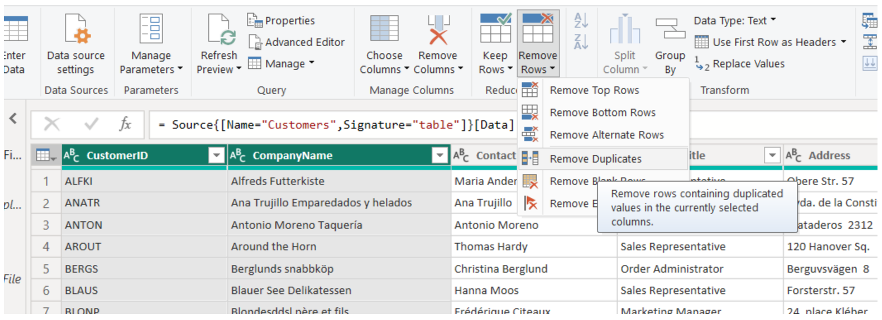
	dress	PostalC
1	Str. 57	12209
2	e la Constitución 2222	05021
3	eros 2312	05023
4	nover Sq.	WA1 1DP
5	svägen 8	S-958 22
6	str. 57	68306
7	e Kléber	67000
8	uil, 67	28023
9	des Bouchers	13008
10	vassen Blvd.	T2F 8M4
11	roy Circus	EC2 5NT
12	333	1010
13	de Granada 9993	05022

Aside from using the column filter, PowerBI have built-in options for removing rows. You can remove the blank rows, rows with errors and even top or bottom rows.



Removing Duplicates

You can also remove the duplicate records from your query. By selecting the columns you want to specify the duplicates, you can set one or more columns to check for duplicates.

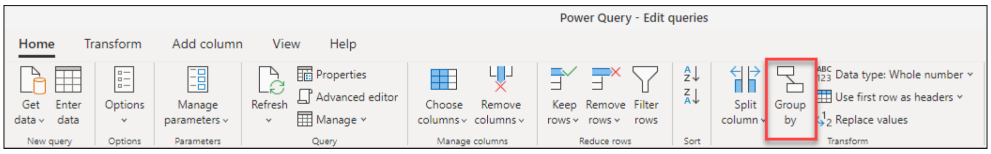

Remove Top Rows
Remove Bottom Rows
Remove Alternate Rows
Remove Duplicates
Remove rows containing duplicated values in the currently selected columns.

= Source{[Name="Customers",Signature="table"]}[Data]

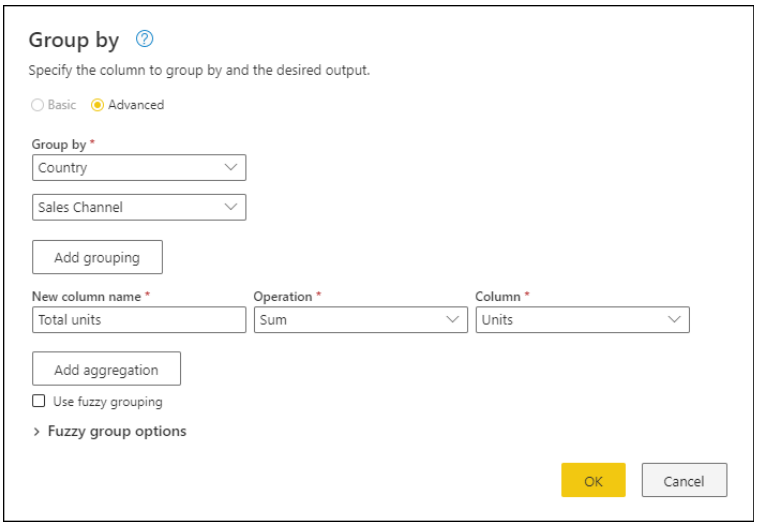
	CustomerID	CompanyName	Contact
1	ALFKI	Alfreds Futterkiste	Maria Ander
2	ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo
3	ANTON	Antonio Moreno Taquería	Antonio Moreno
4	AROUT	Around the Horn	Thomas Hardy
5	BERGS	Berglunds snabbköp	Christina Berglund
6	BLAUS	Blauer See Delikatessen	Hanna Moos

Group By

The Group By command allows you to aggregate or summarize the data based on the column your selected.



This allows you to select an aggregate function to group by one or more columns.


Group by

Specify the column to group by and the desired output.

Basic [✓ Advanced]

Group by *

Country

Sales Channel

Add grouping

New column name *	Operation *	Column *
Total units	Sum	Units

Add aggregation

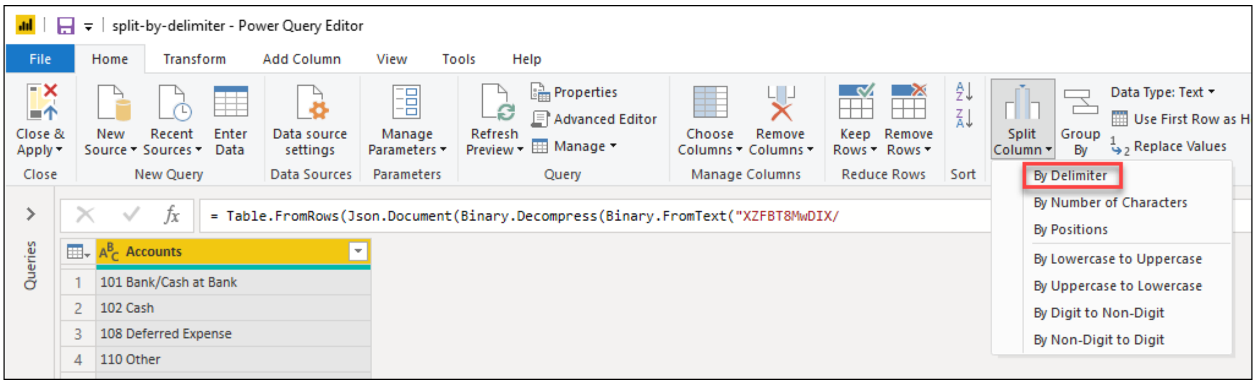
☐ Use fuzzy grouping

› Fuzzy group options

OK Cancel

Splitting Columns

PowerBI allows you to split columns into new columns. There are multiple ways to split a column. The most common is by Delimiter.

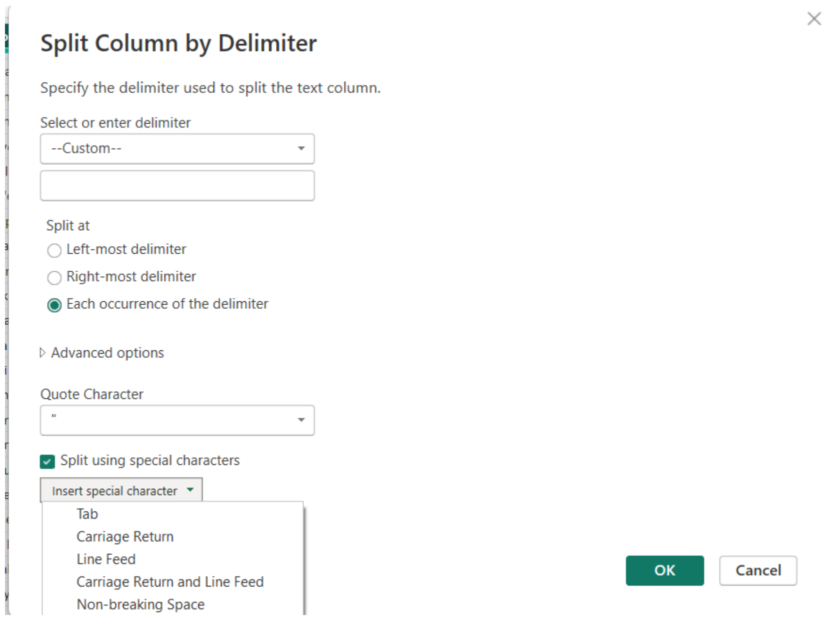

split-by-delimiter - Power Query Editor

```
= Table.FromRows(Json.Document(Binary.Decompress(Binary.FromText("XZFBT8MwDIX/
```

	Accounts
1	101 Bank/Cash at Bank
2	102 Cash
3	108 Deferred Expense
4	110 Other

When selecting By Delimiter, there are options for users to split it by **Left-most delimiter**, **Right-most delimiter**, or by **Each occurrence of the delimiter**.

For special characters as delimiter, you can select Custom as delimiter and check Split using special characters.


Split Column by Delimiter

Specify the delimiter used to split the text column.

Select or enter delimiter

--Custom--

Split at

- ( ) Left-most delimiter
- ( ) Right-most delimiter
- (●) Each occurrence of the delimiter

▷ Advanced options

Quote Character

"

☑ Split using special characters

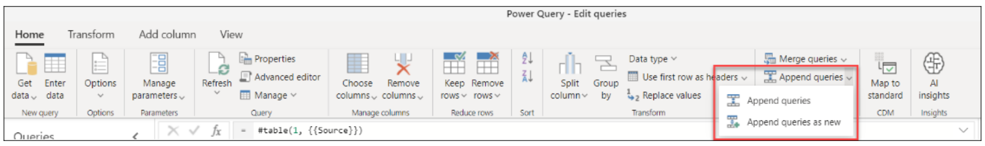
Insert special character ▾

- Tab
- Carriage Return
- Line Feed
- Carriage Return and Line Feed
- Non-breaking Space

OK | Cancel

Combining Queries

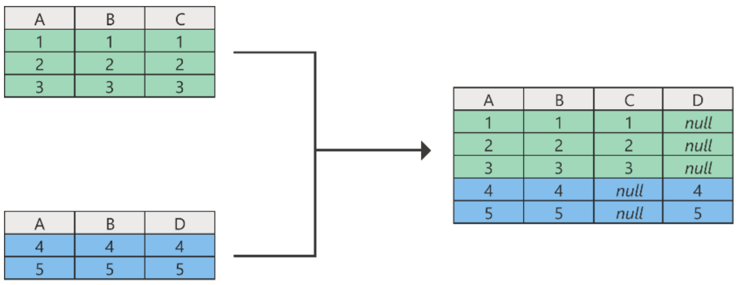
There are two (2) ways of combining queries: Merge Queries and Append Queries. Unlike combining files, combining queries can combine two or more queries with different schema to create a new query.



```
= #table(1, {{Source}})
```

Append Queries

The append operation creates a single query by adding the contents of one or more query to another, and aggregates the column headers from the queries to create the schema for the new table.



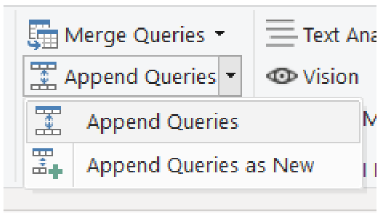
A	B	C
1	1	1
2	2	2
3	3	3

A	B	D
4	4	4
5	5	5

A	B	C	D
1	1	1	*null*
2	2	2	*null*
3	3	3	*null*
4	4	*null*	4
5	5	*null*	5

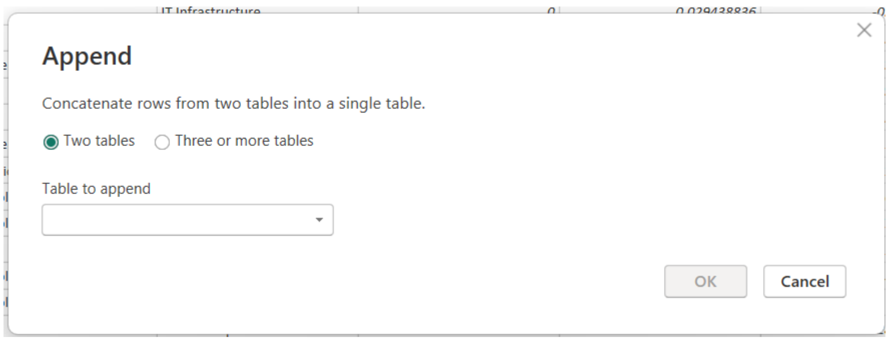
As you can see, if the tables have different column headers all column headers are appended and added to the new table. This will result to null record on the tables that does not have a column header from other tables.

The append operation requires at least two tables. You can append a query to an existing query, or create a new query which appends two or more queries.



Append to existing query

When appending to an existing query using Two Tables, the dialog box will ask you which table you want to append to the existing one.

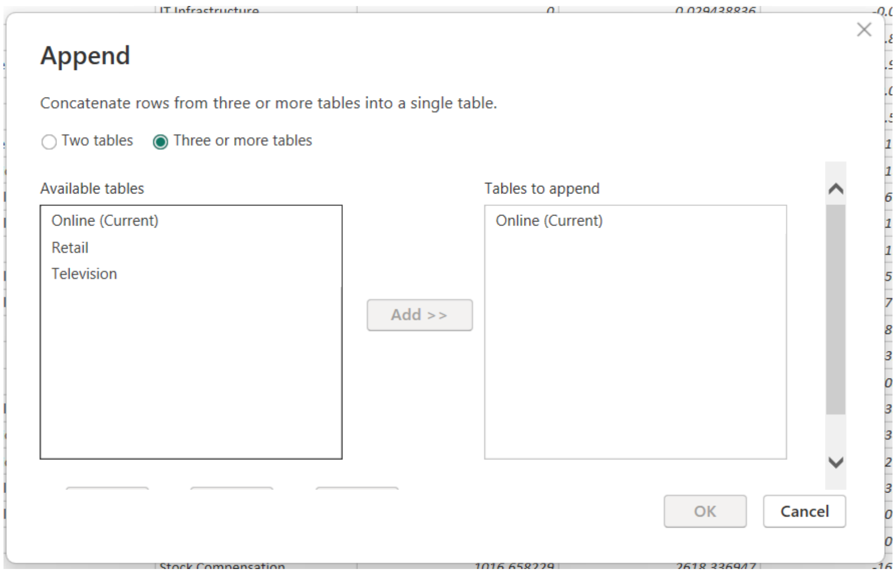

Append

Concatenate rows from two tables into a single table.

[✓ Two tables] Three or more tables

Table to append

While for Three or more tables, you will see which table you are currently in, and you can select the tables your want to append to your current one.


Append

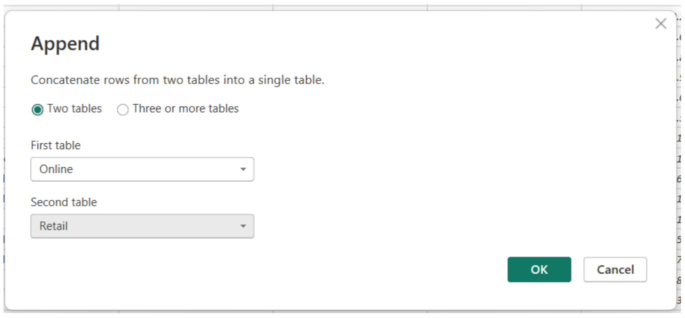
Concatenate rows from three or more tables into a single table.

Two tables [✓ Three or more tables]

Available tables	Tables to append
Online (Current)	Online (Current)
Retail	
Television	

Append to new query

When appending two tables to a new query, the dialog box will ask you for the first and second table you want to append.


Append

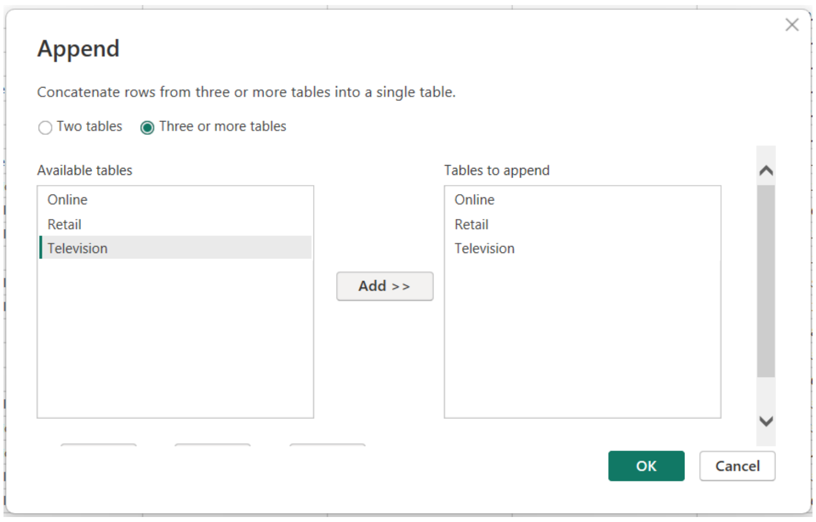
Concatenate rows from two tables into a single table.

[✓ Two tables] Three or more tables

First table: Online

Second table: Retail

While for three or more tables, you can select which tables you want to append by adding it to the **Tables to append** list.


Append

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

Two tables [✓ Three or more tables]

Available tables: Online, Retail, Television

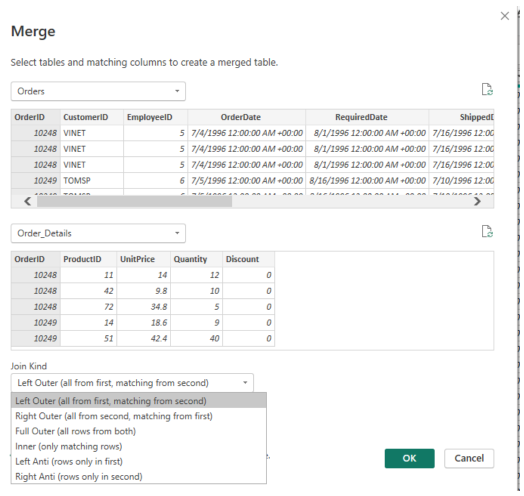
Tables to append: Online, Retail, Television

The tables will be appended in the order in which they're selected, starting with the Primary table for the Two tables mode and from the primary table in the Tables to append list for the Three or more tables mode.

Merge Queries

Merging queries requires the concept of joining tables in database. There are different types of JOIN that you can use in merging queries. The join has different behavior based on the requirement of your query.

Merge queries uses a common field or key to join to one or more table.


Merge

Select tables and matching columns to create a merged table.

Orders

OrderID	CustomerID	EmployeeID	OrderDate	RequiredDate	ShippedI
10248	VINET	5	7/4/1996 12:00:00 AM +00:00	8/1/1996 12:00:00 AM +00:00	7/16/1996 12:00
10248	VINET	5	7/4/1996 12:00:00 AM +00:00	8/1/1996 12:00:00 AM +00:00	7/16/1996 12:00
10248	VINET	5	7/4/1996 12:00:00 AM +00:00	8/1/1996 12:00:00 AM +00:00	7/16/1996 12:00
10249	TOMSP	6	7/5/1996 12:00:00 AM +00:00	8/16/1996 12:00:00 AM +00:00	7/10/1996 12:00

Order_Details

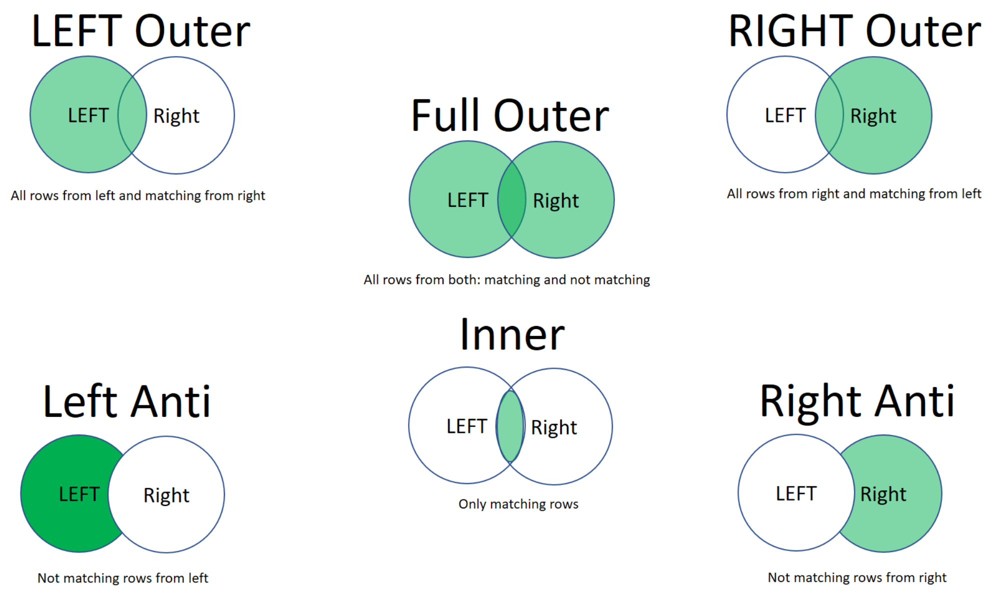
OrderID	ProductID	UnitPrice	Quantity	Discount
10248	11	14	12	0
10248	42	9.8	10	0
10248	72	34.8	5	0
10249	14	18.6	9	0
10249	51	42.4	40	0

Join Kind

Left Outer (all from first, matching from second)

Left Outer (all from first, matching from second)
Right Outer (all from second, matching from first)
Full Outer (all rows from both)
Inner (only matching rows)
Left Anti (rows only in first)
Right Anti (rows only in second)
OK
Cancel

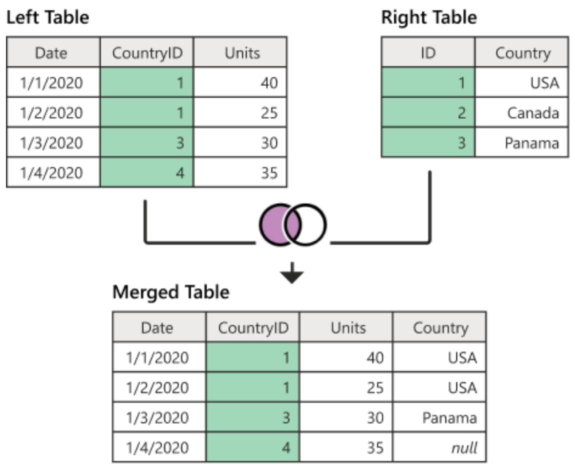
Types of JOIN



LEFT Outer
LEFT
Right
All rows from left and matching from right
RIGHT Outer
LEFT
Right
All rows from right and matching from left
Full Outer
LEFT
Right
All rows from both: matching and not matching
Inner
LEFT
Right
Only matching rows
Left Anti
LEFT
Right
Not matching rows from left
Right Anti
LEFT
Right
Not matching rows from right

LEFT Outer Join

LEFT outer join keeps all the rows from the left table and brings in any matching rows from the right table



Left Table

Date	CountryID	Units
1/1/2020	1	40
1/2/2020	1	25
1/3/2020	3	30
1/4/2020	4	35

Right Table

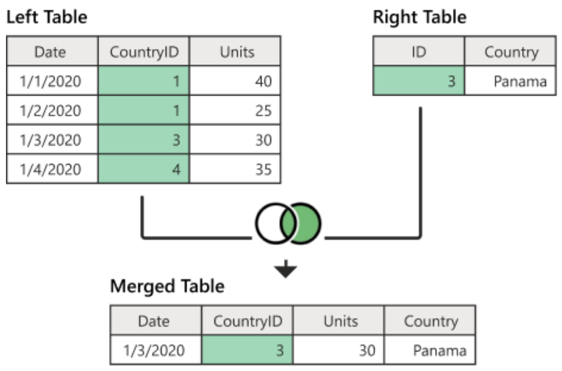
ID	Country
1	USA
2	Canada
3	Panama

Merged Table

Date	CountryID	Units	Country
1/1/2020	1	40	USA
1/2/2020	1	25	USA
1/3/2020	3	30	Panama
1/4/2020	4	35	*null*

RIGHT Outer Join

RIGHT outer join, which keeps all the rows from the right table and brings in any matching rows from the left table



Left Table

Date	CountryID	Units
1/1/2020	1	40
1/2/2020	1	25
1/3/2020	3	30
1/4/2020	4	35

Right Table

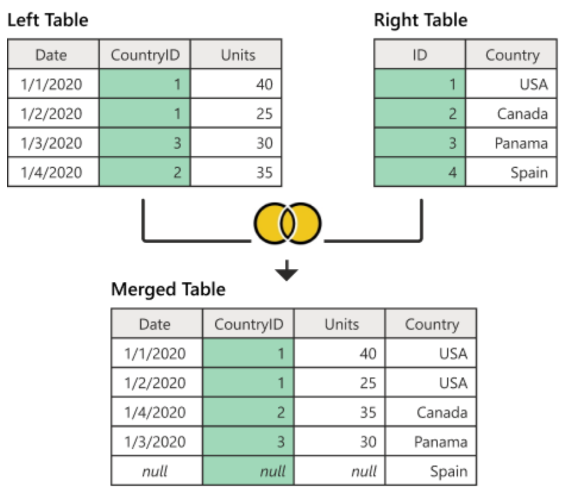
ID	Country
3	Panama

Merged Table

Date	CountryID	Units	Country
1/3/2020	3	30	Panama

FULL Outer Join

FULL outer join, which brings in all the rows from both the left and right tables.



Left Table

Date	CountryID	Units
1/1/2020	1	40
1/2/2020	1	25
1/3/2020	3	30
1/4/2020	2	35

Right Table

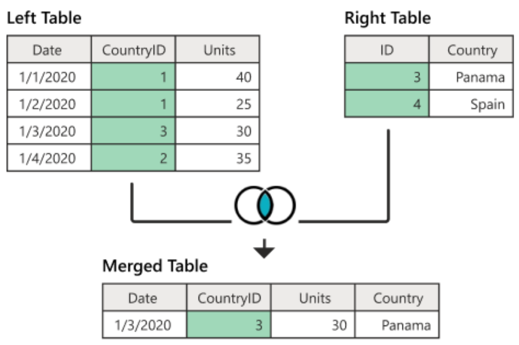
ID	Country
1	USA
2	Canada
3	Panama
4	Spain

Merged Table

Date	CountryID	Units	Country
1/1/2020	1	40	USA
1/2/2020	1	25	USA
1/4/2020	2	35	Canada
1/3/2020	3	30	Panama
null	*null*	*null*	Spain

INNER Join

INNER join, which brings in only matching rows from both the left and right tables.



Left Table

Date	CountryID	Units
1/1/2020	1	40
1/2/2020	1	25
1/3/2020	3	30
1/4/2020	2	35

Right Table

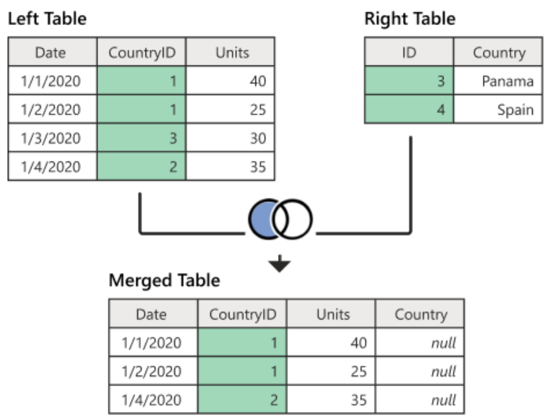
ID	Country
3	Panama
4	Spain

Merged Table

Date	CountryID	Units	Country
1/3/2020	3	30	Panama

LEFT anti join

LEFT Anti Join, which brings in only rows from the left table that don't have any matching rows from the right table.



Left Table

Date	CountryID	Units
1/1/2020	1	40
1/2/2020	1	25
1/3/2020	3	30
1/4/2020	2	35

Right Table

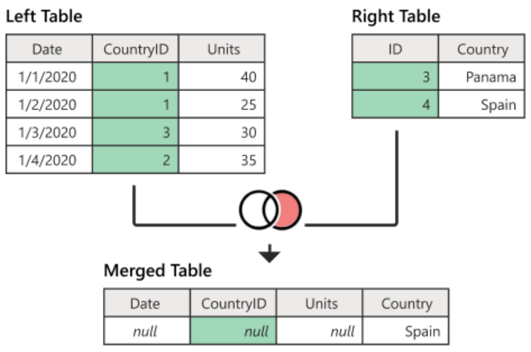
ID	Country
3	Panama
4	Spain

Merged Table

Date	CountryID	Units	Country
1/1/2020	1	40	*null*
1/2/2020	1	25	*null*
1/4/2020	2	35	*null*

RIGHT anti join

RIGHT Anti Join, which brings in only rows from the right table that don't have any matching rows from the left table.



Left Table

Date	CountryID	Units
1/1/2020	1	40
1/2/2020	1	25
1/3/2020	3	30
1/4/2020	2	35

Right Table

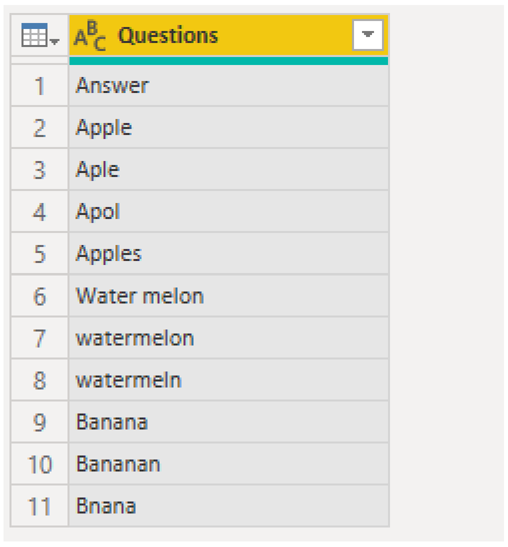
ID	Country
3	Panama
4	Spain

Merged Table

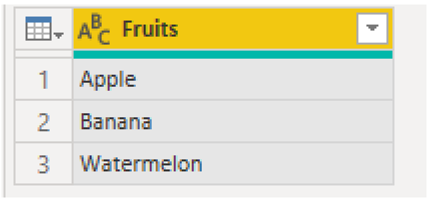
Date	CountryID	Units	Country
null	null	null	Spain

Fuzzy Matching

Fuzzy merge is a smart data preparation feature you can use to apply fuzzy matching algorithms when comparing columns, to try to find matches across the tables that are being merged.



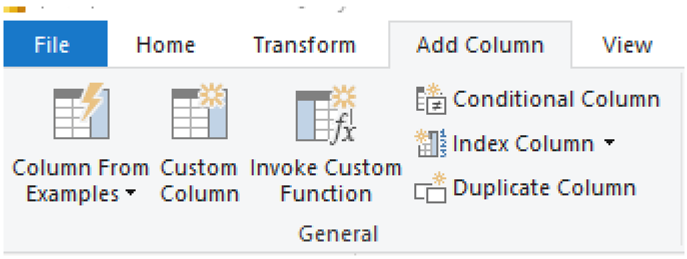
	Questions
1	Answer
2	Apple
3	Aple
4	Apol
5	Apples
6	Water melon
7	watermelon
8	watermeln
9	Banana
10	Bananan
11	Bnana



	Fruits
1	Apple
2	Banana
3	Watermelon

Adding Columns

You can add columns in your existing query. This column will only be loaded in PowerBI and not to your source file. Most of the time you will add a column for derived values. Derived values or columns are columns that are not included in the source of your query. You usually add it by using operators between two different columns.

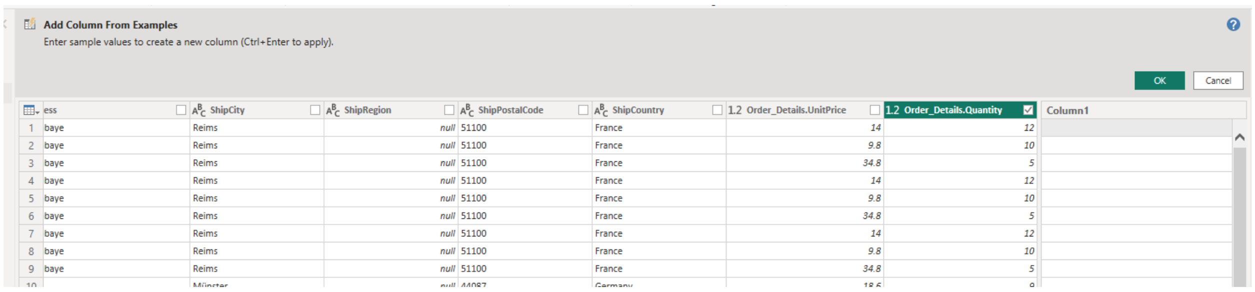


Add a column from examples

When you add columns from examples, you can quickly and easily create new columns that meet your needs. This is useful for the following situations:

- You know the data you want in your new column, but you're not sure which transformation, or collection of transformations, will get you there.
- You already know which transformations you need, but you're not sure what to select in the UI to make them happen.
- You know all about the transformations you need by using a custom column expression in the M language, but one or more of those transformations aren't available in the UI.

When adding a column from example, PowerBI will ask you for some sample values to be used as the reference I creating the column.

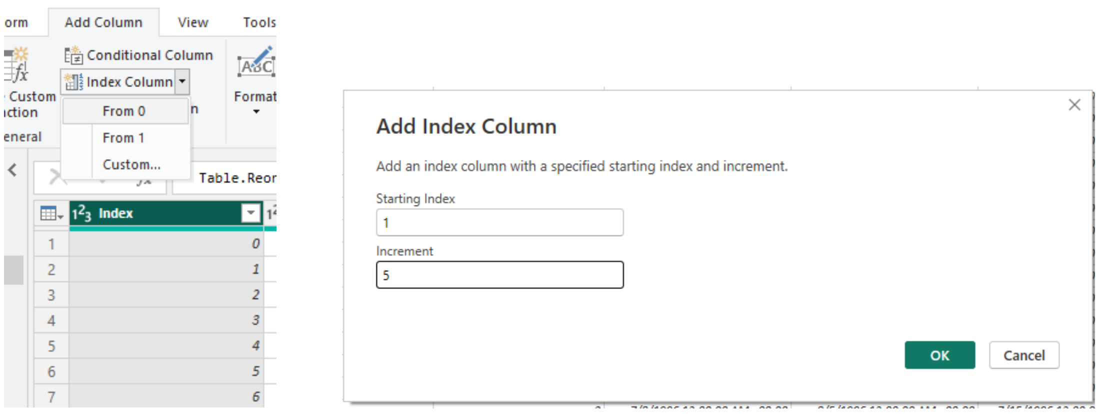

Add Column From Examples

Enter sample values to create a new column (Ctrl+Enter to apply).

	ess	ShipCity	ShipRegion	ShipPostalCode	ShipCountry	1.2 Order_Details.UnitPrice	1.2 Order_Details.Quantity	Column1
1	baye	Reims	null	51100	France	14	12	
2	baye	Reims	null	51100	France	9.8	10	
3	baye	Reims	null	51100	France	34.8	5	
4	baye	Reims	null	51100	France	14	12	
5	baye	Reims	null	51100	France	9.8	10	
6	baye	Reims	null	51100	France	34.8	5	
7	baye	Reims	null	51100	France	14	12	
8	baye	Reims	null	51100	France	9.8	10	
9	baye	Reims	null	51100	France	34.8	5	
10		Münster	null	44087	Germany	18.6	9	

Add Index Columns

The **Index column** command adds a new column to the table with explicit position values, and is usually created to support other transformation patterns.



Add Index Column

Add an index column with a specified starting index and increment.

Starting Index: 1

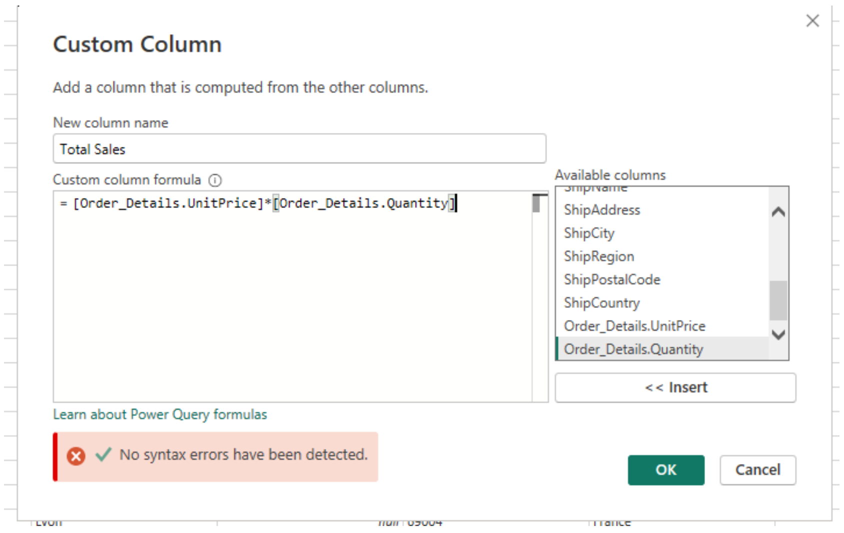
Increment: 5

OK Cancel

By default, the index starts at 0 or 1, but you can specify it using the Custom dialog box.

Add a custom column

This feature allows you to add custom columns using Powe Query M Language. You can use operators or conditional statement to create your custom column.



Custom Column

Add a column that is computed from the other columns.

New column name: Total Sales

Custom column formula:

```
= [Order_Details.UnitPrice]*[Order_Details.Quantity]
```

Available columns: ShipAddress, ShipCity, ShipRegion, ShipPostalCode, ShipCountry, Order_Details.UnitPrice, Order_Details.Quantity

<< Insert

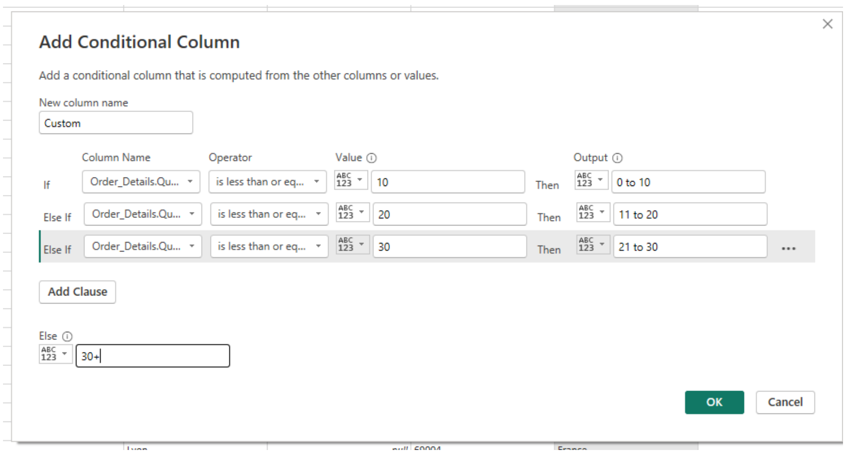
Learn about Power Query formulas

No syntax errors have been detected.

OK Cancel

Add a conditional column

Conditional columns allows you to create columns based on conditions applied to other columns in your table. Multiple conditions uses an **IF ELSEIF** condition, the **ELSE** field at the bottom is where you can put your default value, in case the value does not meet all the conditions stated.


Add Conditional Column

Add a conditional column that is computed from the other columns or values.

New column name: Custom

	Column Name	Operator	Value		Output
If	Order_Details.Qu...	is less than or eq...	10	Then	0 to 10
Else If	Order_Details.Qu...	is less than or eq...	20	Then	11 to 20
Else If	Order_Details.Qu...	is less than or eq...	30	Then	21 to 30

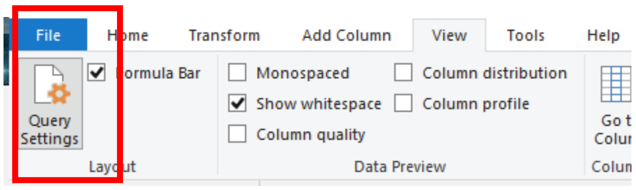
Else: 30+

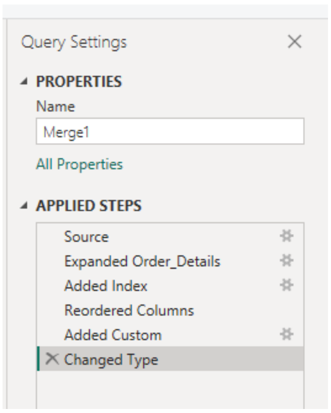
Duplicate Column

This feature creates a copy of the column selected. This is in case you want to transform your column and still preserve the original value.

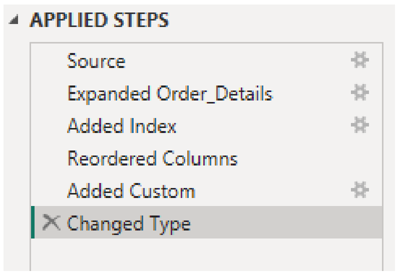
Applied Steps and Advance Editor

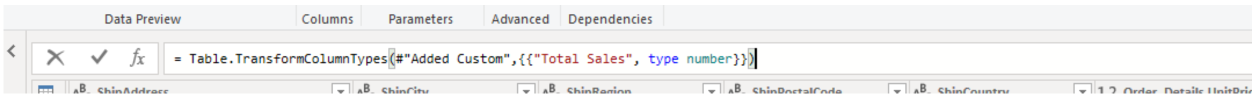
The Applied Steps serves as the record or logs of the steps you did in your data transformation. This also allows you to go back to the previous step and correct the step that causes error in your query.





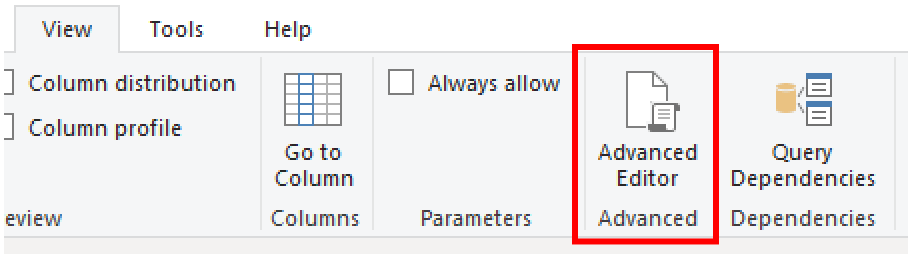
Each step is a visual representation of a Power Query M Language. When you click a step, you can see in the formula bar the corresponding M language equivalent for the step.



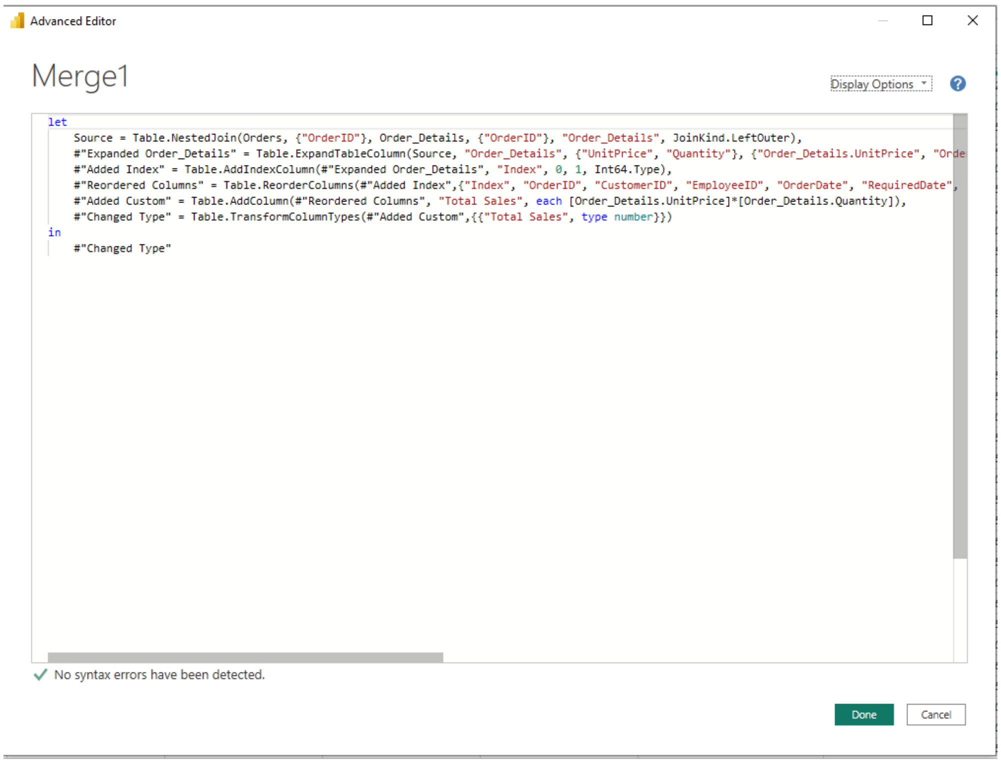


```
= Table.TransformColumnTypes(#"Added Custom",{{"Total Sales", type number}})
```

To view the whole steps as an M Language you can go to the Advanced Editor



M Language is a functional programming language. We will discuss M Language further in the next chapters.



Merge1

```
let
    Source = Table.NestedJoin(Orders, {"OrderID"}, Order_Details, {"OrderID"}, "Order_Details", JoinKind.LeftOuter),
    #"Expanded Order_Details" = Table.ExpandTableColumn(Source, "Order_Details", {"UnitPrice", "Quantity"}, {"Order_Details.UnitPrice", "Orde
    #"Added Index" = Table.AddIndexColumn(#"Expanded Order_Details", "Index", 0, 1, Int64.Type),
    #"Reordered Columns" = Table.ReorderColumns(#"Added Index",{"Index", "OrderID", "CustomerID", "EmployeeID", "OrderDate", "RequiredDate",
    #"Added Custom" = Table.AddColumn(#"Reordered Columns", "Total Sales", each [Order_Details.UnitPrice]*[Order_Details.Quantity]),
    #"Changed Type" = Table.TransformColumnTypes(#"Added Custom",{{"Total Sales", type number}})
in
    #"Changed Type"
```

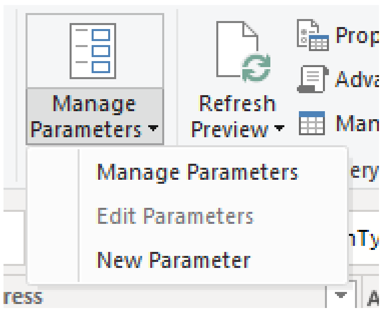
No syntax errors have been detected.

Using Query Parameters

Parameters serves as a way for you to easily store and manage values. These values can be reused in your query.

Parameters give you flexibility in changing the output of your query. This also used for:

- Changing the argument values for particular transforms and data source functions.
- Inputs in custom functions.

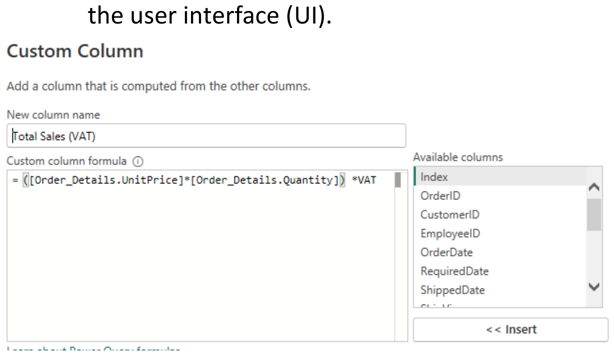


You can create a parameter from the following suggested values:

- Any Value – Allows you to enter your value manually
- List of Values – Allows you to define a list of suggested values which you can later select from for the **Current Value**, and you can set your **Default Value**.
- Query – Uses a list query (a query whose output is a list) to provide the list of suggested values that you can later select for the **Current Value**.

A parameter can be used in many different ways, but it's more commonly used in two scenarios:

- **Step argument**: You can use a parameter as the argument of multiple transformations driven from the user interface (UI).


Custom Column

Add a column that is computed from the other columns.

New column name

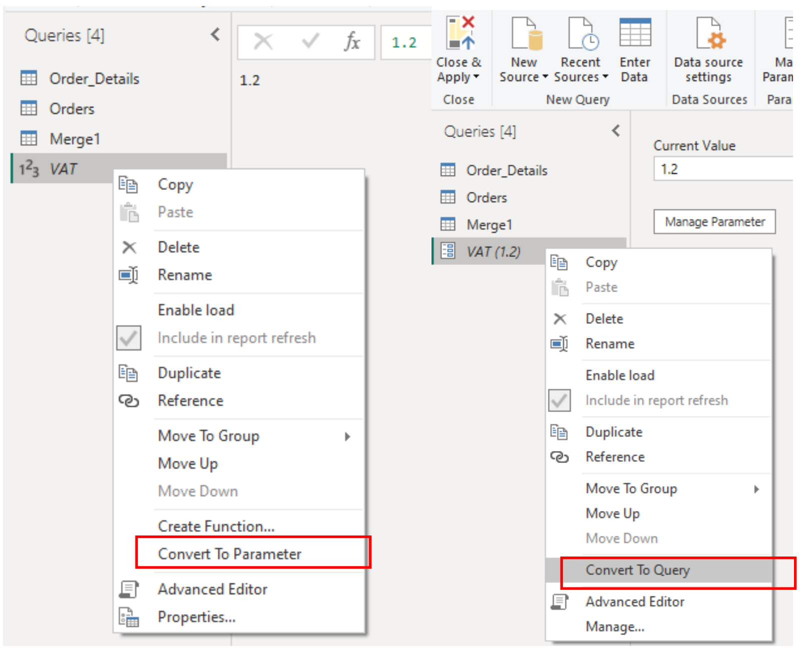
Total Sales (VAT)

Custom column formula

```
= ([Order_Details.UnitPrice]*[Order_Details.Quantity]) *VAT
```

- **Custom Function argument**: You can create a new function from a query and reference parameters as the arguments of your custom function.

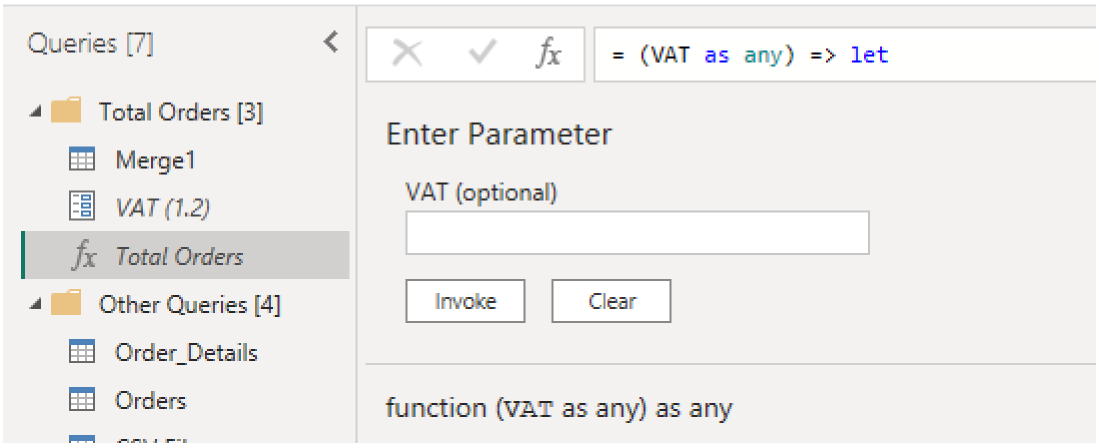
You can also convert an existing query in to a parameter. By right clicking the query and select Convert to Parameter. You can also do the other way around, by converting a parameter into a query. Both can still be used as arguments to a step or function.



Using Functions

Functions allows you to create steps and save it so that it can be reused by another query. This feature allows you not only easily change your query, but also reduce the errors from duplicate transformation steps.

A custom function is a mapping from a set of input values to a single output value, and is created from native M functions and operators.

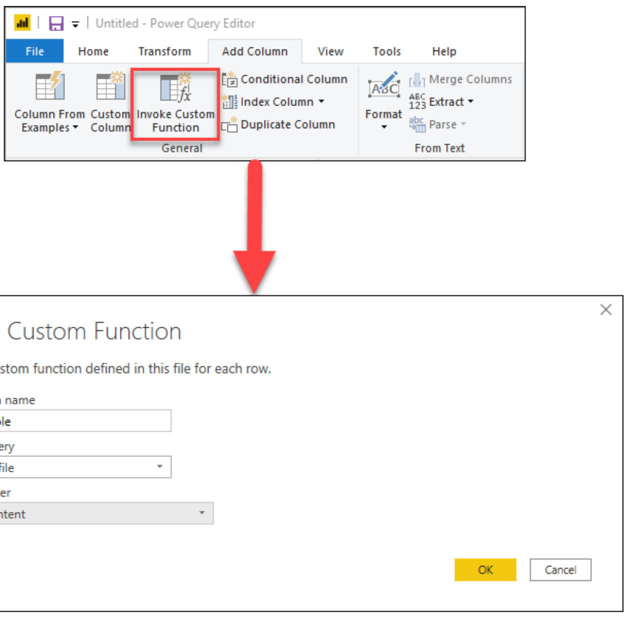


```
= (VAT as any) => let
```

function (VAT as any) as any

Invoking Functions as Column

With a new function created, you can use this function to another query. You will need to enter the data needed for the parameters to perform the function.



Invoke Custom Function

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

New column name: Output Table

Function query: Transform file

File Parameter: Content

Best Practices in Power Query

Choose the right connector

Using the best connector for the task will provide you with the best experience and performance

Filter early

This will let you better focus on your task at hand by only showing data that’s relevant in the data preview section.

Do expensive operations last

When possible, perform such streaming operations first, and do any more expensive operations last. This will help minimize the amount of time you spend waiting for the preview to render each time you add a new step to your query.

Temporarily work against a subset of your data

If adding new steps to your query in the Power Query Editor is slow, consider first doing a "Keep First Rows" operation and limiting the number of rows you're working against. Then, once you've added all the steps you need, remove the "Keep First Rows" step.

Use the correct data types

It's crucial that you always work with the correct data types for your columns. When working with structured data sources such as databases, the data type information will be brought from the table schema found in the database. By default, Power Query offers an automatic data type detection for unstructured data sources.

Explore your data

Utilize Power Query profiling tools to discover your data.

Document your work

We recommend that you document your queries by renaming or adding a description to your steps, queries, or groups as you see fit.

Take a modular approach
If the query contains a large number of steps, then it might be a good idea to split the query into multiple queries, where one query references the next.

Create groups
This is a great way of organizing your work.

Future-proofing queries
It's a best practice to define the scope of your query as to what it should do and what it should account for in terms of structure, layout, column names, data types, and any other component that you consider relevant to the scope.

Use parameters
Parameters in Power Query help you make your queries more dynamic and flexible. A parameter serves as a way to easily store and manage a value that can be reused in many different ways.

Create reusable functions
If you find yourself in a situation where you need to apply the same set of transformations to different queries or values, creating a Power Query custom function that can be reused as many times as you need could be beneficial.

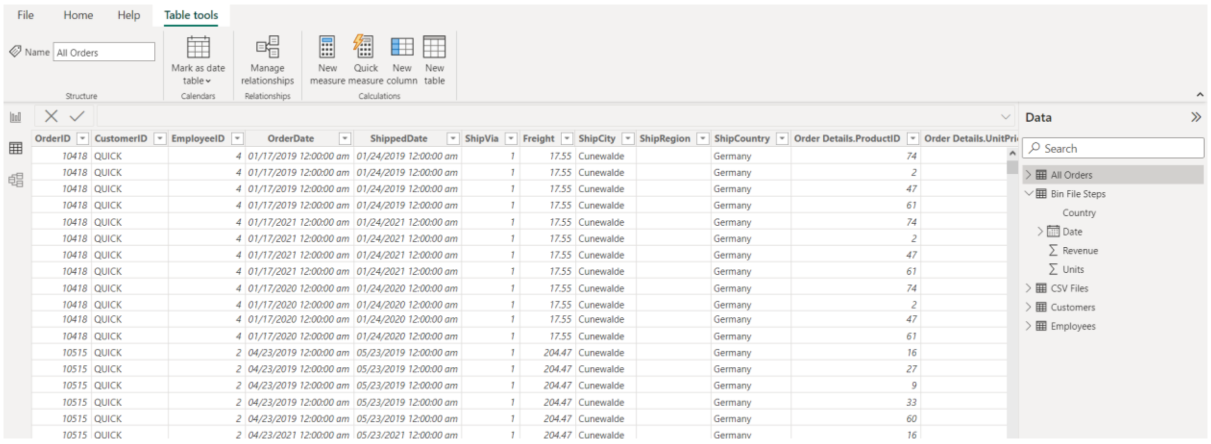
Chapter 3: Data Model Layer

Model View

The next step in our PowerBI journey is the Data Model Layer. After transforming your data using Power Query. It is now ready to be loaded in your Data Model View. The Data Model Layer has 2 views: **Data View** and **Model View**.

Data View

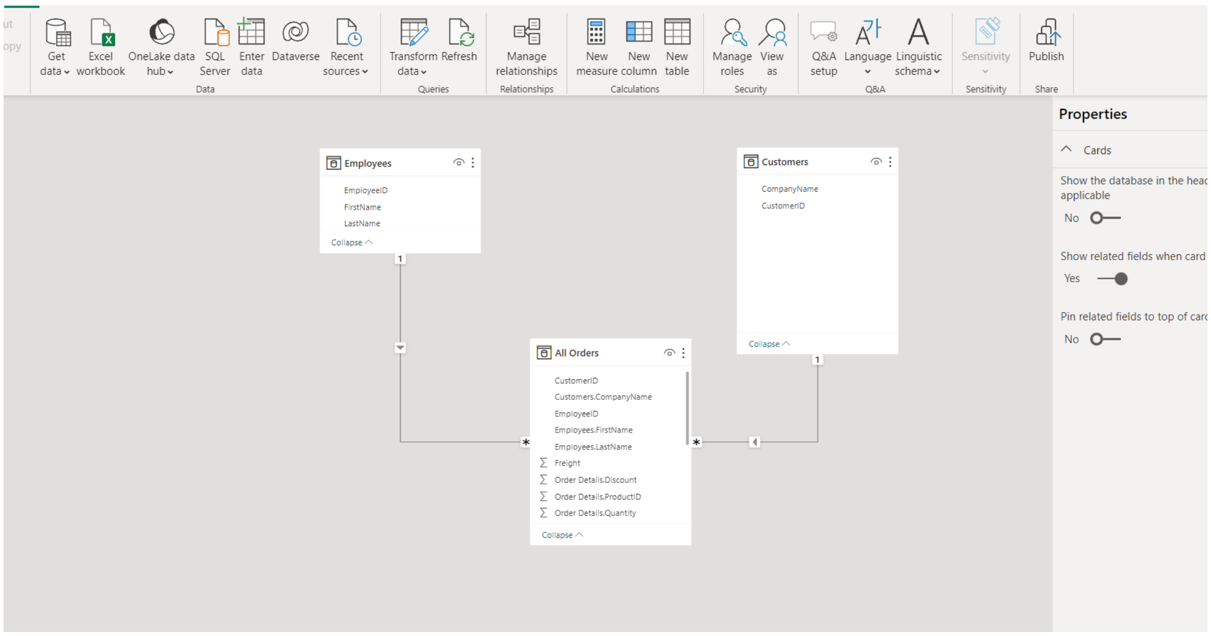
The Data View shows your data in tabular format. It allows you to see the records in your table. This allows you to do additional data modeling like adding columns, filtering rows, and Measures.



OrderID	CustomerID	EmployeeID	OrderDate	ShippedDate	ShipVia	Freight	ShipCity	ShipRegion	ShipCountry	Order Details.ProductID	Order Details.UnitPri
10418	QUICK	4	01/17/2019 12:00:00 am	01/24/2019 12:00:00 am	1	17.55	Cunewalde		Germany	74	
10418	QUICK	4	01/17/2019 12:00:00 am	01/24/2019 12:00:00 am	1	17.55	Cunewalde		Germany	2	
10418	QUICK	4	01/17/2019 12:00:00 am	01/24/2019 12:00:00 am	1	17.55	Cunewalde		Germany	47	
10418	QUICK	4	01/17/2019 12:00:00 am	01/24/2019 12:00:00 am	1	17.55	Cunewalde		Germany	61	
10418	QUICK	4	01/17/2021 12:00:00 am	01/24/2021 12:00:00 am	1	17.55	Cunewalde		Germany	74	
10418	QUICK	4	01/17/2021 12:00:00 am	01/24/2021 12:00:00 am	1	17.55	Cunewalde		Germany	2	
10418	QUICK	4	01/17/2021 12:00:00 am	01/24/2021 12:00:00 am	1	17.55	Cunewalde		Germany	47	
10418	QUICK	4	01/17/2021 12:00:00 am	01/24/2021 12:00:00 am	1	17.55	Cunewalde		Germany	61	
10418	QUICK	4	01/17/2020 12:00:00 am	01/24/2020 12:00:00 am	1	17.55	Cunewalde		Germany	74	
10418	QUICK	4	01/17/2020 12:00:00 am	01/24/2020 12:00:00 am	1	17.55	Cunewalde		Germany	2	
10418	QUICK	4	01/17/2020 12:00:00 am	01/24/2020 12:00:00 am	1	17.55	Cunewalde		Germany	47	
10418	QUICK	4	01/17/2020 12:00:00 am	01/24/2020 12:00:00 am	1	17.55	Cunewalde		Germany	61	
10515	QUICK	2	04/23/2019 12:00:00 am	05/23/2019 12:00:00 am	1	204.47	Cunewalde		Germany	16	
10515	QUICK	2	04/23/2019 12:00:00 am	05/23/2019 12:00:00 am	1	204.47	Cunewalde		Germany	27	
10515	QUICK	2	04/23/2019 12:00:00 am	05/23/2019 12:00:00 am	1	204.47	Cunewalde		Germany	9	
10515	QUICK	2	04/23/2019 12:00:00 am	05/23/2019 12:00:00 am	1	204.47	Cunewalde		Germany	33	
10515	QUICK	2	04/23/2019 12:00:00 am	05/23/2019 12:00:00 am	1	204.47	Cunewalde		Germany	60	
10515	QUICK	2	04/23/2021 12:00:00 am	05/23/2021 12:00:00 am	1	204.47	Cunewalde		Germany	16	

Model View

The Model View allows you see the relationship between your tables/queries. You can also manage the relationships, create, edit, and remove relationships. With the Model View, you can easily see the cardinality of the relationship.



Properties

Cards

Show the database in the header when applicable: No

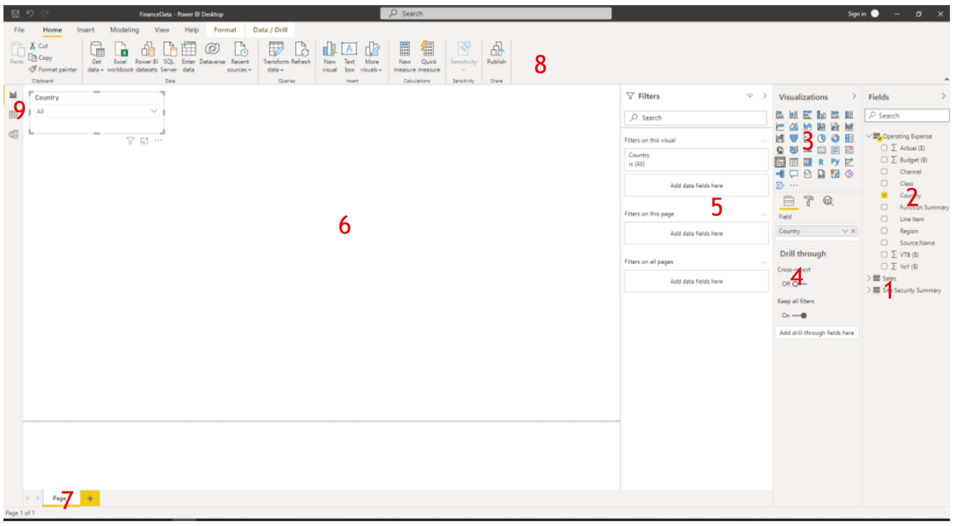
Show related fields when card: Yes

Pin related fields to top of card: No

Report View

This the default view of PowerBI. You can see the different commands you can use to create your dashboard.

1. Tables
2. Fields
3. Visuals
4. Visual Properties
5. Filter Option
6. Canvas
7. Page
8. Menu
9. View Type



DAX – Data Expressions

In this layer you will be looking into a different kind of language for your data modeling. DAX is designed to specifically compute business formulas over a data model.

We can compare DAX with Excel formula. Most likely, if you are familiar with Excel Formulas, DAX will most likely perform the same way an Excel formula does.

DAX vs Excel

Most DAX formulas has the same formula name and behavior as MS Excel formulas. Here are some items we need to remember about DAX.

- Excel formulas take cell address as reference. DAX formula uses table column or table as reference. Just like how Excel Table object works.
- Excel has no function that returns a table, but Excel has some function that can work with arrays. While DAX can reference tables and return tables.
- DAX lookup requires relationship between tables. Excel can use cell reference address.
- Excel supports data with different type in a column, while DAX expects the data to have the same type in each column.

You use DAX to compute values over columns in tables. You can aggregate, calculate, and search for numbers but, at the end, all of the calculations involve tables and columns. Thus, the first syntax to learn is how to reference a column in a table.

```
Sales[SalesAmount] = Sales[ProductPrice] * Sales[ProductQuantity]
```

Many functions in DAX work the same as the equivalent Excel function. The IF function, for example, reads in the same way in DAX and in Excel:

Excel IF ([@SalesAmount] > 10, 1, 0)

DAX IF (Sales[SalesAmount] > 10, 1, 0)

Data Types

PowerBI is strict with data types, unlike MS Excel where data types may vary per column, PowerBI has explicit data type per column.

- Whole Number (Integer)
- Decimal Number (Float)
- Currency (Currency), a fixed decimal number internally stored as an integer
- Date (DateTime)
- Boolean (TRUE/FALSE)
- Text (String)
- Binary large object (BLOB)

Operators

DAX, just like Excel uses operators to perform different calculations. Keep in mind that the operations follows an order of precedence. It does not perform as a first come first serve.

Arithmetic Operators

Operator Type	Symbol	Use	Example
Parenthesis	( )	Precedence order and grouping of arguments	(5 + 2) * 3
Arithmetic	+ - * /	Addition Subtraction/negation Multiplication Division	4 + 2 5 – 3 4 * 2 4 / 2

Conditional Operators

Operator Type	Symbol	Use	Example

Comparison	= <> > >= < <=	Equal to Not equal to Greater than Greater than or equal to Less than Less than or equal to	[CountryRegion] = "USA" [CountryRegion] <> "USA" [Quantity] > 0 [Quantity] >= 100 [Quantity] < 0 [Quantity] <= 100

Logical Operators

Operator Type	Symbol	Use	Example
Text Concatenation	&	Concatenation of strings	"Value is" & [Amount]
Logical	&& || IN NOT	AND condition between two Boolean expressions OR condition between two Boolean expressions Inclusion of an element in a list Boolean negation	[CountryRegion] = "USA" && [Quantity]>0 [CountryRegion] = "USA" || [Quantity] > 0 [CountryRegion] IN {"USA", "Canada"} NOT [Quantity] > 0

Aside from using the logical operators, you can also use the **AND** function and **OR** function to check two conditions.

AND(<logical1>,<logical2>)

```
= IF( AND(  SUM( 'InternetSales_USD'[SalesAmount_USD])
           >SUM('ResellerSales_USD'[SalesAmount_USD])
          , CALCULATE(SUM('InternetSales_USD'[SalesAmount_USD]),
PREVIOUSYEAR('DateTime'[DateKey] ))
           >CALCULATE(SUM('ResellerSales_USD'[SalesAmount_USD]),
PREVIOUSYEAR('DateTime'[DateKey] ))
          )
     , "Internet Hit"
     , ""
     )
```

OR(<logical1>,<logical2>)

```
IF(OR(CALCULATE(SUM('ResellerSales_USD'[SalesAmount_USD]),
'ProductSubcategory'[ProductSubcategoryName]="Touring Bikes") > 1000000
         ,   CALCULATE(SUM('ResellerSales_USD'[SalesAmount_USD]),
'DateTime'[CalendarYear]=2007) > 2500000), "Circle of Excellence"
   , ""  )
```

In using AND and OR you need to be familiar of the condition matrix to know what result you will get.

AND Operator (&&)		
Cond 1	Cond 2	Result
True	True	True
True	False	False
False	True	False
False	False	False

OR Operator (||)		
Cond 1	Cond 2	Result
True	True	True
True	False	True
False	True	True
False	False	False

Conditional Statement

Conditional Statements allows you to perform operation based on specified conditions.

IF Statement

The IF Statement checks for the condition and returns a TRUE statement, and FALSE statement if the condition is not met.

IF(<logical_test>, <value_if_true>[, <value_if_false>])

Nested IF

Nesting IF statement allows you have multiple conditions and have multiple results for each condition

SalesCategory = IF(Sales[GrossMargin]>2000,"Cat 2", IF(Sales[GrossMargin]>1000,"Cat 1","NA"))

Once the first condition is met, it will not proceed to read the following conditions.

SWITCH Function

The SWITCH Function evaluates an expression against a list of values and returns one of multiple possible result expressions. This function can be used to avoid having multiple nested IF statements.

SWITCH(<expression>, <value>, <result>[, <value>, <result>]…[, <else>])

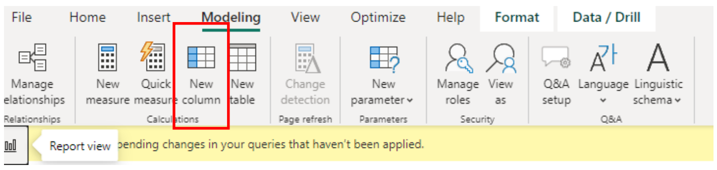
```
= SWITCH (
    [Month Number Of Year],
    1, "January",
    2, "February",
    3, "March",
    4, "April",
    5, "May",
    6, "June",
    7, "July",
    8, "August",
    9, "September",
    10, "October",
    11, "November",
    12, "December",
    "Unknown month number"
   )
```

Calculated Columns and Measures

Calculated Columns

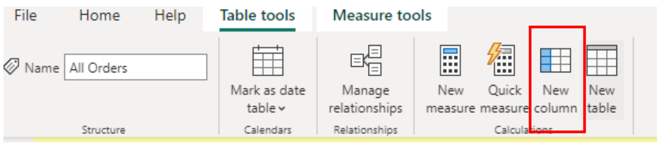
This allows you to create a new column for your table without loading it from your query. Which means that this column is only visible in your Model View and not in your query. You can add a column using the **New Column** command.

If you are in the **Reports View** you can see the Modeling Tab

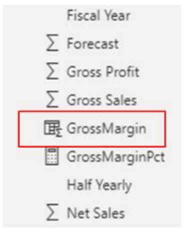


pending changes in your queries that haven't been applied.

If you are in the Model View, you can see the New Column in the Table Tools tab



Calculated columns are created using DAX functions. It is physically added in your table and can be used in creating visuals. Calculated columns you create appear in the Fields list just like any other field, but they'll have a special icon showing its values are the result of a formula.



In this example the name of your column will be **CityState** which is the concatenated value of columns **City** and **State.**

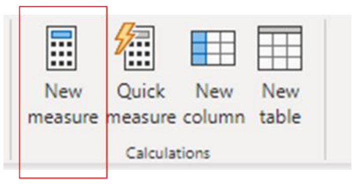
```
CityState = [City] & "," & [State]
```

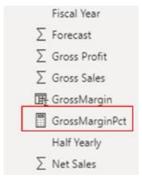
Measures

Measures are is useful whenever you do not want to compute values for each row but, rather, you want to aggregate values from many rows in a table.

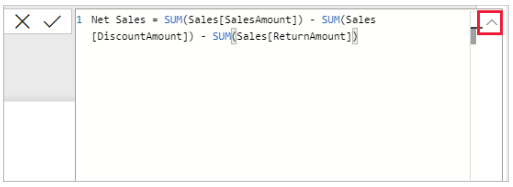
A Measure does not appear in the table when viewed in the Model View. But you can use measures for visualization.

You can easily distinguish a Measure from the other fields, since it has the calculator icon.





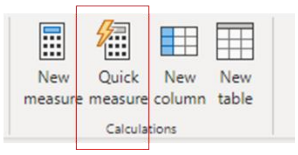
You use DAX formula to create measures.

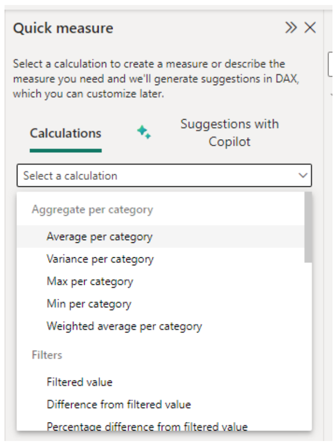


```
Net Sales = SUM(Sales[SalesAmount]) - SUM(Sales
[DiscountAmount]) - SUM(Sales[ReturnAmount])
```

Utilizing Quick Measures

Quick Measures are pre-built measures that you can use, you don’t have to write your own Measure.




Quick measure

Select a calculation to create a measure or describe the measure you need and we'll generate suggestions in DAX, which you can customize later.

Calculations | Suggestions with Copilot

Select a calculation

Aggregate per category
- Average per category
- Variance per category
- Max per category
- Min per category
- Weighted average per category

Filters
- Filtered value
- Difference from filtered value
- Percentage difference from filtered value

You can select different kind of Quick Measures by category. Aggregate, Filters, Time Intelligence, Totals, Mathematical Operations, and Text.

Calculated Columns vs Measures

Calculated columns and Measures are one way of creating DAX statements. They behave quite the same but they are completely different.

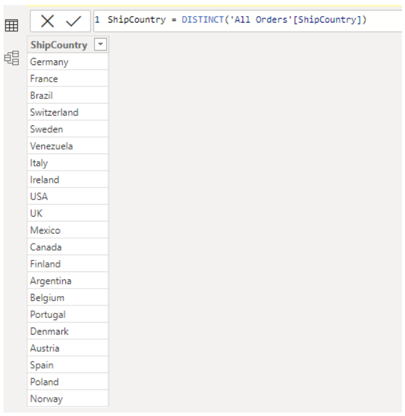
- Columns and measures can be used in visualization
- Columns appear in the table
- Measures do not appear in the table
- The value of a calculated column is computed during data refresh and uses the current row as a context; it does not depend on user activity on the pivot table.
- A measure operates on aggregations of data defined by the current context

When to use

Calculated Columns	Measures
• Place the calculated results in a Slicer, or see results in Rows or Columns in a pivot table (as opposed to the Values area), or use the result as a filter condition in a DAX query. • Define an expression that is strictly bound to the current row. (For example, Price * Quantity cannot work on an average or on a sum of the two columns.) • Categorize text or numbers. (For example, a range of values for a measure, a range of ages of customers, such as 0–18, 18–25, and so on.	• When you calculate profit percentage of a table selection. • When you calculate ratios of a product compared to all products but keeping the filter both by year and region

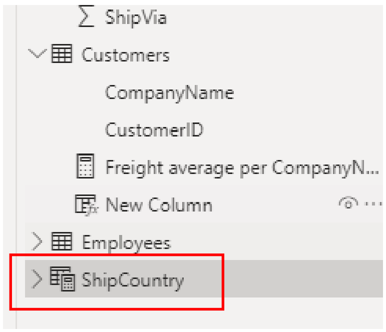
Calculated Tables

Most of the time you create a table using Power Query and import it to the Model View. Calculated Tables allows you to create tables base on the data loaded in your model. You use DAX Table functions to create Calculated Tables like: DISTINCT, VALUES, CROSSJOIN, and UNION.



```
ShipCountry = DISTINCT('All Orders'[ShipCountry])
```

ShipCountry
Germany
France
Brazil
Switzerland
Sweden
Venezuela
Italy
Ireland
USA
UK
Mexico
Canada
Finland
Argentina
Belgium
Portugal
Denmark
Austria
Spain
Poland
Norway



Variables

Variables serves as containers that hold values that you frequently use. Variables are defined with the **VAR** keyword. After you define a variable, you need to provide a RETURN section that defines the result value of the expression. A variable defined in an expression cannot be used outside the expression itself. Variables are computed using lazy evaluation.

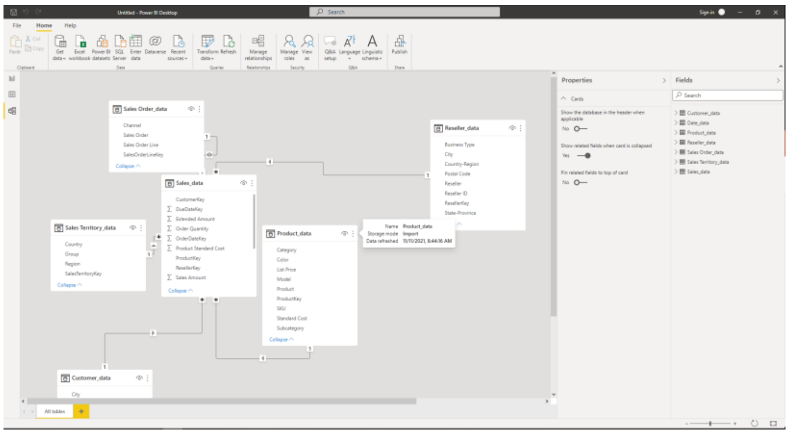
```
Sales YoY Growth % =
VAR SalesPriorYear =
    CALCULATE([Sales], PARALLELPERIOD('Date'[Date], -12, MONTH))
RETURN
    DIVIDE(([Sales] - SalesPriorYear), SalesPriorYear)
```

The advantage of using variables are:

- Improve performance
- Improve readability
- Simplify debugging
- Reduce Complexity

Relationships

Relationship allows your table to be connected and create reference to each other. It allows you to filter data across your tables. PowerBI autodetects relationships when you load your data.



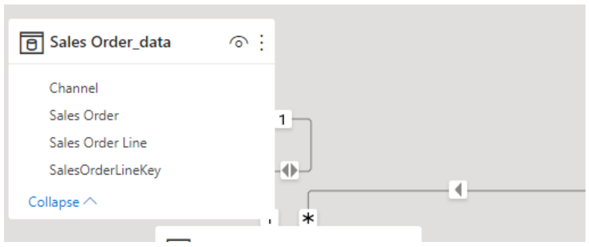
Relationship Cardinality

Cardinality is referred to the relationship between tables. It defines how many instances of one entity are related to instances of another entity. There are four (4) types of cardinality:

- **Many to one (*:1)**: A many-to-one relationship is the most common, default type of relationship. It means the column in a given table can have more than one instance of a value, and the other related table, often know as the lookup table, has only one instance of a value.
- **One to one (1:1)**: In a one-to-one relationship, the column in one table has only one instance of a particular value, and the other related table has only one instance of a particular value.
- **One to many (1:*)**: In a one-to-many relationship, the column in one table has only one instance of a particular value, and the other related table can have more than one instance of a value.
- **Many to many (*:*):** With composite models, you can establish a many-to-many relationship between tables, which removes requirements for unique values in tables. It also removes previous workarounds, such as introducing new tables only to establish relationships. For more information

Cross Filter Direction

Having relationship between tables allows you to create filter between them. In the model view, aside from the relationship, you can also see the cross-filter direction. This allows you to set which table can be used for filtering.

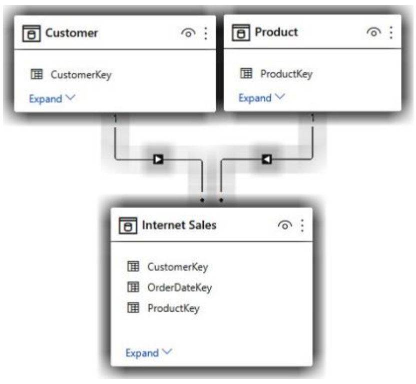


There are two types of Cross Filter:

- Both – For filtering purposes, both tables are treated as if they're a single table
- Single - The most common, default direction, which means filtering choices in connected tables work on the table where values are being aggregated

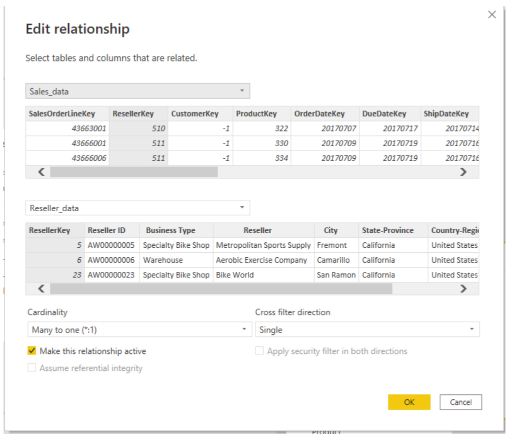
Handling Many-to-Many Relationship

A many-to-many relationship is quite common in your model. For instance, there is always a many-to-many relationship between a customer and a product in a sales system. A customer can buy many products, and a product can end up in many customers' shopping bags. What happens in the sales system is that when we go to the cashier to pay for the products we bought, the cashier scans each product's barcode. So, the system now knows which customer bought which product.



Adding and Editing Relationships

You can create relationships by clicking the Manage Relationship icon in the menu. It will give you an option to Add a new relationship, or edit an existing one.



Edit relationship

Select tables and columns that are related.

Sales_data

SalesOrderLineKey	ResellerKey	CustomerKey	ProductKey	OrderDateKey	DueDateKey	ShipDateKey
43663001	510	-1	322	20170707	20170717	20170714
43666001	511	-1	330	20170709	20170719	20170716
43666006	511	-1	334	20170709	20170719	20170716

Reseller_data

ResellerKey	Reseller ID	Business Type	Reseller	City	State-Province	Country-Regi
5	AW00000005	Specialty Bike Shop	Metropolitan Sports Supply	Fremont	California	United States
6	AW00000006	Warehouse	Aerobic Exercise Company	Camarillo	California	United States
23	AW00000023	Specialty Bike Shop	Bike World	San Ramon	California	United States

Cardinality: Many to one (*:1)

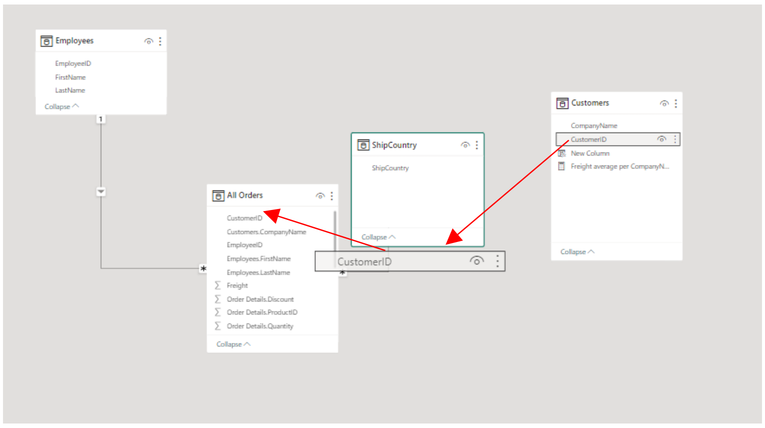
Cross filter direction: Single

☑ Make this relationship active

☐ Apply security filter in both directions

☐ Assume referential integrity

PowerBI also allows you to drag and drop the fields that you want to use to create a relationship.



Chapter 4: DAX Functions

DAX (Data Analysis Expressions) have hundreds of built-in functions in PowerBI. These functions are categorized according to their needs.

This module will not be covering all these functions. We will be discussing some commonly used.

Table functions

Table functions are regular DAX functions that—instead of returning a single value—return a table. These functions are used to create Calculated Tables. Table functions are useful when writing both DAX queries and many advanced calculations that require iterating over tables. Examples of Table functions are:

- FILTER
- ALL
- ALLEXCEPT
- VALUES
- DISTINCT
- ALLSELECTED

FILTER Function

The FILTER function returns a table that represents a subset of another table or expression.

```
FILTER(<table>,<filter>)
```

The first argument is the table you want to filter, then the filter condition to apply.

```
FILTER('InternetSales_USD',
RELATED('SalesTerritory'[SalesTerritoryCountry])<>"United States")
```

ALL and ALLEXCEPT

The ALL Function returns all the rows of a table or all the values of one or more columns, depending on the parameters used. ALL is extremely useful whenever we need to compute percentages or ratios because it ignores the filters automatically introduced by a report.

```
ALL( [<table> | <column>[, <column>[, <column>[,…]]]] )
```

```
= SUMX(ResellerSales_USD,
ResellerSales_USD[SalesAmount_USD])/SUMX(ALL(ResellerSales_USD),
ResellerSales_USD[SalesAmount_USD])
```

ALLEXCEPT function Removes all context filters in the table except filters that have been applied to the specified columns. The function returns a table with all filters removed except for the filters on the specified columns.

```
ALLEXCEPT(<table>,<column>[,<column>[,…]])
```

```
= CALCULATE(SUM(ResellerSales_USD[SalesAmount_USD]), ALLEXCEPT(DateTime,
DateTime[CalendarYear]))
```

VALUE and DISTINCT

VALUE and DISTINCT function return a list of unique values for a column. Both functions are almost identical, the only difference being in how they handle the blank row that might exist in a table. **DISTINCT** always returns all the distinct values of a column. On the other hand, VALUES returns only the distinct visible values. **VALUES** considers the blank row as a valid row, and it returns it. On the other hand, DISTINCT does not return it.

```
VALUES(<TableNameOrColumnName>)
```

```
DISTINCT(<column>)
```

```
DISTINCT(<table>)
```

ALLSELECTED

This function removes context filters from columns and rows in the current query, while retaining all other context filters or explicit filters. This returns the context of the query without any column and row filters.

The ALLSELECTED function gets the context that represents all rows and columns in the query, while keeping explicit filters and contexts other than row and column filters. This function can be used to obtain visual totals in queries.

```
ALLSELECTED([<tableName> | <columnName>[, <columnName>[, <columnName>[,…]]]] )
```

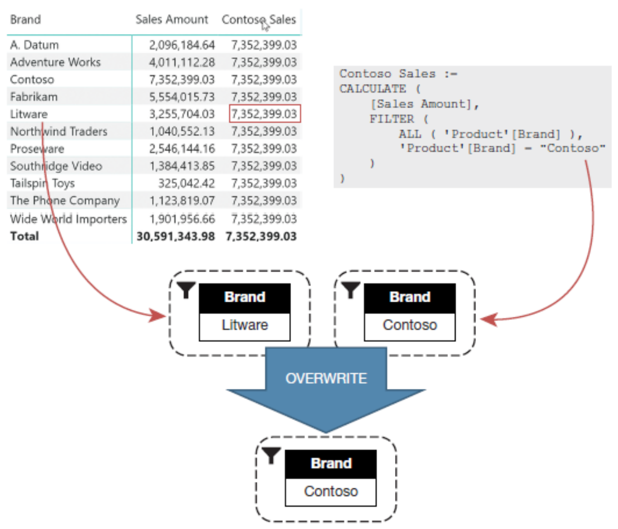
CALCULATE and CALCULATETABLE

Both functions are used to evaluate an expression in a modified filter context. The **CALCULATETABLE** function performs exactly the same functionality and **CALCULATE**, except it modifies the filter context applied to an expression that returns a table object.

```
CALCULATE(<expression>[, <filter1> [, <filter2> [, …]]])
```

```
CALCULATETABLE(<expression>[, <filter1> [, <filter2> [, …]]])
```

The example shows how the CALCULATE function works.



Brand	Sales Amount	Contoso Sales
A. Datum	2,096,184.64	7,352,399.03
Adventure Works	4,011,112.28	7,352,399.03
Contoso	7,352,399.03	7,352,399.03
Fabrikam	5,554,015.73	7,352,399.03
Litware	3,255,704.03	7,352,399.03
Northwind Traders	1,040,552.13	7,352,399.03
Proseware	2,546,144.16	7,352,399.03
Southridge Video	1,384,413.85	7,352,399.03
Tailspin Toys	325,042.42	7,352,399.03
The Phone Company	1,123,819.07	7,352,399.03
Wide World Importers	1,901,956.66	7,352,399.03
Total	**30,591,343.98**	**7,352,399.03**

```
Contoso Sales :=
CALCULATE (
    [Sales Amount],
    FILTER (
        ALL ( 'Product'[Brand] ),
        'Product'[Brand] = "Contoso"
    )
)
```

Things to remember about CALCULATE

- CALCULATE makes a copy of the current filter context.
- CALCULATE evaluates each filter argument and produces, for each condition, the list of valid values for the specified columns.
- If two or more filter arguments affect the same column, they are merged together using an AND operator (or using the set intersection in mathematical terms).
- CALCULATE uses the new condition to replace existing filters on the columns in the model. If a column already has a filter, then the new filter replaces the existing one. On the other hand, if the column does not have a filter, then CALCULATE adds the new filter to the filter
- context.
- Once the new filter context is ready, CALCULATE applies the filter context to the model, and it computes the first argument: the expression. In the end, CALCULATE restores the original filter context, returning the computed result.

Iterator Functions

These functions enumerate all rows of a given table and evaluate a given expression for each row. They provide you with flexibility and control over how your model calculations will summarize data. Common iterator functions are:

- SUMX
- AVERAGEX
- MINX
- MAXX
- COUNTX

Aggregate vs Iterator Functions

There is a big difference on how these functions behave. Iterating functions go through every single row of a table to add logic to each of these rows. Aggregating functions look at the entire column left over after the context is placed in a formula.

Aggregating Function

Sales = SUM(ProductSales[ordersales])

Iterating Function

TotalSales = SUMX(ProductSales, Products[unitprice]* RELATED(ProductSales[orderqty]))

TotalSalesUpper = SUMX(ProductSales,IF([Total Sales] >1000,[Total Sales],0))

When to use?

- You can use aggregate function if you need to do a simple aggregation
  Sales = SUM(ProductSales[ordersales])

- You can use iterating function if you need to make aggregation that includes some complex logic.
  Sales = SUMX(ProductSales, Products[unitprice]* RELATED(ProductSales[orderqty]))

More DAX Functions

You can check all DAX functions from this link: https://learn.microsoft.com/en-us/dax/new-dax-functions

Chapter 5: M Language

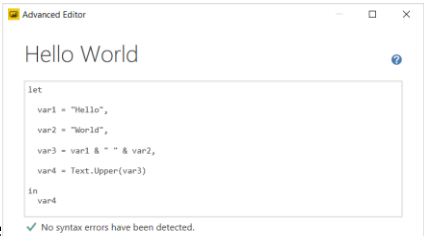
Introduction to M Language

"M" stands for Mashup Language. This is the programming language used by Power Query. Creating a M query can be useful in many ways which allows you to accomplish things that cannot be done in a regulate query editor. M is a functional programming language, computation through evaluation of mathematical functions. Programming involves writing expressions instead of statements. M does not support changing state or mutable data. Every query is a single expression that returns a single value. Every query has a return type. M Language is case sensitive. A query expression can reference other queries by name.

Let Statement

Queries usually created using let statement. Here are some items you need to remember:

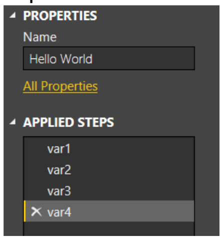
- Allows a single expressions to contain inner expressions
- Each line in let block represents a separate expression
- Each line in let block has variable which is named step
- Each line in let block requires comma at end except for last line



```
let
  var1 = "Hello",
  var2 = "World",
  var3 = var1 & " " & var2,
  var4 = Text.Upper(var3)
in
  var4
```

✓ No syntax errors have been detected.

- Expression inside in block is returned as let statement value



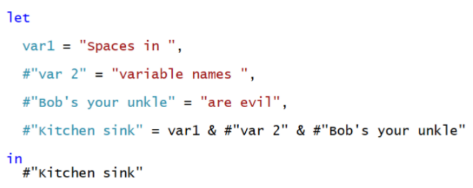
Comments and Variable names

Creating comments in the query editor allows you to document your query properly. When you comment a step, it allows you to not include it in your step.

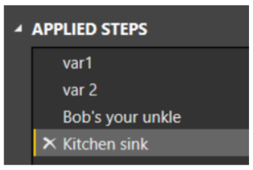
```
/*
  This is my most excellent query
*/
let

  var1 = 42, // the secret of life
```

Variable names with spaces must be enclosed in "#". These variables are created automatically by query designer.

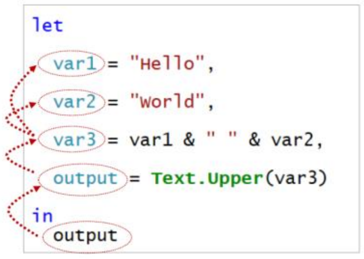


```
let
   var1 = "Spaces in ",
   #"var 2" = "variable names ",
   #"Bob's your unkle" = "are evil",
   #"Kitchen sink" = var1 & #"var 2" & #"Bob's your unkle"
in
   #"Kitchen sink"
```



Statement Evaluation

Since M language is a functional programming language, the order of the lines does not matter. This can be executed depending on where the expression is triggered.

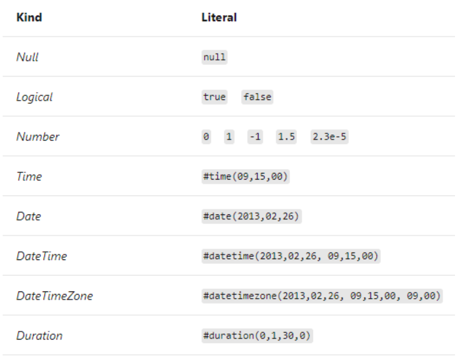


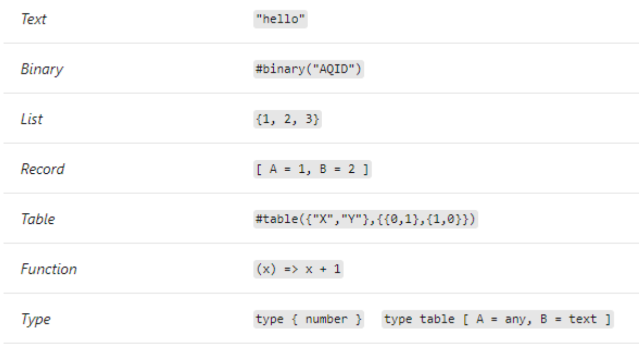
```
let
  var1 = "Hello",
  var2 = "World",
  var3 = var1 & " " & var2,
  output = Text.Upper(var3)
in
  output
```

Data Types

M language have different data types assigned to variables. M language is strict in its data type, which means it needs to assign the correct data type that will be used in the expression. There are different data types:

- **Primitive types**, which classify primitive values (binary, date, datetime, datetimezone, duration, list, logical, null, number, record, text, time, type) and also include a number of abstract types (function, table, any, and none).
- **Record types**, which classify record values based on field names and value types.
- **List types**, which classify lists using a single item base type.
- **Function types**, which classify function values based on the types of their parameters and return values
- **Table types**, which classify table values based on column names, column types, and keys
- **Nullable types**, which classifies the value null in addition to all the values classified by a base type
- **Type types**, which classify values that are types





Kind	**Literal**
Null	`null`
Logical	`true` `false`
Number	`0` `1` `-1` `1.5` `2.3e-5`
Time	`#time(09,15,00)`
Date	`#date(2013,02,26)`
DateTime	`#datetime(2013,02,26, 09,15,00)`
DateTimeZone	`#datetimezone(2013,02,26, 09,15,00, 09,00)`
Duration	`#duration(0,1,30,0)`
Text	`"hello"`
Binary	`#binary("AQID")`
List	`{1, 2, 3}`
Record	`[ A = 1, B = 2 ]`
Table	`#table({"X","Y"},{{0,1},{1,0}})`
Function	`(x) => x + 1`
Type	`type { number }` `type table [ A = any, B = text ]`

```
let

  // primitives
  var1 = 123,       // number
  var2 = true,      // boolean
  var3 = "hello",   // text
  var4 = null,      // null

  // creating lists
  list1 = {1, 2, 3},          // list of three numbers

  // accessing list elements
  var5 = list1{1},

  // create records
  record1 = [ FirstName="Soupy", LastName="Sales", ID=3 ],

  // accessing records
  var6 = record1[FirstName],

  // table
  table1 = #table( {"A", "B"}, { {1, 2}, {3, 4} } ),

  // creating function
  function1 = (x) => x * 2,

  // calling function
  output = function1(var1)

  in
    output
```

```
// time
var1 = #time(09,15,00),

// date
var2 = #date(2013,02,26),

// date and time
var3 = #datetime(2013,02,26, 09,15,00),

// date and time in specific timezone
var4 = #datetimezone(2013,02,26, 09,15,00, 09,00),

// time durection
var5 = #duration(0,1,30,0),
```

Operators

Just like DAX, M language has its own operators that it uses to perform the operations. It also follows an order of precedence when executed:

Types of operators arranged by precedence

- Primary – i, ( ), x[i],x{y}, x(..),{x,y,...},[i=x...]
- Unary - +x, -x, not x
- Metadata – x meta y
- Multiplicative - *,/
- Additive - +,-
- Relational - <, >, <, <=,>=
- Equality - =,<>
- Type assertion – x as y
- Type conformance – x is y
- Logical AND – x and y
- Logical OR – x or y

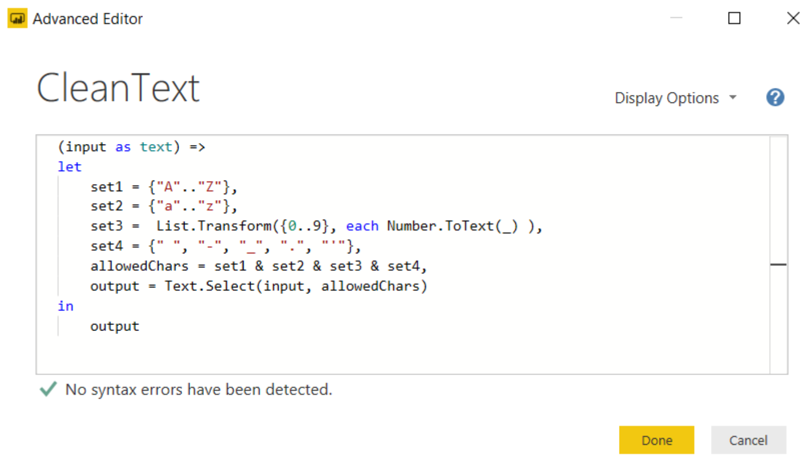
Conditional Statement

The ***if-expression*** selects from two expressions based on the value of a logical input value and evaluates only the selected expression.

`if` *if-condition* `then` *true-expression* `else` *false-expression*

Custom Function Queries

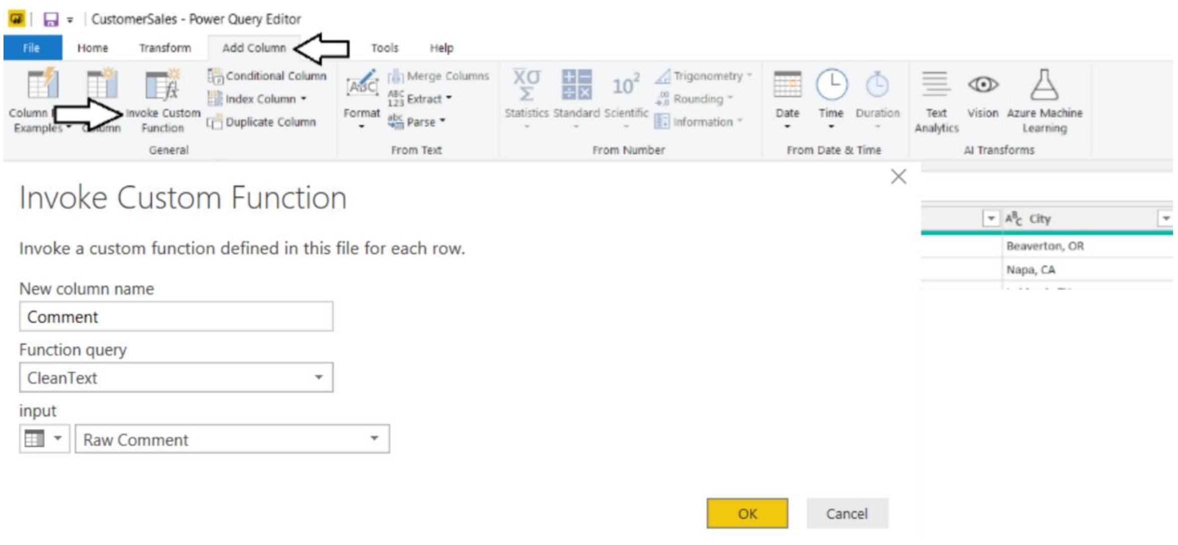
You can create your own functions in M language. This allows you to reuse operations to improve the performance of your query. We learned how to create custom functions in Power Query using the graphical interface. Using M Language you can create a more flexible function that you can use in your query.



```
(input as text) =>
let
    set1 = {"A".."Z"},
    set2 = {"a".."z"},
    set3 =  List.Transform({0..9}, each Number.ToText(_) ),
    set4 = {" ", "-", "_", ".", "'"},
    allowedChars = set1 & set2 & set3 & set4,
    output = Text.Select(input, allowedChars)
in
    output
```

✓ No syntax errors have been detected.

To use your function, you need to invoke it from the Add Column tab.



Invoke Custom Function

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

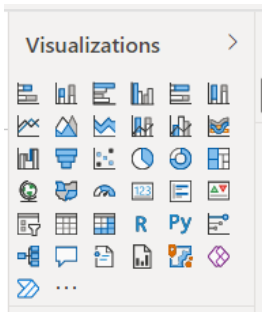
New column name: Comment

Function query: CleanText

input: Raw Comment

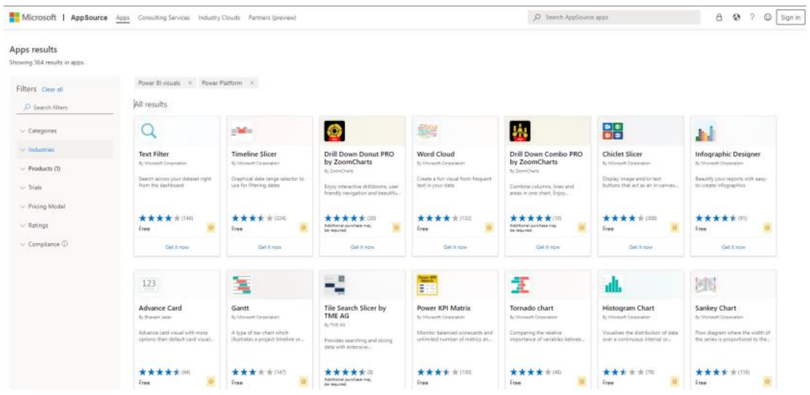
Chapter 6: Data Visualization

The last part of your PowerBI journey is the Data Visualization. Data Visualization brings your data to life. It should not just simplify your information, but to clarify them. Visualization gives meaning to your data. Charts and graphs summarize your data. Being able to see the story within the numbers makes data visualization a powerful tool for sharing and communicating information.



MS Office Store Visuals

PowerBI have built-in functions that you can use. You can also go to the Office Store to get custom visuals.



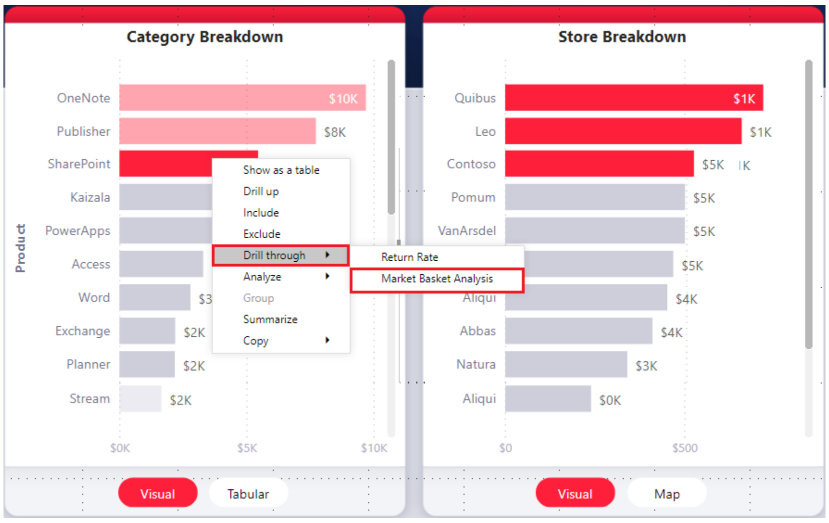
Each visualization type can be used to represent your data effectively.

Chart Type	When to use
Column	It is used to compare values across categories.
Line	This is used to display a trend over time
Pie	It is best used to compare parts of a whole.
Bar	It is best used to compare multiple value
Area	Used to emphasize the difference between several data points over a period of time.

Scatter	It is used to determine the relationship between two different parameters (X,Y).

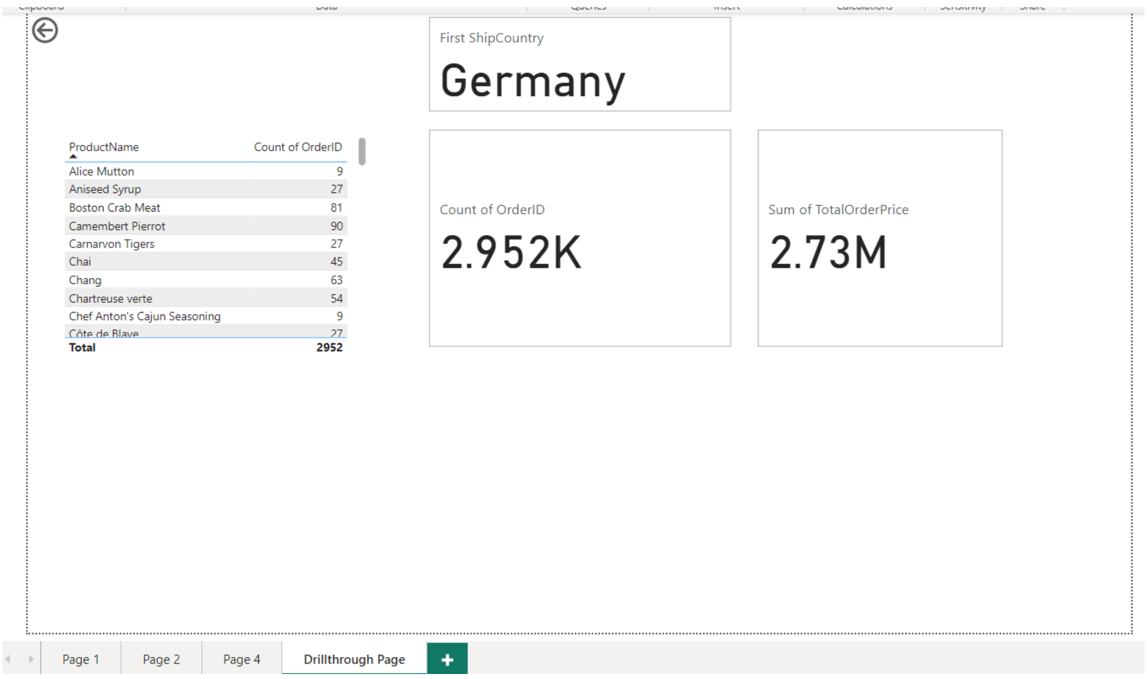
Drill Through in Visuals

With drill through in Power BI reports, you can create a page in your report that focuses on a specific entity. It allows you to go deeper into your data and see the breakdown of the data you set.



You need to setup a drill through page that has the visuals you want for the type of entity that you're going to provide drill through for.

Drillthrough Page



First ShipCountry

Germany

ProductName	Count of OrderID
Alice Mutton	9
Aniseed Syrup	27
Boston Crab Meat	81
Camembert Pierrot	90
Carnarvon Tigers	27
Chai	45
Chang	63
Chartreuse verte	54
Chef Anton's Cajun Seasoning	9
Côte de Blaye	27
Total	**2952**

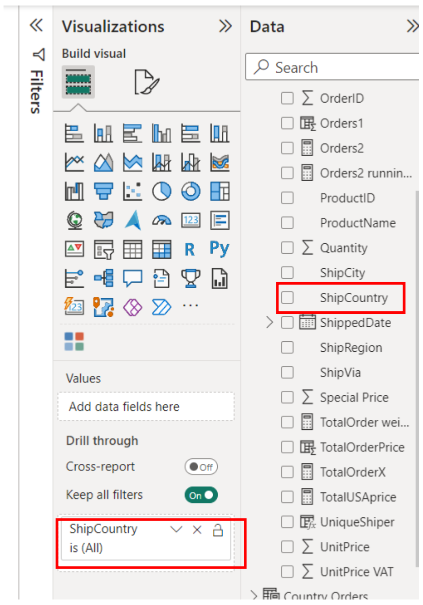
Count of OrderID

2.952K

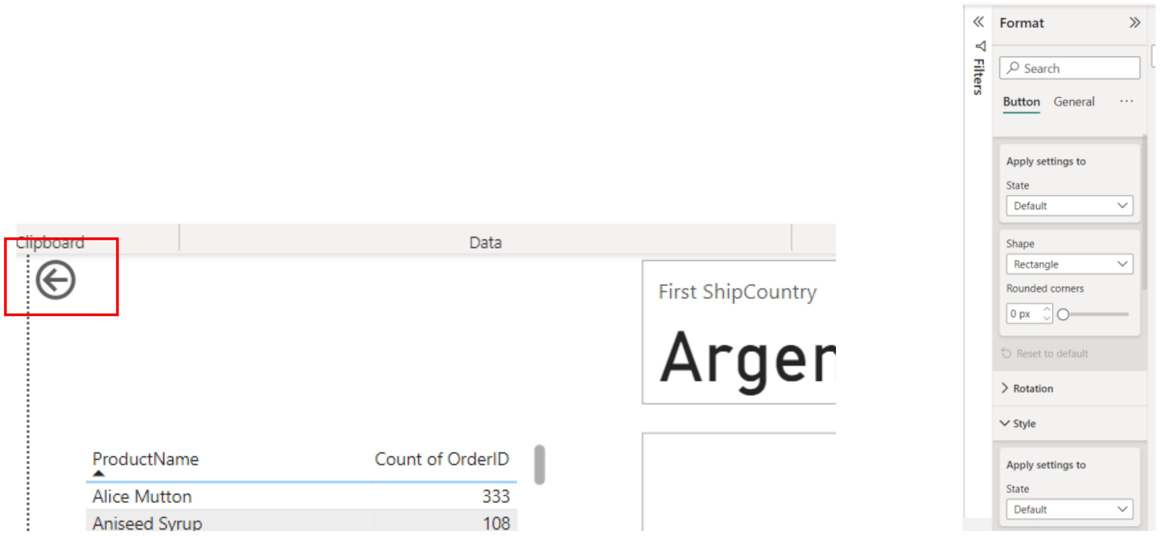
Sum of TotalOrderPrice

2.73M

Still in the drillthrouh page, in the field section of the Visualization pane, you can select the field that will enable the drillthrough. For this example, we selected **ShipCountry**.



When you add the field in the drillthrough, you will notice that a back button is added to the page.



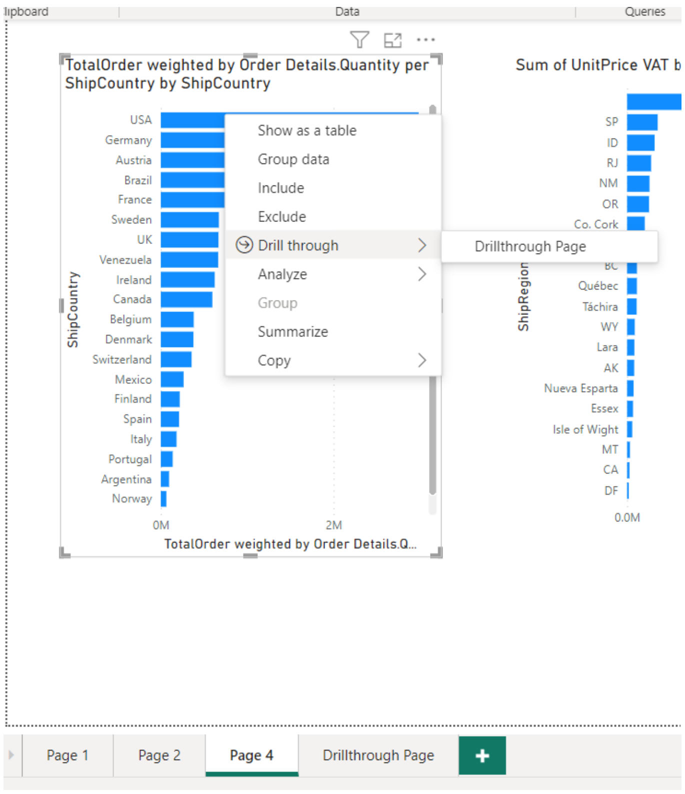
This allows users to go back to the previous page during drillthrough. This button will function when the dashboard is published. When you are still in PowerBI Desktop, you can enable it by pressing and holding the CTRL button.

You can set your own back image, and do some editing from the Format pane in the left navigation pane.

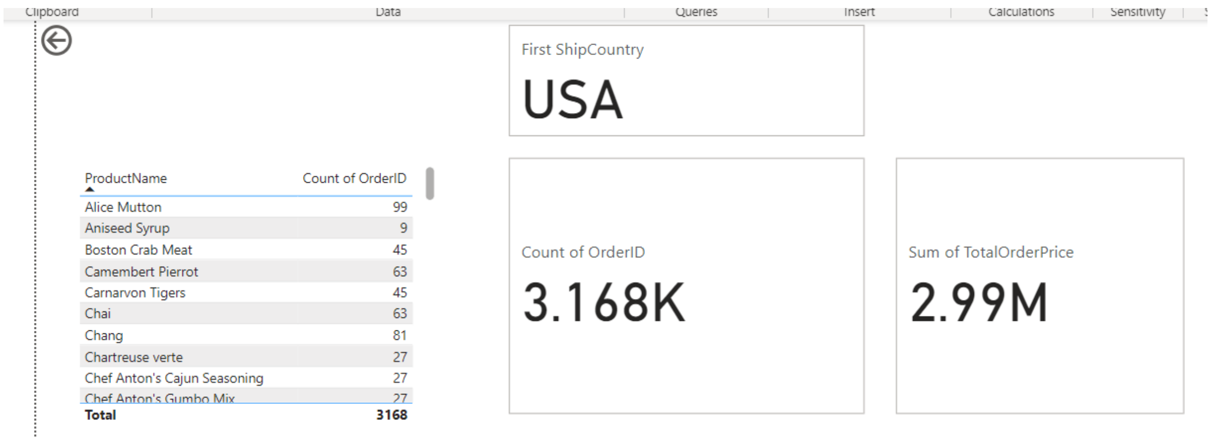
You can now use your drillthrough page by going to a page where the ShipCountry field is being used.

Right click the visualization and you will see a Drill through option. This will display the available drillthrough pages in your dashboard.

In this example, we drill through by country selecting USA.



This should display the information for USA only.



First ShipCountry

USA

ProductName	Count of OrderID
Alice Mutton	99
Aniseed Syrup	9
Boston Crab Meat	45
Camembert Pierrot	63
Carnarvon Tigers	45
Chai	63
Chang	81
Chartreuse verte	27
Chef Anton's Cajun Seasoning	27
Chef Anton's Gumbo Mix	27
Total	**3168**

Count of OrderID

3.168K

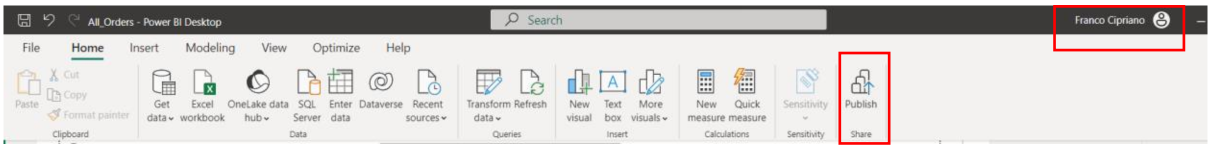
Sum of TotalOrderPrice

2.99M

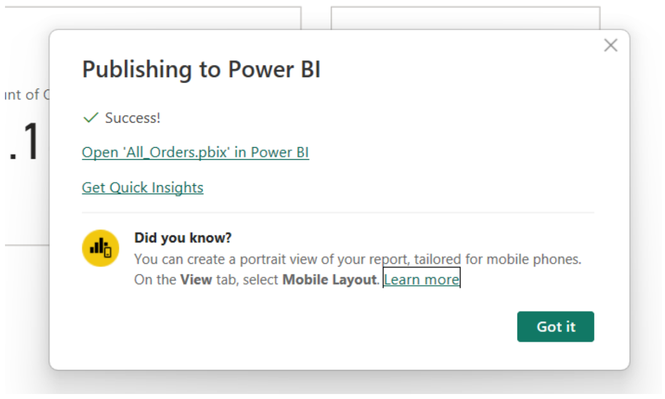
Drill through pages is very useful in enhancing your data visualization. You can setup an Overview page and have a drill through to see the more detailed information.

Chapter 7: Publishing your data

After completing your dashboard in your PowerBI Desktop, you are now ready to publish it to your team using the PowerBI Services. You need to be logged in to your Microsoft account to be able to share your dashboard.



Once published you will be notified that your dashboard is available.



Publishing to Power BI

✓ Success!

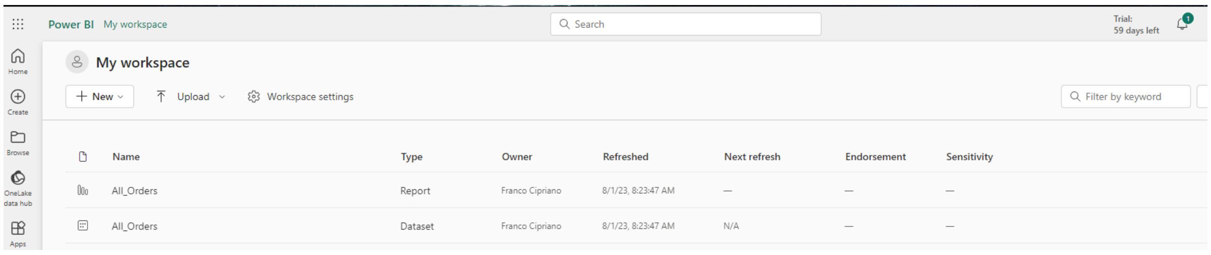
Open 'All_Orders.pbix' in Power BI

Get Quick Insights

Did you know?
You can create a portrait view of your report, tailored for mobile phones. On the **View** tab, select **Mobile Layout**. Learn more

Got it

You login to https://app.powerbi.com/ to check your published dashboards.



My workspace

Name	Type	Owner	Refreshed	Next refresh	Endorsement	Sensitivity
All_Orders	Report	Franco Cipriano	8/1/23, 8:23:47 AM	—	—	—
All_Orders	Dataset	Franco Cipriano	8/1/23, 8:23:47 AM	N/A	—	—

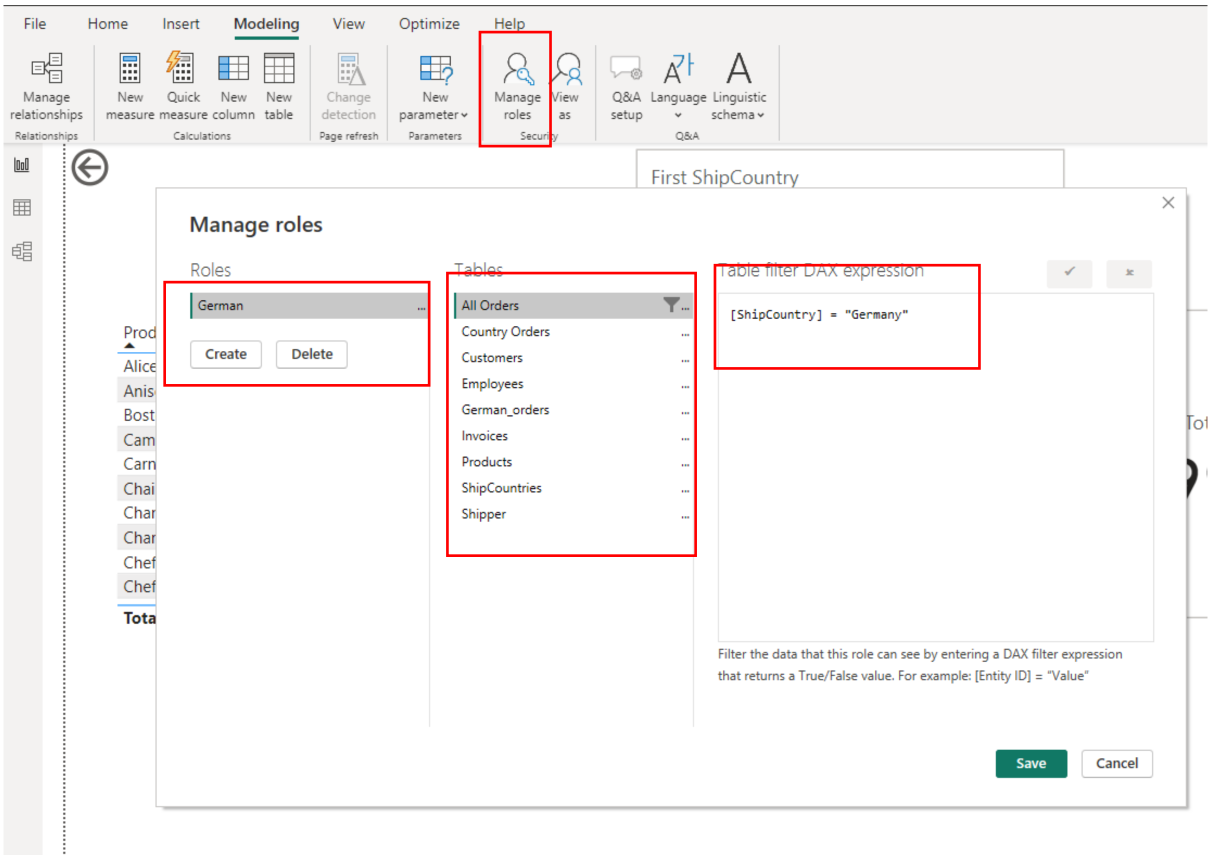
Row Level Security

RLS (Row Level Security) allows you to restrict data that can be given to users. Filters restrict data access at the row level, and you can define filters within roles.

Define Roles

You can create and manage roles from PowerBi Desktop. By going to the Modeling tab, you can select Manage Roles.

You need to select the table and apply the filter using DAX expression.



Manage roles

Roles: German

Create | Delete

Tables: All Orders, Country Orders, Customers, Employees, German_orders, Invoices, Products, ShipCountries, Shipper

Table filter DAX expression

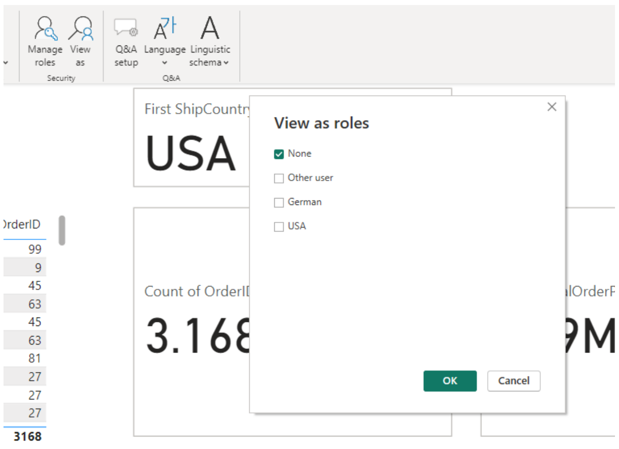
```
[ShipCountry] = "Germany"
```

Filter the data that this role can see by entering a DAX filter expression that returns a True/False value. For example: [Entity ID] = "Value"

Save | Cancel

Validating Your Roles

To test your roles created, you can click the **View as** command. This will give you a list of your roles and lets you select which role you want to view your dashboard as.



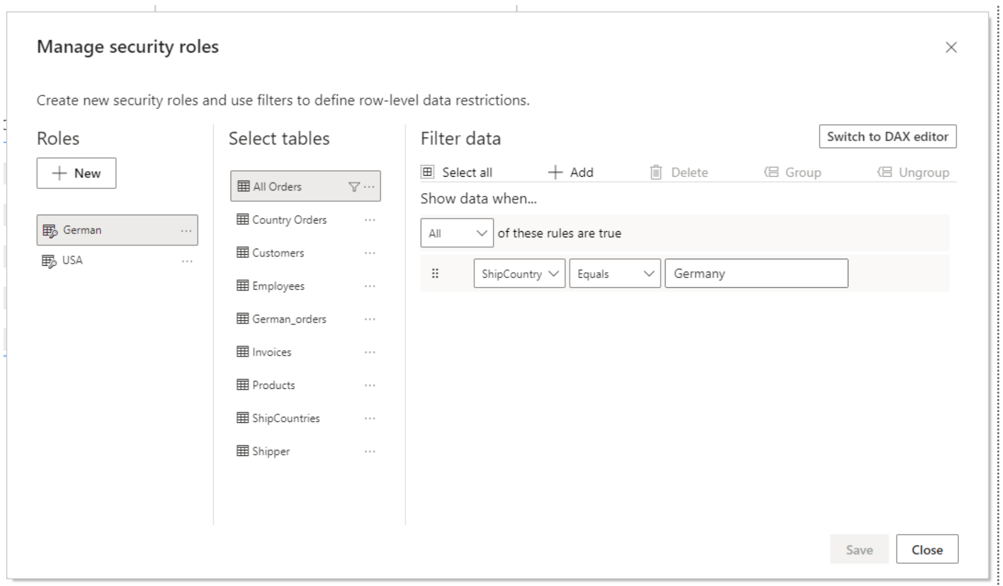
View as roles

- [✓] None
- [ ] Other user
- [ ] German
- [ ] USA

OK | Cancel

Enhanced Row-Level Security Editor

PowerBI gives us an easier way to create table filters. If you are not comfortable writing the DAX expression, you can use the RLS Editor, which gives you a user interface that lets you easily add filters.



Manage security roles

Create new security roles and use filters to define row-level data restrictions.

Roles: New; German; USA

Select tables: All Orders; Country Orders; Customers; Employees; German_orders; Invoices; Products; ShipCountries; Shipper

Filter data — Switch to DAX editor

Select all | Add | Delete | Group | Ungroup

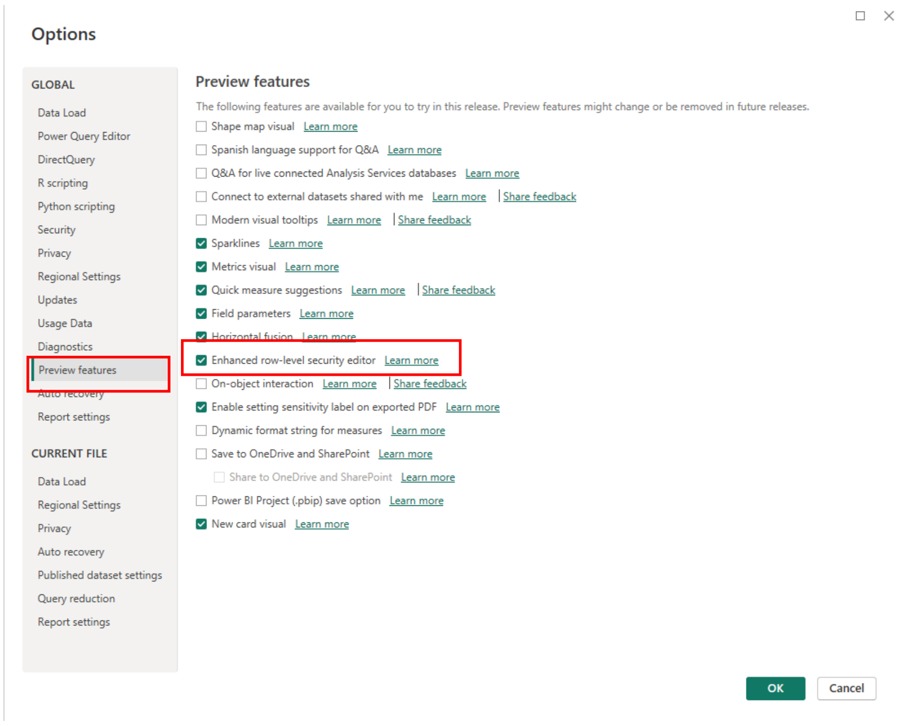
Show data when...

All of these rules are true

ShipCountry | Equals | Germany

Save | Close

To use this feature you need to Enable the preview by going to **Files > Options and Settings > Options > Preview features and turn on "Enhanced row-level security editor".**



Options

GLOBAL: Data Load; Power Query Editor; DirectQuery; R scripting; Python scripting; Security; Privacy; Regional Settings; Updates; Usage Data; Diagnostics; Preview features; Auto recovery; Report settings

CURRENT FILE: Data Load; Regional Settings; Privacy; Auto recovery; Published dataset settings; Query reduction; Report settings

Preview features

The following features are available for you to try in this release. Preview features might change or be removed in future releases.

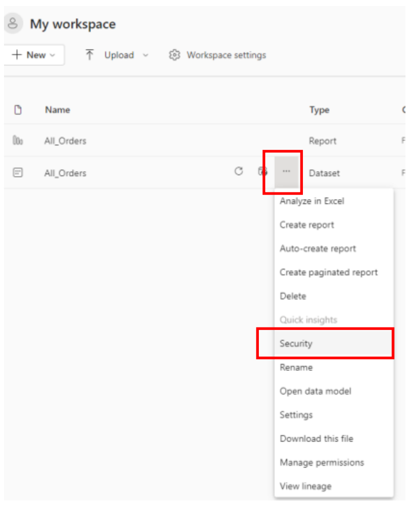
- ☐ Shape map visual Learn more
- ☐ Spanish language support for Q&A Learn more
- ☐ Q&A for live connected Analysis Services databases Learn more
- ☐ Connect to external datasets shared with me Learn more | Share feedback
- ☐ Modern visual tooltips Learn more | Share feedback
- ☑ Sparklines Learn more
- ☑ Metrics visual Learn more
- ☑ Quick measure suggestions Learn more | Share feedback
- ☑ Field parameters Learn more
- ☑ Horizontal fusion Learn more
- ☑ Enhanced row-level security editor Learn more
- ☐ On-object interaction Learn more | Share feedback
- ☑ Enable setting sensitivity label on exported PDF Learn more
- ☐ Dynamic format string for measures Learn more
- ☐ Save to OneDrive and SharePoint Learn more
  - ☐ Share to OneDrive and SharePoint Learn more
- ☐ Power BI Project (.pbip) save option Learn more
- ☑ New card visual Learn more

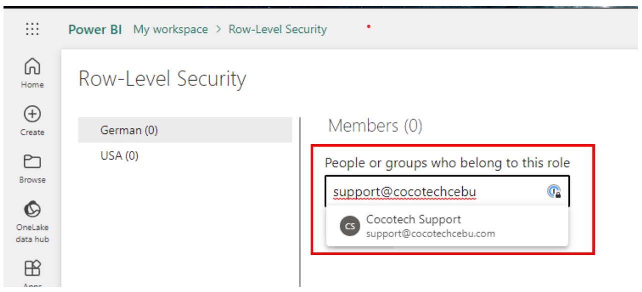
OK | Cancel

Managing Security in your Model

Once you publish your dashboard you can start sharing it to you organization. With RLS, you are able to restrict the users' role to whom you share your dashboard.

Go to your workspace and locate the dataset you want to set. Click the ellipsis icon on the left side of the Name column, and select **Security**.





You can select the Role and add the email of the user to that role, in this example we selected German and assign user support@cocotechcebu.com to German role.

Considerations and limitations

You can see the current limitations for row-level security on cloud models here:

- If you previously defined roles and rules in the Power BI service, you must re-create them in Power BI Desktop.
- You can define RLS only on the datasets created with Power BI Desktop. If you want to enable RLS for datasets created with Excel, you must convert your files into Power BI Desktop (PBIX) files first. Learn more.
- Service principals can't be added to an RLS role. Accordingly, RLS won't be applied for apps using a service principal as the final effective identity.
- Only Import and DirectQuery connections are supported. Live connections to Analysis Services are handled in the on-premises model.
- The Test as role/View as role feature doesn't work for DirectQuery models with single sign-on (SSO) enabled.