

Power BI Notes

Introduction

The **Home** tab contains **general settings** and **common table transformation tools**.

The **Transform** tab, contains tools to **modify existing columns** (splitting/grouping, transposing, extracting text, etc.)

The **Add Column** tab **creates new columns** (based on conditional rules, text operations, calculations, etc.)

Connecting & Shaping Data

Storage and connection modes

Power BI supports several types of **storage** and **connection modes**:

- **Import:** Tables are stored in-memory within Power BI, and queries are fulfilled by cached data (default).
 - Dataset is less than 1GB.
 - Storage data does not change frequently.
 - No restrictions on Power Query, data modelling, and DAX functions.
- **DirectQuery:** Tables are connected directly to the source, and queries are executed on-demand at the data source.
 - Dataset is too large to be stored in-memory.
 - Source data changes frequently, and reports must reflect changes.
 - Company policies states that data can only be accessed from the original source.
- **Composite Model (Dual):** Tables come from a mix of import and DirectQuery modes, or integrate multiple DirectQuery tables.
 - Boost performance by setting appropriate storage for each table.
 - Combine a DirectQuery model with additional imported data.
 - Create a single model from two or more DirectQuery models.
- **Live Connection:** Connects to a Pre-published Power BI datasets in Power BI Service or Azure Analysis Services.

- Create one dataset that serves as a central source of truth.
- Analyst teams can create different reports from the same source.
- Multi-developer teams where one user builds the model, and another works on visualization.

Note: Many tools can be accessed from both the **Transform** and **Add Column** menus, the difference is whether we want to **ADD** a new column, or **OVERWRITE** an existing one.

Date & Time Tools

They include:

- **Age:** Difference between the current date and the date in each row.
- **Date Only:** Removes the time component from a date/time field.
- **Year/Month/Quarter/Week/Day:** Extracts individual components from a date field (time-specific options include Hour, Minute, Second, etc.)
- **Earliest/Latest:** Evaluates the earliest or latest date from a column as a single value (can only be accessed from the “*Transform*” menu)

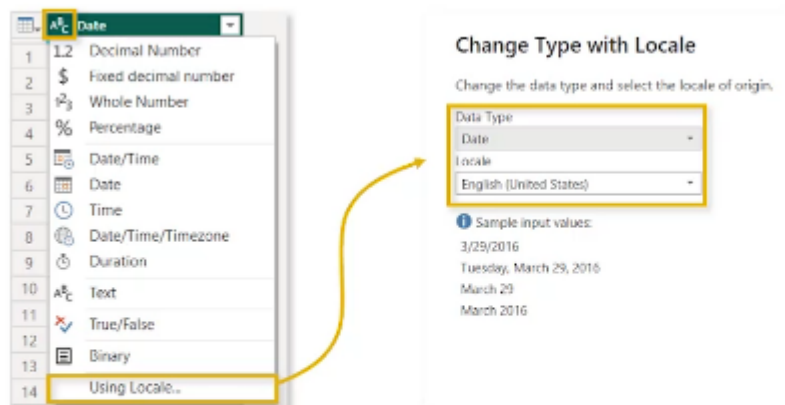
If we have a Start of Week step, and we want to set the start of the week to be Monday, we need to press the gear icon, and add an optional secondary parameter (which can be 1 for Monday):

```
Date.StartOfWeek([Date], 1)
```

Or we can also do:

```
Date.StartOfWeek([Date], Day.Monday)
```

Change type with Locale

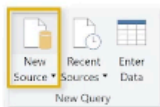


1) Left click the data type icon in the column header and select the **Using Locale** option

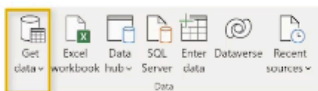
2) Select **Date** as the data type and **English (United States)** as the locale for all datasets in this course (regardless of your actual location)

Rolling calendars

1 Create a new **blank query** & name it **"Rolling Calendar"**

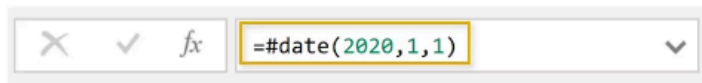


Power Query: New Source > Blank Query



Front end: Get Data > Blank Query

2 In the formula bar, type a **"literal"** to generate a start date:



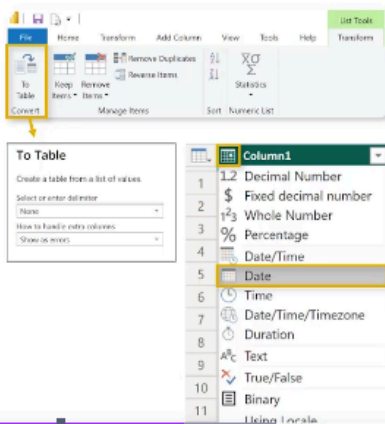
Format as: YYYY, MM, DD

3 Click the **fx** icon to **add a custom step**, and enter the following formula to generate a list of dates between the start date and the current day:

```
= List.Dates(
    Source,
    Number.From(DateTime.LocalNow()) - Number.From(Source),
    #duration(1, 0, 0, 0)
)
```

Note: If your first applied step is named something other than **"Source"**, use that name in your formula (this is common for non-US users)

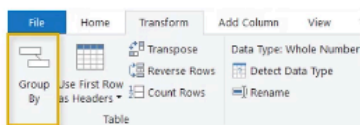
4 Convert the resulting list into a **Table** and set the data type as a **Date**



5 Rename the column to **"Date"** and add calculated date columns (year, month, quarter, etc.) using the **Add Column** tools

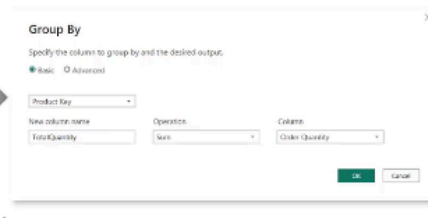
	Date	Year	Start of Quarter	Start of Month
1	1/1/2020	2020	1/1/2020	1/1/2020
2	1/2/2020	2020	1/1/2020	1/1/2020
3	1/3/2020	2020	1/1/2020	1/1/2020
4	1/4/2020	2020	1/1/2020	1/1/2020
5	1/5/2020	2020	1/1/2020	1/1/2020
6	1/6/2020	2020	1/1/2020	1/1/2020
7	1/7/2020	2020	1/1/2020	1/1/2020
8	1/8/2020	2020	1/1/2020	1/1/2020
9	1/9/2020	2020	1/1/2020	1/1/2020
10	1/10/2020	2020	1/1/2020	1/1/2020
11	1/11/2020	2020	1/1/2020	1/1/2020
12	1/12/2020	2020	1/1/2020	1/1/2020
13	1/13/2020	2020	1/1/2020	1/1/2020
14	1/14/2020	2020	1/1/2020	1/1/2020
15	1/15/2020	2020	1/1/2020	1/1/2020
16	1/16/2020	2020	1/1/2020	1/1/2020
17	1/17/2020	2020	1/1/2020	1/1/2020
18	1/18/2020	2020	1/1/2020	1/1/2020
19	1/19/2020	2020	1/1/2020	1/1/2020
20	1/20/2020	2020	1/1/2020	1/1/2020
21	1/21/2020	2020	1/1/2020	1/1/2020

Grouping & Aggregating



Group By allows you to aggregate data at a different level or "grain" (i.e. group daily records into monthly, aggregate transactions by store, etc.)

	Order Date	Product Key	Customer Key	Order Quantity
1	6/25/2013	214	34729	1
2	6/26/2013	214	31530	1
3	12/30/2013	214	27898	1
4	6/26/2013	214	22198	1
5	6/16/2013	214	27621	1
6	2/5/2013	214	21435	1
7	6/9/2013	214	34851	1
8	6/10/2013	214	21813	1
9	6/20/2013	214	27438	1
10	3/15/2013	214	24236	1
11	6/25/2013	214	28292	1
12	6/13/2013	214	32721	1
13	6/25/2013	214	33636	1
14	7/25/2013	214	22640	1
15	11/24/2013	214	22833	1
16	8/2/2013	214	20582	1
17	10/12/2013	214	20612	1
18	6/16/2013	214	35027	1
19	10/22/2013	214	20642	1
20	8/13/2013	214	31696	1



	Product Key	TotalQuantity
1	214	2020
2	214	1940
3	214	1995
4	214	2111
5	214	880
6	229	400
7	230	424
8	245	881
9	810	160
10	811	199
11	312	170
12	313	100
13	818	157
14	820	65
15	822	89
16	324	72
17	326	85

Here we're transforming a daily, transaction-level table into a summary of **Total Quantity** by **Product Key**

NOTE: Any fields not specified in the Group By settings are lost

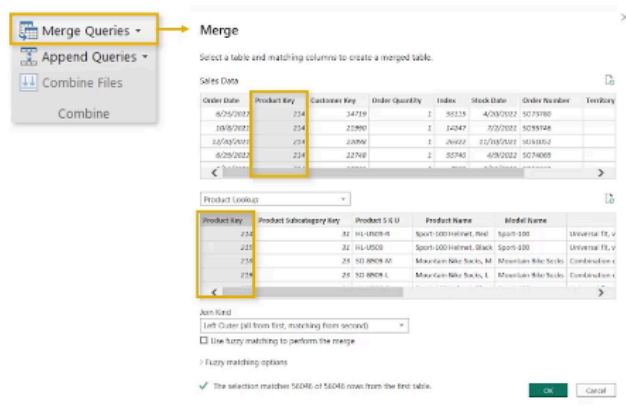
Pivoting & Unpivoting

Pivoting describes the process of turning **distinct row values into columns**, and **unpivoting** describes the process of turning **distinct columns into rows**.

Date	Product Category	North Region	Central Region	South Region
7/1/2022	Bikes	10	19	25
7/1/2022	Components	14	32	16
7/1/2022	Clothing	35	32	46

Date	Product Category	Region	Quantity Sold
7/1/2022	Bikes	North Region	10
7/1/2022	Bikes	Central Region	19
7/1/2022	Bikes	South Region	25
7/1/2022	Components	North Region	14
7/1/2022	Components	Central Region	32
7/1/2022	Components	South Region	16
7/1/2022	Clothing	North Region	35
7/1/2022	Clothing	Central Region	32
7/1/2022	Clothing	South Region	46

Merging & Appending Queries



Merging queries allows you to **join tables** based on a common column (like a lookup in Excel)

In this case we're merging the **Sales Data** table with the **Product Lookup** table, which share a common **Product Key** column

NOTE: Merging **adds columns** to an existing table/query

HEY THIS IS IMPORTANT!

Just because you can merge tables, doesn't mean you should!

In many cases, it's better to keep tables separate and define **relationships** between them in the data model (*more on that soon!*)

We have two options:

- **Merge Queries:** Will append the other columns to the current query, preserving the items that it originally had.
- **Merge Queries as New:** Will create a new query with only the original, and merged queries

After that, we can expand the table using the arrows icon on the right.

And appending allows us to **combine** or **stack** tables sharing the exact same column structure and data types. Grouping makes the table wider, whereas appending makes it taller.

Appending makes a query dependant on the queries that it used to create the table, to avoid this dependencies on other queries, and having redundant data, the files can be added to a folder, and the folder can be loaded.

Data source parameters

Use **parameters** to dynamically manage and update connection paths in the Power Query editor

The image shows two screenshots from the Power Query Editor. The left screenshot displays the 'Manage Parameters' dialog box. It has a 'Name' field (labeled 'Parameter name (Name of the query/table)'), a 'Type' dropdown (labeled 'Parameter type (Any value, text, date, etc.)'), a 'List of values' field (labeled 'Parameter value (Any value, list, query)'), and 'Default Value' and 'Current Value' dropdowns (labeled 'Parameter type (Default & current)'). The right screenshot shows the 'MySQL database' configuration window. It has a 'Server' dropdown (labeled 'Server (fuzzy factory)') and a 'Text' dropdown (labeled 'Parameter (fuzzy factory)'). Below these, it says 'Update Server & Database connection text values with parameters'.

That can be used to switch from a development to a production environment, for example.

Refreshing Queries

The image shows two screenshots from the Power Query Editor. The left screenshot shows the 'Home' tab ribbon with the 'Refresh' button highlighted. Below it, text says: 'By default, **all queries** will refresh when you use the **Refresh** command from the **Home** tab'. The right screenshot shows the 'Queries' pane with a context menu open for 'Customer Lookup'. The 'Include in report refresh' checkbox is highlighted. Below it, text says: 'From the Query Editor, uncheck **Include in report refresh** to exclude individual queries from the refresh'. At the bottom, a 'PRO TIP' box says: 'PRO TIP: Exclude queries from refresh that don't change often (like lookups or static data tables)'.

Best practices:

★ Get organized before connecting and loading data

- Define clear and intuitive table/query names from the start, and establish an organized file/folder structure if you are working with local flat files to avoid changes to file names or paths

★ Disable report refresh for any static data sources

- There's no need to constantly refresh data sources that don't change, like lookups or static data tables

★ When working with large tables, only load the data you need

- Don't include hourly data when you only need daily, or transaction-level data when only need a product-level summary (extra data will only slow your report down!)

Creating a Data Model

Fact & Dimension Tables

Data models generally contain two types: **fact** (*data*) tables, and **dimension** (*lookup*) tables:

- **Fact tables** contain **numerical values** or **metrics** used for summarization (*sales, orders, transactions, pageviews, etc.*)
- **Dimension tables** contain **descriptive attributes** used for filtering or grouping (*products, customers, dates, stores, etc.*)

date	product_id	quantity
1/1/1997	809	5
1/1/1997	1477	3
1/1/1997	76	4
1/1/1997	320	3
1/1/1997	4	4
1/1/1997	952	4
1/1/1997	1222	4
1/1/1997	517	4
1/1/1997	1359	4
1/1/1997	357	4
1/1/1997	1426	5
1/1/1997	190	4
1/1/1997	367	4
1/1/1997	250	5
1/1/1997	600	4
1/1/1997	702	5

This **Fact** table contains **quantity** values, along with **date** and **product_id** fields

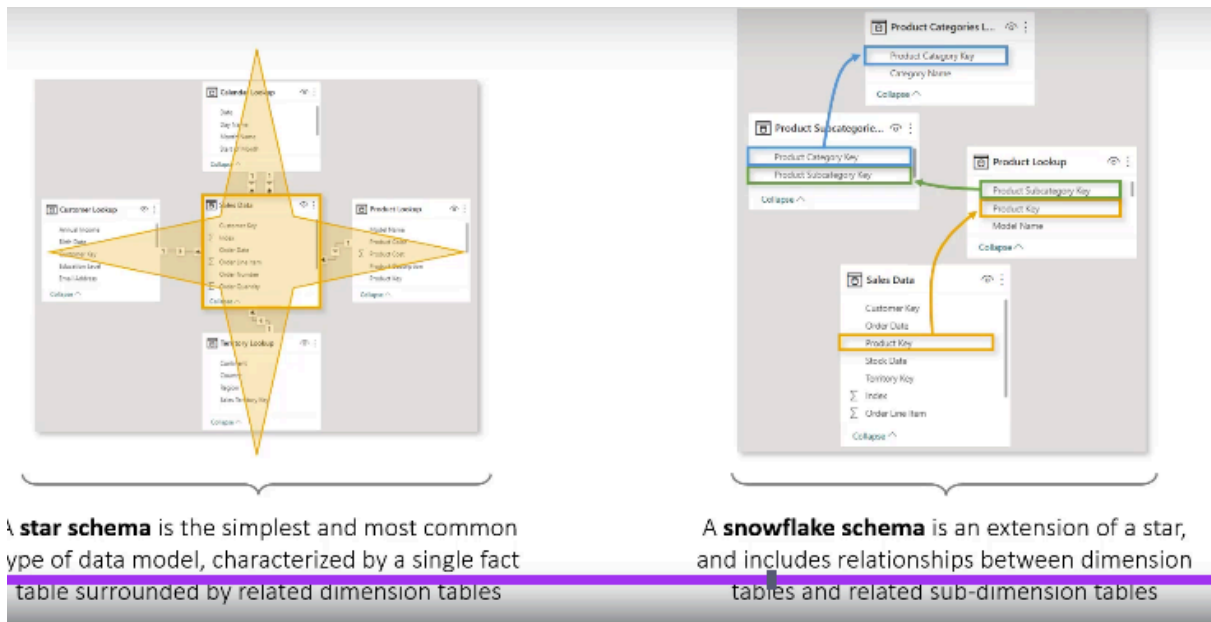
date	day_of_month	month	year	weekday	week_of_year	week_ending	month_name	quarter
1/1/1997	1	1	1997	Wednesday	1	1/5/1997	January	Q1
1/2/1997	2	1	1997	Thursday	1	1/5/1997	January	Q1
1/3/1997	3	1	1997	Friday	1	1/5/1997	January	Q1
1/4/1997	4	1	1997	Saturday	1	1/5/1997	January	Q1
1/5/1997	5	1	1997	Sunday	2	1/5/1997	January	Q1
1/6/1997	6	1	1997	Monday	2	1/12/1997	January	Q1

This **Calendar Lookup** table contains attributes about each **date** (month, year, quarter, etc.)

product_id	product_brand	product_name	product_sku	product retail price	product cost	product weight
1	Washington	Washington Berry Juice	90026586676	2.85	0.54	8.93
2	Washington	Washington Mango Drink	9000600099	0.25	0.26	7.42
3	Washington	Washington Strawberry Drink	5802221905	0.88	0.4	14.1
4	Washington	Washington Green Soda	65012155267	3.65	1.64	10.6
5	Washington	Washington Beer Soda	85540199489	2.19	0.22	6.99
6	Washington	Washington Cola	2980042296	5.15	0.27	15.8
7	Washington	Washington Diet Cola	20191666756	2.61	0.64	18
8	Washington	Washington Orange Juice	8977020250	2.59	0.8	8.92

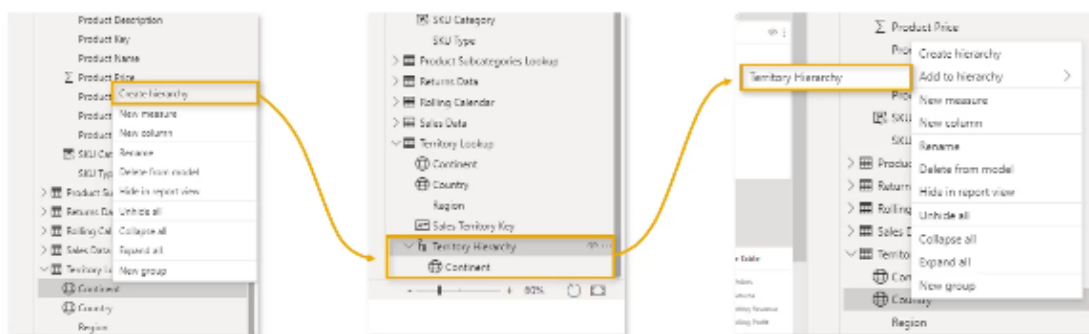
This **Product Lookup** table contains attributes about each **product_id** (brand, SKU, price, etc.)

Star & Snowflake schemas



Hierarchies

Hierarchies are groups of columns that reflect multiple levels of granularity. For example, a **Geographic hierarchy** might include **Country**, **state**, and **city** fields. They are treated as a **single item** in tables and reports, allowing users to “*drill up*” and “*drill down*” through each level.



In the **Data** pane, right-click a field and select **Create hierarchy**

This hierarchy contains “Continent”, and is named “**Territory Hierarchy**”

Right-click another field (like “Country”) and select **Add to Hierarchy** (or drag it in!)

Data Model Best Practices



Focus on building a normalized model from the start

- Leverage relationships and make sure that each table serves a clear, distinct purpose



Organize dimension tables above data tables in your model

- This serves as a visual reminder that filters always flow “downstream”



Avoid complex relationships unless absolutely necessary

- Aim to use 1-to-many table relationships and one-way filters whenever possible



Hide fields from report view to prevent invalid filter context

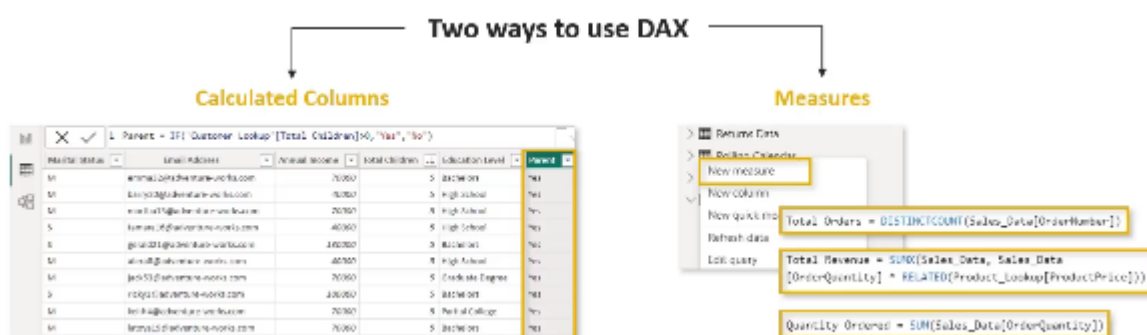
- This forces report users to filter using primary keys from dimension tables

Calculated Fields with DAX

Data Analysis Expressions (DAX)

Data Analysis Expressions (commonly known as **DAX**) is the formula language that drives the Power BI front-end. With DAX, we can:

- Go beyond the capabilities of traditional spreadsheets formulas, with powerful and flexible functions built specifically to work with relational data models.
- Add **calculated columns** (for filtering) and **measures** (for aggregation) to enhance data models.




M and **DAX** are two distinct functional languages used within Power BI Desktop. **M** is used in the Power Query editor, and is designed specifically for extracting, transforming and loading data. Whereas **DAX** is used in the Power BI front-end, and is designed specifically for analysing relational data models.

Calculated Columns

Calculated columns allow you to add new, formula-based columns to tables in a model.

- They refer to **entire tables** or **columns** (no *A1-style* references.)
- Calculated columns **generate values for each row**, which are visible within tables in the Data view.
- Calculated columns understand **row context**; they are great for defining properties based on information in each row, but generally useless for aggregation (sum, count, etc.)

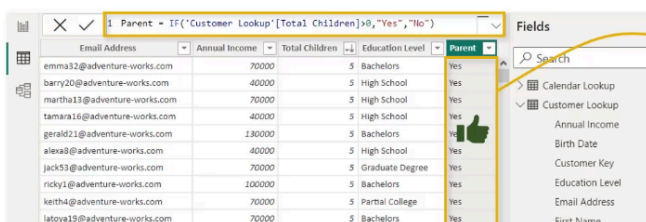


HEY THIS IS IMPORTANT!

As a rule of thumb, use calculated columns to “stamp” static, fixed values to each row in a table (*or go upstream and use the Query Editor!*)

DO NOT use calculated columns for aggregation – this is what **measures** are for!

For example:



Email Address	Annual Income	Total Children	Education Level	Parent
emma32@adventure-works.com	70000	5	Bachelors	Yes
barry20@adventure-works.com	40000	5	High School	Yes
martha13@adventure-works.com	70000	5	High School	Yes
tamara16@adventure-works.com	40000	5	High School	Yes
gerald21@adventure-works.com	130000	5	Bachelors	Yes
alexa8@adventure-works.com	40000	5	High School	Yes
jack53@adventure-works.com	70000	5	Graduate Degree	Yes
ricky1@adventure-works.com	100000	5	Bachelors	Yes
keith4@adventure-works.com	70000	5	Partial College	Yes
latoya19@adventure-works.com	70000	5	Bachelors	Yes

In this case we've added a **calculated column** named **Parent**, which equals “**Yes**” if the [Total Children] field is greater than 0, and “**No**” otherwise

- Since calculated columns understand **row context**, a new value is calculated in each row based on the value in the [Total Children] column
- This is a **valid use** of calculated columns; it creates a new row “property” that we can use to filter or segment any related data within the model

For example:

```
Quantity Type = IF (
    'Sales Data'[OrderQuantity] > 1,
    "Multiple Items",
```

"Single Item"

)

That will create a column that will contain *Multiple Items* if OrderQuantity is greater than one, or *Single Item* otherwise.

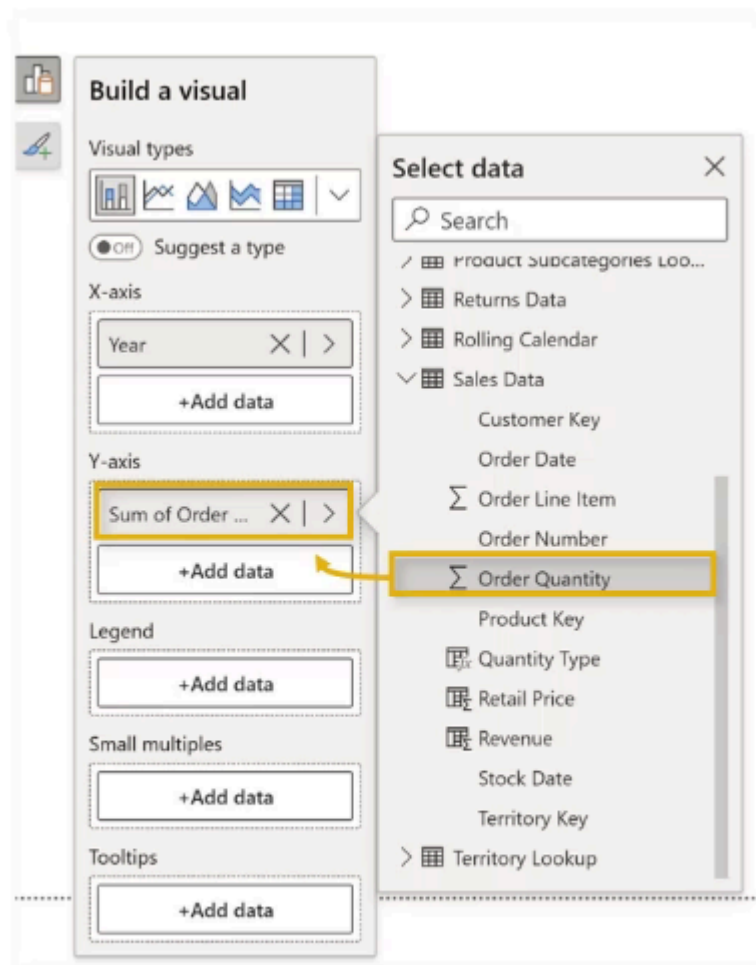
DAX Measures

Measures are DAX formulas used to generate new calculated values.

- They also reference **entire tables** or **columns** (no *A1-style* cell references)
- Unlike calculated columns, **measures** aren't visible within tables; they can only be "seen" within a visualization like a chart or a matrix (similar to a calculated field in a PivotTable)
- Measures evaluate based on **filter context**, which means they recalculate when the fields or filters around them change.

Implicit vs Explicit Measures

Implicit measures are created when we drag raw numerical fields into a report visual and manually select an aggregation mode (Sum, Average, Min, Max, Count, etc):



Example of an **implicit measure**

Whereas the **explicit measures** are created when we actually write a DAX formula and define a new measure that can be used within the model.



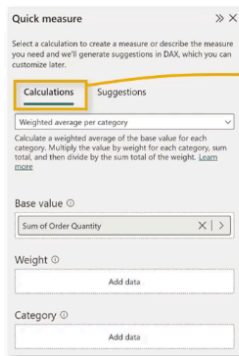
HEY THIS IS IMPORTANT!

Implicit measures are only accessible within the **specific visualization** in which they were created, and cannot be referenced elsewhere

Explicit measures can be used **anywhere in the report**, and referenced by other DAX calculations to create “measure trees”

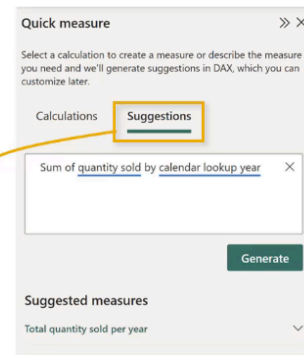
Quick Measures

Quick measures automatically create formulas based on pre-built templates or natural language prompts



Quick measure **calculations** can be used to build measures using **predefined templates** (*weighted averages, percent difference, time intelligence, etc.*)

Quick measure **suggestions** can be used to find suggested measures based on **natural language queries** (i.e. "sum of quantity sold by calendar year")



Calculated columns vs Measures

CALCULATED COLUMNS

- Values are calculated based on information from each row of a table (**row context**)
- Appends static values to each row in a table and stores them in the model (*which increases file size*)
- Recalculate on data source refresh or when changes are made to component columns
- Primarily used for **filtering** data in reports

Calculated columns "live" in **tables**

MEASURES

- Values are calculated based on information from any filters in the report (**filter context**)
- Does not create new data in the tables themselves (*doesn't increase file size*)
- Recalculate in response to any change to filters within the report
- Primarily used for **aggregating values** in report visuals

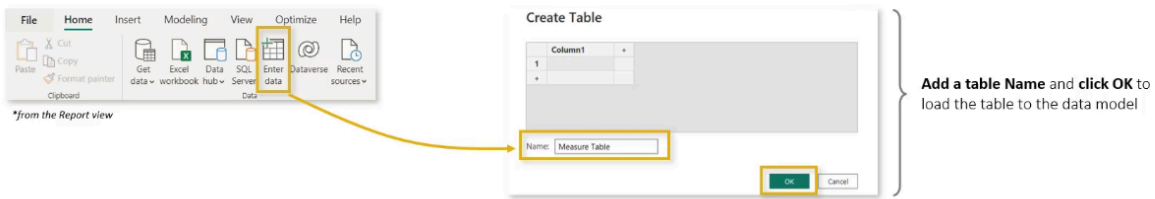


Measures "live" in **visuals**

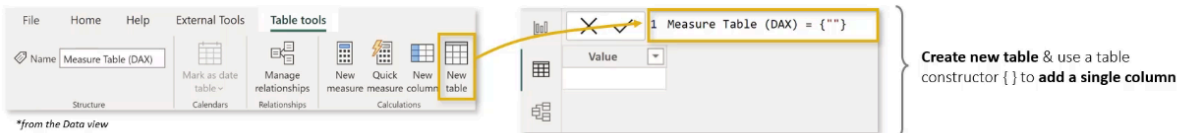
Measure Tables

It's a common best practice to **create a dedicated table to store your measures**; this will help you stay organised, find measures quickly, and allow you to group related measures into folders. We can create them in two ways:

Option 1: Enter Data into Power Query (loads the table to the data model – table is visible in Power Query)



Option 2: Create a calculated table using DAX directly in the model (table is not visible in Power Query)

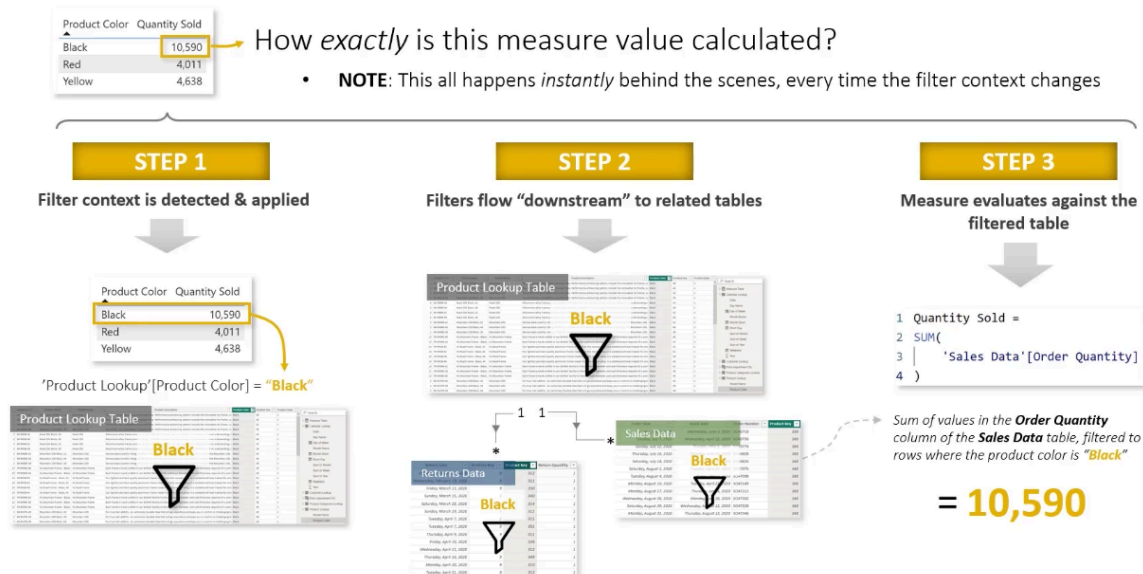


To create a blank table, we can write the DAX:

Table Name = { \"\" }

Which uses table constructors to basically create a blank table.

Measure Calculation



DAX Syntax

Consider the following piece of DAX:

Total Quantity: = SUM(Transactions[quantity])

In **grey** we have *Measure Name*. They are always surrounded by brackets (i.e. **[Total Quantity]**) when referenced in formulas, so spaces are OK.

The component immediately following the equals sign (the one in **yellow**) is the *function name*. Calculated columns don't always use functions but measures do. In a measure column, we couldn't do something like = *Transactions[quantity]* but we can in a calculated column, as it's evaluated row by row. In a measure, Power BI wouldn't know how to convert that into a single value.

The table name is in **orange**, and the **blue** is the column name. If we were referencing a different table with spaces, we'd need to enclose it within quotes, so we'd end up with something like:

Total Quantity: = SUM ("Transactions Table"[quantity])

The following are the DAX operators:

Arithmetic Operator	Meaning	Example
+	Addition	2 + 7
-	Subtraction	5 - 3
*	Multiplication	2 * 6
/	Division	4 / 2
^	Exponent	2 ^ 5

Comparison operators:

Comparison Operator	Meaning	Example
=	Equal to	[City] = "Boston"
>	Greater than	[Quantity] > 10
<	Less than	[Quantity] < 10
≥	Greater than or equal	[Unit Price] ≥ 2.5
≤	Less than or equal to	[Unit Price] ≤ 2.5
<>	Not equal to	[Country] <> "Mexico"

Text operators and logical operators:

Text/Logical Operator	Meaning	Example

&	Concatenates two values to produce one string	[City] & “ “ & [State]
&&	Creates an AND condition between two logical expressions	([State]=”MA”) && ([Quantity] > 10)
	Creates an OR condition between two logical expressions	([State]=”MA”) ([State]=”CT”)
IN	Creates a logical OR condition based on a given list (using curly brackets)	“Store Lookup”[State] IN {”MA”, “CT”, “NY”}

Common Function Categories

MATH & STATS Functions	LOGICAL Functions	TEXT Functions	FILTER Functions	TABLE Functions	DATE & TIME Functions	RELATIONSHIP Functions
Functions used for aggregation or iterative, row-level calculations	Functions that use conditional expressions (IF/THEN statements)	Functions used to manipulate text strings or value formats	Functions used to manipulate table and filter contexts	Functions that create or manipulate tables and output tables vs. scalar values	Functions used to manipulate date & time values or handle time intelligence calculations	Functions used to manage & modify table relationships
Common Examples: <ul style="list-style-type: none"> SUM AVERAGE MAX/MIN DIVIDE COUNT/COUNTA COUNTROWS DISTINCTCOUNT Iterator Functions: <ul style="list-style-type: none"> SUMX AVERAGEX MAXX/MINX RANKX COUNTX 	Common Examples: <ul style="list-style-type: none"> IF IFERROR AND OR NOT SWITCH TRUE FALSE 	Common Examples: <ul style="list-style-type: none"> CONCATENATE COMBINEVALUES FORMAT LEFT/MID/RIGHT UPPER/LOWER LEN SEARCH/FIND REPLACE SUBSTITUTE TRIM 	Common Examples: <ul style="list-style-type: none"> CALCULATE FILTER ALL ALLEXCEPT ALLSELECTED KEEPFILTERS REMOVEFILTERS SELECTEDVALUE 	Common Examples: <ul style="list-style-type: none"> SUMMARIZE ADDCOLUMNS GENERATESERIES DISTINCT VALUES UNION INTERSECT TOPN 	Common Examples: <ul style="list-style-type: none"> DATE DATEDIFF YEARFRAC YEAR/MONTH DAY/HOUR TODAY/NOW WEEKDAY WEEKNUM NETWORKDAYS Time Intelligence: <ul style="list-style-type: none"> DATESYTD DATESMTD DATEADD DATESBETWEEN 	Common Examples: <ul style="list-style-type: none"> RELATED RELATEDTABLE CROSSFILTER USERELATIONSHIP

Basic maths & stats functions

- **SUM:** Evaluates the sum of a column: $=SUM(ColumnName)$
- **AVERAGE:** Returns the average (arithmetic mean) of all the numbers in a column: $=AVERAGE(ColumnName)$
- **MAX:** Returns the largest value in a column or between two scalar expressions: $=MAX(ColumnNameOrScalar1, [Scalar2])$
- **MIN:** Returns the smallest value in a column or between two scalar expressions: $=MIN(ColumnNameOrScalar1, [Scalar2])$
- **DIVIDE:** Performs division and returns the alternate result (or blank) if DIV/0: $=DIVIDE(Numerator, Denominator, [AlternateResult])$

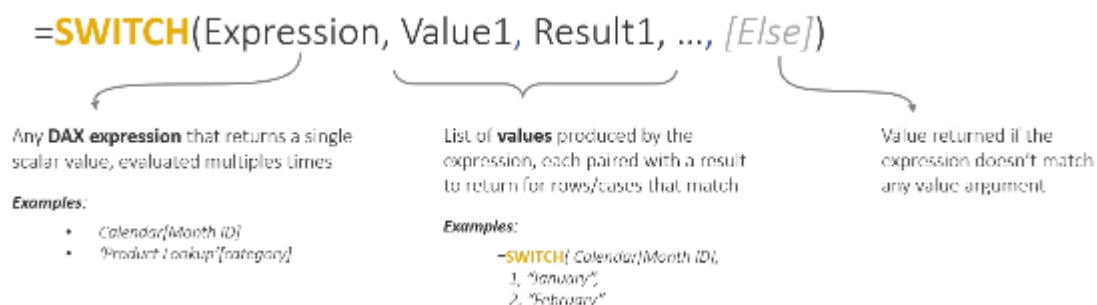
Count Functions

- **COUNT:** Counts the number of non-empty cells in a column (excluding Boolean values), can only be used with numbers. `=COUNT (ColumnName)`
- **COUNTA:** Counts the number of non-empty cells in a column (excluding Boolean values), can be used with both numbers, and text: `=COUNTA (ColumnName)`
- **DISTINCTCOUNT:** Counts the number of distinct values in a column: `=DISTINCTCOUNT (ColumnName)`
- **COUNTROWS:** Counts the number of rows in the specified table, or a table defined by an expression: `=COUNTROWS ([Table])`
 - **Note:** The definition of *Table* in Power BI is either a column, or an entire table.

Conditional & Logical Functions

- **IF:** Checks if a given condition is met and returns one value if the condition is *TRUE* and another if the condition is *FALSE*: `=IF (LogicalTest, ResultIfTrue, [ResultIfFalse])`
- **IFERROR:** Evaluates an expression and returns a specified value if it returns an error, otherwise returns the expression itself: `=IFERROR (Value, ValueIfError)`
- **SWITCH:** Evaluates an expression against a list of values and returns one of multiple possible expressions. `=SWITCH (Expression, Value1, Result1, ..., [Else])`
- **AND:** Checks whether both arguments are TRUE to return TRUE, otherwise returns FALSE. `=AND (Logical1, Logical2)`
- **OR:** Checks if at least one argument is TRUE to return TRUE, otherwise returns FALSE. `=OR (Logical1, Logical2)`

The switch function



Switch is used mainly to test for equality, if we wanted to use other operators, we could do:

```
Price Point = SWITCH (
    TRUE (),
    'Product Lookup'[ProductPrice] > 500, "High",
    'Product Lookup'[ProductPrice] > 100, "Mid-Ran
ge",
    "Low"
)
```

Text Functions

- **LEN:** Returns the number of characters in a string: `=LEN (Text)`
- **CONCATENATE:** Joins two strings into one: `=CONCATENATE (Text1, Text2)`
- **UPPER/LOWER:** Converts a string to upper or lower case: `=UPPER/LOWER (Text)`
- **LEFT/RIGHT/MID:** Returns a specified number of characters starting from the left/right/end of a string.
 - `=LEFT/RIGHT (Text, [NumChars])`
 - `=MID (Text, StartPosition, NumChars)`
- **SUBSTITUTE:** Replaces an instance of existing text with new text in a string: `=SUBSTITUTE (Text, OldText, NewText, [InstanceNumber])`
- **SEARCH:** Returns the position where a specified string or character is found, reading left to right: `=SEARCH (FindText, WithinText, [StartPosition], [NotFoundValue])`

Basic Date & Time Functions

- **TODAY/NOW:** Returns the current date or exact time: `=TODAY/NOW ()`
- **DAY/MONTH/YEAR:** Returns the day of the month (1-31), month of the year (1-12), or year of a given date: `=DAY/MONTH/YEAR (Date)`
- **HOURL/MINUTE/SECOND:** Returns the hour (0-23), minute (0-59), or second (059) of a given datetime value: `=HOURL/MINUTE/SECOND (Datetime)`
- **WEEKDAY/WEEKNUM:** Returns a weekday number from 1 (Sunday) to 7 (Saturday), or the week number of the year: `=WEEKDAY/WEEKNUM (Date, [ReturnType])`
- **EOMONTH:** Returns the date of the last day of the month, +/- a specified number of months: `=EOMONTH (StartDate, Months)`

- **DATEDIFF**: Returns the difference between two dates, based on a given interval (day, hour, year, etc): $=DATEDIFF (Date1, Date2, Interval)$

Joining Data with RELATED

The **RELATED** function returns related values in each row of a table based on relationships with other tables:

=RELATED(ColumnName)

The **column** from a related table containing the values you want to retrieve

Examples:

- 'Product Lookup'[Product Name]
- 'Territory Lookup'[Country]

HEY THIS IS IMPORTANT!

RELATED works like a **VLOOKUP** function in Excel – it uses the relationship between tables (*defined by primary and foreign keys*) to pull values from one table into a new column of another.

Since this function requires row context, it can only be used as a **calculated column** or as part of an **iterator function** that cycles through all rows in a table (*FILTER, SUMX, MAXX, etc.*)

The CALCULATE Function

The **CALCULATE** function evaluates an expression in a context that is modified by filters.

=CALCULATE(Expression, [Filter1], [Filter2],...)

Name of an **existing measure** or a **DAX formula** for a valid measure

Examples:

- [Total Orders]
- SUM('Returns Data'[Return Quantity])

A Boolean (True/False) expression or a table expression that defines a filter

Note: these require fixed values or aggregation functions that return a scalar value (you cannot create filters based on measures)

Examples:

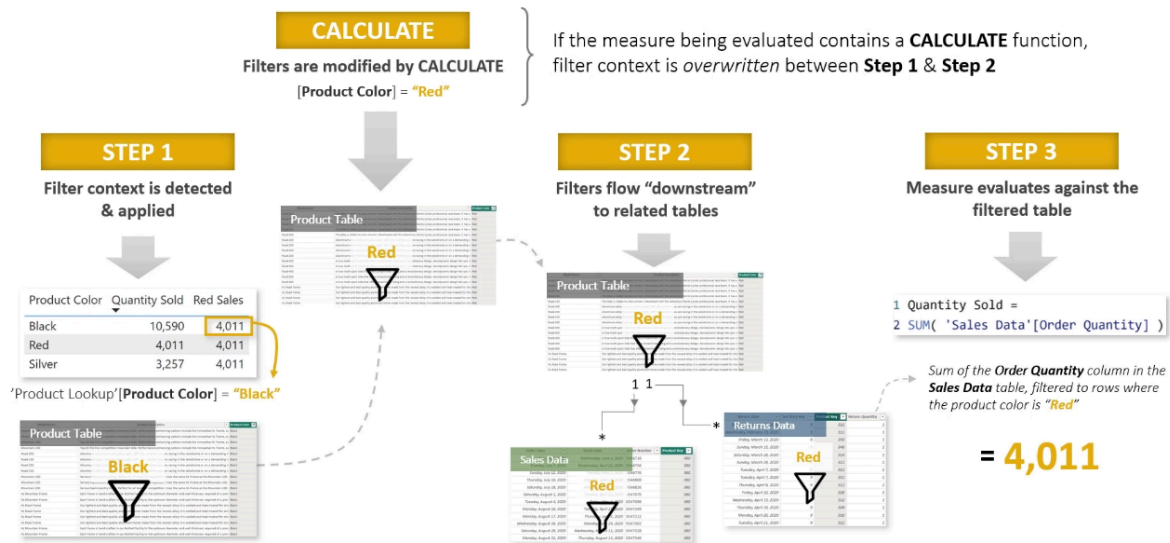
- 'Territory Lookup'[Country] = "USA"
- Calendar[Year] <> MAX(Calendar[Year])



PRO TIP:

Think of **CALCULATE** as a **filter modifier**; it allows you to overrule existing report filters and “force” new filter context

Take into account this:



Note: This sum doesn't add up to the correct value:

CategoryName	Total Orders	Weekend Orders
Accessories	16,983	4,913
Bikes	13,929	3,995
Clothing	6,976	1,962
Total	25,164	7,214

because the total is a **DISTINCTCOUNT** so, for example, if in an order an accessory and a bike was bought, it'd add one to both of them, but only one to the total.

The ALL Function

The **ALL** function returns all rows in a table, or all values in a column, ignoring any filters that have been applied.

ALL

Returns all rows in a table, or all values in a column, ignoring any filters that have been applied

=ALL(**Table or Column**, [Column2], [Column3],...)

The **table** or **column** that you want to clear filters on

Examples:

- Transactions
- Products[Category]

Additional columns that you want to clear filters on (optional)

- Cannot specify columns if your first parameter is a **table**
- All columns must include the **table name** and come from the **same table**

Examples:

- 'Customer Lookup'[City], 'Customer Lookup'[Country]
- Products[Product Name]

If we want to do calculations over all the elements of a column without the filter context, we could do this:

```
Overall Average Price = CALCULATE (
    [Average Retail Price],
    ALL ('Product Lookup')
)
```

The FILTER Function

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

=FILTER(**Table**, FilterExpression)

Table to be filtered

Examples:

- Territory Lookup
- Customer Lookup

A Boolean (True/False) filter expression to be evaluated for each row of the table

Examples:

- 'Territory Lookup'[Country] = "USA"
- Calendar[Year] = 1998
- Products[Price] > [Overall Avg Price]

HEY THIS IS IMPORTANT!

FILTER is used to add new filter context, and can handle **more complex filter expressions** than CALCULATE (by referencing measures, for example)

Since FILTER returns an entire table, it's often **nested within other functions**, like CALCULATE or SUMX

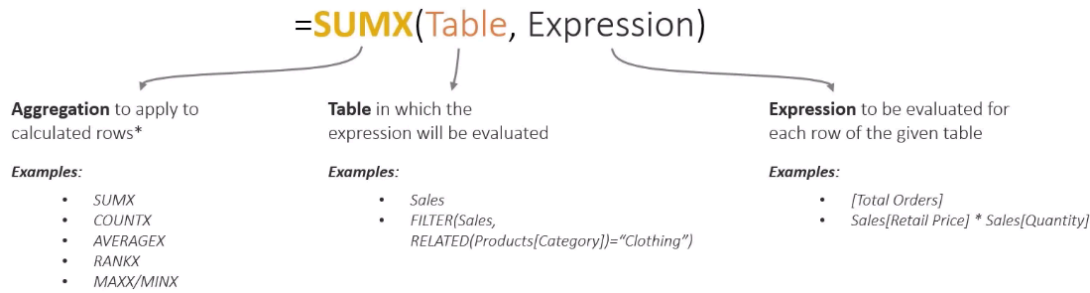


PRO TIP:

Since FILTER **iterates through each row in a table**, it can be slow and computationally expensive; only use FILTER if a simple CALCULATE function won't get the job done!

ITERATOR (X) Functions

Iterator (or "X") **functions** allow you to loop through the same expression on each row of a table, then simply apply some sort of aggregation to the results (SUM, MAX, etc):



PRO TIP:

Imagine that iterator functions **add a temporary new column** to a table, calculate a value in each row based on the given expression, then aggregate the values within that temporary column (similar to **SUMPRODUCT** in Excel)

Time Intelligence

Time Intelligence patterns are used to calculate common date-based comparisons:

Performance To-Date	=CALCULATE (Measure, DATESYTD (Calendar[Date])) <i>Use DATESYTD for Years, DATESQTD for Quarters, DATESMTD for Months</i>
Previous Period	=CALCULATE (Measure, DATEADD (Calendar[Date], -1, MONTH)) <i>Select an interval (DAY, MONTH, QUARTER, or YEAR) and the # of intervals to compare (e.g. previous month, rolling 10-day)</i>
Running Total	=CALCULATE (Measure, DATESINPERIOD (Calendar[Date], MAX (Calendar[Date]), -10, DAY))

DAX Best Practices



Know when to use calculated columns vs. measures

- Use calculated columns for filtering, and measures for aggregating values



Use explicit measures, even for simple calculations

- Explicit measures can be referenced anywhere, and nested within other measures



Use fully-qualified column references in measures

- This makes your DAX more readable, and differentiates column references from measure references



Move column calculations “upstream” when possible

- Adding calculated columns at the source or in Power Query improves report speed and efficiency



Minimize the use of “expensive” iterator functions

- Use iterators with caution, especially if you are working with large tables or complex models

HASONEVALUE

The **HASONEVALUE** function is used to determine if a column has a single distinct value in the current context: `=HASONEVALUE (Column)`

For example, if we want to display only one customer in a customer lookup table, we could do:

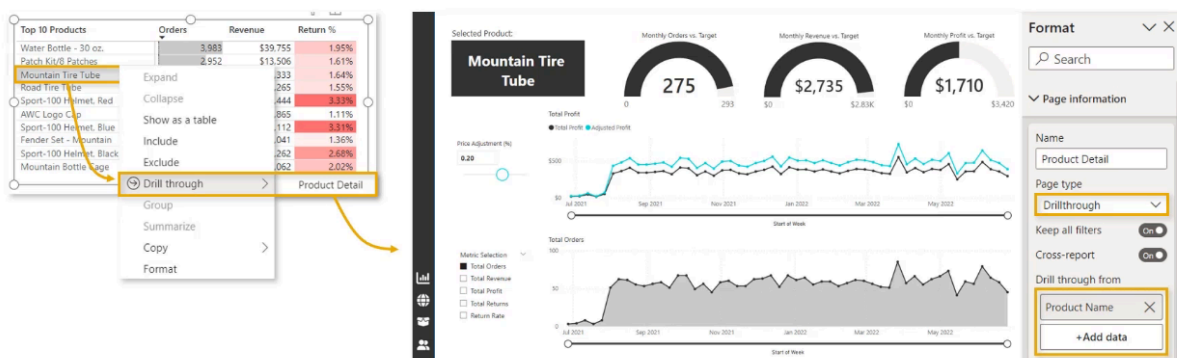
```
Customer Full name (Customer Details) = IF (
    HASONEVALUE (
        'Customer Lookup'[CustomerKey]
    ),
    MAX (
        'Customer Lookup'[Full Name]
    ),
    "Multiple Customers"
)
```

Visualizing Data with Reports

Drill Through Filters

Drill through filters allow users to navigate to a specific report page, pre-filtered on the item selected

- Here we've created a **Product Detail** page, set the type to **Drillthrough**, and configured drill through from **Product Name**
- This means that users can right-click any instance of product name (i.e. in a matrix visual) and use the Drill through option to navigate straight to the Product Detail report filtered on that product (in this case "Mountain Tire Tube")

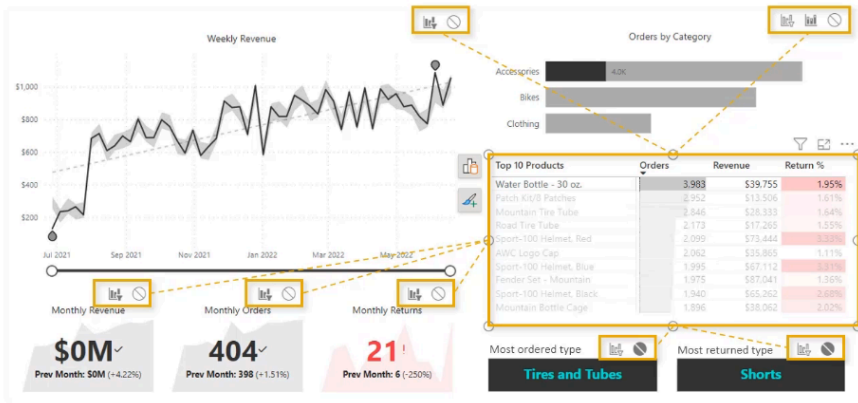


Report Interactions

Edit **report interactions** to customize how filters applied to one visual impact other visuals on the page

- Cross-filter options include **filter** (🔍), **highlight** (👉) and **none** (🚫), depending on the visual type

Format > Edit Interactions



In this example, selecting a product in the matrix visual:

- **Filters** the line chart & KPIs
- **Highlights** the bar chart
- **Doesn't impact** the text cards



To change the **report interactions**, we find its button on the Format > Edit Interactions button.

Note: If two or more filters want to be applied, we need to press CTRL when we apply them so that they are stacked.

Slicer Panels

Slicer Panels can be created using buttons that lead to bookmarks with the Data checkbox disabled.

Numeric Ranges

Numeric Ranges can be used with slicers to change a value. They are created under the Modelling > New Parameter tab. They will create two measures: one containing the value, and another one containing the possible values.

Note: We can set the title of an element to one of a range (i.e. a fields) using the Fx button.

Managing & Viewing Roles

We can create custom roles under the Modelling > Manage Roles button.

Data Visualization Best Practices



Always ask yourself the three key questions

- *What type of data are you visualizing, what are you communicating, and who is the end user?*



Strive for clarity and simplicity above all else

- *"Perfection is achieved not when there's nothing more to add, but when there's nothing left to take away"*



Focus on creating clear narratives and intuitive user experiences

- *Use bookmarks, drillthroughs, tooltips and navigation buttons to seamlessly guide users through reports*



Create optimized layouts for mobile viewers

- *Create custom mobile layouts if you plan to publish reports to Power BI Service or use the Power BI app*