



# CALCULATED FIELDS WITH DAX

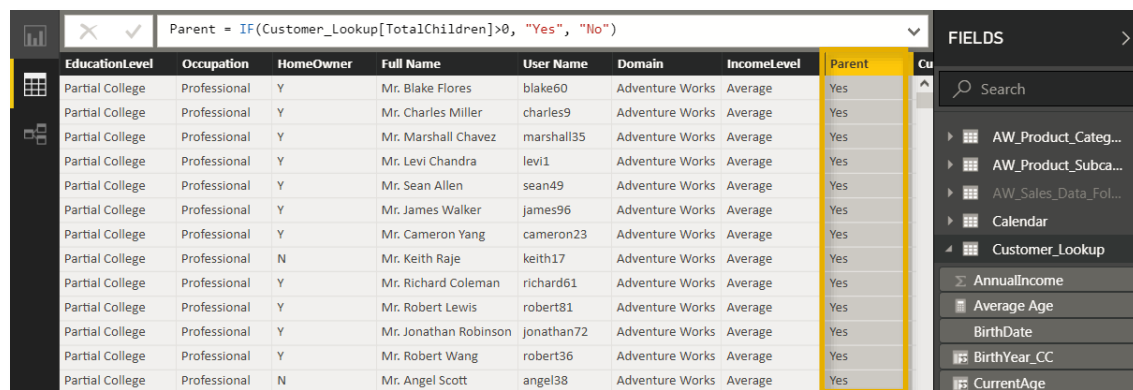
# MEET DAX

**Data Analysis Expressions**, commonly known as **DAX**, is the formula language that drives Power BI. With DAX, you can:

- Add *calculated columns* and *measures* to your model, using intuitive syntax
- Go beyond the capabilities of traditional “grid-style” formulas, with powerful and flexible functions built specifically to work with relational data models

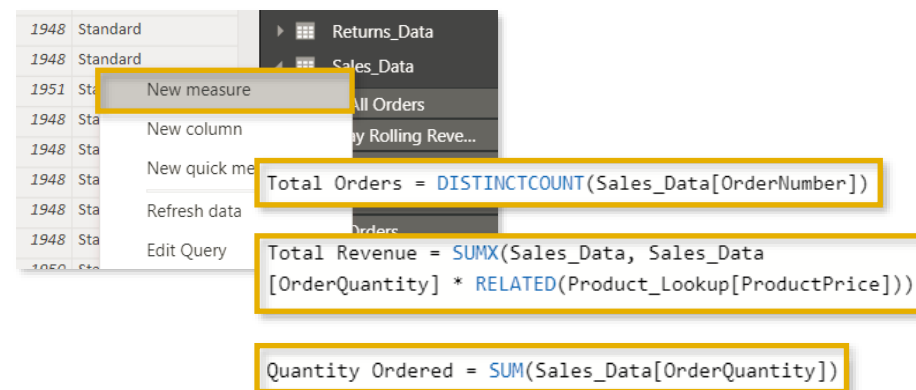
## Two ways to use DAX

### 1) Calculated Columns



EducationLevel	Occupation	HomeOwner	Full Name	User Name	Domain	IncomeLevel	Parent
Partial College	Professional	Y	Mr. Blake Flores	blake60	Adventure Works	Average	Yes
Partial College	Professional	Y	Mr. Charles Miller	charles9	Adventure Works	Average	Yes
Partial College	Professional	Y	Mr. Marshall Chavez	marshall35	Adventure Works	Average	Yes
Partial College	Professional	Y	Mr. Levi Chandra	levi1	Adventure Works	Average	Yes
Partial College	Professional	Y	Mr. Sean Allen	sean49	Adventure Works	Average	Yes
Partial College	Professional	Y	Mr. James Walker	james96	Adventure Works	Average	Yes
Partial College	Professional	Y	Mr. Cameron Yang	cameron23	Adventure Works	Average	Yes
Partial College	Professional	N	Mr. Keith Raj	keith17	Adventure Works	Average	Yes
Partial College	Professional	Y	Mr. Richard Coleman	richard61	Adventure Works	Average	Yes
Partial College	Professional	Y	Mr. Robert Lewis	robert81	Adventure Works	Average	Yes
Partial College	Professional	Y	Mr. Jonathan Robinson	jonathan72	Adventure Works	Average	Yes
Partial College	Professional	Y	Mr. Robert Wang	robert36	Adventure Works	Average	Yes
Partial College	Professional	N	Mr. Angel Scott	angel38	Adventure Works	Average	Yes

### 2) Measures



New measure

New column

New quick measure

Refresh data

Edit Query

Total Orders = `DISTINCTCOUNT(Sales_Data[OrderNumber])`

Total Revenue = `SUMX(Sales_Data, Sales_Data[OrderQuantity] * RELATED(Product_Lookup[ProductPrice]))`

Quantity Ordered = `SUM(Sales_Data[OrderQuantity])`

# CALCULATED COLUMNS

**Calculated columns** allow you to add new, formula-based columns to tables

- No “A1-style” references; calculated columns refer to **entire tables** or **columns**
- Calculated columns generate values for each row, which are **visible within tables in the Data view**
- Calculated columns understand **row context**; they’re 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 when you want to “stamp” static, fixed values to each row in a table (*or use the Query Editor!*)

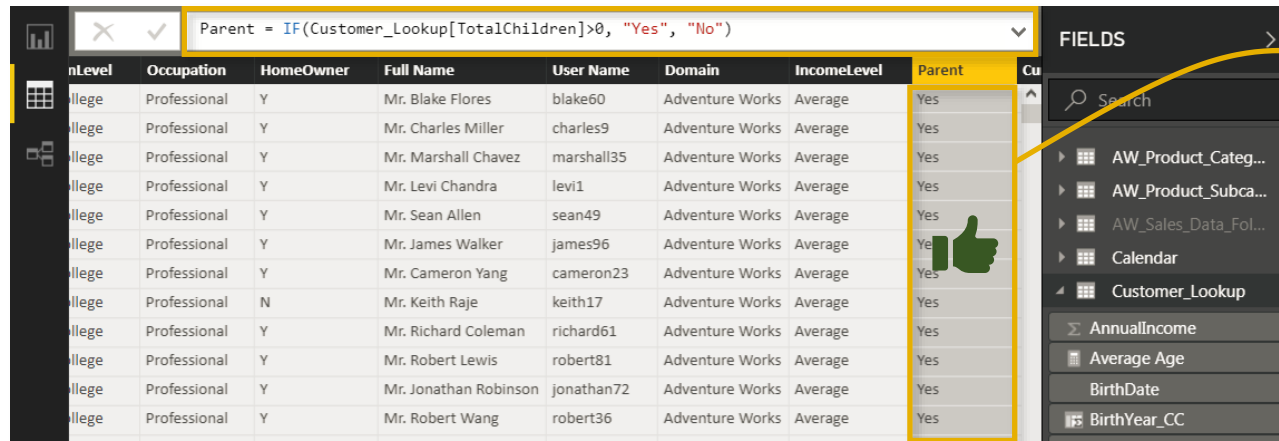
**DO NOT** use calculated columns for aggregation formulas, or to calculate fields for the “Values” area of a visualization (use **measures** instead)



## PRO TIP:

*Calculated columns are typically used for **filtering** data, rather than creating numerical values*

# CALCULATED COLUMNS (EXAMPLES)



The screenshot shows a table with columns: nLevel, Occupation, HomeOwner, Full Name, User Name, Domain, IncomeLevel, and Parent. The formula bar at the top displays: `Parent = IF(Customer_Lookup[TotalChildren]>0, "Yes", "No")`. The 'Parent' column contains 'Yes' for most rows and 'No' for one. A green thumbs-up icon is placed over the 'Parent' column header. A yellow arrow points from the formula bar to the 'Parent' column.

nLevel	Occupation	HomeOwner	Full Name	User Name	Domain	IncomeLevel	Parent
Illege	Professional	Y	Mr. Blake Flores	blake60	Adventure Works	Average	Yes
Illege	Professional	Y	Mr. Charles Miller	charles9	Adventure Works	Average	Yes
Illege	Professional	Y	Mr. Marshall Chavez	marshall35	Adventure Works	Average	Yes
Illege	Professional	Y	Mr. Levi Chandra	levi1	Adventure Works	Average	Yes
Illege	Professional	Y	Mr. Sean Allen	sean49	Adventure Works	Average	Yes
Illege	Professional	Y	Mr. James Walker	james96	Adventure Works	Average	Yes
Illege	Professional	Y	Mr. Cameron Yang	cameron23	Adventure Works	Average	Yes
Illege	Professional	N	Mr. Keith Raj	keith17	Adventure Works	Average	Yes
Illege	Professional	Y	Mr. Richard Coleman	richard61	Adventure Works	Average	Yes
Illege	Professional	Y	Mr. Robert Lewis	robert81	Adventure Works	Average	Yes
Illege	Professional	Y	Mr. Jonathan Robinson	jonathan72	Adventure Works	Average	Yes
Illege	Professional	Y	Mr. Robert Wang	robert36	Adventure Works	Average	Yes

In this case we've added a **calculated column** named **"Parent"**, which equals **"Yes"** if the [TotalChildren] field is greater than 0, and **"No"** otherwise (*just like Excel!!*)

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

Here we're using an aggregation function (SUM) to calculate a new column named **TotalQuantity**

- Since calculated columns do not understand **filter context**, the same grand total is returned in *every single row* of the table
- This is **not a valid use** of calculated columns; these values are statically "stamped" onto the table and can't be filtered, sliced, subdivided, etc.



The screenshot shows a table with columns: kDate, OrderNumber, ProductKey, CustomerKey, TerritoryKey, OrderLineItem, OrderQuantity, QuantityType, and TotalQuantity. The formula bar at the top displays: `TotalQuantity = SUM(AW_Sales_Data[OrderQuantity])`. The 'TotalQuantity' column contains the same value (84174) for every row. A red thumbs-down icon is placed over the 'TotalQuantity' column header. A yellow arrow points from the formula bar to the 'TotalQuantity' column.

kDate	OrderNumber	ProductKey	CustomerKey	TerritoryKey	OrderLineItem	OrderQuantity	QuantityType	TotalQuantity
5/3/2002	SO46718	360	12570	9	1	1	Single Item	84174
2/22/2002	SO46736	360	12341	9	1	1	Single Item	84174
5/5/2002	SO46776	360	12356	9	1	1	Single Item	84174
2/22/2002	SO46808	360	12347	9	1	1	Single Item	84174
2/11/2002	SO46826	360	12575	9	1	1	Single Item	84174
2/21/2002	SO47075	360	12685	9	1	1	Single Item	84174
5/1/2002	SO47098	360	12667	9	1	1	Single Item	84174
2/21/2002	SO47149	360	12669	9	1	1	Single Item	84174
5/4/2002	SO47212	360	12580	9	1	1	Single Item	84174
2/29/2002	SO47302	360	12670	9	1	1	Single Item	84174
2/12/2002	SO47328	360	12681	9	1	1	Single Item	84174
2/13/2002	SO47346	360	12585	9	1	1	Single Item	84174
2/12/2002	SO47744	360	12989	9	1	1	Single Item	84174
2/28/2002	SO47745	360	12998	9	1	1	Single Item	84174

# MEASURES

**Measures** are DAX formulas used to generate new calculated values

- Like calculated columns, measures reference **entire tables** or **columns** (*no A1-style or “grid” references*)
- *Unlike* calculated columns, **measure** values aren’t visible within tables; they can only be “*seen*” within a visualization like a chart or matrix (*similar to a calculated field in an Excel pivot*)
- Measures are evaluated based on **filter context**, which means they recalculate when the fields or filters around them change (*like when new row or column labels are pulled into a matrix or when new filters are applied to a report*)



## HEY THIS IS IMPORTANT!

As a rule of thumb, use measures (vs. *calculated columns*) when a single row can’t give you the answer (*in other words, when you need to **aggregate***)



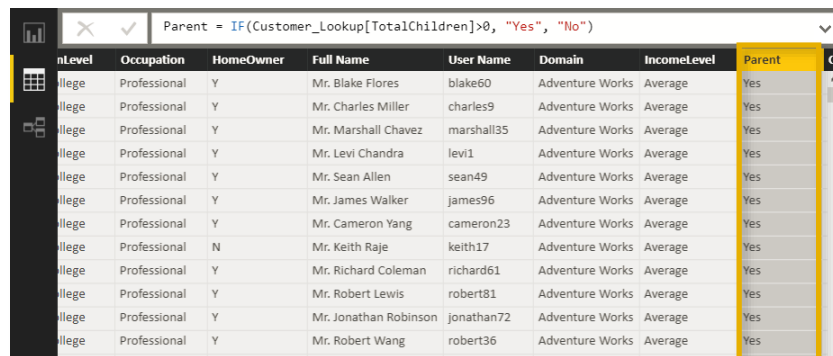
## PRO TIP:

Use measures to create **numerical, calculated values** that can be analyzed in the “**values**” field of a report visual

# RECAP: CALCULATED COLUMNS VS. MEASURES

## CALCULATED COLUMNS

- Values are calculated based on information from each row of a table (has **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 as **rows**, **columns**, **sliders** or **filters**



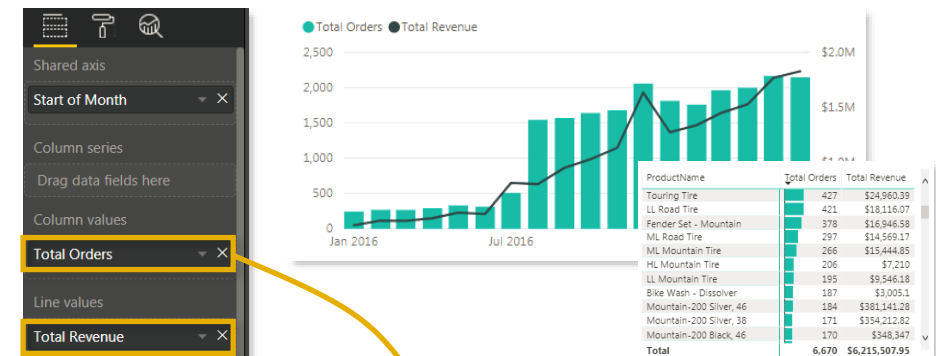
The screenshot shows a table with columns: nLevel, Occupation, HomeOwner, Full Name, User Name, Domain, IncomeLevel, and Parent. The formula bar at the top displays: Parent = IF(Customer\_Lookup[TotalChildren]>0, "Yes", "No"). The 'Parent' column contains 'Yes' for most rows and 'No' for one row (Mr. Keith Raje).

nLevel	Occupation	HomeOwner	Full Name	User Name	Domain	IncomeLevel	Parent
illeg	Professional	Y	Mr. Blake Flores	blake60	Adventure Works	Average	Yes
illeg	Professional	Y	Mr. Charles Miller	charles9	Adventure Works	Average	Yes
illeg	Professional	Y	Mr. Marshall Chavez	marshall35	Adventure Works	Average	Yes
illeg	Professional	Y	Mr. Levi Chandra	levi1	Adventure Works	Average	Yes
illeg	Professional	Y	Mr. Sean Allen	sean49	Adventure Works	Average	Yes
illeg	Professional	Y	Mr. James Walker	james96	Adventure Works	Average	Yes
illeg	Professional	Y	Mr. Cameron Yang	cameron23	Adventure Works	Average	Yes
illeg	Professional	N	Mr. Keith Raje	keith17	Adventure Works	Average	Yes
illeg	Professional	Y	Mr. Richard Coleman	richard61	Adventure Works	Average	Yes
illeg	Professional	Y	Mr. Robert Lewis	robert81	Adventure Works	Average	Yes
illeg	Professional	Y	Mr. Jonathan Robinson	jonathan72	Adventure Works	Average	Yes
illeg	Professional	Y	Mr. Robert Wang	robert36	Adventure Works	Average	Yes

Calculated columns "live" in **tables**

## MEASURES

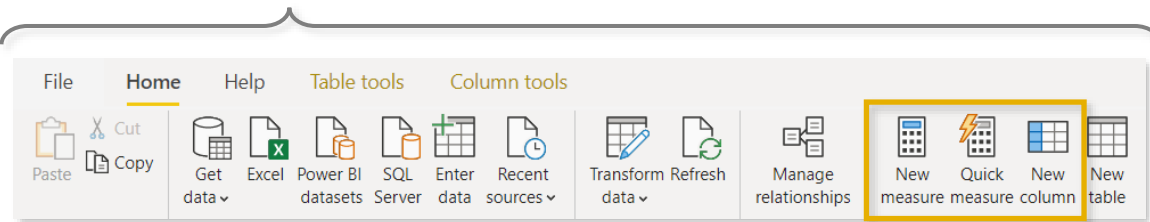
- Values are calculated based on information from any filters in the report (has **filter context**)
- Does not create new data in the tables themselves (*doesn't increase file size*)
- Recalculate in response to any change to filters within the report
- Almost *always* used within the **values** field of a visual



Measures "live" in **visuals**

# ADDING COLUMNS & MEASURES

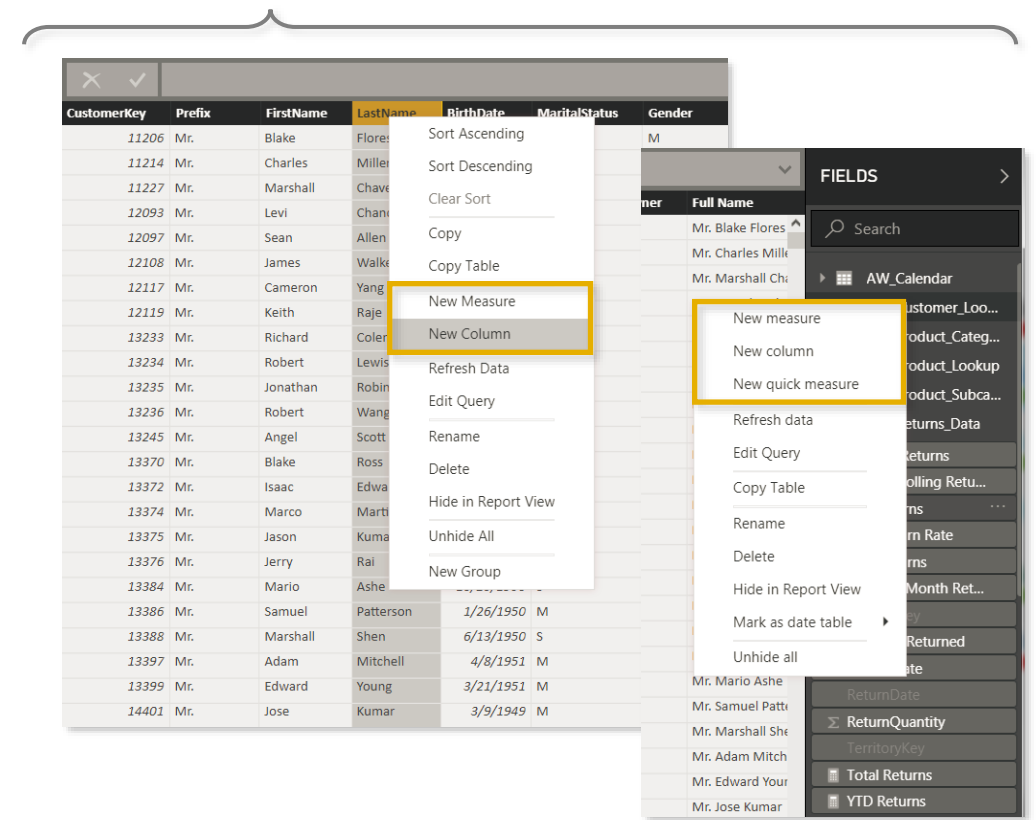
**Option 1:** Select “**New Measure**” or “**New Column**” from the **Home** tab\*



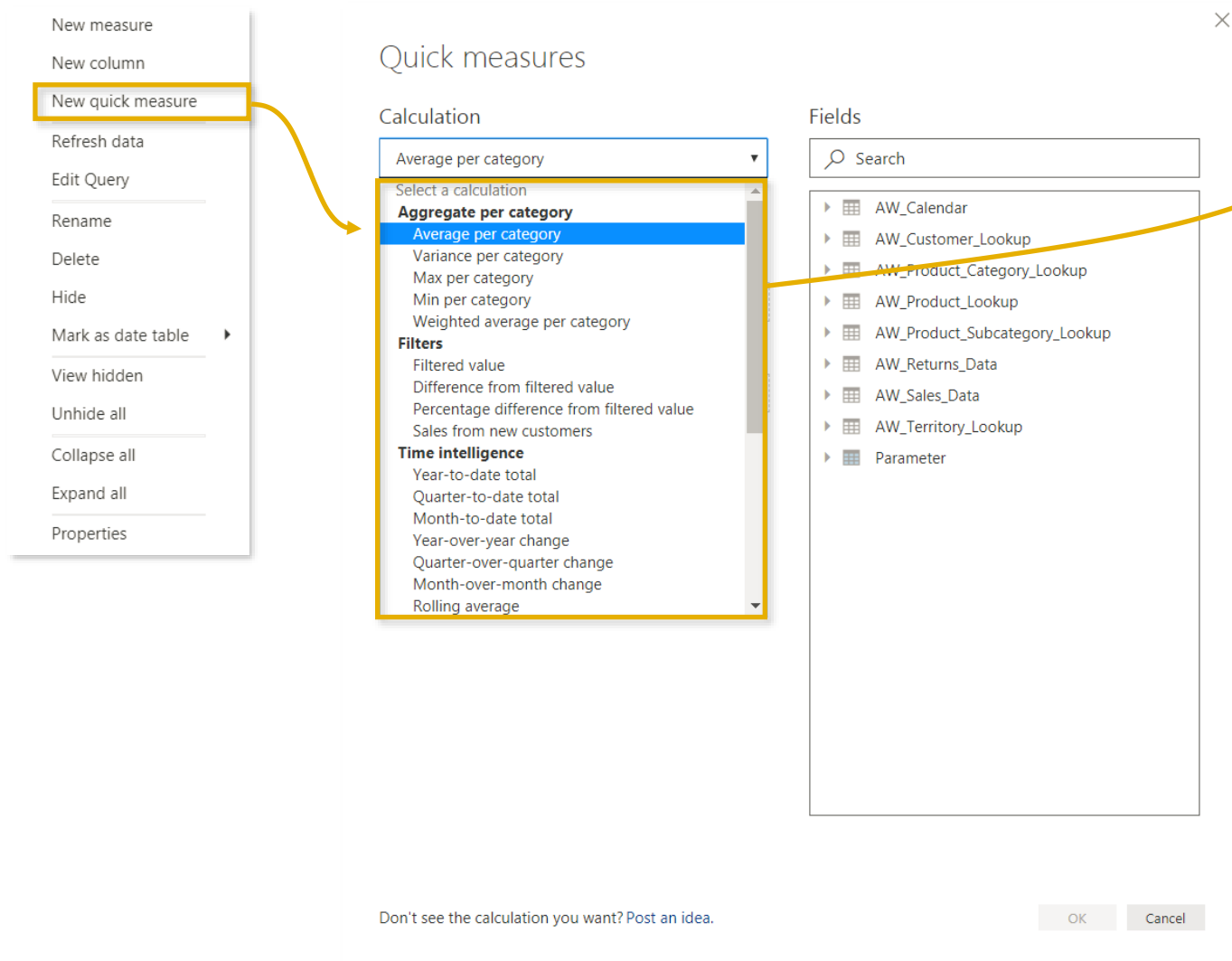
When you insert Columns or Measures using the **Home** tab (Option 1), they are assigned to whichever table is *currently selected*, or the *first table in the field list* by default

- Measures can be reassigned to new “Home” tables (under the “**Structure**” options in the contextual **Measure Tools** tab), but Option 2 allows you to be more deliberate about placing them
- **NOTE:** Assigning measures to specific tables doesn’t have ANY impact on functionality – it’s just a way to keep them organized

**Option 2:** Right-click within the **table** (in the **Data** view) or the **Field List** (in either the **Data** or **Report** view)



# QUICK MEASURES



**Quick Measures** are pre-built formula templates that allow you to drag and drop fields, rather than write DAX from scratch

While these tools can be helpful for defining more complex measures (*like weighted averages or time intelligence formulas*), they encourage laziness and don't help you understand the fundamentals of DAX

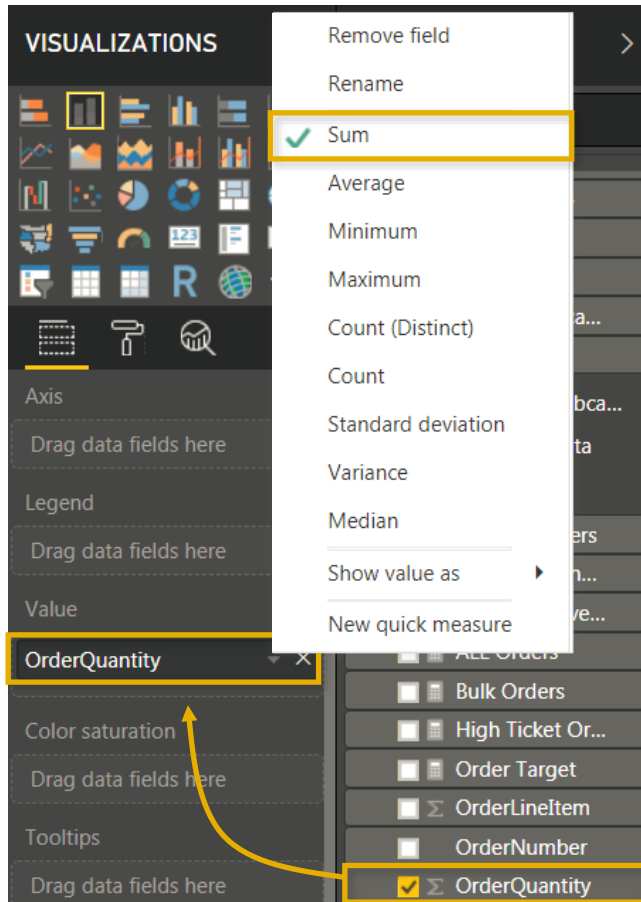


## PRO TIP:

Just say **“NO”** to quick measures  
(you're better than that)



# IMPLICIT VS. EXPLICIT MEASURES



Example of an **implicit measure**

**Implicit measures** are created when you drag raw numerical fields (like “*OrderQuantity*”) into the values pane of a visual and manually select the aggregation mode (*Sum, Average, Min/Max, etc*)

**Explicit measures** are created by actually entering DAX functions (or adding “*quick measures*”) to define calculated columns or measures

## HEY THIS IS IMPORTANT!



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

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

# UNDERSTANDING FILTER CONTEXT

Remember that measures are evaluated based on **filter context**, which means that they recalculate whenever the fields or filters around them change

ProductName	Total Orders	Return Rate
Water Bottle - 30 oz.	1,164	1.96 %
Road Tire Tube	829	1.62 %
AWC Logo Cap	803	0.93 %
Patch Kit/8 Patches	798	1.57 %
Sport-100 Helmet, Red	753	2.79 %
Touring Tire Tube	702	1.35 %
Sport-100 Helmet, Blue	666	3.15 %
Sport-100 Helmet, Black	626	3.67 %
Road Bottle Cage	560	1.58 %
Mountain Tire Tube	554	1.95 %
Mountain Bottle Cage	539	1.38 %
Touring Tire	427	1.16 %
LL Road Tire	421	2.02 %
Fender Set - Mountain	378	1.82 %
ML Road Tire	297	1.72 %
ML Mountain Tire	266	1.94 %
HL Mountain Tire	206	3.40 %
Mountain-200 Silver, 46	199	1.51 %
Mountain-200 Black, 46	196	3.06 %
LL Mountain Tire	195	2.09 %
Mountain-200 Silver, 38	189	2.65 %
Bike Wash - Dissolver	187	2.38 %
Mountain-200 Black, 42	182	3.85 %
Mountain-200 Black, 38	180	3.33 %
Long-Sleeve Logo Jersey, M	161	4.35 %
HL Road Tire	156	5.06 %
Mountain-200 Silver, 42	156	1.28 %
Hydration Pack - 70 oz.	147	4.08 %
Long-Sleeve Logo Jersey, L	147	2.72 %
Long-Sleeve Logo Jersey, S	130	2.31 %
Total	7,380	2.17 %

For this particular value in the matrix, the **Total Orders** measure is calculated based on the following filter context: *Products[ProductName] = "Touring Tire Tube"*

- This allows the measure to return the total order quantity for each product specifically (or whatever the row and column labels dictate – *years, countries, product categories, customer names, etc*)

This Total is **not** calculated by summing the values above; it evaluates as its own measure, with **no filter context** (*since we aren't calculating orders for a specific product*)



## HEY THIS IS IMPORTANT!

Each measure value in a report is **like an island**, and calculates according to it's own filter context (*even Totals and Grand Totals*)

# FILTER CONTEXT (EXAMPLES)

MEASURE: **Total Revenue**

FILTER CONTEXT:

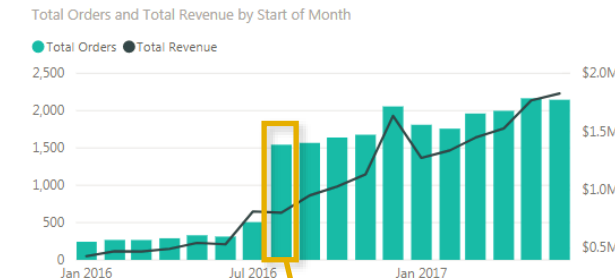
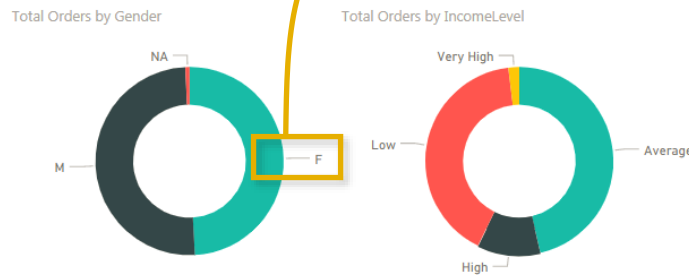
- **Calendar[Year] = 2016 or 2017**
- **Customers[Full Name] = Mr. Larry Munoz**

Full Name	Total Orders	Total Revenue
Mr. Maurice Shan	6	\$12,407.95
Mrs. Janet Munoz	6	\$12,015.39
Mrs. Lisa Chen	7	\$11,330.44
Mrs. Lacey Zheng	7	\$11,085.74
Mr. Jordan Turner	7	\$11,022.38
Mr. Larry Munoz	7	\$10,852.04
Mrs. Ariana Gray	6	\$10,391.42
Mr. Marco Lopez	6	\$10,289.68
Mr. Franklin Xu	5	\$10,164.34
Mrs. Margaret He	4	\$9,266.74
Mrs. Kaitlyn Henderson	4	\$9,258.92
Mrs. Nichole Nara	4	\$9,234.66
Mr. Randall Dominguez	4	\$9,210.36
Mrs. Rosa Hu	4	\$9,201.2
Adriana Gonzalez	4	\$9,195.69
Mrs. Dominique Prasad	6	\$9,180.93
Mrs. Brandi Gill	4	\$9,166.18
Mr. Brad She	4	\$9,161.01
Mr. Francisco Sara	4	\$9,125.54
Mr. Kevin Coleman	4	\$7,750.53
Mr. Johnathan Suri	4	\$7,721.33
Mrs. Crystal Zeng	4	\$7,706.81
Mrs. Felicia Blanco	4	\$7,669.66
Mrs. Jill Suarez	4	\$7,652.61
Mr. Preston Raman	4	\$7,599.49
Mr. Willie Xu	4	\$7,353.55
Mrs. Abby Subram	4	\$7,308.39
Mr. Lance Blanco	4	\$7,207.07
Mrs. Audrey Blanco	4	\$7,139.17
Mr. Ricky Navarro	3	\$7,119.63
Mr. Eddie Dominguez	3	\$7,044.38
Mrs. Molly Madan	3	\$7,043.25
Mr. Jarrod Mehta	3	\$7,038.37
Ms. Susan Zhou	3	\$7,027.98
Mr. Brent Zhang	3	\$7,018.99
Ms. Alyssa Bradley	3	\$7,018.84
<b>Total</b>	<b>22,534</b>	<b>\$18,509,633.2</b>

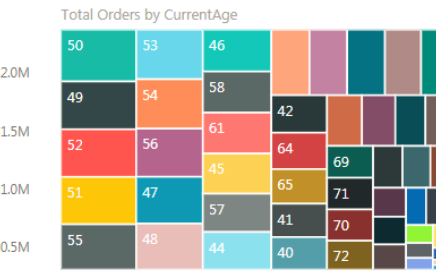
MEASURE: **Total Orders**

FILTER CONTEXT:

- **Calendar[Year] = 2016 or 2017**
- **Customers[Gender] = F (Female)**



**Mr. Maurice Shan**  
Top Customer

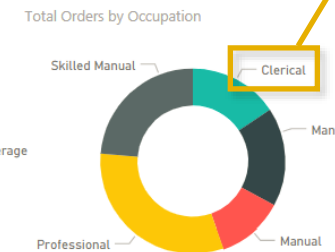


**23K**  
Total Orders

MEASURE: **Total Orders**

FILTER CONTEXT:

- **Calendar[Year] = 2016 or 2017**
- **Customers[Occupation] = Clerical**



Values

Drag data fields here

**FILTERS**

Page level filters

Drag data fields here

Drillthrough filters

Drag drillthrough fields here

Report level filters

Year is 2016 or 2017

*This is a **page-level filter**, which impact **ALL** visuals on the report page*

MEASURE: **Total Orders**

FILTER CONTEXT:

- **Calendar[Year] = 2016 or 2017**

MEASURE: **Total Orders**

FILTER CONTEXT:

- **Calendar[Year] = 2016 or 2017**
- **Calendar[Month] = August 2016**

MEASURE: **Total Revenue**

FILTER CONTEXT:

- **Calendar[Year] = 2016 or 2017**

**\$18.51M**  
Total Revenue

# STEP-BY-STEP MEASURE CALCULATION

CategoryName	Total Returns
Accessories	1,115
Bikes	342
Clothing	267

How *exactly* is this measure calculated?

- REMEMBER: This all happens *instantly* behind the scenes, every time the filter context changes

## STEP 1

Filter context is detected & applied

CategoryName	Total Returns
Accessories	1,115
Bikes	342
Clothing	267

Product[CategoryName] = "Accessories"

Product Table	
Accessories	

## STEP 2

Filters flow "downstream" to all related tables

Product Table	
Accessories	

AW_Sales_Data	
Accessories	

AW_Returns_Data	
Accessories	

## STEP 3

Measure formula evaluates against the filtered table

Total Returns = COUNTROWS(AW\_Returns\_Data)

Count of rows in the AW\_Returns\_Data table, filtered down to only rows where the product category is "Accessories" = 1,115



# COMMON DAX FUNCTIONS

# DAX SYNTAX

## MEASURE NAME

- **Note:** Measures are always surrounded in brackets (i.e. **[Total Quantity]**) when referenced in formulas, so spaces are OK

Referenced  
**TABLE NAME**

Referenced  
**COLUMN NAME**

Total Quantity: =SUM(Transactions[quantity])

## FUNCTION NAME

- Calculated columns don't always use functions, but measures do:
  - In a **Calculated Column**, =Transactions[quantity] returns the value from the quantity column in each row (since it evaluates one row at a time)
  - In a **Measure**, =Transactions[quantity] will return an **error** since Power BI doesn't know how to translate that as a single value (you need some sort of aggregation)

**Note:** This is a “fully qualified” column, since it's preceded by the table name -- table names with spaces must be surrounded by **single quotes**:

- Without a space: **Transactions[quantity]**
- With a space: **'Transactions Table'[quantity]**



## PRO TIP:

For **column** references, use the fully qualified name (i.e. **Table[Column]**)  
For **measure** references, just use the measure name (i.e. **[Measure]**)

# DAX OPERATORS

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

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

*Pay attention to these!*

Text/Logical Operator	Meaning	Example
&	Concatenates two values to produce one text string	[City] & " " & [State]
&&	Create an AND condition between two logical expressions	([State]="MA") && ([Quantity]>10)
(double pipe)	Create 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" }

\*Head to [www.msdn.microsoft.com](http://www.msdn.microsoft.com) for more information about DAX syntax, operators, troubleshooting, etc

# COMMON FUNCTION CATEGORIES

## MATH & STATS Functions

Basic **aggregation** functions as well as “**iterators**” evaluated at the row-level

### Common Examples:

- SUM
- AVERAGE
- MAX/MIN
- DIVIDE
- COUNT/COUNTA
- COUNTROWS
- DISTINCTCOUNT

### Iterator Functions:

- SUMX
- AVERAGEX
- MAXX/MINX
- RANKX
- COUNTX

## LOGICAL Functions

Functions for returning information about values in a given **conditional expression**

### Common Examples:

- IF
- IFERROR
- AND
- OR
- NOT
- SWITCH
- TRUE
- FALSE

## TEXT Functions

Functions to manipulate **text strings** or **control formats** for dates, times or numbers

### Common Examples:

- CONCATENATE
- FORMAT
- LEFT/MID/RIGHT
- UPPER/LOWER
- PROPER
- LEN
- SEARCH/FIND
- REPLACE
- REPT
- SUBSTITUTE
- TRIM
- UNICHAR

## FILTER Functions

**Lookup** functions based on related tables and **filtering** functions for dynamic calculations

### Common Examples:

- CALCULATE
- FILTER
- ALL
- ALLEXCEPT
- RELATED
- RELATEDTABLE
- DISTINCT
- VALUES
- EARLIER/EARLIEST
- HASONESVALUE
- HASONEFILTER
- ISFILTERED
- USERRELATIONSHIP

## DATE & TIME Functions

Basic **date and time** functions as well as advanced **time intelligence** operations

### Common Examples:

- DATEDIFF
- YEARFRAC
- YEAR/MONTH/DAY
- HOUR/MINUTE/SECOND
- TODAY/NOW
- WEEKDAY/WEEKNUM

### Time Intelligence Functions:

- DATESYTD
- DATESQTD
- DATESMTD
- DATEADD
- DATESINPERIOD

**\*Note:** This is NOT a comprehensive list (does not include trigonometry functions, parent/child functions, information functions, or other less common functions)



# BASIC DATE & TIME FUNCTIONS

**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)

**HOUR/MINUTE/  
SECOND()**

*Returns the hour (0-23), minute (0-59), or second (0-59) of a given datetime value*

=**HOUR/MINUTE/SECOND**(Datetime)

**TODAY/NOW()**

*Returns the current date or exact time*

=**TODAY/NOW**()

**WEEKDAY/  
WEEKNUM()**

*Returns a weekday number from 1 (Sunday) to 7 (Saturday), or the week # 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 selected interval*

=**DATEDIFF**(Date1, Date2, Interval)

# BASIC LOGICAL FUNCTIONS (IF/AND/OR)

**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 the expression returns an error, otherwise returns the expression itself*

=**IFERROR**(Value, ValueIfError)

**AND()**

*Checks whether both arguments are TRUE, and returns TRUE if both arguments are TRUE, otherwise returns FALSE*

=**AND**(Logical1, Logical2)

**OR()**

*Checks whether one of the arguments is TRUE to return TRUE, and returns FALSE if both arguments are FALSE*

=**OR**(Logical1, Logical2)

**Note:** Use the **&&** and **||** operators if you want to include more than two conditions!

# TEXT FUNCTIONS

**LEN()**

*Returns the number of characters in a string*

=**LEN**(Text)

*Note: Use the & operator as a shortcut,  
or to combine more than two strings!*

**CONCATENATE()**

*Joins two text strings into one*

=**CONCATENATE**(Text1, Text2)

**LEFT/MID/  
RIGHT()**

*Returns a number of characters from the  
start/middle/end of a text string*

=**LEFT/RIGHT**(Text, [NumChars])

=**MID**(Text, StartPosition, NumChars)

**UPPER/LOWER/  
PROPER()**

*Converts letters in a string to  
upper/lower/proper case*

=**UPPER/LOWER/PROPER**(Text)

**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])

# RELATED

## RELATED()

*Returns related values in each row of a table based on relationships with other tables*

=RELATED(ColumnName)



*The column that contains the values you want to retrieve*

**Examples:**

- *Product\_Lookup[ProductName]*
- *Territory\_Lookup[Country]*



### HEY THIS IS IMPORTANT!

**RELATED** works almost *exactly* like a **VLOOKUP** function – 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*)



### PRO TIP:

*Avoid using RELATED to create redundant calculated columns unless you absolutely need them, since those extra columns increase file size; instead, use RELATED within a measure like FILTER or SUMX*

# BASIC MATH & 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**(ColumnName) or =**MAX**(Scalar1, [Scalar2])

**MIN()**

*Returns the smallest value in a column or between two scalar expressions*

=**MIN**(ColumnName) or =**MIN**(Scalar1, [Scalar2])

**DIVIDE()**

*Performs division and returns the alternate result (or blank) if div/0*

=**DIVIDE**(Numerator, Denominator, [AlternateResult])

# COUNT, COUNTA, DISTINCTCOUNT & COUNTROWS

---

**COUNT()**

*Counts the number of cells in a column that contain numbers*

=**COUNT**(ColumnName)

**COUNTA()**

*Counts the number of non-empty cells in a column (numerical and non-numerical)*

=**COUNTA**(ColumnName)

**DISTINCTCOUNT()**

*Counts the number of distinct or unique 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)

# CALCULATE

## CALCULATE()

*Evaluates a given expression or formula under a set of defined filters*

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

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

**Examples:**

- [Total Orders]
- SUM>Returns\_Data[ReturnQuantity])

*List of simple Boolean (True/False) filter expressions  
(**note:** these require simple, fixed values; you cannot create filters based on measures)*

**Examples:**




- Territory\_Lookup[Country] = "USA"
- Calendar[Year] > 1998



### PRO TIP:

*CALCULATE works just like **SUMIF** or **COUNTIF** in Excel, except it can evaluate measures based on ANY sort of calculation (not just a sum, count, etc); it may help to think of it like "**CALCULATEIF**"*

# CALCULATE (EXAMPLE)

  Bike Returns = CALCULATE([Total Returns], Products[CategoryName] = "Bikes") 

CategoryName	Total Returns	Bike Returns
Accessories	1,115	342
Bikes	342	342
Clothing	267	342
Components		342
Total	1,724	342

Here we've defined a new measure named "**Bike Returns**", which evaluates the "**Total Returns**" measure when the *CategoryName* in the **Products** table equals "**Bikes**"

Wait, why do we see the **same repeating values** when we view a matrix with different categories on rows?

Shouldn't these cells have different filter contexts for **Accessories**, **Clothing**, **Components**, etc?



## HEY THIS IS IMPORTANT!

CALCULATE **modifies** and **overrides** any competing filter context!

In this example, the "Clothing" row has filter context of **CategoryName = "Clothing"** (defined by the row label) **and** **CategoryName = "Bikes"** (defined by the CALCULATE function)

Both cannot be true at the same time, so the "**Clothing**" filter is overwritten and the "**Bikes**" filter (from CALCULATE) takes priority



# CALCULATE CHANGES THE FILTER CONTEXT

## CALCULATE

*Filters modified by CALCULATE*

[CategoryName] = "Bikes"

If the measure being evaluated contains a **CALCULATE** function, filter context is *overwritten* between **Step 1** & **Step 2**

### STEP 1

*Filter context is detected & applied*

CategoryName	Total Returns	Bike Returns
Accessories	1,115	342
Bikes	342	342
Clothing	267	342
Components	342	342
Total	1,724	342

Products[CategoryName] = "Accessories"

### STEP 2

*Filters flow "downstream" to all related tables*

Product Table

Bikes

Product Table

Bikes

### STEP 3

*Measure formula evaluates against the filtered table*

Total Returns = COUNTROWS(AW\_Returns\_Data)

Count of rows in the **AW\_Returns\_Data** table, filtered down to only rows where the product category is "Bikes" = **342**

Product Table

Accessories

AW\_Sales\_Data

Bikes

AW\_Returns\_Data

Bikes

# ALL

## ALL()

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

=**ALL**(**Table** or **ColumnName**, [ColumnName1], [ColumnName2],...)

The table or column that you want to clear filters on

**Examples:**

- Transactions
- Products[ProductCategory]

List of columns that you want to clear filters on (optional)

**Notes:**

- If your first parameter is a table, you can't specify additional columns
- All columns must include the table name, and come from the same table

**Examples:**

- Customer\_Lookup[CustomerCity], Customer\_Lookup[CustomerCountry]
- Products[ProductName]



**PRO TIP:**

Instead of adding filter context, ALL **removes it**. This is often used when you need unfiltered values that won't react to changes in filter context (i.e. **% of Total**, where the denominator needs to remain fixed)

# FILTER

## FILTER()

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 almost always used as an *input* to other functions, like CALCULATE or SUMX



### PRO TIP:

Since FILTER iterates through each row in a table, it can be slow and processor-intensive; don't use FILTER if a CALCULATE function will accomplish the same thing

# ITERATOR (“X”) FUNCTIONS

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

**=SUMX**(Table, Expression)

Aggregation to apply to calculated rows\*

**Examples:**

- SUMX
- COUNTX
- AVERAGEX
- RANKX
- MAXX/MINX

Table in which the expression will be evaluated

**Examples:**

- Sales
- FILTER(Sales, RELATED(Products[Category])="Clothing")

Expression to be evaluated for each row of the given table

**Examples:**

- [Total Orders]
- Sales[RetailPrice] \* Sales[Quantity]



## PRO TIP:

Imagine the function **adding a temporary new column** to the table, calculating the value in each row (based on the expression) and then applying the aggregation to that new column (like SUMPRODUCT)

\*In this example we're looking at **SUMX**, but other “X” functions follow a similar syntax

# TIME INTELLIGENCE FORMULAS

**Time Intelligence** functions allow you to easily calculate common time comparisons:

Performance  
To-Date

=**CALCULATE**(Measure, **DATESYTD**(Calendar[Date]))

Use **DATESQTD** for Quarters or **DATESMTD** for Months

Previous  
Period

=**CALCULATE**(Measure, **DATEADD**(Calendar[Date], -1, **MONTH**))

Select an interval (**DAY**, **MONTH**, **QUARTER**, or **YEAR**) and the # of intervals to compare (i.e. previous month, rolling 10-day)

Running  
Total

=**CALCULATE**(Measure, **DATESINPERIOD**(Calendar[Date], **MAX**(Calendar[Date]), -10, **DAY**))



## PRO TIP:

To calculate a **moving average**, use the running total calculation above and divide by the number of intervals