atomcamp

MICROSOFT POWER BI MANUAL FOR DESKTOP

Calculated Fields with DAX

What is DAX?

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

With DAX, you can:

- Go beyond the capabilities of traditional spreadsheet 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

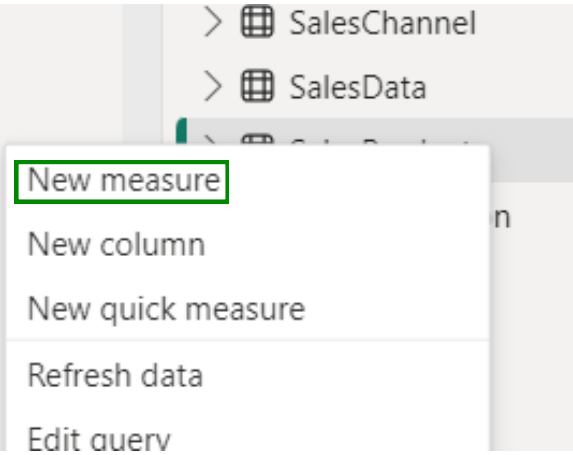
Two ways to use DAX

Calculated Columns

```
Week = WEEKNUM(SalesData[Order Date])
```

Order ID	Order Date	Unit Cost	Price	Order Qty	Cost of Sales	ChannelKey	PromotionKey	ProductKey	ProductSubCategoryKey	StateID	Week
1563	Friday, 6 April 2012	41.9689762040848	198	10	419.689762040848	1	1	984	13	6	14
213	Sunday, 17 March 2013	43.8564334991178	148	10	438.564334991178	1	1	960	13	37	12
7481	Saturday, 22 June 2013	63.7753476353409	196.9	10	637.753476353408	1	1	1011	13	4	25
7496	Friday, 13 September 2013	46.1767546733516	281	10	461.767546733516	1	1	951	13	26	37
7617	Friday, 1 June 2012	10.2749513905361	184.5	10	102.749513905361	1	1	952	13	27	22
284	Wednesday, 25 April 2012	122.766049653059	198	10	1227.66049653059	1	1	1012	13	34	17
906	Sunday, 13 May 2012	152.177079021059	188.5	10	1521.77079021059	1	1	967	13	3	20
367	Sunday, 25 March 2012	131.323258169443	268	10	1313.23258169443	1	1	1019	13	26	13
404	Tuesday, 4 September 2012	140.667781370824	186.9	10	1406.67781370824	1	1	954	13	32	36
419	Saturday, 4 May 2013	128.845850714887	184.5	10	1288.45850714887	1	1	1022	13	35	18
1569	Friday, 6 September 2013	78.3518181064417	268	10	783.518181064417	1	1	949	13	4	36
1735	Sunday, 15 April 2012	37.0437910382909	196.9	10	370.437910382909	1	1	955	13	21	16
1759	Saturday, 6 April 2013	40.3275641157312	198	10	403.275641157312	1	1	970	13	18	14
1268	Tuesday, 28 May 2013	49.5238242603922	268	10	495.238242603922	1	1	977	13	15	22
1201	Thursday, 8 March 2012	42.028096907257	188.5	10	420.28096907257	1	1	995	13	4	10
1062	Sunday, 8 April 2012	51.6393751178306	268	10	516.393751178306	1	1	1033	13	29	15

Measures




```
Order_Quantity = SUM(SalesData[Order Qty])
```

```
Total Profit = SUMX(SalesData, SalesData[Cost of Sales] - SalesData[Price])
```

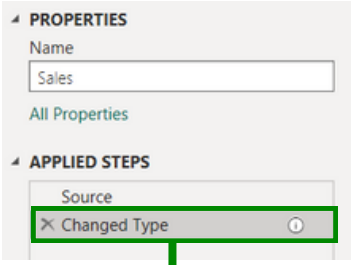
M vs. DAX

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
- DAX is used in the Power BI front-end, and is designed specifically for analyzing relational data models

M

Query Editor:




```
let
    Source = SalesData,

    // Change the datatype of the "Price" column to Int64
    #"Changed Type" = Table.TransformColumnTypes(
        Source,
        {{"Price", Int64.Type}}
    )
in
    #"Changed Type"
```

DAX

Report View:

Product Category	Sum of Order Qty	Computers Orders
Audio	9067	
Cameras and camcorders	43893	
Cell phones	96074	
Computers	69425	69425
Music, Movies and Audio Books	9953	
TV and Video	22714	
Total	251126	

```
Computers Orders =
IF(
    SELECTEDVALUE('ProductSubCategory'[Product Category]) = "Computers",
    CALCULATE(
        [Order_Quantity],                                   // Counting total order quantity
        'ProductSubCategory'[Product Category] = "Computers"    // filtered for Computers only
    ),
    BLANK()
)
```

Calculated Columns

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

- Calculated columns refer to entire tables or columns (no A1-style cell references)
- 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.)

Tip: Calculated columns are typically used for **filtering** & **grouping** data, rather than creating aggregate numerical values

Example: Calculated Columns

```
Week = WEEKNUM(SalesData[Order Date])
```

Order ID	Order Date	Unit Cost	Price	Order Qty	Cost of Sales	ChannelKey	PromotionKey	ProductKey	ProductSubCategoryKey	StateID	Week
1563	Friday, 6 April 2012	41.9689762040848	198	10	419.689762040848	1	1	984	13	6	14
213	Sunday, 17 March 2013	43.8564334991178	148	10	438.564334991178	1	1	960	13	37	12
7481	Saturday, 22 June 2013	63.7753476353409	196.9	10	637.753476353408	1	1	1011	13	4	25
7496	Friday, 13 September 2013	46.1767546733516	281	10	461.767546733516	1	1	951	13	26	37
7617	Friday, 1 June 2012	10.2749513905361	184.5	10	102.749513905361	1	1	952	13	27	22
284	Wednesday, 25 April 2012	122.766049653059	198	10	1227.66049653059	1	1	1012	13	34	17
906	Sunday, 13 May 2012	152.177079021059	188.5	10	1521.77079021059	1	1	967	13	3	20
367	Sunday, 25 March 2012	131.323258169443	268	10	1313.23258169443	1	1	1019	13	26	13
404	Tuesday, 4 September 2012	140.667781370824	186.9	10	1406.67781370824	1	1	954	13	32	36
419	Saturday, 4 May 2013	128.845850714887	184.5	10	1288.45850714887	1	1	1022	13	35	18
1569	Friday, 6 September 2013	78.3518181064417	268	10	783.518181064417	1	1	949	13	4	36
1735	Sunday, 15 April 2012	37.0437910382909	196.9	10	370.437910382909	1	1	955	13	21	16
1759	Saturday, 6 April 2013	40.3275641157312	198	10	403.275641157312	1	1	970	13	18	14
1268	Tuesday, 28 May 2013	49.5238242603922	268	10	495.238242603922	1	1	977	13	15	22
1201	Thursday, 8 March 2012	42.028096907257	188.5	10	420.28096907257	1	1	995	13	4	10
1062	Sunday, 8 April 2012	51.6393751178306	268	10	516.393751178306	1	1	1033	13	29	15

In this case we've added a **calculated column** named **Week**, which calculates the week number from the [Order date]

- Since calculated columns understand **row context**, a new value is calculated in each row based on the value in the [Order Date] 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 mode

DAX Measures

Measures are DAX formulas used to generate new calculated values

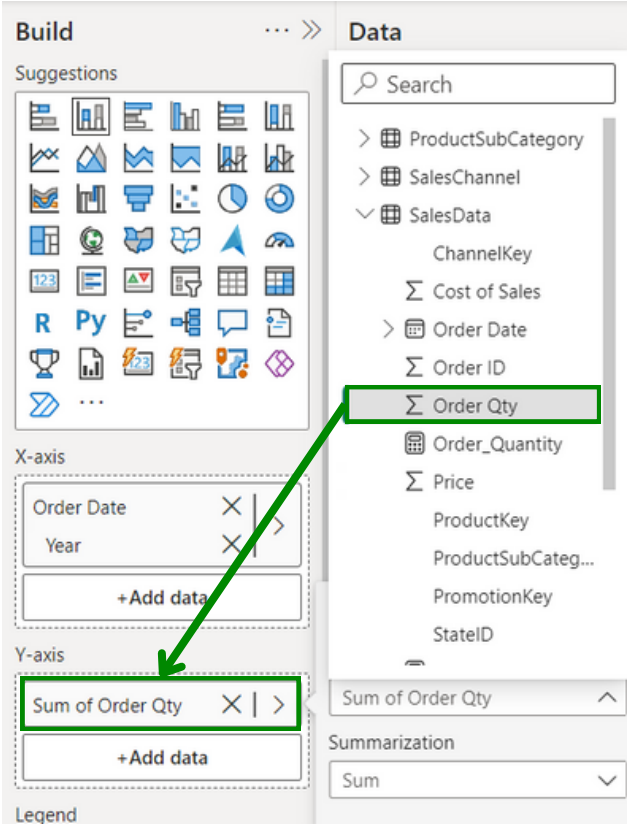
- Like calculated columns, measures reference **entire tables** or **columns** (no A1-style cell references)
- Unlike calculated columns, **measures** aren't visible within tables; they can only be "seen" within a visualization like a chart or matrix (similar to a calculated field in a PivotTable)
-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

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

Implicit vs. Explicit Measures

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

Explicit measures are created when you actually write a DAX formula and define a new measure that can be used within the model



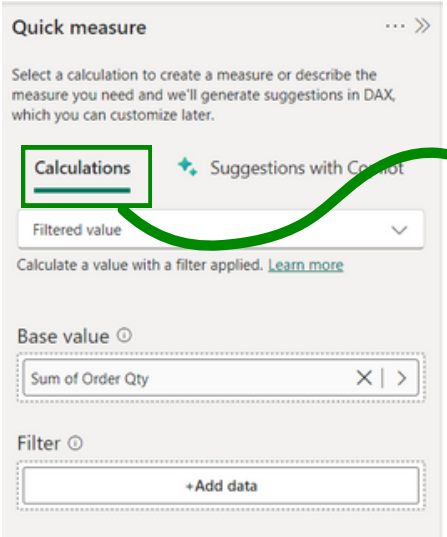
*Example of an **Implicit Measure***

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"

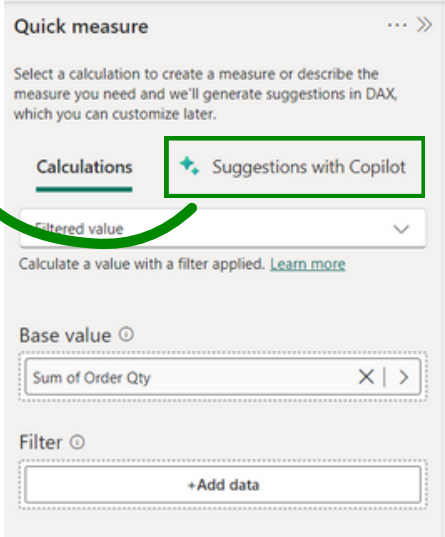
Quick Measures

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")



Tip: Quick measures can be a great learning tool for beginners or for building more complex formulas but use them with caution; **mastering DAX requires a deep understanding of the underlying theory**!

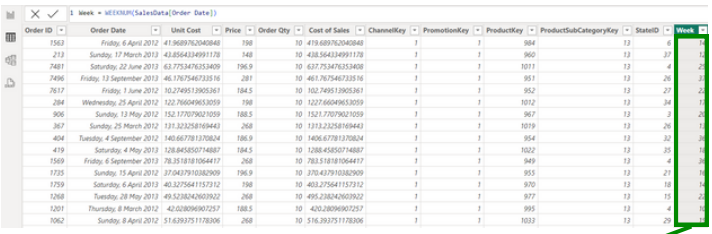
Calculated Columns vs. Measures

Calculated Columns

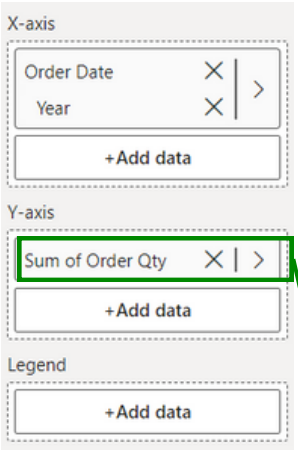
- Values are calculated based on information from each row of a table (**row context**)
- Appends static values to each row in a table and stores them in the model (which increases file size)
- Recalculate on data source refresh or when changes are made to component columns
- Primarily used for **filtering**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

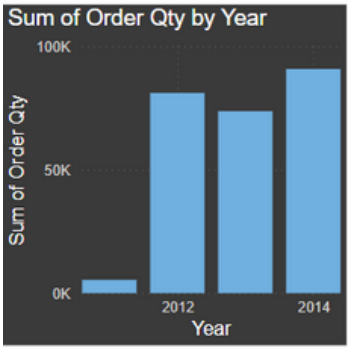
Measures

- 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



*Calculated Columns "live" in **tables***





*Measures "live" in **visuals***

Filter Context

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

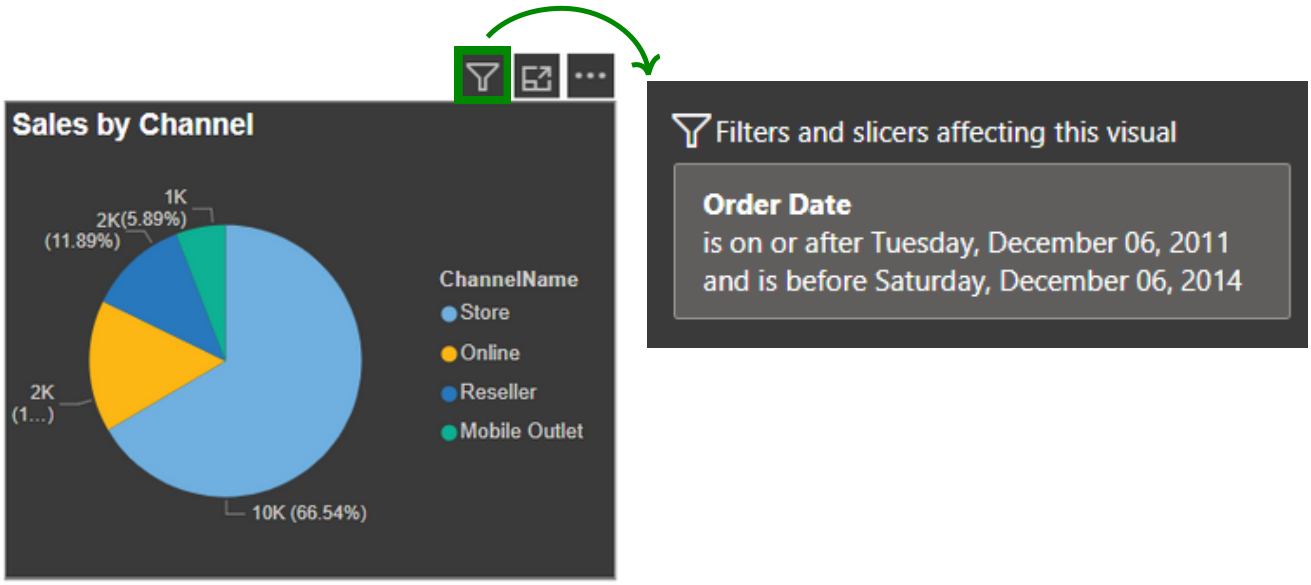
Product Category	Sum of Order Qty
Audio	9067
Cameras and camcorders	43893
Cell phones	96074
Computers	69425
Music, Movies and Audio Books	9953
TV and Video	22714
Total	**251126**

For this value in the matrix (96074), the **Sum of Order Qty** measure is calculated based on the following filter context:
ProductSubCategory[Product Category] = "Cell phones"

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

This total (251126) does **NOT** calculate by summing the values above; it evaluates as an independent measure with **no filter context** applied

- **IMPORTANT**: Every measure value in a report evaluates **independently** (like an island) and calculates based on its own filter context



Tip: Clicking the **filter icon** will show you the filters currently applied to a selected visua

DAX Syntax

MEASURE NAME
Measures are always surrounded by brackets (i.e. [Total Quantity]) when referenced in formulas, so spaces are OK

Referenced **TABLE NAME**

Referenced **COLUMN NAME**

Order_Quantity = **SUM**(SalesData[Order Qty])

This is a **"fully qualified"** column, since it's preceded by the table name.
NOTE: Table names with spaces must be surrounded by **single quotes**:

- Without a space: **SalesData**[Order Qty]
- With a space: **'Sales Data'**[Order Qty]

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

Tip: Column references use fully qualified names (i.e. **'Table'[Column]**)
Measure references just use the measure name (i.e. **[Measure]**) and can be called by typing an open square bracket " ["

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" }

Common Function Categories

Math & Stats Functions

*Functions used for **aggregation** or iterative, row-level calculations*

Common Examples:

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

Iterator Functions:

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

Logical Functions

*Functions that use **conditional expressions** (IF/THEN statements)*

Common Examples:

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

Common Function Categories

Text Functions

*Functions used to manipulate **text strings** or **value formats***

Common Examples:

- CONCATENATE
- COMBINEVALUES
- FORMAT
- LEFT/MID/RIGHT
- UPPER/LOWER
- LEN
- SEARCH/FIND
- REPLACE
- SUBSTITUTE
- TRIM

Filter Functions

*Functions used to **manipulate table** and **filter contexts***

Common Examples:

- CALCULATE
- FILTER
- ALL
- ALLEXCEPT
- ALLSELECTED
- KEEPFILTERS
- REMOVEFILTERS
- SELECTEDVALUE

Common Function Categories

Table Functions

*Functions that **create** or **manipulate tables** and output tables vs. scalar values*

Common Examples:

- SUMMARIZE
- ADDCOLUMNS
- GENERATESERIES
- DISTINCT
- VALUES
- UNION
- INTERSECT
- TOPN

Date & Time Functions

*Functions used to manipulate **date & time values** or handle time intelligence calculations*

Common Examples:

- DATE
- DATEDIFF
- YEARFRAC
- YEAR/MONTH
- DAY/HOUR
- TODAY/NOW
- WEEKDAY
- WEEKNUM
- NETWORKDAYS

Time Intelligence:

- DATESYTD
- DATESMTD
- DATEADD
- DATESBETWEEN

Relationship Functions

*Functions used to **manage & modify table relationships***

Common Examples:

- RELATED
- RELATEDTABLE
- CROSSFILTER
- USERELATIONSHIP

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(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]*)

Counting Functions

COUNT	Counts the number of non-empty cells in a column (excluding Boolean values)	=COUNT(ColumnName)
COUNTA	Counts the number of non-empty cells in a column (including Boolean values)	=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])

Basic 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 whether any argument is TRUE to return TRUE, otherwise returns FALSE	=OR(Logical1, Logical2)

Note: *Use the **&&** and **||** operators to include more than two conditions*

Switch

SWITCH	Evaluates an expression against a list of values and returns one of multiple possible expressions

=SWITCH(Expression, Value1, Result1, ..., *[Else]*)

Any **DAX expression** that returns a single scalar value, evaluated multiples times

Examples:

- *Calendar[Month ID]*
- *'Product Lookup'[category]*

List of **values** produced by the expression, each paired with a result to return for rows/cases that match

Examples:

=SWITCH(Calendar[Month ID],
1, "January",
2, "February"

Value returned if the expression doesn't match any value argument

Tip: SWITCH(TRUE) is a common DAX pattern to replace multiple nested IF statements

Text Functions

LEN	Returns the number of characters in a string	=LEN(Text)
CONCATENATE	Joins two text strings into one	=CONCATENATE(Text1, Text2)
UPPER/LOWER	Converts a string to upper or lower case	=UPPER/LOWER (Text)
LEFT/RIGHT/MID	Returns a number of characters from the start/middle/end of a text 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])

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

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)
HOUR/MINUTE/ SECOND	Returns the hour (0-23), minute (0-59), or second (0-59) of a given datetime value	=HOUR/MINUTE/SECOND(Datetime)
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 given interval (day, hour, year, etc.)	=DATEDIFF(Date1, Date2, Interval)

Related

RELATED	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]*

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.)

Tip: Instead of using RELATED to create extra columns (which increases file size), **nest it within measures like FILTER or SUMX**

Calculate

CALCULATE	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])*

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

ALL

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]*

Tip: Instead of adding filter context, **the ALL function removes it**. This is often used in **"% of Total"** calculations, when the denominator needs to remain fixed regardless of filter context

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]*

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

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 Functions

Iterator (or **"X"**) **functions** allow you to loop through the same expression on each row of a table, 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[Retail Price] * Sales[Quantity]*

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 comparison

Performance To-Date

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

*Use **DATESYTD** for Years, **DATESQTD** for Quarters, **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 (e.g. previous month, rolling 10-day)*

Running Total

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

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

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*