

Contents

Data Analysis Expressions (DAX) Reference

Learn

[DAX overview](#)

[Videos](#)

[Use DAX in Power BI Desktop Learn path](#)

[Sample model](#)

[Best practices](#)

[Appropriate use of error functions](#)

[Avoid converting BLANKs to values](#)

[Avoid using FILTER as a filter argument](#)

[Column and measure references](#)

[DIVIDE function vs divide operator \(/\)](#)

[Use SELECTEDVALUE instead of VALUES](#)

[Use COUNTROWS instead of COUNT](#)

[Use variables to improve formulas](#)

DAX functions

[DAX function reference](#)

[New DAX functions](#)

[Aggregation functions](#)

[Aggregation functions overview](#)

[APPROXIMATEDISTINCTCOUNT](#)

[AVERAGE](#)

[AVERAGEA](#)

[AVERAGEX](#)

[COUNT](#)

[COUNTA](#)

[COUNTAX](#)

[COUNTBLANK](#)

[COUNTROWS](#)

COUNTX

DISTINCTCOUNT

DISTINCTCOUNTNOBLANK

MAX

MAXA

MAXX

MIN

MINA

MINX

PRODUCT

PRODUCTX

SUM

SUMX

Date and time functions

Date and time functions overview

CALENDAR

CALENDARAUTO

DATE

DATEDIFF

DATEVALUE

DAY

EDATE

EOMONTH

HOUR

MINUTE

MONTH

NETWORKDAYS

NOW

QUARTER

SECOND

TIME

TIMEVALUE

TODAY

UTCNOW

UTCTODAY

WEEKDAY

WEEKNUM

YEAR

YEARFRAC

Filter functions

[Filter functions overview](#)

ALL

ALLCROSSFILTERED

ALLEXCEPT

ALLNOBLANKROW

ALLSELECTED

CALCULATE

CALCULATETABLE

EARLIER

EARLIEST

FILTER

KEEPFILTERS

LOOKUPVALUE

REMOVEFILTERS

SELECTEDVALUE

Financial functions

[Financial functions overview](#)

ACCRINT

ACCRINTM

AMORDEGRC

AMORLINC

COUPDAYBS

COUPDAYS

COUPDAYSNC

COUPNCD
COUPNUM
COUPPCD
CUMIPMT
CUMPRINC
DB
DDB
DISC
DOLLARDE
DOLLARFR
DURATION
EFFECT
FV
INTRATE
IPMT
ISPMT
MDURATION
NOMINAL
NPER
ODDFPRICE
ODDFYIELD
ODDLPRICE
ODDLYIELD
PDURATION
PMT
PPMT
PRICE
PRICEDISC
PRICEMAT
PV
RATE
RECEIVED

RRI
SLN
SYD
TBILLEQ
TBILLPRICE
TBILLYIELD
VDB
XIRR
XNPV
YIELD
YIELDDISC
YIELDMAT

Information functions

[Information functions overview](#)

COLUMNSTATISTICS
CONTAINS
CONTAINSROW
CONTAINSSTRING
CONTAINSSTRINGEXACT
CUSTOMDATA
HASONEFILTER
HASONEVALUE
ISAFTER
ISBLANK
ISCROSSFILTERED
ISEMPTY
ISERROR
ISEVEN
ISFILTERED
ISINSCOPE
ISLOGICAL
ISNONTEXT

ISNUMBER

ISODD

ISONORAFTER

ISSELECTEDMEASURE

ISSUBTOTAL

ISTEXT

NONVISUAL

SELECTEDMEASURE

SELECTEDMEASUREFORMATSTRING

SELECTEDMEASURENAME

USERCULTURE

USERNAME

USEROBJECTID

USERPRINCIPALNAME

Logical functions

Logical functions overview

AND

BITAND

BITLSHIFT

BITOR

BITRSHIFT

BITXOR

COALESCE

FALSE

IF

IF.EAGER

IFERROR

NOT

OR

SWITCH

TRUE

Math and trig functions

Math and trig functions overview

ABS

ACOS

ACOSH

ACOT

ACOTH

ASIN

ASINH

ATAN

ATANH

CEILING

CONVERT

COS

COSH

COT

COTH

CURRENCY

DEGREES

DIVIDE

EVEN

EXP

FACT

FLOOR

GCD

INT

ISO.CEILING

LCM

LN

LOG

LOG10

MOD

MROUND

[ODD](#)

[PI](#)

[POWER](#)

[QUOTIENT](#)

[RADIANS](#)

[RAND](#)

[RANDBETWEEN](#)

[ROUND](#)

[ROUNDDOWN](#)

[ROUNDUP](#)

[SIGN](#)

[SIN](#)

[SINH](#)

[SQRT](#)

[SQRTPI](#)

[TAN](#)

[TANH](#)

[TRUNC](#)

[Other functions](#)

[Other functions overview](#)

[BLANK](#)

[ERROR](#)

[Parent and child functions](#)

[Parent and child functions overview](#)

[Understanding functions for parent-child hierarchies](#)

[PATH](#)

[PATHCONTAINS](#)

[PATHITEM](#)

[PATHITEMREVERSE](#)

[PATHLENGTH](#)

[Relationship functions](#)

[Relationship functions](#)

[CROSSFILTER](#)

[RELATED](#)

[RELATEDTABLE](#)

[USERELATIONSHIP](#)

[Statistical functions](#)

[Statistical functions overview](#)

[BETA.DIST](#)

[BETA.INV](#)

[CHISQ.DIST](#)

[CHISQ.DIST.RT](#)

[CHISQ.INV](#)

[CHISQ.INV.RT](#)

[COMBIN](#)

[COMBINA](#)

[CONFIDENCE.NORM](#)

[CONFIDENCE.T](#)

[EXPON.DIST](#)

[GEOMEAN](#)

[GEOMEANX](#)

[MEDIAN](#)

[MEDIANX](#)

[NORM.DIST](#)

[NORM.INV](#)

[NORM.S.DIST](#)

[NORM.S.INV](#)

[PERCENTILE.EXC](#)

[PERCENTILE.INC](#)

[PERCENTILEX.EXC](#)

[PERCENTILEX.INC](#)

[PERMUT](#)

[POISSON.DIST](#)

[RANK.EQ](#)

RANKX
SAMPLE
STDEV.S
STDEV.P
STDEVX.S
STDEVX.P
T.DIST
T.DIST.2T
T.DIST.RT
T.INV
T.INV.2T
VAR.S
VAR.P
VARX.S
VARX.P

Table manipulation functions

[Table manipulation functions overview](#)

ADDCOLUMNS
ADDMISSINGITEMS
CROSSJOIN
CURRENTGROUP
DATATABLE
DETAILROWS
DISTINCT (column)
DISTINCT (table)
EXCEPT
FILTERS
GENERATE
GENERATEALL
GENERATESERIES
GROUPBY
IGNORE

[INTERSECT](#)

[NATURALINNERJOIN](#)

[NATURALLEFTOUTERJOIN](#)

[ROLLUP](#)

[ROLLUPADDISUBTOTAL](#)

[ROLLUPGROUP](#)

[ROLLUPISUBTOTAL](#)

[ROW](#)

[SELECTCOLUMNS](#)

[SUBSTITUTEWITHINDEX](#)

[SUMMARIZE](#)

[SUMMARIZECOLUMNS](#)

[Table constructor](#)

[TOPN](#)

[TREATAS](#)

[UNION](#)

[VALUES](#)

[Text functions](#)

[Text functions overview](#)

[COMBINEVALUES](#)

[CONCATENATE](#)

[CONCATENATEX](#)

[EXACT](#)

[FIND](#)

[FIXED](#)

[FORMAT](#)

[LEFT](#)

[LEN](#)

[LOWER](#)

[MID](#)

[REPLACE](#)

[REPT](#)

RIGHT
SEARCH
SUBSTITUTE
TRIM
UNICHAR
UNICODE
UPPER
VALUE

Time intelligence functions

[Time intelligence functions overview](#)

CLOSINGBALANCEMONTH
CLOSINGBALANCEQUARTER
CLOSINGBALANCEYEAR
DATEADD
DATESBETWEEN
DATESINPERIOD
DATESMTD
DATESQTD
DATESYTD
ENDOFMONTH
ENDOFQUARTER
ENDOFYEAR
FIRSTDATE
FIRSTNONBLANK
FIRSTNONBLANKVALUE
LASTDATE
LASTNONBLANK
LASTNONBLANKVALUE
NEXTDAY
NEXTMONTH
NEXTQUARTER
NEXTYEAR

OPENINGBALANCEMONTH
OPENINGBALANCEQUARTER
OPENINGBALANCEYEAR
PARALLELPERIOD
PREVIOUSDAY
PREVIOUSMONTH
PREVIOUSQUARTER
PREVIOUSYEAR
SAMEPERIODLASTYEAR
STARTOFMONTH
STARTOFQUARTER
STARTOFYEAR
TOTALMTD
TOTALQTD
TOTALYTD

DAX statements

Statements overview

DEFINE

EVALUATE

MEASURE

ORDER BY

START AT

VAR

DAX glossary

DAX operators

DAX queries

DAX parameter-naming

DAX syntax

DAX overview

10/31/2022 • 30 minutes to read • [Edit Online](#)

Data Analysis Expressions (DAX) is a formula expression language used in Analysis Services, Power BI, and Power Pivot in Excel. DAX formulas include functions, operators, and values to perform advanced calculations and queries on data in related tables and columns in tabular data models.

This article provides only a basic introduction to the most important concepts in DAX. It describes DAX as it applies to all the products that use it. Some functionality may not apply to certain products or use cases. Refer to your product's documentation describing its particular implementation of DAX.

Calculations

DAX formulas are used in measures, calculated columns, calculated tables, and row-level security.

Measures

Measures are dynamic calculation formulas where the results change depending on context. Measures are used in reporting that support combining and filtering model data by using multiple attributes such as a Power BI report or Excel PivotTable or PivotChart. Measures are created by using the DAX formula bar in the model designer.

A formula in a measure can use standard aggregation functions automatically created by using the Autosum feature, such as COUNT or SUM, or you can define your own formula by using the DAX formula bar. Named measures can be passed as an argument to other measures.

When you define a formula for a measure in the formula bar, a Tooltip feature shows a preview of what the results would be for the total in the current context, but otherwise the results are not immediately output anywhere. The reason you cannot see the (filtered) results of the calculation immediately is because the result of a measure cannot be determined without context. To evaluate a measure requires a reporting client application that can provide the context needed to retrieve the data relevant to each cell and then evaluate the expression for each cell. That client might be an Excel PivotTable or PivotChart, a Power BI report, or a table expression in a DAX query in SQL Server Management Studio (SSMS).

Regardless of the client, a separate query is run for each cell in the results. That is to say, each combination of row and column headers in a PivotTable, or each selection of slicers and filters in a Power BI report, generates a different subset of data over which the measure is calculated. For example, using this very simple measure formula:

```
Total Sales = SUM([Sales Amount])
```

When a user places the TotalSales measure in a report, and then places the Product Category column from a Product table into Filters, the sum of Sales Amount is calculated and displayed for each product category.

Unlike calculated columns, the syntax for a measure includes the measure's name preceding the formula. In the example just provided, the name **Total Sales** appears preceding the formula. After you've created a measure, the name and its definition appear in the reporting client application Fields list, and depending on perspectives and roles is available to all users of the model.

To learn more, see:

[Measures in Power BI Desktop](#)

[Measures in Analysis Services](#)

Calculated columns

A calculated column is a column that you add to an existing table (in the model designer) and then create a DAX formula that defines the column's values. When a calculated column contains a valid DAX formula, values are calculated for each row as soon as the formula is entered. Values are then stored in the in-memory data model. For example, in a Date table, when the formula is entered into the formula bar:

```
= [Calendar Year] & " Q" & [Calendar Quarter]
```

A value for each row in the table is calculated by taking values from the Calendar Year column (in the same Date table), adding a space and the capital letter Q, and then adding the values from the Calendar Quarter column (in the same Date table). The result for each row in the calculated column is calculated immediately and appears, for example, as **2017 Q1**. Column values are only recalculated if the table or any related table is processed (refresh) or the model is unloaded from memory and then reloaded, like when closing and reopening a Power BI Desktop file.

To learn more, see:

[Calculated columns in Power BI Desktop](#)

[Calculated columns in Analysis Services](#)

[Calculated Columns in Power Pivot.](#)

Calculated tables

A calculated table is a computed object, based on a formula expression, derived from all or part of other tables in the same model. Instead of querying and loading values into your new table's columns from a data source, a DAX formula defines the table's values.

Calculated tables can be helpful in a role-playing dimension. An example is the Date table, as OrderDate, ShipDate, or DueDate, depending on the foreign key relationship. By creating a calculated table for ShipDate explicitly, you get a standalone table that is available for queries, as fully operable as any other table. Calculated tables are also useful when configuring a filtered rowset, or a subset or superset of columns from other existing tables. This allows you to keep the original table intact while creating variations of that table to support specific scenarios.

Calculated tables support relationships with other tables. The columns in your calculated table have data types, formatting, and can belong to a data category. Calculated tables can be named, and surfaced or hidden just like any other table. Calculated tables are re-calculated if any of the tables it pulls data from are refreshed or updated.

To learn more, see:

[Calculated tables in Power BI Desktop](#)

[Calculated tables in Analysis Services.](#)

Row-level security

With row-level security, a DAX formula must evaluate to a Boolean TRUE/FALSE condition, defining which rows can be returned by the results of a query by members of a particular role. For example, for members of the Sales role, the Customers table with the following DAX formula:

```
= Customers[Country] = "USA"
```

Members of the Sales role will only be able to view data for customers in the USA, and aggregates, such as SUM are returned only for customers in the USA. Row-level security is not available in Power Pivot in Excel.

When defining row-level security by using DAX formula, you are creating an allowed row set. This does not

deny access to other rows; rather, they are simply not returned as part of the allowed row set. Other roles can allow access to the rows excluded by the DAX formula. If a user is a member of another role, and that role's row-level security allows access to that particular row set, the user can view data for that row.

Row-level security formulas apply to the specified rows as well as related rows. When a table has multiple relationships, filters apply security for the relationship that is active. Row-level security formulas will be intersected with other formulas defined for related tables.

To learn more, see:

[Row-level security \(RLS\) with Power BI Roles in Analysis Services](#)

Queries

DAX queries can be created and run in SQL Server Management Studio (SSMS) and open-source tools like DAX Studio (daxstudio.org). Unlike DAX calculation formulas, which can only be created in tabular data models, DAX queries can also be run against Analysis Services Multidimensional models. DAX queries are often easier to write and more efficient than Multidimensional Data Expressions (MDX) queries.

A DAX query is a statement, similar to a SELECT statement in T-SQL. The most basic type of DAX query is an *evaluate* statement. For example,

```
EVALUATE
( FILTER ( 'DimProduct', [SafetyStockLevel] < 200 ) )
ORDER BY [EnglishProductName] ASC
```

Returns in Results a table listing only those products with a SafetyStockLevel less than 200, in ascending order by EnglishProductName.

You can create measures as part of the query. Measures exist only for the duration of the query. To learn more, see [DAX queries](#).

Formulas

DAX formulas are essential for creating calculations in calculated columns and measures, and securing your data by using row-level security. To create formulas for calculated columns and measures, use the formula bar along the top of the model designer window or the DAX Editor. To create formulas for row-level security, use the Role Manager or Manage roles dialog box. Information in this section is meant to get you started with understanding the basics of DAX formulas.

Formula basics

DAX formulas can be very simple or quite complex. The following table shows some examples of simple formulas that could be used in a calculated column.

| FORMULA | DEFINITION |
|--------------------------------------|---|
| <code>= TODAY()</code> | Inserts today's date in every row of a calculated column. |
| <code>= 3</code> | Inserts the value 3 in every row of a calculated column. |
| <code>= [Column1] + [Column2]</code> | Adds the values in the same row of [Column1] and [Column2] and puts the results in the calculated column of the same row. |

Whether the formula you create is simple or complex, you can use the following steps when building a formula:

1. Each formula must begin with an equal sign (=).
2. You can either type or select a function name, or type an expression.
3. Begin to type the first few letters of the function or name you want, and AutoComplete displays a list of available functions, tables, and columns. Press TAB to add an item from the AutoComplete list to the formula.

You can also click the Fx button to display a list of available functions. To select a function from the dropdown list, use the arrow keys to highlight the item, and click **OK** to add the function to the formula.

4. Supply the arguments to the function by selecting them from a dropdown list of possible tables and columns, or by typing in values.
5. Check for syntax errors: ensure that all parentheses are closed and columns, tables and values are referenced correctly.
6. Press ENTER to accept the formula.

NOTE

In a calculated column, as soon as you enter the formula and the formula is validated, the column is populated with values. In a measure, pressing ENTER saves the measure definition with the table. If a formula is invalid, an error is displayed.

In this example, let's look at a formula in a measure named **Days in Current Quarter**:

```
Days in Current Quarter = COUNTROWS( DATESBETWEEN( 'Date'[Date], STARTOFQUARTER( LASTDATE('Date'[Date])),  
ENDOFQUARTER('Date'[Date])))
```

This measure is used to create a comparison ratio between an incomplete period and the previous period. The formula must take into account the proportion of the period that has elapsed, and compare it to the same proportion in the previous period. In this case, [Days Current Quarter to Date]/[Days in Current Quarter] gives the proportion elapsed in the current period.

This formula contains the following elements:

| FORMULA ELEMENT | DESCRIPTION |
|-------------------------|--|
| Days in Current Quarter | The name of the measure. |
| = | The equals sign (=) begins the formula. |
| COUNTROWS | COUNTROWS counts the number of rows in the Date table |
| () | Open and closing parenthesis specifies arguments. |
| DATESBETWEEN | The DATESBETWEEN function returns the dates between the last date for each value in the Date column in the Date table. |
| 'Date' | Specifies the Date table. Tables are in single quotes. |
| [Date] | Specifies the Date column in the Date table. Columns are in brackets. |

| FORMULA ELEMENT | DESCRIPTION |
|-----------------|---|
| , | |
| STARTOFQUARTER | The STARTOFQUARTER function returns the date of the start of the quarter. |
| LASTDATE | The LASTDATE function returns the last date of the quarter. |
| 'Date' | Specifies the Date table. |
| [Date] | Specifies the Date column in the Date table. |
| , | |
| ENDOFQUARTER | The ENDOFQUARTER function |
| 'Date' | Specifies the Date table. |
| [Date] | Specifies the Date column in the Date table. |

Using formula AutoComplete

AutoComplete helps you enter a valid formula syntax by providing you with options for each element in the formula.

- You can use formula AutoComplete in the middle of an existing formula with nested functions. The text immediately before the insertion point is used to display values in the drop-down list, and all of the text after the insertion point remains unchanged.
- AutoComplete does not add the closing parenthesis of functions or automatically match parentheses. You must make sure that each function is syntactically correct or you cannot save or use the formula.

Using multiple functions in a formula

You can nest functions, meaning that you use the results from one function as an argument of another function. You can nest up to 64 levels of functions in calculated columns. However, nesting can make it difficult to create or troubleshoot formulas. Many functions are designed to be used solely as nested functions. These functions return a table, which cannot be directly saved as a result; it must be provided as input to a table function. For example, the functions SUMX, AVERAGEX, and MINX all require a table as the first argument.

Functions

A function is a named formula within an expression. Most functions have required and optional arguments, also known as parameters, as input. When the function is executed, a value is returned. DAX includes functions you can use to perform calculations using dates and times, create conditional values, work with strings, perform lookups based on relationships, and the ability to iterate over a table to perform recursive calculations. If you are familiar with Excel formulas, many of these functions will appear very similar; however, DAX formulas are different in the following important ways:

- A DAX function always references a complete column or a table. If you want to use only particular values from a table or column, you can add filters to the formula.
- If you need to customize calculations on a row-by-row basis, DAX provides functions that let you use the current row value or a related value as a kind of parameter, to perform calculations that vary by context. To understand how these functions work, see [Context](#) in this article.

- DAX includes many functions that return a table, rather than a value. The table is not displayed in a reporting client, but is used to provide input to other functions. For example, you can retrieve a table and then count the distinct values in it, or calculate dynamic sums across filtered tables or columns.
- DAX functions include a variety of *time intelligence* functions. These functions let you define or select date ranges, and perform dynamic calculations based on these dates or range. For example, you can compare sums across parallel periods.

Aggregation functions

Aggregation functions calculate a (scalar) value such as count, sum, average, minimum, or maximum for all rows in a column or table as defined by the expression. To learn more, see [Aggregation functions](#).

Date and time functions

The date and time functions in DAX are similar to date and time functions in Microsoft Excel. However, DAX functions are based on a **datetime** data type starting March 1, 1900. To learn more, see [Date and time functions](#).

Filter functions

The filter functions in DAX return specific data types, look up values in related tables, and filter by related values. The lookup functions work by using tables and relationships, like a database. The filtering functions let you manipulate data context to create dynamic calculations. To learn more, see [Filter functions](#).

Financial functions

The financial functions in DAX are used in formulas that perform financial calculations, such as net present value and rate of return. These functions are similar to financial functions used in Microsoft Excel. To learn more, see [Financial functions](#).

Information functions

An information function looks at the cell or row that is provided as an argument and tells you whether the value matches the expected type. For example, the ISERROR function returns TRUE if the value that you reference contains an error. To learn more, see [Information functions](#).

Logical functions

Logical functions act upon an expression to return information about the values in the expression. For example, the TRUE function lets you know whether an expression that you are evaluating returns a TRUE value. To learn more, see [Logical functions](#).

Mathematical and trigonometric functions

The mathematical functions in DAX are very similar to the Excel mathematical and trigonometric functions. Some minor differences exist in the numeric data types used by DAX functions. To learn more, see [Math and trig functions](#).

Other functions

These functions perform unique actions that cannot be defined by any of the categories most other functions belong to. To learn more, see [Other functions](#).

Relationship functions

Relationship functions in DAX allow you to return values from another related table, specify a particular relationship to use in an expression, and specify cross filtering direction. To learn more, see [Relationship functions](#).

Statistical functions

Statistical functions calculate values related to statistical distributions and probability, such as standard deviation and number of permutations. To learn more, see [Statistical functions](#).

Text functions

Text functions in DAX are very similar to their counterparts in Excel. You can return part of a string, search for text within a string, or concatenate string values. DAX also provides functions for controlling the formats for dates, times, and numbers. To learn more, see [Text functions](#).

Time intelligence functions

The time intelligence functions provided in DAX let you create calculations that use built-in knowledge about calendars and dates. By using time and date ranges in combination with aggregations or calculations, you can build meaningful comparisons across comparable time periods for sales, inventory, and so on. To learn more, see [Time intelligence functions \(DAX\)](#).

Table manipulation functions

These functions return a table or manipulate existing tables. For example, by using ADDCOLUMNS you can add calculated columns to a specified table, or you can return a summary table over a set of groups with the SUMMARIZECOLUMNS function. To learn more, see [Table manipulation functions](#).

Variables

You can create variables within an expression by using [VAR](#). VAR is technically not a function, it's a keyword to store the result of an expression as a named variable. That variable can then be passed as an argument to other measure expressions. For example:

```
VAR
    TotalQty = SUM ( Sales[Quantity] )

Return
    IF (
        TotalQty > 1000,
        TotalQty * 0.95,
        TotalQty * 1.25
    )
```

In this example, TotalQty can be passed as a named variable to other expressions. Variables can be of any scalar data type, including tables. Using variables in your DAX formulas can be incredibly powerful.

Data types

You can import data into a model from many different data sources that might support different data types. When you import data into a model, the data is converted to one of the tabular model data types. When the model data is used in a calculation, the data is then converted to a DAX data type for the duration and output of the calculation. When you create a DAX formula, the terms used in the formula will automatically determine the value data type returned.

DAX supports the following data types:

| DATA TYPE IN MODEL | DATA TYPE IN DAX | DESCRIPTION |
|--------------------|--|---|
| Whole Number | A 64 bit (eight-bytes) integer value ^{1, 2} | Numbers that have no decimal places. Integers can be positive or negative numbers, but must be whole numbers between -9,223,372,036,854,775,808 (-2^63) and 9,223,372,036,854,775,807 (2^63-1). |

| DATA TYPE IN MODEL | DATA TYPE IN DAX | DESCRIPTION |
|--------------------|--|---|
| Decimal Number | A 64 bit (eight-bytes) real number ^{1, 2} | <p>Real numbers are numbers that can have decimal places. Real numbers cover a wide range of values:</p> <p>Negative values from -1.79E +308 through -2.23E -308</p> <p>Zero</p> <p>Positive values from 2.23E -308 through 1.79E + 308</p> <p>However, the number of significant digits is limited to 17 decimal digits.</p> |
| Boolean | Boolean | Either a True or False value. |
| Text | String | A Unicode character data string. Can be strings, numbers or dates represented in a text format. |
| Date | Date/time | <p>Dates and times in an accepted date-time representation.</p> <p>Valid dates are all dates after March 1, 1900.</p> |
| Currency | Currency | Currency data type allows values between -922,337,203,685,477.5808 to 922,337,203,685,477.5807 with four decimal digits of fixed precision. |
| N/A | Blank | A blank is a data type in DAX that represents and replaces SQL nulls. You can create a blank by using the BLANK function, and test for blanks by using the logical function, ISBLANK. |

Tabular data models also include the *Table* data type as the input or output to many DAX functions. For example, the FILTER function takes a table as input and outputs another table that contains only the rows that meet the filter conditions. By combining table functions with aggregation functions, you can perform complex calculations over dynamically defined data sets.

While data types are typically automatically set, it is important to understand data types and how they apply, in-particular, to DAX formulas. Errors in formulas or unexpected results, for example, are often caused by using a particular operator that cannot be used with a data type specified in an argument. For example, the formula, `= 1 & 2`, returns a string result of 12. The formula, `= "1" + "2"`, however, returns an integer result of 3.

Context

Context is an important concept to understand when creating DAX formulas. Context is what enables you to perform dynamic analysis, as the results of a formula change to reflect the current row or cell selection and also any related data. Understanding context and using context effectively are critical for building high-performing, dynamic analyses, and for troubleshooting problems in formulas.

Formulas in tabular models can be evaluated in a different context, depending on other design elements:

- Filters applied in a PivotTable or report
- Filters defined within a formula
- Relationships specified by using special functions within a formula

There are different types of context: *row context*, *query context*, and *filter context*.

Row context

Row context can be thought of as "the current row". If you create a formula in a calculated column, the row context for that formula includes the values from all columns in the current row. If the table is related to another table, the content also includes all the values from the other table that are related to the current row.

For example, suppose you create a calculated column, `= [Freight] + [Tax]`, that adds together values from two columns, Freight and Tax, from the same table. This formula automatically gets only the values from the current row in the specified columns.

Row context also follows any relationships that have been defined between tables, including relationships defined within a calculated column by using DAX formulas, to determine which rows in related tables are associated with the current row.

For example, the following formula uses the RELATED function to fetch a tax value from a related table, based on the region that the order was shipped to. The tax value is determined by using the value for region in the current table, looking up the region in the related table, and then getting the tax rate for that region from the related table.

```
= [Freight] + RELATED('Region'[TaxRate])
```

This formula gets the tax rate for the current region from the Region table and adds it to the value of the Freight column. In DAX formulas, you do not need to know or specify the specific relationship that connects the tables.

Multiple row context

DAX includes functions that iterate calculations over a table. These functions can have multiple current rows, each with its own row context. In essence, these functions let you create formulas that perform operations recursively over an inner and outer loop.

For example, suppose your model contains a **Products** table and a **Sales** table. Users might want to go through the entire sales table, which is full of transactions involving multiple products, and find the largest quantity ordered for each product in any one transaction.

With DAX you can build a single formula that returns the correct value, and the results are automatically updated any time a user adds data to the tables.

```
= MAXX(FILTER(Sales,[ProdKey] = EARLIER([ProdKey])),Sales[OrderQty])
```

For a detailed example of this formula, see [EARLIER](#).

To summarize, the EARLIER function stores the row context from the operation that preceded the current operation. At all times, the function stores in memory two sets of context: one set of context represents the current row for the inner loop of the formula, and another set of context represents the current row for the outer loop of the formula. DAX automatically feeds values between the two loops so that you can create complex aggregates.

Query context

Query context refers to the subset of data that is implicitly retrieved for a formula. For example, when a user places a measure or field into a report, the engine examines row and column headers, slicers, and report filters to determine the context. The necessary queries are then run against model data to get the correct subset of

data, make the calculations defined by the formula, and then populate values in the report.

Because context changes depending on where you place the formula, the results of the formula can also change. For example, suppose you create a formula that sums the values in the **Profit** column of the **Sales** table:

`= SUM('Sales'[Profit])`. If you use this formula in a calculated column within the **Sales** table, the results for the formula will be the same for the entire table, because the query context for the formula is always the entire data set of the **Sales** table. Results will have profit for all regions, all products, all years, and so on.

However, users typically don't want to see the same result hundreds of times, but instead want to get the profit for a particular year, a particular country, a particular product, or some combination of these, and then get a grand total.

In a report, context is changed by filtering, adding or removing fields, and using slicers. For each change, the query context in which the measure is evaluated. Therefore, the same formula, used in a measure, is evaluated in a different *query context* for each cell.

Filter context

Filter context is the set of values allowed in each column, or in the values retrieved from a related table. Filters can be applied to the column in the designer, or in the presentation layer (reports and PivotTables). Filters can also be defined explicitly by filter expressions within the formula.

Filter context is added when you specify filter constraints on the set of values allowed in a column or table, by using arguments to a formula. Filter context applies on top of other contexts, such as row context or query context.

In tabular models, there are many ways to create filter context. Within the context of clients that can consume the model, such as Power BI reports, users can create filters on the fly by adding slicers or report filters on the row and column headings. You can also specify filter expressions directly within the formula, to specify related values, to filter tables that are used as inputs, or to dynamically get context for the values that are used in calculations. You can also completely clear or selectively clear the filters on particular columns. This is very useful when creating formulas that calculate grand totals.

To learn more about how to create filters within formulas, see the [FILTER Function \(DAX\)](#).

For an example of how filters can be cleared to create grand totals, see the [ALL Function \(DAX\)](#).

For examples of how to selectively clear and apply filters within formulas, see [ALLEXCEPT](#).

Determining context in formulas

When you create a DAX formula, the formula is first tested for valid syntax, and then tested to make sure the names of the columns and tables included in the formula can be found in the current context. If any column or table specified by the formula cannot be found, an error is returned.

Context during validation (and recalculation operations) is determined as described in the preceding sections, by using the available tables in the model, any relationships between the tables, and any filters that have been applied.

For example, if you have just imported some data into a new table and it is not related to any other tables (and you have not applied any filters), the *current context* is the entire set of columns in the table. If the table is linked by relationships to other tables, the current context includes the related tables. If you add a column from the table to a report that has Slicers and maybe some report filters, the context for the formula is the subset of data in each cell of the report.

Context is a powerful concept that can also make it difficult to troubleshoot formulas. We recommend that you begin with simple formulas and relationships to see how context works. The following section provides some examples of how formulas use different types of context to dynamically return results.

Operators

The DAX language uses four different types of calculation operators in formulas:

- Comparison operators to compare values and return a logical TRUE\FALSE value.
- Arithmetic operators to perform arithmetic calculations that return numeric values.
- Text concatenation operators to join two or more text strings.
- Logical operators that combine two or more expressions to return a single result.

For detailed information about operators used in DAX formulas, see [DAX operators](#).

Working with tables and columns

Tables in tabular data models look like Excel tables, but are different in the way they work with data and with formulas:

- Formulas work only with tables and columns, not with individual cells, range references, or arrays.
- Formulas can use relationships to get values from related tables. The values that are retrieved are always related to the current row value.
- You cannot have irregular or "ragged" data like you can in an Excel worksheet. Each row in a table must contain the same number of columns. However, you can have empty values in some columns. Excel data tables and tabular model data tables are not interchangeable.
- Because a data type is set for each column, each value in that column must be of the same type.

Referring to tables and columns in formulas

You can refer to any table and column by using its name. For example, the following formula illustrates how to refer to columns from two tables by using the *fully qualified* name:

```
= SUM('New Sales'[Amount]) + SUM('Past Sales'[Amount])
```

When a formula is evaluated, the model designer first checks for general syntax, and then checks the names of columns and tables that you provide against possible columns and tables in the current context. If the name is ambiguous or if the column or table cannot be found, you will get an error on your formula (an #ERROR string instead of a data value in cells where the error occurs). To learn more about naming requirements for tables, columns, and other objects, see Naming Requirements in [DAX syntax](#).

Table relationships

By creating relationships between tables, you gain the ability for related values in other tables to be used in calculations. For example, you can use a calculated column to determine all the shipping records related to the current reseller, and then sum the shipping costs for each. In many cases, however, a relationship might not be necessary. You can use the [LOOKUPVALUE](#) function in a formula to return the value in *result_columnName* for the row that meets criteria specified in the *search_column* and *search_value* arguments.

Many DAX functions require that a relationship exist between the tables, or among multiple tables, in order to locate the columns that you have referenced and return results that make sense. Other functions will attempt to identify the relationship; however, for best results you should always create a relationship where possible.

Tabular data models support multiple relationships among tables. To avoid confusion or incorrect results, only one relationship at a time is designated as the active relationship, but you can change the active relationship as necessary to traverse different connections in the data in calculations. [USERELATIONSHIP](#) function can be used to specify one or more relationships to be used in a specific calculation.

It's important to observe these formula design rules when using relationships:

- When tables are connected by a relationship, you must ensure the two columns used as keys have values that match. Referential integrity is not enforced, therefore it is possible to have non-matching values in a key column and still create a relationship. If this happens, you should be aware that blank values or non-matching values might affect the results of formulas.
- When you link tables in your model by using relationships, you enlarge the scope, or *context*, in which your formulas are evaluated. Changes in context resulting from the addition of new tables, new relationships, or from changes in the active relationship can cause your results to change in ways that you might not anticipate. To learn more, see [Context](#) in this article.

Process and refresh

Process and *recalculation* are two separate but related operations. You should thoroughly understand these concepts when designing a model that contains complex formulas, large amounts of data, or data that is obtained from external data sources.

Process (refresh) is updating the data in a model with new data from an external data source.

Recalculation is the process of updating the results of formulas to reflect any changes to the formulas themselves and to reflect changes in the underlying data. Recalculation can affect performance in the following ways:

- The values in a calculated column are computed and stored in the model. To update the values in the calculated column, you must process the model using one of three processing commands – Process Full, Process Data, or Process Recalc. The result of the formula must always be recalculated for the entire column, whenever you change the formula.
- The values calculated by measures are dynamically evaluated whenever a user adds the measure to a PivotTable or open a report; as the user modifies the context, values returned by the measure change. The results of the measure always reflect the latest in the in-memory cache.

Processing and recalculation have no effect on row-level security formulas unless the result of a recalculation returns a different value, thus making the row queryable or not queryable by role members.

Updates

DAX is constantly being improved. [New and updated functions](#) are released with the next available update, which is usually monthly. Services are updated first, followed by installed applications like Power BI Desktop, Excel, SQL Server Management Studio (SSMS), and Analysis Services project extension for Visual Studio (SSDT). SQL Server Analysis Services is updated with the next cumulative update. New functions are first announced and described in the DAX function reference coinciding with Power BI Desktop updates.

Not all functions are supported in earlier versions of SQL Server Analysis Services and Excel.

Troubleshooting

If you get an error when defining a formula, the formula might contain either a *syntactic error*, *semantic error*, or *calculation error*.

Syntactic errors are the easiest to resolve. They typically involve a missing parenthesis or comma.

The other type of error occurs when the syntax is correct, but the value or a column referenced does not make sense in the context of the formula. Such semantic and calculation errors might be caused by any of the following problems:

- The formula refers to a non-existing column, table, or function.
- The formula appears to be correct, but when the data engine fetches the data, it finds a type mismatch and

raises an error.

- The formula passes an incorrect number or type of arguments to a function.
- The formula refers to a different column that has an error, and therefore its values are invalid.
- The formula refers to a column that has not been processed, meaning it has metadata but no actual data to use for calculations.

In the first four cases, DAX flags the entire column that contains the invalid formula. In the last case, DAX grays out the column to indicate that the column is in an unprocessed state.

Apps and tools

Power BI Desktop



[Power BI Desktop](#) is a free data modeling and reporting application. The model designer includes a DAX editor for creating DAX calculation formulas.

Power Pivot in Excel



The [Power Pivot in Excel](#) models designer includes a DAX editor for creating DAX calculation formulas.

Visual Studio



Visual Studio with [Analysis Services projects](#) extension (VSIX) is used to create Analysis Services model projects. Tabular model designer, installed with the projects extension includes a DAX editor.

SQL Server Management Studio



[SQL Server Management Studio](#) (SSMS) is an essential tool for working with Analysis Services. SSMS includes a DAX query editor for querying both tabular and multidimensional models.

DAX Studio



[DAX Studio](#) is an open-source client tool for creating and running DAX queries against Analysis Services, Power BI Desktop, and Power Pivot in Excel models.

Tabular Editor



[Tabular Editor](#) is an open-source tool that provides an intuitive, hierarchical view of every object in tabular model metadata. Tabular Editor includes a DAX Editor with syntax highlighting, which provides an easy way to edit measures, calculated column, and calculated table expressions.

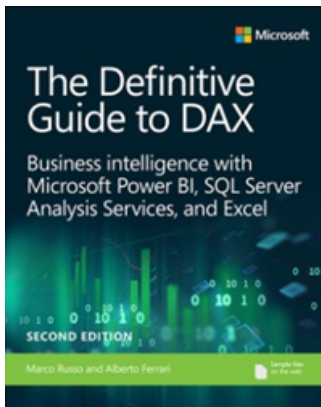
Learning resources

When learning DAX, it's best to use the application you'll be using to create your data models. Analysis Services, Power BI Desktop, and Power Pivot in Excel all have articles and tutorials that include lessons on creating measures, calculated columns, and row-filters by using DAX. Here are some additional resources:

Videos

[Use DAX in Power BI Desktop](#) learning path.

The [Definitive Guide to DAX](#) by Alberto Ferrari and Marco Russo (Microsoft Press). Now in its second edition, this extensive guide provides basics to innovative high-performance techniques for beginning data modelers and BI professionals.



Community

DAX has a vibrant community always willing to share their expertise. Microsoft [Power BI Community](#) has a special discussion forum just for DAX, [DAX Commands and Tips](#).

Videos

10/31/2022 • 2 minutes to read • [Edit Online](#)

Whether you're using Power BI Desktop, Power Pivot in Excel, or Analysis Services, learning Data Analysis Expressions (DAX) is essential to creating effective data models. Here are some videos to help you get started using this powerful expression language.

DAX 101

In this DAX 101 video, Microsoft Partner, Alberto Ferrari introduces essential concepts in DAX. With practical and clear examples, you will learn about measures, calculated columns, and basic data modeling expressions with DAX.

Advanced DAX

In this advanced DAX video, Microsoft Partner, Alberto Ferrari describes DAX theory, filter and row context, and other essential concepts in DAX.

DAX sample model

10/31/2022 • 2 minutes to read • [Edit Online](#)

The **Adventure Works DW 2020** Power BI Desktop sample model is designed to support your DAX learning. The model is based on the [Adventure Works data warehouse sample](#) for AdventureWorksDW2017—however, the data has been modified to suit the objectives of the sample model.

The sample model does not contain any DAX formulas. It does however support hundreds or even thousands of potential calculation formulas and queries. Some function examples, like those in **CALCULATE**, **DATESBETWEEN**, **DATESIN PERIOD**, **IF**, and **LOOKUPVALUE** can be added to the sample model without modification. We're working on including more examples in other function reference articles that work with the sample model.

Scenario



The Adventure Works company represents a bicycle manufacturer that sells bicycles and accessories to global markets. The company has their data warehouse data stored in an Azure SQL Database.

Model structure

The model has seven tables:

| TABLE | DESCRIPTION |
|-------------|---|
| Customer | Describes customers and their geographic location. Customers purchase products online (Internet sales). |
| Date | There are three relationships between the Date and Sales tables, for order date, ship date, and due date. The order date relationship is active. The company's reports sales using a fiscal year that commences on July 1 of each year. The table is marked as a date table using the Date column. |
| Product | Stores finished products only. |
| Reseller | Describes resellers and their geographic location. Reseller on sell products to their customers. |
| Sales | Stores rows at sales order line grain. All financial values are in US dollars (USD). The earliest order date is July 1, 2017, and the latest order date is June 15, 2020. |
| Sales Order | Describes sales order and order line numbers, and also the sales channel, which is either Reseller or Internet . This table has a one-to-one relationship with the Sales table. |

| TABLE | DESCRIPTION |
|-----------------|--|
| Sales Territory | Sales territories are organized into groups (North America, Europe, and Pacific), countries, and regions. Only the United States sells products at the region level. |

Download sample

Download the Power BI Desktop sample model file [here](#).

See also

- Learning path: [Use DAX in Power BI Desktop](#)
- Questions? [Try asking the Power BI Community](#)
- Suggestions? [Contribute ideas to improve Power BI](#)

Appropriate use of error functions

10/31/2022 • 2 minutes to read • [Edit Online](#)

As a data modeler, when you write a DAX expression that might raise an evaluation-time error, you can consider using two helpful DAX functions.

- The **ISERROR** function, which takes a single expression and returns TRUE if that expression results in error.
- The **IFERROR** function, which takes two expressions. Should the first expression result in error, the value for the second expression is returned. It is in fact a more optimized implementation of nesting the ISERROR function inside an IF function.

However, while these functions can be helpful and can contribute to writing easy-to-understand expressions, they can also significantly degrade the performance of calculations. It can happen because these functions increase the number of storage engine scans required.

Most evaluation-time errors are due to unexpected BLANKs or zero values, or invalid data type conversion.

Recommendations

It's better to avoid using the ISERROR and IFERROR functions. Instead, apply defensive strategies when developing the model and writing expressions. Strategies can include:

- **Ensuring quality data is loaded into the model:** Use Power Query transformations to remove or substitute invalid or missing values, and to set correct data types. A Power Query transformation can also be used to filter rows when errors, like invalid data conversion, occur.

Data quality can also be controlled by setting the model column **Is Nullable** property to Off, which will fail the data refresh should BLANKs be encountered. If this failure occurs, data loaded as a result of a successful refresh will remain in the tables.

- **Using the IF function:** The IF function logical test expression can determine whether an error result would occur. Note, like the ISERROR and IFERROR functions, this function can result in additional storage engine scans, but will likely perform better than them as no error needs to be raised.
- **Using error-tolerant functions:** Some DAX functions will test and compensate for error conditions. These functions allow you to enter an alternate result that would be returned instead. The **DIVIDE** function is one such example. For additional guidance about this function, read the [DAX: DIVIDE function vs divide operator \(/\)](#) article.

Example

The following measure expression tests whether an error would be raised. It returns BLANK in this instance (which is the case when you do not provide the IF function with a value-if-false expression).

```
Profit Margin
= IF(ISERROR([Profit] / [Sales]))
```

This next version of the measure expression has been improved by using the IFERROR function in place of the IF and ISERROR functions.


```
Profit Margin  
= IFERROR([Profit] / [Sales], BLANK())
```

However, this final version of the measure expression achieves the same outcome, yet more efficiently and elegantly.

```
Profit Margin  
= DIVIDE([Profit], [Sales])
```

See also

- Learning path: [Use DAX in Power BI Desktop](#)
- Questions? [Try asking the Power BI Community](#)
- Suggestions? [Contribute ideas to improve Power BI](#)

Avoid converting BLANKs to values

10/31/2022 • 2 minutes to read • [Edit Online](#)

As a data modeler, when writing measure expressions you might come across cases where a meaningful value can't be returned. In these instances, you may be tempted to return a value—like zero—instead. It's suggested you carefully determine whether this design is efficient and practical.

Consider the following measure definition that explicitly converts BLANK results to zero.

```
Sales (No Blank) =  
IF(  
    ISBLANK([Sales]),  
    0,  
    [Sales]  
)
```

Consider another measure definition that also converts BLANK results to zero.

```
Profit Margin =  
DIVIDE([Profit], [Sales], 0)
```

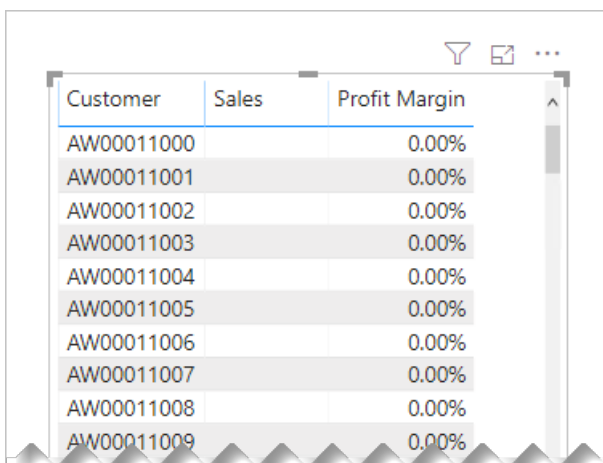
The **DIVIDE** function divides the **Profit** measure by the **Sales** measure. Should the result be zero or BLANK, the third argument—the alternate result (which is optional)—is returned. In this example, because zero is passed as the alternate result, the measure is guaranteed to always return a value.

These measure designs are inefficient and lead to poor report designs.

When they're added to a report visual, Power BI attempts to retrieve all groupings within the filter context. The evaluation and retrieval of large query results often leads to slow report rendering. Each example measure effectively turns a sparse calculation into a dense one, forcing Power BI to use more memory than necessary.

Also, too many groupings often overwhelm your report users.

Let's see what happens when the **Profit Margin** measure is added to a table visual, grouping by customer.



| Customer | Sales | Profit Margin |
|------------|-------|---------------|
| AW00011000 | | 0.00% |
| AW00011001 | | 0.00% |
| AW00011002 | | 0.00% |
| AW00011003 | | 0.00% |
| AW00011004 | | 0.00% |
| AW00011005 | | 0.00% |
| AW00011006 | | 0.00% |
| AW00011007 | | 0.00% |
| AW00011008 | | 0.00% |
| AW00011009 | | 0.00% |

The table visual displays an overwhelming number of rows. (There are in fact 18,484 customers in the model, and so the table attempts to display all of them.) Notice that the customers in view haven't achieved any sales. Yet, because the **Profit Margin** measure always returns a value, they are displayed.

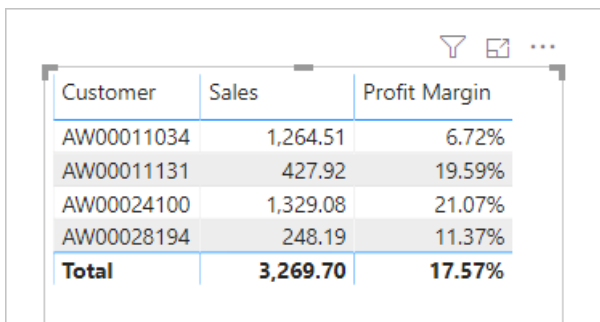
NOTE

When there are too many data points to display in a visual, Power BI may use data reduction strategies to remove or summarize large query results. For more information, see [Data point limits and strategies by visual type](#).

Let's see what happens when the **Profit Margin** measure definition is improved. It now returns a value only when the **Sales** measure isn't BLANK (or zero).

```
Profit Margin =  
DIVIDE([Profit], [Sales])
```

The table visual now displays only customers who have made sales within the current filter context. The improved measure results in a more efficient and practical experience for your report users.



A screenshot of a Power BI table visual. The table has three columns: 'Customer', 'Sales', and 'Profit Margin'. It displays five rows of data, including a 'Total' row at the bottom. The 'Sales' column is formatted with commas as thousands separators, and the 'Profit Margin' column is formatted as percentages. The table is part of a larger visual, as indicated by the filter and view icons at the top right.

| Customer | Sales | Profit Margin |
|--------------|-----------------|---------------|
| AW00011034 | 1,264.51 | 6.72% |
| AW00011131 | 427.92 | 19.59% |
| AW00024100 | 1,329.08 | 21.07% |
| AW00028194 | 248.19 | 11.37% |
| Total | 3,269.70 | 17.57% |

TIP

When necessary, you can configure a visual to display all groupings (that return values or BLANK) within the filter context by enabling the [Show Items With No Data](#) option.

Recommendation

It's recommended that your measures return BLANK when a meaningful value cannot be returned.

This design approach is efficient, allowing Power BI to render reports faster. Also, returning BLANK is better because report visuals—by default—eliminate groupings when summarizations are BLANK.

See also

- Learning path: [Use DAX in Power BI Desktop](#)
- Questions? [Try asking the Power BI Community](#)
- Suggestions? [Contribute ideas to improve Power BI](#)

Avoid using FILTER as a filter argument

10/31/2022 • 2 minutes to read • [Edit Online](#)

As a data modeler, it's common you'll write DAX expressions that need to be evaluated in a modified filter context. For example, you can write a measure definition to calculate sales for "high margin products". We'll describe this calculation later in this article.

NOTE

This article is especially relevant for model calculations that apply filters to Import tables.

The [CALCULATE](#) and [CALCULATETABLE](#) DAX functions are important and useful functions. They let you write calculations that remove or add filters, or modify relationship paths. It's done by passing in filter arguments, which are either Boolean expressions, table expressions, or special filter functions. We'll only discuss Boolean and table expressions in this article.

Consider the following measure definition, which calculates red product sales by using a table expression. It will replace any filters that might be applied to the **Product** table.

```
Red Sales =  
CALCULATE(  
    [Sales],  
    FILTER('Product', 'Product'[Color] = "Red")  
)
```

The [CALCULATE](#) function accepts a table expression returned by the [FILTER](#) DAX function, which evaluates its filter expression for each row of the **Product** table. It achieves the correct result—the sales result for red products. However, it could be achieved much more efficiently by using a Boolean expression.

Here's an improved measure definition, which uses a Boolean expression instead of the table expression. The [KEEPFILTERS](#) DAX function ensures any existing filters applied to the **Color** column are preserved, and not overwritten.

```
Red Sales =  
CALCULATE(  
    [Sales],  
    KEEPFILTERS('Product'[Color] = "Red")  
)
```

It's recommended you pass filter arguments as Boolean expressions, whenever possible. It's because Import model tables are in-memory column stores. They are explicitly optimized to efficiently filter columns in this way.

There are, however, restrictions that apply to Boolean expressions when they're used as filter arguments. They:

- Cannot reference columns from multiple tables
- Cannot reference a measure
- Cannot use nested [CALCULATE](#) functions
- Cannot use functions that scan or return a table

It means that you'll need to use table expressions for more complex filter requirements.

Consider now a different measure definition. The requirement is to calculate sales, but only for months that have

achieved a profit.

```
Sales for Profitable Months =  
CALCULATE(  
    [Sales],  
    FILTER(  
        VALUES('Date'[Month]),  
        [Profit] > 0  
    )  
)
```

In this example, the FILTER function must be used. It's because it requires evaluating the **Profit** measure to eliminate those months that didn't achieve a profit. It's not possible to use a measure in a Boolean expression when it's used as a filter argument.

Recommendations

For best performance, it's recommended you use Boolean expressions as filter arguments, whenever possible.

Therefore, the FILTER function should only be used when necessary. You can use it to perform filter complex column comparisons. These column comparisons can involve:

- Measures
- Other columns
- Using the [OR](#) DAX function, or the OR logical operator (||)

See also

- [Filter functions \(DAX\)](#)
- Learning path: [Use DAX in Power BI Desktop](#)
- Questions? [Try asking the Power BI Community](#)
- Suggestions? [Contribute ideas to improve Power BI](#)

Column and measure references

10/31/2022 • 2 minutes to read • [Edit Online](#)

As a data modeler, your DAX expressions will refer to model columns and measures. Columns and measures are always associated with model tables, but these associations are different, so we have different recommendations on how you'll reference them in your expressions.

Columns

A column is a table-level object, and column names must be unique within a table. So it's possible that the same column name is used multiple times in your model—providing they belong to different tables. There's one more rule: a column name cannot have the same name as a measure name or hierarchy name that exists in the same table.

In general, DAX will not force using a *fully qualified* reference to a column. A fully qualified reference means that the table name precedes the column name.

Here's an example of a calculated column definition using only column name references. The **Sales** and **Cost** columns both belong to a table named **Orders**.

```
Profit = [Sales] - [Cost]
```

The same definition can be rewritten with fully qualified column references.

```
Profit = Orders[Sales] - Orders[Cost]
```

Sometimes, however, you'll be required to use fully qualified column references when Power BI detects ambiguity. When entering a formula, a red squiggly and error message will alert you. Also, some DAX functions like the [LOOKUPVALUE](#) DAX function, require the use of fully qualified columns.

It's recommended you always fully qualify your column references. The reasons are provided in the [Recommendations](#) section.

Measures

A measure is a model-level object. For this reason, measure names must be unique within the model. However, in the **Fields** pane, report authors will see each measure associated with a single model table. This association is set for cosmetic reasons, and you can configure it by setting the **Home Table** property for the measure. For more information, see [Measures in Power BI Desktop \(Organizing your measures\)](#).

It's possible to use a fully qualified measure in your expressions. DAX intellisense will even offer the suggestion. However, it isn't necessary, and it's not a recommended practice. If you change the home table for a measure, any expression that uses a fully qualified measure reference to it will break. You'll then need to edit each broken formula to remove (or update) the measure reference.

It's recommended you never qualify your measure references. The reasons are provided in the [Recommendations](#) section.

Recommendations

Our recommendations are simple and easy to remember:

- Always use fully qualified column references
- Never use fully qualified measure references

Here's why:

- **Formula entry:** Expressions will be accepted, as there won't be any ambiguous references to resolve. Also, you'll meet the requirement for those DAX functions that require fully qualified column references.
- **Robustness:** Expressions will continue to work, even when you change a measure home table property.
- **Readability:** Expressions will be quick and easy to understand—you'll quickly determine that it's a column or measure, based on whether it's fully qualified or not.

See also

- Learning path: [Use DAX in Power BI Desktop](#)
- Questions? [Try asking the Power BI Community](#)
- Suggestions? [Contribute ideas to improve Power BI](#)

DIVIDE function vs. divide operator (/)

10/31/2022 • 2 minutes to read • [Edit Online](#)

As a data modeler, when you write a DAX expression to divide a numerator by a denominator, you can choose to use the **DIVIDE** function or the divide operator (/ - forward slash).

When using the DIVIDE function, you must pass in numerator and denominator expressions. Optionally, you can pass in a value that represents an *alternate result*.

```
DIVIDE(<numerator>, <denominator> [, <alternateresult>])
```

The DIVIDE function was designed to automatically handle division by zero cases. If an alternate result is not passed in, and the denominator is zero or BLANK, the function returns BLANK. When an alternate result is passed in, it's returned instead of BLANK.

The DIVIDE function is convenient because it saves your expression from having to first test the denominator value. The function is also better optimized for testing the denominator value than the **IF** function. The performance gain is significant since checking for division by zero is expensive. Further using DIVIDE results in a more concise and elegant expression.

Example

The following measure expression produces a safe division, but it involves using four DAX functions.

```
Profit Margin =  
IF(  
    OR(  
        ISBLANK([Sales]),  
        [Sales] == 0  
    ),  
    BLANK(),  
    [Profit] / [Sales]  
)
```

This measure expression achieves the same outcome, yet more efficiently and elegantly.

```
Profit Margin =  
DIVIDE([Profit], [Sales])
```

Recommendations

It's recommended that you use the DIVIDE function whenever the denominator is an expression that *could* return zero or BLANK.

In the case that the denominator is a constant value, we recommend that you use the divide operator. In this case, the division is guaranteed to succeed, and your expression will perform better because it will avoid unnecessary testing.

Carefully consider whether the DIVIDE function should return an alternate value. For measures, it's usually a better design that they return BLANK. Returning BLANK is better because report visuals—by default—eliminate groupings when summarizations are BLANK. It allows the visual to focus attention on groups where data exists.

When necessary, in Power BI, you can configure the visual to display all groups (that return values or BLANK) within the filter context by enabling the [Show items with no data](#) option.

See also

- Learning path: [Use DAX in Power BI Desktop](#)
- Questions? [Try asking the Power BI Community](#)
- Suggestions? [Contribute ideas to improve Power BI](#)

Use SELECTEDVALUE instead of VALUES

10/31/2022 • 2 minutes to read • [Edit Online](#)

As a data modeler, sometimes you might need to write a DAX expression that tests whether a column is filtered by a specific value.

In earlier versions of DAX, this requirement was safely achieved by using a pattern involving three DAX functions; [IF](#), [HASONEVALUE](#) and [VALUES](#). The following measure definition presents an example. It calculates the sales tax amount, but only for sales made to Australian customers.

```
Australian Sales Tax =  
IF(  
    HASONEVALUE(Customer[Country-Region]),  
    IF(  
        VALUES(Customer[Country-Region]) = "Australia",  
        [Sales] * 0.10  
    )  
)
```

In the example, the HASONEVALUE function returns TRUE only when a single value of the **Country-Region** column is visible in the current filter context. When it's TRUE, the VALUES function is compared to the literal text "Australia". When the VALUES function returns TRUE, the **Sales** measure is multiplied by 0.10 (representing 10%). If the HASONEVALUE function returns FALSE—because more than one value filters the column—the first IF function returns BLANK.

The use of the HASONEVALUE is a defensive technique. It's required because it's possible that multiple values filter the **Country-Region** column. In this case, the VALUES function returns a table of multiple rows. Comparing a table of multiple rows to a scalar value results in an error.

Recommendation

It's recommended that you use the [SELECTEDVALUE](#) function. It achieves the same outcome as the pattern described in this article, yet more efficiently and elegantly.

Using the SELECTEDVALUE function, the example measure definition is now rewritten.

```
Australian Sales Tax =  
IF(  
    SELECTEDVALUE(Customer[Country-Region]) = "Australia",  
    [Sales] * 0.10  
)
```

TIP

It's possible to pass an *alternate result* value into the SELECTEDVALUE function. The alternate result value is returned when either no filters—or multiple filters—are applied to the column.

See also

- Learning path: [Use DAX in Power BI Desktop](#)
- Questions? [Try asking the Power BI Community](#)

- Suggestions? [Contribute ideas to improve Power BI](#)

Use COUNTROWS instead of COUNT

10/31/2022 • 2 minutes to read • [Edit Online](#)

As a data modeler, sometimes you might need to write a DAX expression that counts table rows. The table could be a model table or an expression that returns a table.

Your requirement can be achieved in two ways. You can use the [COUNT](#) function to count column values, or you can use the [COUNTROWS](#) function to count table rows. Both functions will achieve the same result, providing that the counted column contains no BLANKs.

The following measure definition presents an example. It calculates the number of **OrderDate** column values.

```
Sales Orders =  
COUNT(Sales[OrderDate])
```

Providing that the granularity of the **Sales** table is one row per sales order, and the **OrderDate** column does not contain BLANKs, then the measure will return a correct result.

However, the following measure definition is a better solution.

```
Sales Orders =  
COUNTROWS(Sales)
```

There are three reasons why the second measure definition is better:

- It's more efficient, and so it will perform better.
- It doesn't consider BLANKs contained in any column of the table.
- The intention of formula is clearer, to the point of being self-describing.

Recommendation

When it's your intention to count table rows, it's recommended you always use the [COUNTROWS](#) function.

See also

- Learning path: [Use DAX in Power BI Desktop](#)
- Questions? [Try asking the Power BI Community](#)
- Suggestions? [Contribute ideas to improve Power BI](#)

Use variables to improve your DAX formulas

10/31/2022 • 3 minutes to read • [Edit Online](#)

As a data modeler, writing and debugging some DAX calculations can be challenging. It's common that complex calculation requirements often involve writing compound or complex expressions. Compound expressions can involve the use of many nested functions, and possibly the reuse of expression logic.

Using variables in your DAX formulas can help you write more complex and efficient calculations. Variables can improve performance, reliability, readability, and reduce complexity.

In this article, we'll demonstrate the first three benefits by using an example measure for year-over-year (YoY) sales growth. (The formula for YoY sales growth is period sales, minus sales for the same period last year, divided by sales for the same period last year.)

Let's start with the following measure definition.

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

The measure produces the correct result, yet let's now see how it can be improved.

Improve performance

Notice that the formula repeats the expression that calculates "same period last year". This formula is inefficient, as it requires Power BI to evaluate the same expression twice. The measure definition can be made more efficient by using a variable, [VAR](#).

The following measure definition represents an improvement. It uses an expression to assign the "same period last year" result to a variable named **SalesPriorYear**. The variable is then used twice in the RETURN expression.

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

The measure continues to produce the correct result, and does so in about half the query time.

Improve readability

In the previous measure definition, notice how the choice of variable name makes the RETURN expression simpler to understand. The expression is short and self-describing.

Simplify debugging

Variables can also help you debug a formula. To test an expression assigned to a variable, you temporarily rewrite the RETURN expression to output the variable.

The following measure definition returns only the **SalesPriorYear** variable. Notice how it comments-out the

intended RETURN expression. This technique allows you to easily revert it back once your debugging is complete.

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

Reduce complexity

In earlier versions of DAX, variables were not yet supported. Complex expressions that introduced new filter contexts were required to use the [EARLIER](#) or [EARLIEST](#) DAX functions to reference outer filter contexts. Unfortunately, data modelers found these functions difficult to understand and use.

Variables are always evaluated outside the filters your RETURN expression applies. For this reason, when you use a variable within a modified filter context, it achieves the same result as the EARLIEST function. The use of the EARLIER or EARLIEST functions can therefore be avoided. It means you can now write formulas that are less complex, and that are easier to understand.

Consider the following calculated column definition added to the **Subcategory** table. It evaluates a rank for each product subcategory based on the **Subcategory Sales** column values.

```
Subcategory Sales Rank =  
COUNTROWS(  
    FILTER(  
        Subcategory,  
        EARLIER(Subcategory[Subcategory Sales]) < Subcategory[Subcategory Sales]  
    )  
)+ 1
```

The EARLIER function is used to refer to the **Subcategory Sales** column value *in the current row context*.

The calculated column definition can be improved by using a variable instead of the EARLIER function. The **CurrentSubcategorySales** variable stores the **Subcategory Sales** column value *in the current row context*, and the RETURN expression uses it within a modified filter context.

```
Subcategory Sales Rank =  
VAR CurrentSubcategorySales = Subcategory[Subcategory Sales]  
RETURN  
    COUNTROWS(  
        FILTER(  
            Subcategory,  
            CurrentSubcategorySales < Subcategory[Subcategory Sales]  
        )  
    ) + 1
```

See also

- [VAR DAX article](#)
- Learning path: [Use DAX in Power BI Desktop](#)
- Questions? [Try asking the Power BI Community](#)

DAX function reference

10/31/2022 • 2 minutes to read • [Edit Online](#)

The DAX function reference provides detailed information including syntax, parameters, return values, and examples for each of the over 250 functions used in Data Analysis Expression (DAX) formulas.

IMPORTANT

Not all DAX functions are supported or included in earlier versions of Power BI Desktop, Analysis Services, and Power Pivot in Excel.

In this section

New DAX functions - These functions are new or are existing functions that have been significantly updated.

Aggregation functions - These functions calculate a (scalar) value such as count, sum, average, minimum, or maximum for all rows in a column or table as defined by the expression.

Date and time functions - These functions in DAX are similar to date and time functions in Microsoft Excel. However, DAX functions are based on the datetime data types used by Microsoft SQL Server.

Filter functions - These functions help you return specific data types, look up values in related tables, and filter by related values. Lookup functions work by using tables and relationships between them. Filtering functions let you manipulate data context to create dynamic calculations.

Financial functions - These functions are used in formulas that perform financial calculations, such as net present value and rate of return.

Information functions - These functions look at a table or column provided as an argument to another function and returns whether the value matches the expected type. For example, the ISERROR function returns TRUE if the value you reference contains an error.

Logical functions - These functions return information about values in an expression. For example, the TRUE function lets you know whether an expression that you are evaluating returns a TRUE value.

Math and Trig functions - Mathematical functions in DAX are similar to Excel's mathematical and trigonometric functions. However, there are some differences in the numeric data types used by DAX functions.

Other functions - These functions perform unique actions that cannot be defined by any of the categories most other functions belong to.

Parent and Child functions - These functions help users manage data that is presented as a parent/child hierarchy in their data models.

Relationship functions - These functions are for managing and utilizing relationships between tables. For example, you can specify a particular relationship to be used in a calculation.

Statistical functions - These functions calculate values related to statistical distributions and probability, such as standard deviation and number of permutations.

Table manipulation functions - These functions return a table or manipulate existing tables.

Text functions - With these functions, you can return part of a string, search for text within a string, or concatenate string values. Additional functions are for controlling the formats for dates, times, and numbers.

[Time intelligence functions](#) - These functions help you create calculations that use built-in knowledge about calendars and dates. By using time and date ranges in combination with aggregations or calculations, you can build meaningful comparisons across comparable time periods for sales, inventory, and so on.

See also

[DAX Syntax Reference](#)

[DAX Operator Reference](#)

[DAX Parameter-Naming Conventions](#)

New DAX functions

10/31/2022 • 2 minutes to read • [Edit Online](#)

DAX is continuously being improved with new functions and functionality to support new features. New functions and updates are included in service, application, and tool updates which in most cases are monthly.

While functions and functionality are being updated all the time, only those updates that have a visible and functional change exposed to users are described in documentation. New functions and updates to existing functions within the past year are shown here.

IMPORTANT

Not all functions are supported in all versions of Power BI Desktop, Analysis Services, and Power Pivot in Excel. New and updated functions are typically first introduced in Power BI Desktop, and then later in Analysis Services, Power Pivot in Excel, and tools.

New functions

| FUNCTION | MONTH | DESCRIPTION |
|-----------------------------|----------------|---|
| NETWORKDAYS | July, 2022 | Returns the number of whole workdays between two dates. |
| BITAND | November, 2021 | Returns a bitwise 'AND' of two numbers. |
| BITLSHIFT | November, 2021 | Returns a number shifted left by the specified number of bits. |
| BITOR | November, 2021 | Returns a bitwise 'OR' of two numbers. |
| BITRSHIFT | November, 2021 | Returns a number shifted right by the specified number of bits. |
| BITXOR | November, 2021 | Returns a bitwise 'XOR' of two numbers. |

Updated functions

| FUNCTION | MONTH | DESCRIPTION |
|--------------------------------|-----------------|--|
| CALCULATE | September, 2021 | Support for aggregation functions in boolean filter expressions. |
| CALCULATETABLE | September, 2021 | Support for aggregation functions in boolean filter expressions. |

| FUNCTION | MONTH | DESCRIPTION |
|----------|-----------------|--|
| FORMAT | September, 2021 | Additional locale_name parameter that specifies the name of a locale to be used by format_string. |
| XIRR | September, 2021 | Additional alternateResult parameter that specifies a result to be returned instead of an error if XIRR cannot determine a solution. |

Aggregation functions

10/31/2022 • 2 minutes to read • [Edit Online](#)

Aggregation functions calculate a (scalar) value such as count, sum, average, minimum, or maximum for all rows in a column or table as defined by the expression.

In this category

| FUNCTION | DESCRIPTION |
|--|--|
| APPROXIMATEDISTINCTCOUNT | Returns the <i>approximate</i> number of rows that contain distinct values in a column. |
| AVERAGE | Returns the average (arithmetic mean) of all the numbers in a column. |
| AVERAGEA | Returns the average (arithmetic mean) of the values in a column. |
| AVERAGEX | Calculates the average (arithmetic mean) of a set of expressions evaluated over a table. |
| COUNT | Counts the number of rows in the specified column that contain non-blank values. |
| COUNTA | Counts the number of rows in the specified column that contain non-blank values. |
| COUNTAX | Counts non-blank results when evaluating the result of an expression over a table. |
| COUNTBLANK | Counts the number of blank cells in a column. |
| COUNTROWS | Counts the number of rows in the specified table, or in a table defined by an expression. |
| COUNTX | Counts the number of rows that contain a number or an expression that evaluates to a number, when evaluating an expression over a table. |
| DISTINCTCOUNT | Counts the number of distinct values in a column. |
| DISTINCTCOUNTNOBLANK | Counts the number of distinct values in a column. |
| MAX | Returns the largest numeric value in a column, or between two scalar expressions. |
| MAXA | Returns the largest value in a column. |
| MAXX | Evaluates an expression for each row of a table and returns the largest numeric value. |

| FUNCTION | DESCRIPTION |
|----------|--|
| MIN | Returns the smallest numeric value in a column, or between two scalar expressions. |
| MINA | Returns the smallest value in a column, including any logical values and numbers represented as text. |
| MINX | Returns the smallest numeric value that results from evaluating an expression for each row of a table. |
| PRODUCT | Returns the product of the numbers in a column. |
| PRODUCTX | Returns the product of an expression evaluated for each row in a table. |
| SUM | Adds all the numbers in a column. |
| SUMX | Returns the sum of an expression evaluated for each row in a table. |

APPROXIMATEDISTINCTCOUNT

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the *approximate* number of rows that contain distinct values in a column. This function can query large amounts of data with potentially better performance than DISTINCTCOUNT, with slight deviation from the exact result.

Syntax

```
APPROXIMATEDISTINCTCOUNT(<columnName>)
```

Parameters

| TERM | DESCRIPTION |
|--------|--|
| column | The column that contains the values to be counted. This cannot be an expression. |

Return value

The approximate number of distinct values in *column*.

Remarks

The only argument to this function is a column. You can use columns containing any type of data. When the function finds no rows to count, it returns a BLANK, otherwise it returns the count of distinct values.

AVERAGE

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the average (arithmetic mean) of all the numbers in a column.

Syntax

```
AVERAGE(<column>)
```

Parameters

| TERM | DEFINITION |
|--------|--|
| column | The column that contains the numbers for which you want the average. |

Return value

Returns a decimal number that represents the arithmetic mean of the numbers in the column.

Remarks

- This function takes the specified column as an argument and finds the average of the values in that column. If you want to find the average of an expression that evaluates to a set of numbers, use the AVERAGEX function instead.
- Nonnumeric values in the column are handled as follows:
 - If the column contains text, no aggregation can be performed, and the function returns blanks.
 - If the column contains logical values or empty cells, those values are ignored.
 - Cells with the value zero are included.
- When you average cells, you must keep in mind the difference between an empty cell and a cell that contains the value 0 (zero). When a cell contains 0, it is added to the sum of numbers and the row is counted among the number of rows used as the divisor. However, when a cell contains a blank, the row is not counted.
- Whenever there are no rows to aggregate, the function returns a blank. However, if there are rows, but none of them meet the specified criteria, the function returns 0. Excel also returns a zero if no rows are found that meet the conditions.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following formula returns the average of the values in the column, ExtendedSalesAmount, in the table, InternetSales.

```
= AVERAGE(InternetSales[ExtendedSalesAmount])
```

Related functions

The AVERAGEX function can take as its argument an expression that is evaluated for each row in a table. This enables you to perform calculations and then take the average of the calculated values.

The AVERAGEA function takes a column as its argument, but otherwise is like the Excel function of the same name. By using the AVERAGEA function, you can calculate a mean on a column that contains empty values.

AVERAGEA

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the average (arithmetic mean) of the values in a column. Handles text and non-numeric values.

Syntax

```
AVERAGEA(<column>)
```

Parameters

| TERM | DEFINITION |
|--------|---|
| column | A column that contains the values for which you want the average. |

Return value

A decimal number.

Remarks

- The AVERAGEA function takes a column and averages the numbers in it, but also handles non-numeric data types according to the following rules:
 - Values that evaluates to TRUE count as 1.
 - Values that evaluate to FALSE count as 0 (zero).
 - Values that contain non-numeric text count as 0 (zero).
 - Empty text ("") counts as 0 (zero).
- If you do not want to include logical values and text representations of numbers in a reference as part of the calculation, use the AVERAGE function.
- Whenever there are no rows to aggregate, the function returns a blank. However, if there are rows, but none of them meet the specified criteria, the function returns 0. Microsoft Excel also returns a zero if no rows are found that meet the conditions.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following example returns the average of non-blank cells in the referenced column, given the following table. If you used the AVERAGE function, the mean would be 21/2; with the AVERAGEA function, the result is 22/5.

| TRANSACTION ID | AMOUNT | RESULT |
|----------------|--------|-------------|
| 0000123 | 1 | Counts as 1 |

| TRANSACTION ID | AMOUNT | RESULT |
|----------------|--------|--------------|
| 0000124 | 20 | Counts as 20 |
| 0000125 | n/a | Counts as 0 |
| 0000126 | | Counts as 0 |
| 0000126 | TRUE | Counts as 1 |

= AVERAGEA([Amount])

See also

[AVERAGE function](#)

[AVERAGEX function](#)

[Statistical functions](#)

AVERAGEX

10/31/2022 • 2 minutes to read • [Edit Online](#)

Calculates the average (arithmetic mean) of a set of expressions evaluated over a table.

Syntax

```
AVERAGEX(<table>,<expression>)
```

Parameters

| TERM | DEFINITION |
|------------|--|
| table | Name of a table, or an expression that specifies the table over which the aggregation can be performed. |
| expression | An expression with a scalar result, which will be evaluated for each row of the table in the first argument. |

Return value

A decimal number.

Remarks

- The AVERAGEX function enables you to evaluate expressions for each row of a table, and then take the resulting set of values and calculate its arithmetic mean. Therefore, the function takes a table as its first argument, and an expression as the second argument.
- In all other respects, AVERAGEX follows the same rules as AVERAGE. You cannot include non-numeric or null cells. Both the table and expression arguments are required.
- When there are no rows to aggregate, the function returns a blank. When there are rows, but none of them meet the specified criteria, then the function returns 0.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following example calculates the average freight and tax on each order in the InternetSales table, by first summing Freight plus TaxAmt in each row, and then averaging those sums.

```
= AVERAGEX(InternetSales, InternetSales[Freight]+ InternetSales[TaxAmt])
```

If you use multiple operations in the expression used as the second argument, you must use parentheses to control the order of calculations. For more information, see [DAX Syntax Reference](#).

See also

AVERAGE function
AVERAGEA function
Statistical functions

COUNT

10/31/2022 • 2 minutes to read • [Edit Online](#)

Counts the number of rows in the specified column that contain non-blank values.

Syntax

```
COUNT(<column>)
```

Parameters

| TERM | DEFINITION |
|--------|--|
| column | The column that contains the values to be counted. |

Return value

A whole number.

Remarks

- The only argument allowed to this function is a column. The COUNT function counts rows that contain the following kinds of values:
 - Numbers
 - Dates
 - Strings
- When the function finds no rows to count, it returns a blank.
- Blank values are skipped. TRUE/FALSE values are not supported.
- If you want to evaluate a column of TRUE/FALSE values, use the COUNTA function.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.
- For best practices when using COUNT, see [Use COUNTROWS instead of COUNT](#).

Example

The following example shows how to count the number of values in the column, ShipDate.

```
= COUNT([ShipDate])
```

To count logical values or text, use the COUNTA or COUNTAX functions.

See also

[COUNTA function](#)

[COUNTAX function](#)

COUNTA

10/31/2022 • 2 minutes to read • [Edit Online](#)

Counts the number of rows in the specified column that contain non-blank values.

Syntax

```
COUNTA(<column>)
```

Parameters

| TERM | DEFINITION |
|--------|--|
| column | The column that contains the values to be counted. |

Return value

A whole number.

Remarks

- When the function does not find any rows to count, the function returns a blank.
- Unlike [COUNT](#), COUNTA supports Boolean data type.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following example returns all rows in the `Reseller` table that have any kind of value in the column that stores phone numbers.

```
= COUNTA(Reseller[Phone])
```

See also

[COUNT function](#)

[COUNTAX function](#)

[COUNTX function](#)

[Statistical functions](#)

COUNTAX

10/31/2022 • 2 minutes to read • [Edit Online](#)

The COUNTAX function counts non-blank results when evaluating the result of an expression over a table. That is, it works just like the COUNTA function, but is used to iterate through the rows in a table and count rows where the specified expressions results in a non-blank result.

Syntax

```
COUNTAX(<table>,<expression>)
```

Parameters

| TERM | DEFINITION |
|------------|---|
| table | The table containing the rows for which the expression will be evaluated. |
| expression | The expression to be evaluated for each row of the table. |

Return value

A whole number.

Remarks

- Like the COUNTA function, the COUNTAX function counts cells containing any type of information, including other expressions. For example, if the column contains an expression that evaluates to an empty string, the COUNTAX function treats that result as non-blank. Usually the COUNTAX function does not count empty cells but in this case the cell contains a formula, so it is counted.
- Whenever the function finds no rows to aggregate, the function returns a blank.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following example counts the number of nonblank rows in the column, Phone, using the table that results from filtering the Reseller table on [Status] = **Active**.

```
= COUNTAX(FILTER('Reseller',[Status]="Active"),[Phone])
```

See also

[COUNT function](#)

[COUNTA function](#)

[COUNTX function](#)

[Statistical functions](#)

COUNTBLANK

10/31/2022 • 2 minutes to read • [Edit Online](#)

Counts the number of blank cells in a column.

Syntax

```
COUNTBLANK(<column>)
```

Parameters

| TERM | DEFINITION |
|--------|---|
| column | The column that contains the blank cells to be counted. |

Return value

A whole number. If no rows are found that meet the condition, blanks are returned.

Remarks

- The only argument allowed to this function is a column. You can use columns containing any type of data, but only blank cells are counted. Cells that have the value zero (0) are not counted, as zero is considered a numeric value and not a blank.
- Whenever there are no rows to aggregate, the function returns a blank. However, if there are rows, but none of them meet the specified criteria, the function returns 0. Microsoft Excel also returns a zero if no rows are found that meet the conditions.
- In other words, if the COUNTBLANK function finds no blanks, the result will be zero, but if there are no rows to check, the result will be blank.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following example shows how to count the number of rows in the table Reseller that have blank values for BankName.

```
= COUNTBLANK(Reseller[BankName])
```

To count logical values or text, use the COUNTA or COUNTAX functions.

See also

[COUNT function](#)

[COUNTA function](#)

[COUNTAX function](#)

COUNTROWS

10/31/2022 • 2 minutes to read • [Edit Online](#)

The COUNTROWS function counts the number of rows in the specified table, or in a table defined by an expression.

Syntax

```
COUNTROWS([<table>])
```

Parameters

| TERM | DEFINITION |
|-------|---|
| table | (Optional) The name of the table that contains the rows to be counted, or an expression that returns a table. When not provided, the default value is the home table of the current expression. |

Return value

A whole number.

Remarks

- This function can be used to count the number of rows in a base table, but more often is used to count the number of rows that result from filtering a table, or applying context to a table.
- Whenever there are no rows to aggregate, the function returns a blank. However, if there are rows, but none of them meet the specified criteria, the function returns 0. Microsoft Excel also returns a zero if no rows are found that meet the conditions.
- To learn more about best practices when using COUNT and COUNTROWS, see [Use COUNTROWS instead of COUNT in DAX](#).
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example 1

The following example shows how to count the number of rows in the table Orders. The expected result is 52761.

```
= COUNTROWS('Orders')
```

Example 2

The following example demonstrates how to use COUNTROWS with a row context. In this scenario, there are two sets of data that are related by order number. The table Reseller contains one row for each reseller; the table

ResellerSales contains multiple rows for each order, each row containing one order for a particular reseller. The tables are connected by a relationship on the column, ResellerKey.

The formula gets the value of ResellerKey and then counts the number of rows in the related table that have the same reseller ID. The result is output in the column, **CalculatedColumn1**.

```
= COUNTROWS(RELATEDTABLE(ResellerSales))
```

The following table shows a portion of the expected results:

| RESELLERKEY | CALCULATEDCOLUMN1 |
|-------------|-------------------|
| 1 | 73 |
| 2 | 70 |
| 3 | 394 |

See also

[COUNT function](#)

[COUNTA function](#)

[COUNTAX function](#)

[COUNTX function](#)

[Statistical functions](#)

COUNTX

10/31/2022 • 2 minutes to read • [Edit Online](#)

Counts the number of rows that contain a non-blank value or an expression that evaluates to a non-blank value, when evaluating an expression over a table.

Syntax

```
COUNTX(<table>,<expression>)
```

Parameters

| TERM | DEFINITION |
|------------|--|
| table | The table containing the rows to be counted. |
| expression | An expression that returns the set of values that contains the values you want to count. |

Return value

An integer.

Remarks

- The COUNTX function takes two arguments. The first argument must always be a table, or any expression that returns a table. The second argument is the column or expression that is searched by COUNTX.
- The COUNTX function counts only values, dates, or strings. If the function finds no rows to count, it returns a blank.
- If you want to count logical values, use the COUNTAX function.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example 1

The following formula returns a count of all rows in the Product table that have a list price.

```
= COUNTX(Product,[ListPrice])
```

Example 2

The following formula illustrates how to pass a filtered table to COUNTX for the first argument. The formula uses a filter expression to get only the rows in the Product table that meet the condition, ProductSubCategory = "Caps", and then counts the rows in the resulting table that have a list price. The FILTER expression applies to the table Products but uses a value that you look up in the related table, ProductSubCategory.

```
= COUNTX(FILTER(Product,RELATED(ProductSubcategory[EnglishProductSubcategoryName])="Caps"),  
Product[ListPrice])
```

See also

[COUNT function](#)

[COUNTA function](#)

[COUNTAX function](#)

[Statistical functions](#)

DISTINCTCOUNT

10/31/2022 • 2 minutes to read • [Edit Online](#)

Counts the number of distinct values in a column.

Syntax

```
DISTINCTCOUNT(<column>)
```

Parameters

| TERM | DESCRIPTION |
|--------|---|
| column | The column that contains the values to be counted |

Return value

The number of distinct values in *column*.

Remarks

- The only argument allowed to this function is a column. You can use columns containing any type of data. When the function finds no rows to count, it returns a BLANK, otherwise it returns the count of distinct values.
- DISTINCTCOUNT function counts the BLANK value. To skip the BLANK value, use the [DISTINCTCOUNTNOBLANK](#) function.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following example shows how to count the number of distinct sales orders in the column ResellerSales_USD[SalesOrderNumber].

```
= DISTINCTCOUNT(ResellerSales_USD[SalesOrderNumber])
```

Using the above measure in a table with calendar year in the side and product category on top returns the following results:

| ROW LABELS | ACCESSORIES | BIKES | CLOTHING | COMPONENTS | - | GRAND TOTAL |
|------------|-------------|-------|----------|------------|---|-------------|
| 2005 | 135 | 345 | 242 | 205 | | 366 |
| 2006 | 356 | 850 | 644 | 702 | | 1015 |
| 2007 | 531 | 1234 | 963 | 1138 | | 1521 |

| ROW LABELS | ACCESSORIES | BIKES | CLOTHING | COMPONENTS | - | GRAND TOTAL |
|-------------|-------------|-------|----------|------------|---|-------------|
| 2008 | 293 | 724 | 561 | 601 | | 894 |
| | | | | | 1 | 1 |
| Grand Total | 1315 | 3153 | 2410 | 2646 | 1 | 3797 |

In the above example, note that the rows Grand Total numbers do not add up, this happens because the same order might contain line items, in the same order, from different product categories.

See also

[COUNT function](#)

[COUNTA function](#)

[COUNTAX function](#)

[COUNTX function](#)

[Statistical functions](#)

DISTINCTCOUNTNOBLANK

10/31/2022 • 2 minutes to read • [Edit Online](#)

Counts the number of distinct values in a column.

Syntax

```
DISTINCTCOUNTNOBLANK (<column>)
```

Parameters

| TERM | DESCRIPTION |
|--------|---|
| column | The column that contains the values to be counted |

Return value

The number of distinct values in *column*.

Remarks

- Unlike [DISTINCTCOUNT](#) function, DISTINCTCOUNTNOBLANK does not count the BLANK value.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following example shows how to count the number of distinct sales orders in the column ResellerSales_USD[SalesOrderNumber].

```
= DISTINCTCOUNT(ResellerSales_USD[SalesOrderNumber])
```

DAX query

```
EVALUATE
    ROW(
        "DistinctCountNoBlank", DISTINCTCOUNTNOBLANK(DimProduct[EndDate]),
        "DistinctCount", DISTINCTCOUNT(DimProduct[EndDate])
    )
```

| [DISTINCTCOUNTNOBLANK] | [DISTINCTCOUNT] |
|------------------------|-----------------|
| 2 | 3 |

See also

[DISTINCTCOUNT](#)

MAX

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the largest value in a column, or between two scalar expressions.

Syntax

```
MAX(<column>)
```

```
MAX(<expression1>, <expression2>)
```

Parameters

| TERM | DEFINITION |
|------------|---|
| column | The column in which you want to find the largest value. |
| expression | Any DAX expression which returns a single value. |

Return value

The largest value.

Remarks

- When comparing two expressions, blank is treated as 0 when comparing. That is, Max(1, Blank()) returns 1, and Max(-1, Blank()) returns 0. If both arguments are blank, MAX returns a blank. If either expression returns a value which is not allowed, MAX returns an error.
- TRUE/FALSE values are not supported. If you want to evaluate a column of TRUE/FALSE values, use the MAXA function.

Example 1

The following example returns the largest value found in the ExtendedAmount column of the InternetSales table.

```
= MAX(InternetSales[ExtendedAmount])
```

Example 2

The following example returns the largest value between the result of two expressions.

```
= Max([TotalSales], [TotalPurchases])
```

See also

MAXA function
MAXX function
Statistical functions

MAXA

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the largest value in a column.

Syntax

```
MAXA(<column>)
```

Parameters

| TERM | DEFINITION |
|--------|---|
| column | The column in which you want to find the largest value. |

Return value

The largest value.

Remarks

- The MAXA function takes as argument a column, and looks for the largest value among the following types of values:
 - Numbers
 - Dates
- Logical values, such as TRUE and FALSE. Rows that evaluate to TRUE count as 1; rows that evaluate to FALSE count as 0 (zero).
- Empty cells are ignored. If the column contains no values that can be used, MAXA returns 0 (zero).
- If you want to compare text values, use the MAX function.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example 1

The following example returns the greatest value from a calculated column, named **ResellerMargin**, that computes the difference between list price and reseller price.

```
= MAXA([ResellerMargin])
```

Example 2

The following example returns the largest value from a column that contains dates and times. Therefore, this formula gets the most recent transaction date.

```
= MAXA([TransactionDate])
```

See also

[MAX function](#)

[MAXX function](#)

[Statistical functions](#)

MAXX

10/31/2022 • 2 minutes to read • [Edit Online](#)

Evaluates an expression for each row of a table and returns the largest value.

Syntax

```
MAXX(<table>,<expression>)
```

Parameters

| TERM | DEFINITION |
|------------|---|
| table | The table containing the rows for which the expression will be evaluated. |
| expression | The expression to be evaluated for each row of the table. |

Return value

The largest value.

Remarks

- The **table** argument to the MAXX function can be a table name, or an expression that evaluates to a table. The second argument indicates the expression to be evaluated for each row of the table.
- Of the values to evaluate, only the following are counted:
 - Numbers
 - Texts
 - Dates
- Blank values are skipped. TRUE/FALSE values are not supported.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example 1

The following formula uses an expression as the second argument to calculate the total amount of taxes and shipping for each order in the table, InternetSales. The expected result is 375.7184.

```
= MAXX(InternetSales, InternetSales[TaxAmt]+ InternetSales[Freight])
```

Example 2

The following formula first filters the table InternetSales, by using a FILTER expression, to return a subset of orders for a specific sales region, defined as [SalesTerritory] = 5. The MAXX function then evaluates the expression used as the second argument for each row of the filtered table, and returns the highest amount for

taxes and shipping for just those orders. The expected result is 250.3724.

```
= MAXX(FILTER(InternetSales,[SalesTerritoryCode]="5"), InternetSales[TaxAmt]+ InternetSales[Freight])
```

See also

[MAX function](#)

[MAXA function](#)

[Statistical functions](#)

MIN

10/31/2022 • 2 minutes to read • [Edit Online](#)

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

Syntax

```
MIN(<column>)
```

```
MIN(<expression1>, <expression2>)
```

Parameters

| TERM | DEFINITION |
|------------|--|
| column | The column in which you want to find the smallest value. |
| expression | Any DAX expression which returns a single value. |

Return value

The smallest value.

Remarks

- The MIN function takes a column or two expressions as an argument, and returns the smallest value. The following types of values in the columns are counted:
 - Numbers
 - Texts
 - Dates
 - Blanks
- When comparing expressions, blank is treated as 0 when comparing. That is, `Min(1,Blank())` returns 0, and `Min(-1, Blank())` returns -1. If both arguments are blank, MIN returns a blank. If either expression returns a value which is not allowed, MIN returns an error.
- TRUE/FALSE values are not supported. If you want to evaluate a column of TRUE/FALSE values, use the MINA function.

Example 1

The following example returns the smallest value from the calculated column, ResellerMargin.

```
= MIN([ResellerMargin])
```

Example 2

The following example returns the smallest value from a column that contains dates and times, TransactionDate. This formula therefore returns the date that is earliest.

```
= MIN([TransactionDate])
```

Example 3

The following example returns the smallest value from the result of two scalar expressions.

```
= Min([TotalSales], [TotalPurchases])
```

See also

[MINA function](#)

[MINX function](#)

[Statistical functions](#)

MINA

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the smallest value in a column.

Syntax

```
MINA(<column>)
```

Parameters

| TERM | DEFINITION |
|--------|--|
| column | The column for which you want to find the minimum value. |

Return value

The smallest value.

Remarks

- The MINA function takes as argument a column that contains numbers, and determines the smallest value as follows:
 - If the column contains no values, MINA returns 0 (zero).
 - Rows in the column that evaluates to logical values, such as TRUE and FALSE are treated as 1 if TRUE and 0 (zero) if FALSE.
 - Empty cells are ignored.
- If you want to compare text values, use the MIN function.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example 1

The following expression returns the minimum freight charge from the table, InternetSales.

```
= MINA(InternetSales[Freight])
```

Example 2

The following expression returns the minimum value in the column, PostalCode. Because the data type of the column is text, the function does not find any values, and the formula returns zero (0).

```
= MINA([PostalCode])
```

See also

MIN function
MINX function
Statistical functions

MINX

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the smallest value that results from evaluating an expression for each row of a table.

Syntax

```
MINX(<table>, < expression>)
```

Parameters

| TERM | DEFINITION |
|------------|---|
| table | The table containing the rows for which the expression will be evaluated. |
| expression | The expression to be evaluated for each row of the table. |

Return value

A smallest value.

Remarks

- The MINX function takes as its first argument a table, or an expression that returns a table. The second argument contains the expression that is evaluated for each row of the table.
- Blank values are skipped. TRUE/FALSE values are not supported.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example 1

The following example filters the table, InternetSales, and returns only rows for a specific sales territory. The formula then finds the minimum value in the column, Freight.

```
= MINX( FILTER(InternetSales, [SalesTerritoryKey] = 5),[Freight])
```

Example 2

The following example uses the same filtered table as in the previous example, but instead of merely looking up values in the column for each row of the filtered table, the function calculates the sum of two columns, Freight and TaxAmt, and returns the smallest value resulting from that calculation.

```
= MINX( FILTER(InternetSales, InternetSales[SalesTerritoryKey] = 5), InternetSales[Freight] +  
InternetSales[TaxAmt])
```

In the first example, the names of the columns are unqualified. In the second example, the column names are fully qualified.

See also

[MIN function](#)

[MINA function](#)

[Statistical functions](#)

PRODUCT

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the product of the numbers in a column.

Syntax

```
PRODUCT(<column>)
```

Parameters

| TERM | DEFINITION |
|--------|---|
| column | The column that contains the numbers for which the product is to be computed. |

Return value

A decimal number.

Remarks

- To return the product of an expression evaluated for each row in a table, use [PRODUCTX function](#).
- Only the numbers in the column are counted. Blanks, logical values, and text are ignored. For example,

`PRODUCT(Table[Column])` is equivalent to `PRODUCTX(Table, Table[Column])`.

- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following computes the product of the AdjustedRates column in an Annuity table:

```
= PRODUCT( Annuity[AdjustedRates] )
```

See also

[PRODUCTX](#)

PRODUCTX

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the product of an expression evaluated for each row in a table.

Syntax

```
PRODUCTX(<table>, <expression>)
```

Parameters

| TERM | DEFINITION |
|------------|---|
| table | The table containing the rows for which the expression will be evaluated. |
| expression | The expression to be evaluated for each row of the table. |

Return value

A decimal number.

Remarks

- To return the product of the numbers in a column, use [PRODUCT](#).
- The PRODUCTX function takes as its first argument a table, or an expression that returns a table. The second argument is a column that contains the numbers for which you want to compute the product, or an expression that evaluates to a column.
- Only the numbers in the column are counted. Blanks, logical values, and text are ignored.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following computes the future value of an investment:

```
= [PresentValue] * PRODUCTX( AnnuityPeriods, 1+[FixedInterestRate] )
```

See also

[PRODUCT](#)

SUM

10/31/2022 • 2 minutes to read • [Edit Online](#)

Adds all the numbers in a column.

Syntax

```
SUM(<column>)
```

Parameters

| TERM | DEFINITION |
|--------|--|
| column | The column that contains the numbers to sum. |

Return value

A decimal number.

Remarks

If you want to filter the values that you are summing, you can use the SUMX function and specify an expression to sum over.

Example

The following example adds all the numbers that are contained in the column, Amt, from the table, Sales.

```
= SUM(Sales[Amt])
```

See also

[SUMX](#)

SUMX

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the sum of an expression evaluated for each row in a table.

Syntax

```
SUMX(<table>, <expression>)
```

Parameters

| TERM | DEFINITION |
|------------|---|
| table | The table containing the rows for which the expression will be evaluated. |
| expression | The expression to be evaluated for each row of the table. |

Return value

A decimal number.

Remarks

- The SUMX function takes as its first argument a table, or an expression that returns a table. The second argument is a column that contains the numbers you want to sum, or an expression that evaluates to a column.
- Only the numbers in the column are counted. Blanks, logical values, and text are ignored.
- For more complex examples of SUMX in formulas, see [ALL](#) and [CALCULATETABLE](#).
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following example first filters the table, InternetSales, on the expression, 'InternetSales[SalesTerritoryID] = 5', and then returns the sum of all values in the Freight column. In other words, the expression returns the sum of freight charges for only the specified sales area.

```
= SUMX(FILTER(InternetSales, InternetSales[SalesTerritoryID]=5),[Freight])
```

If you do not need to filter the column, use the SUM function. The SUM function is similar to the Excel function of the same name, except that it takes a column as a reference.

See also

[SUM](#)

[Statistical functions](#)

Date and time functions

10/31/2022 • 2 minutes to read • [Edit Online](#)

These functions help you create calculations based on dates and time. Many of the functions in DAX are similar to the Excel date and time functions. However, DAX functions use a **datetime** data type, and can take values from a column as an argument.

In this category

| FUNCTION | DESCRIPTION |
|------------------------------|---|
| CALENDAR | Returns a table with a single column named "Date" that contains a contiguous set of dates. |
| CALENDARAUTO | Returns a table with a single column named "Date" that contains a contiguous set of dates. |
| DATE | Returns the specified date in datetime format. |
| DATEDIFF | Returns the number of interval boundaries between two dates. |
| DATEVALUE | Converts a date in the form of text to a date in datetime format. |
| DAY | Returns the day of the month, a number from 1 to 31. |
| EDATE | Returns the date that is the indicated number of months before or after the start date. |
| EOMONTH | Returns the date in datetime format of the last day of the month, before or after a specified number of months. |
| HOUR | Returns the hour as a number from 0 (12:00 A.M.) to 23 (11:00 P.M.). |
| MINUTE | Returns the minute as a number from 0 to 59, given a date and time value. |
| MONTH | Returns the month as a number from 1 (January) to 12 (December). |
| NETWORKDAYS | Returns the number of whole workdays between two dates. |
| NOW | Returns the current date and time in datetime format. |
| QUARTER | Returns the quarter as a number from 1 to 4. |
| SECOND | Returns the seconds of a time value, as a number from 0 to 59. |

| FUNCTION | DESCRIPTION |
|-----------|--|
| TIME | Converts hours, minutes, and seconds given as numbers to a time in datetime format. |
| TIMEVALUE | Converts a time in text format to a time in datetime format. |
| TODAY | Returns the current date. |
| UTCNOW | Returns the current UTC date and time |
| UTCTODAY | Returns the current UTC date. |
| WEEKDAY | Returns a number from 1 to 7 identifying the day of the week of a date. |
| WEEKNUM | Returns the week number for the given date and year according to the return_type value. |
| YEAR | Returns the year of a date as a four digit integer in the range 1900-9999. |
| YEARFRAC | Calculates the fraction of the year represented by the number of whole days between two dates. |

CALENDAR

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns a table with a single column named "Date" that contains a contiguous set of dates. The range of dates is from the specified start date to the specified end date, inclusive of those two dates.

Syntax

```
CALENDAR(<start_date>, <end_date>)
```

Parameters

| TERM | DEFINITION |
|------------|---|
| start_date | Any DAX expression that returns a datetime value. |
| end_date | Any DAX expression that returns a datetime value. |

Return value

Returns a table with a single column named "Date" containing a contiguous set of dates. The range of dates is from the specified start date to the specified end date, inclusive of those two dates.

Remarks

- An error is returned if start_date is greater than end_date.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Examples

The following formula returns a table with dates between January 1st, 2015 and December 31st, 2021.

```
= CALENDAR (DATE (2015, 1, 1), DATE (2021, 12, 31))
```

For a data model which includes actual sales data and future sales forecasts, the following expression returns a date table covering the range of dates in both the Sales and Forecast tables.

```
= CALENDAR (MINX (Sales, [Date]), MAXX (Forecast, [Date]))
```

CALENDARAUTO

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns a table with a single column named "Date" that contains a contiguous set of dates. The range of dates is calculated automatically based on data in the model.

Syntax

```
CALENDARAUTO([fiscal_year_end_month])
```

Parameters

| TERM | DEFINITION |
|-----------------------|--|
| fiscal_year_end_month | Any DAX expression that returns an integer from 1 to 12. If omitted, defaults to the value specified in the calendar table template for the current user, if present; otherwise, defaults to 12. |

Return value

Returns a table with a single column named "Date" that contains a contiguous set of dates. The range of dates is calculated automatically based on data in the model.

Remarks

- The date range is calculated as follows:
 - The earliest date in the model which is not in a calculated column or calculated table is taken as the MinDate.
 - The latest date in the model which is not in a calculated column or calculated table is taken as the MaxDate.
 - The date range returned is dates between the beginning of the fiscal year associated with MinDate and the end of the fiscal year associated with MaxDate.
- An error is returned if the model does not contain any datetime values which are not in calculated columns or calculated tables.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

In this example, the MinDate and MaxDate in the data model are July 1, 2010 and June 30, 2011.

`CALENDARAUTO()` will return all dates between January 1, 2010 and December 31, 2011.

`CALENDARAUTO(3)` will return all dates between March 1, 2010 and March 31, 2012.

DATE

10/31/2022 • 3 minutes to read • [Edit Online](#)

Returns the specified date in **datetime** format.

Syntax

```
DATE(<year>, <month>, <day>)
```

Parameters

| TERM | DEFINITION |
|-------|---|
| year | <p>A number representing the year.</p> <p>The value of the year argument can include one to four digits. The year argument is interpreted according to the date system used by your computer.</p> <p>Dates beginning with March 1, 1900 are supported.</p> <p>If you enter a number that has decimal places, the number is rounded.</p> <p>For values greater than 9999 or less than zero (negative values), the function returns a #VALUE! error.</p> <p>If the year value is between 0 and 1899, the value is added to 1900 to produce the final value. See the examples below. Note: You should use four digits for the year argument whenever possible to prevent unwanted results. For example, using 07 returns 1907 as the year value.</p> |
| month | <p>A number representing the month or a calculation according to the following rules:</p> <p>Negative integers are not supported. Valid values are 1-12.</p> <p>If month is a number from 1 to 12, then it represents a month of the year. 1 represents January, 2 represents February, and so on until 12 that represents December.</p> <p>If you enter an integer larger than 12, the following computation occurs: the date is calculated by adding the value of month to the year. For example, if you have DATE(2008, 18, 1), the function returns a datetime value equivalent to June 1st of 2009, because 18 months are added to the beginning of 2008 yielding a value of June 2009. See examples below.</p> |

| TERM | DEFINITION |
|------|---|
| day | <p>A number representing the day or a calculation according to the following rules:</p> <p>Negative integers are not supported. Valid values are 1-31.</p> <p>If day is a number from 1 to the last day of the given month then it represents a day of the month.</p> <p>If you enter an integer larger than last day of the given month, the following computation occurs: the date is calculated by adding the value of day to month. For example, in the formula <code>DATE(2008, 3, 32)</code>, the DATE function returns a datetime value equivalent to April 1st of 2008, because 32 days are added to the beginning of March yielding a value of April 1st.</p> <p>If day contains a decimal portion, it is rounded to the nearest integer value.</p> |

Return value

Returns the specified date (**datetime**).

Remarks

- The DATE function takes the integers that are input as arguments, and generates the corresponding date. The DATE function is most useful in situations where the year, month, and day are supplied by formulas. For example, the underlying data might contain dates in a format that is not recognized as a date, such as YYYYMMDD. You can use the DATE function in conjunction with other functions to convert the dates to a number that can be recognized as a date.
- In contrast to Microsoft Excel, which stores dates as a serial number, DAX date functions always return a **datetime** data type. However, you can use formatting to display dates as serial numbers if you want.
- Date and datetime can also be specified as a literal in the format `dt"YYYY-MM-DD"`, `dt"YYYY-MM-DDThh:mm:ss"`, or `dt"YYYY-MM-DD hh:mm:ss"`. When specified as a literal, using the DATE function in the expression is not necessary. To learn more, see [DAX Syntax | Date and time](#).
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Examples

Simple Date

The following formula returns the date July 8, 2009:

```
= DATE(2009,7,8)
```

Years before 1899

If the value that you enter for the **year** argument is between 0 (zero) and 1899 (inclusive), that value is added to 1900 to calculate the year. The following formula returns January 2, 1908: (1900+08).

```
= DATE(08,1,2)
```

Years after 1899

If **year** is between 1900 and 9999 (inclusive), that value is used as the year. The following formula returns January 2, 2008:

```
= DATE(2008,1,2)
```

Months

If **month** is greater than 12, **month** adds that number of months to the first month in the year specified. The following formula returns the date February 2, 2009:

```
= DATE(2008,14,2)
```

Days

If **day** is greater than the number of days in the month specified, **day** adds that number of days to the first day in the month. The following formula returns the date February 4, 2008:

```
= DATE(2008,1,35)
```

See also

[Date and time functions](#)

[DAY function](#)

[TODAY function](#)

DATEDIFF

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the number of interval boundaries between two dates.

Syntax

```
DATEDIFF(<Date1>, <Date2>, <Interval>)
```

Parameters

| TERM | DEFINITION |
|----------|--|
| Date1 | A scalar datetime value. |
| Date2 | A scalar datetime value. |
| Interval | <p>The interval to use when comparing dates. The value can be one of the following:</p> <ul style="list-style-type: none">- SECOND- MINUTE- HOUR- DAY- WEEK- MONTH- QUARTER- YEAR |

Return value

The count of interval boundaries between two dates.

Remarks

A positive result is returned if Date2 is larger than Date1. A negative result is returned if Date1 is larger than Date2.

Example

Examples in this article can be used with the sample Adventure Works DW 2020 Power BI Desktop model. To get the model, see [DAX sample model](#).

The following DAX query:

```

EVALUATE
VAR StartDate = DATE ( 2019, 07, 01 )
VAR EndDate = DATE ( 2021, 12, 31 )
RETURN
{
    ( "Year", DATEDIFF ( StartDate, EndDate, YEAR ) ),
    ( "Quarter", DATEDIFF ( StartDate, EndDate, QUARTER ) ),
    ( "Month", DATEDIFF ( StartDate, EndDate, MONTH ) ),
    ( "Week", DATEDIFF ( StartDate, EndDate, WEEK ) ),
    ( "Day", DATEDIFF ( StartDate, EndDate, DAY ) )
}

```

Returns the following:

| VALUE1 | VALUE2 |
|---------|--------|
| Year | 2 |
| Quarter | 9 |
| Month | 29 |
| Week | 130 |
| Day | 914 |

DATEVALUE

10/31/2022 • 2 minutes to read • [Edit Online](#)

Converts a date in text format to a date in datetime format.

Syntax

```
DATEVALUE(date_text)
```

Parameters

| TERM | DEFINITION |
|-----------|------------------------------|
| date_text | Text that represents a date. |

Property Value/Return value

A date in **datetime** format.

Remarks

- When converting, DATEVALUE uses the locale and date/time settings of the model to determine a date value. If the model date/time settings represent dates in the format of Month/Day/Year, then the string, "1/8/2009", is converted to a **datetime** value equivalent to January 8th of 2009. However, if the model date/time settings represent dates in the format of Day/Month/Year, the same string is converted as a **datetime** value equivalent to August 1st of 2009.
- If conversion using the locale and date/time settings of the model fails, DATEVALUE will attempt to use other date formats. In this case, some rows may be converted using one format and other rows are converted using a different format. For example, "5/4/2018" may convert to May 4th of 2018, and "20/4/2018" may convert to April 20th.
- If the year portion of the **date_text** argument is omitted, the DATEVALUE function uses the current year from your computer's built-in clock. Time information in the **date_text** argument is ignored.
- Model locale and data/time settings are initially determined by the application and computer when the model is created.
- Date and datetime can also be specified as a literal in the format `dt"YYYY-MM-DD"`, `dt"YYYY-MM-DDThh:mm:ss"`, or `dt"YYYY-MM-DD hh:mm:ss"`. When specified as a literal, using the DATEVALUE function in the expression is not necessary. To learn more, see [DAX Syntax | Date and time](#).

Example

The following example returns a different **datetime** value depending on the model locale and settings for how dates and times are presented.

- In date/time settings where the day precedes the month, the example returns a **datetime** value corresponding to January 8th of 2009.
- In date/time settings where the month precedes the day, the example returns a **datetime** value

corresponding to August 1st of 2009.

```
= DATEVALUE("8/1/2009")
```

See also

[Date and time functions](#)

DAY

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the day of the month, a number from 1 to 31.

Syntax

```
DAY(<date>)
```

Parameters

| TERM | DEFINITION |
|------|---|
| date | A date in datetime format, or a text representation of a date. |

Return value

An integer number indicating the day of the month.

Remarks

- The DAY function takes as an argument the date of the day you are trying to find. Dates can be provided to the function by using another date function, by using an expression that returns a date, or by typing a date in a **datetime** format. You can also type a date in one of the accepted string formats for dates.
- Values returned by the YEAR, MONTH and DAY functions will be Gregorian values regardless of the display format for the supplied date value. For example, if the display format of the supplied date is Hijri, the returned values for the YEAR, MONTH and DAY functions will be values associated with the equivalent Gregorian date.
- When the date argument is a text representation of the date, the day function uses the locale and date/time settings of the client computer to understand the text value in order to perform the conversion. If the current date/time settings represent dates in the format of Month/Day/Year, then the string, "1/8/2009", is interpreted as a **datetime** value equivalent to January 8th of 2009, and the function returns 8. However, if the current date/time settings represent dates in the format of Day/Month/Year, the same string would be interpreted as a **datetime** value equivalent to August 1st of 2009, and the function returns 1.

Example: Getting the day from a date column

The following formula returns the day from the date in the column, [Birthdate].

```
= DAY([Birthdate])
```

Example - Getting the day from a string date

The following formulas return the day, 4, using dates that have been supplied as strings in an accepted text format.

```
= DAY("3-4-1007")  
= DAY("March 4 2007")
```

Example - Using a day value as a condition

The following expression returns the day that each sales order was placed, and flags the row as a promotional sale item if the order was placed on the 10th of the month.

```
= IF( DAY([SalesDate])=10,"promotion","")
```

See also

[Date and time functions](#)

[TODAY function](#)

[DATE function](#)

EDATE

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the date that is the indicated number of months before or after the start date. Use EDATE to calculate maturity dates or due dates that fall on the same day of the month as the date of issue.

Syntax

```
EDATE(<start_date>, <months>)
```

Parameters

| TERM | DEFINITION |
|------------|---|
| start_date | A date in datetime or text format that represents the start date. |
| months | An integer that represents the number of months before or after start_date . |

Return value

A date (**datetime**).

Remarks

- In contrast to Microsoft Excel, which stores dates as sequential serial numbers, DAX works with dates in a **datetime** format. Dates stored in other formats are converted implicitly.
- If **start_date** is not a valid date, EDATE returns an error. Make sure that the column reference or date that you supply as the first argument is a date.
- If **months** is not an integer, it is truncated.
- When the date argument is a text representation of the date, the EDATE function uses the locale and date time settings of the client computer to understand the text value in order to perform the conversion. If the current date time settings represent a date in the format of Month/Day/Year, then the following string "1/8/2009" is interpreted as a datetime value equivalent to January 8th of 2009. However, if the current date time settings represent a date in the format of Day/Month/Year, the same string would be interpreted as a datetime value equivalent to August 1st of 2009.
- If the requested date is past the last day of the corresponding month, then the last day of the month is returned. For example, the following functions: EDATE("2009-01-29", 1), EDATE("2009-01-30", 1), EDATE("2009-01-31", 1) return February 28th of 2009; that corresponds to one month after the start date.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following example returns the date three months after the order date, which is stored in the column [TransactionDate].

```
= EDATE([TransactionDate],3)
```

See also

[EOMONTH function](#)

[Date and time functions](#)

EOMONTH

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the date in **datetime** format of the last day of the month, before or after a specified number of months. Use EOMONTH to calculate maturity dates or due dates that fall on the last day of the month.

Syntax

```
EOMONTH(<start_date>, <months>)
```

Parameters

| TERM | DEFINITION |
|------------|--|
| start_date | The start date in datetime format, or in an accepted text representation of a date. |
| months | A number representing the number of months before or after the start_date . Note: If you enter a number that is not an integer, the number is rounded up or down to the nearest integer. |

Return value

A date (**datetime**).

Remarks

- In contrast to Microsoft Excel, which stores dates as sequential serial numbers, DAX works with dates in a **datetime** format. The EOMONTH function can accept dates in other formats, with the following restrictions:
- If **start_date** is not a valid date, EOMONTH returns an error.
- If **start_date** is a numeric value that is not in a **datetime** format, EOMONTH will convert the number to a date. To avoid unexpected results, convert the number to a **datetime** format before using the EOMONTH function.
- If **start_date** plus months yields an invalid date, EOMONTH returns an error. Dates before March 1st of 1900 and after December 31st of 9999 are invalid.
- When the date argument is a text representation of the date, the EDATE function uses the locale and date time settings, of the client computer, to understand the text value in order to perform the conversion. If current date time settings represent a date in the format of Month/Day/Year, then the following string "1/8/2009" is interpreted as a datetime value equivalent to January 8th of 2009. However, if the current date time settings represent a date in the format of Day/Month/Year, the same string would be interpreted as a datetime value equivalent to August 1st of 2009.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following expression returns May 31, 2008, because the **months** argument is rounded to 2.

```
= EOMONTH("March 3, 2008",1.5)
```

See also

[EDATE function](#)

[Date and time functions](#)

HOUR

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the hour as a number from 0 (12:00 A.M.) to 23 (11:00 P.M.).

Syntax

```
HOUR(<datetime>)
```

Parameters

| TERM | DEFINITION |
|----------|---|
| datetime | A datetime value, such as 16:48:00 or 4:48 PM. |

Return value

An integer number from 0 to 23.

Remarks

- The HOUR function takes as argument the time that contains the hour you want to find. You can supply the time by using a date/time function, an expression that returns a **datetime**, or by typing the value directly in one of the accepted time formats. Times can also be entered as any accepted text representation of a time.
- When the **datetime** argument is a text representation of the date and time, the function uses the locale and date/time settings of the client computer to understand the text value in order to perform the conversion. Most locales use the colon (:) as the time separator and any input text using colons as time separators will parse correctly. Review your locale settings to understand your results.

Example 1

The following example returns the hour from the **TransactionTime** column of a table named **Orders**.

```
= HOUR('Orders'[TransactionTime])
```

Example 2

The following example returns 15, meaning the hour corresponding to 3 PM in a 24-hour clock. The text value is automatically parsed and converted to a date/time value.

```
= HOUR("March 3, 2008 3:00 PM")
```

See also

[Date and time functions](#)

MINUTE function

YEAR function

SECOND function

MINUTE

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the minute as a number from 0 to 59, given a date and time value.

Syntax

```
MINUTE(<datetime>)
```

Parameters

| TERM | DEFINITION |
|----------|--|
| datetime | A datetime value or text in an accepted time format, such as 16:48:00 or 4:48 PM. |

Return value

An integer number from 0 to 59.

Remarks

- In contrast to Microsoft Excel, which stores dates and times in a serial numeric format, DAX uses a **datetime** data type for dates and times. You can provide the **datetime** value to the MINUTE function by referencing a column that stores dates and times, by using a date/time function, or by using an expression that returns a date and time.
- When the **datetime** argument is a text representation of the date and time, the function uses the locale and date/time settings of the client computer to understand the text value in order to perform the conversion. Most locales use the colon (:) as the time separator and any input text using colons as time separators will parse correctly. Verify your locale settings to understand your results.

Example 1

The following example returns the minute from the value stored in the **TransactionTime** column of the **Orders** table.

```
= MINUTE(Orders[TransactionTime])
```

Example 2

The following example returns 45, which is the number of minutes in the time 1:45 PM.

```
= MINUTE("March 23, 2008 1:45 PM")
```

See also

Date and time functions

HOUR function

YEAR function

SECOND function

MONTH

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the month as a number from 1 (January) to 12 (December).

Syntax

```
MONTH(<datetime>)
```

Parameters

| TERM | DEFINITION |
|------|---|
| date | A date in datetime or text format. |

Return value

An integer number from 1 to 12.

Remarks

- In contrast to Microsoft Excel, which stores dates as serial numbers, DAX uses a **datetime** format when working with dates. You can enter the date used as argument to the MONTH function by typing an accepted **datetime** format, by providing a reference to a column that contains dates, or by using an expression that returns a date.
- Values returned by the YEAR, MONTH and DAY functions will be Gregorian values regardless of the display format for the supplied date value. For example, if the display format of the supplied date is Hijri, the returned values for the YEAR, MONTH and DAY functions will be values associated with the equivalent Gregorian date.
- When the date argument is a text representation of the date, the function uses the locale and date time settings of the client computer to understand the text value in order to perform the conversion. If the current date time settings represent a date in the format of Month/Day/Year, then the following string "1/8/2009" is interpreted as a datetime value equivalent to January 8th of 2009, and the function yields a result of 1. However, if the current date time settings represent a date in the format of Day/Month/Year, then the same string would be interpreted as a datetime value equivalent to August 1st of 2009, and the function yields a result of 8.
- If the text representation of the date cannot be correctly converted to a datetime value, the function returns an error.

Example 1

The following expression returns 3, which is the integer corresponding to March, the month in the **date** argument.

```
= MONTH("March 3, 2008 3:45 PM")
```

Example 2

The following expression returns the month from the date in the **TransactionDate** column of the **Orders** table.

```
= MONTH(Orders[TransactionDate])
```

See also

[Date and time functions](#)

[HOUR function](#)

[MINUTE function](#)

[YEAR function](#)

[SECOND function](#)

NETWORKDAYS

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the number of whole workdays between two dates (inclusive). Parameters specify which and how many days are weekend days. Weekend days and days specified as holidays are not considered as workdays.

Syntax

```
NETWORKDAYS(<start_date>, <end_date>[, <weekend>, <holidays>])
```

Parameters

| TERM | DEFINITION |
|------------|--|
| start_date | A date that represents the start date. The dates for which the difference is to be computed. The start_date can be earlier than, the same as, or later than the end_date. |
| end_date | A date that represents the end date. The dates for which the difference is to be computed. The start_date can be earlier than, the same as, or later than the end_date. |
| weekend | Indicates the days of the week that are weekend days and are not included in the number of whole working days between start_date and end_date. Weekend is a weekend number that specifies when weekends occur. Weekend number values indicate the following weekend days: 1 or omitted: Saturday, Sunday 2: Sunday, Monday 3: Monday, Tuesday 4: Tuesday, Wednesday 5: Wednesday, Thursday 6: Thursday, Friday 7: Friday, Saturday 11: Sunday only 12: Monday only 13: Tuesday only 14: Wednesday only 15: Thursday only 16: Friday only 17: Saturday only |
| holidays | A column table of one or more dates that are to be excluded from the working day calendar. |

Return Value

An integer number of whole workdays.

Remarks

- This DAX function is similar to Excel NETWORKDAYS.INTL and NETWORKDAYS functions.

- If start_date and end_date both are BLANK, the output value is also BLANK.
- If either start_date or end_date is BLANK, the BLANK start_date or end_date will be treated as Date(1899, 12, 30).
- Dates must be specified by using [DATE function](#) or as the result of another expression. For example, `DATE (2022, 5, 30)`, specifies May 30th, 2022. Dates can also be specified as a [literal](#) in format, `(dt"2022-05-30")`. Do not specify dates as text.

Example

The following expression:

```
= NETWORKDAYS (
    DATE ( 2022, 5, 28 ),
    DATE ( 2022, 5, 30 ),
    1,
    {
        DATE ( 2022, 5, 30 )
    }
)
```

Returns:

| [VALUE] |
|---------|
| 0 |

In this example, 0 is returned because the start date is a Saturday and the end date is a Monday. The weekend parameter specifies that the weekend is Saturday and Sunday, so those are not work days. The holiday parameter marks the 30th (the end date) as a holiday, so no working days remain.

NOW

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the current date and time in **datetime** format.

The NOW function is useful when you need to display the current date and time on a worksheet or calculate a value based on the current date and time, and have that value updated each time you open the worksheet.

Syntax

```
NOW()
```

Return value

A date (**datetime**).

Remarks

- The result of the NOW function changes only when the column that contains the formula is refreshed. It is not updated continuously.
- In the Power BI Service, the result of the NOW function is always in the UTC timezone.
- The TODAY function returns the same date but is not precise with regard to time; the time returned is always 12:00:00 AM and only the date is updated.

Example

The following example returns the current date and time plus 3.5 days:

```
= NOW()+3.5
```

See also

[UTCNOW function](#)

[TODAY function](#)

QUARTER

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the quarter as a number from 1 (January – March) to 4 (October – December).

Syntax

```
QUARTER(<date>)
```

Parameters

| TERM | DEFINITION |
|------|------------|
| date | A date. |

Return value

An integer number from 1 to 4.

Remarks

If the input value is BLANK, the output value is also BLANK.

Example 1

The following DAX query:

```
EVALUATE { QUARTER(DATE(2019, 2, 1)), QUARTER(DATE(2018, 12, 31)) }
```

Returns:

| [VALUE] |
|---------|
| 1 |
| 4 |

Example 2

The following DAX query:

```
EVALUATE
ADDCOLUMNS(
    FILTER(
        VALUES(
            FactInternetSales[OrderDate]),
            [OrderDate] >= DATE(2008, 3, 31) && [OrderDate] <= DATE(2008, 4, 1)
        ),
        "Quarter", QUARTER([OrderDate])
    )
)
```

Returns:

| FACTINTERNETSALES[ORDERDATE] | [QUARTER] |
|------------------------------|-----------|
| 3/31/2008 | 1 |
| 4/1/2008 | 2 |

See also

[YEAR](#)

[MONTH](#)

[DAY](#)

SECOND

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the seconds of a time value, as a number from 0 to 59.

Syntax

```
SECOND(<time>)
```

Parameters

| TERM | DEFINITION |
|------|---|
| time | A time in datetime format, such as 16:48:23 or 4:48:47 PM. |

Return value

An integer number from 0 to 59.

Remarks

- In contrast to Microsoft Excel, which stores dates and times as serial numbers, DAX uses a **datetime** format when working with dates and times. If the source data is not in this format, DAX implicitly converts the data. You can use formatting to display the dates and times as a serial number if you need to.
- The date/time value that you supply as an argument to the SECOND function can be entered as a text string within quotation marks (for example, "6:45 PM"). You can also provide a time value as the result of another expression, or as a reference to a column that contains times.
- If you provide a numeric value of another data type, such as 13.60, the value is interpreted as a serial number and is represented as a **datetime** data type before extracting the value for seconds. To make it easier to understand your results, you might want to represent such numbers as dates before using them in the SECOND function. For example, if you use SECOND with a column that contains a numeric value such as, 25.56, the formula returns 24. That is because, when formatted as a date, the value 25.56 is equivalent to January 25, 1900, 1:26:24 PM.
- When the **time** argument is a text representation of a date and time, the function uses the locale and date/time settings of the client computer to understand the text value in order to perform the conversion. Most locales use the colon (:) as the time separator and any input text using colons as time separators will parse correctly. Review your locale settings to understand your results.

Example 1

The following formula returns the number of seconds in the time contained in the **TransactionTime** column of a table named **Orders**.

```
= SECOND('Orders'[TransactionTime])
```

Example 2

The following formula returns 3, which is the number of seconds in the time represented by the value, **March 3, 2008 12:00:03**.

```
= SECOND("March 3, 2008 12:00:03")
```

See also

[Date and time functions](#)

[HOUR](#)

[MINUTE](#)

[YEAR](#)

TIME

10/31/2022 • 2 minutes to read • [Edit Online](#)

Converts hours, minutes, and seconds given as numbers to a time in **datetime** format.

Syntax

```
TIME(hour, minute, second)
```

Parameters

| TERM | DEFINITION |
|--------|--|
| hour | <p>Import mode: A number from 0 to 32767 representing the hour.</p> <p>Any value greater than 23 will be divided by 24 and the remainder will be treated as the hour value, represented as a fraction of a day.</p> <p>For example, TIME(27,0,0) = TIME(3,0,0) = 3:00:00 AM</p> <p>DirectQuery mode: A number from 0 to 23 representing the hour.</p> |
| minute | <p>Import mode: A number from 0 to 32767 representing the minute.</p> <p>Any value greater than 59 minutes will be converted to hours and minutes.</p> <p>Any value greater than 1440 (24 hours) does not alter the date portion - instead, it will be divided by 1440 and the remainder will be treated as the minute value, represented as a fraction of a day.</p> <p>For example, TIME(0,2190,0) = TIME(0,750,0) = TIME(12,30,0) = 12:30:00 PM</p> <p>DirectQuery mode: A number from 0 to 59 representing the minute.</p> |
| second | <p>Import mode: A number from 0 to 32767 representing the second.</p> <p>Any value greater than 59 will be converted to hours, minutes, and seconds.</p> <p>For example, TIME(0,0,2000) = TIME(0,33,20) = 12:33:20 AM</p> <p>DirectQuery mode: A number from 0 to 59 representing the second.</p> |

Return value

A time (**datetime**) ranging from 00:00:00 (12:00:00 AM) to 23:59:59 (11:59:59 PM).

Remarks

- In contrast to Microsoft Excel, which stores dates and times as serial numbers, DAX works with date and

time values in a **datetime** format. Numbers in other formats are implicitly converted when you use a date/time value in a DAX function. If you need to use serial numbers, you can use formatting to change the way that the numbers are displayed.

- Time values are a portion of a date value, and in the serial number system are represented by a decimal number. Therefore, the **datetime** value 12:00 PM is equivalent to 0.5, because it is half of a day.
- You can supply the arguments to the TIME function as values that you type directly, as the result of another expression, or by a reference to a column that contains a numeric value.
- Date and datetime can also be specified as a literal in the format `dt"YYYY-MM-DD"`, `dt"YYYY-MM-DDThh:mm:ss"`, or `dt"YYYY-MM-DD hh:mm:ss"`. When specified as a literal, using the TIME function in the expression is not necessary. To learn more, see [DAX Syntax | Date and time](#).
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example 1

The following examples both return the time, 3:00 AM:

```
= TIME(27,0,0)
```

```
= TIME(3,0,0)
```

Example 2

The following examples both return the time, 12:30 PM:

```
= TIME(0,750,0)
```

```
= TIME(12,30,0)
```

Example 3

The following example creates a time based on the values in the columns, `intHours`, `intMinutes`, `intSeconds` :

```
= TIME([intHours],[intMinutes],[intSeconds])
```

See also

[DATE](#)

[Date and time functions](#)

TIMEVALUE

10/31/2022 • 2 minutes to read • [Edit Online](#)

Converts a time in text format to a time in datetime format.

Syntax

```
TIMEVALUE(time_text)
```

Parameters

| TERM | DEFINITION |
|-----------|---|
| time_text | A text string that represents a certain time of the day. Any date information included in the time_text argument is ignored. |

Return value

A date (**datetime**).

Remarks

- Time values are a portion of a date value and represented by a decimal number. For example, 12:00 PM is represented as 0.5 because it is half of a day.
- When the **time_text** argument is a text representation of the date and time, the function uses the locale and date/time settings of the model to understand the text value in order to perform the conversion. Most locales use the colon (:) as the time separator, and any input text using colons as time separators will parse correctly. Review your locale settings to understand your results.
- Date and datetime can also be specified as a literal in the format `dt"YYYY-MM-DD"`, `dt"YYYY-MM-DDThh:mm:ss"`, or `dt"YYYY-MM-DD hh:mm:ss"`. When specified as a literal, using the TIMEVALUE function in the expression is not necessary. To learn more, see [DAX Syntax | Date and time](#).

Example

```
= TIMEVALUE("20:45:30")
```

See also

[Date and time functions](#)

TODAY

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the current date.

Syntax

```
TODAY()
```

Return value

A date (**datetime**).

Remarks

- The TODAY function is useful when you need to have the current date displayed on a worksheet, regardless of when you open the workbook. It is also useful for calculating intervals.
- If the TODAY function does not update the date when you expect it to, you might need to change the settings that control when the column or workbook is refreshed..
- The NOW function is similar but returns the exact time, whereas TODAY returns the time value 12:00:00 PM for all dates.

Example

If you know that someone was born in 1963, you might use the following formula to find that person's age as of this year's birthday:

```
= YEAR(TODAY())-1963
```

This formula uses the TODAY function as an argument for the YEAR function to obtain the current year, and then subtracts 1963, returning the person's age.

See also

[Date and time functions](#)

[NOW](#)

UTCNOW

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the current UTC date and time.

Syntax

```
UTCNOW()
```

Return value

A **(datetime)**.

Remarks

The result of the UTCNOW function changes only when the formula is refreshed. It is not continuously updated.

Example

The following:

```
EVALUATE { FORMAT(UTCNOW(), "General Date") }
```

Returns:

```
[VALUE]
```

2/2/2018 4:48:08 AM

See also

[NOW function](#)

[UTCTODAY function](#)

UTCTODAY

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the current UTC date.

Syntax

```
UTCTODAY()
```

Return value

A date.

Remarks

- UTCTODAY returns the time value 12:00:00 PM for all dates.
- The UTCNOW function is similar but returns the exact time and date.

Example

The following:

```
EVALUATE { FORMAT(UTCTODAY(), "General Date") }
```

Returns:

```
[VALUE]
```

```
2/2/2018
```

See also

[NOW function](#)

[UTCNOW function](#)

WEEKDAY

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns a number from 1 to 7 identifying the day of the week of a date. By default the day ranges from 1 (Sunday) to 7 (Saturday).

Syntax

```
WEEKDAY(<date>, <return_type>)
```

Parameters

| TERM | DEFINITION |
|-------------|---|
| date | <p>A date in datetime format.</p> <p>Dates should be entered by using the DATE function, by using expressions that result in a date, or as the result of other formulas.</p> |
| return_type | <p>A number that determines the Return value:</p> <p>Return type: 1, week begins on Sunday (1) and ends on Saturday (7). numbered 1 through 7.</p> <p>Return type: 2, week begins on Monday (1) and ends on Sunday (7).</p> <p>Return type: 3, week begins on Monday (0) and ends on Sunday (6).numbered 1 through 7.</p> |

Return value

An integer number from 1 to 7.

Remarks

- In contrast to Microsoft Excel, which stores dates as serial numbers, DAX works with dates and times in a **datetime** format. If you need to display dates as serial numbers, you can use the formatting options in Excel.
- You can also type dates in an accepted text representation of a date, but to avoid unexpected results, it is best to convert the text date to a **datetime** format first.
- When the date argument is a text representation of the date, the function uses the locale and date/time settings of the client computer to understand the text value in order to perform the conversion. If the current date/time settings represent dates in the format of Month/Day/Year, then the string, "1/8/2009", is interpreted as a **datetime** value equivalent to January 8th of 2009. However, if the current date/time settings represent dates in the format of Day/Month/Year, then the same string would be interpreted as a **datetime** value equivalent to August 1st of 2009.

Example

The following example gets the date from the [HireDate] column, adds 1, and displays the weekday corresponding to that date. Because the **return_type** argument has been omitted, the default format is used, in which 1 is Sunday and 7 is Saturday. If the result is 4, the day would be Wednesday.

```
= WEEKDAY([HireDate]+1)
```

See also

[Date and time functions](#)

[WEEKNUM function](#)

[YEARFRAC function](#)

WEEKNUM

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the week number for the given date according to the **return_type** value. The week number indicates where the week falls numerically within a year.

There are two *systems* used for this function:

- **System 1** - The week containing January 1 is the first week of the year and is numbered week 1.
- **System 2** - The week containing the first Thursday of the year is the first week of the year and is numbered as week 1. This system is the methodology specified in ISO 8601, which is commonly known as the European week numbering system.

Syntax

```
WEEKNUM(<date>[, <return_type>])
```

Parameters

| TERM | DEFINITION |
|-------------|--|
| date | The date in datetime format. |
| return_type | (Optional) A number that determines on which day the week begins. Default is 1. See Remarks. |
| | |

Return value

An integer number.

Remarks

- By default, the WEEKNUM function uses a calendar convention in which the week containing January 1 is considered to be the first week of the year. However, the ISO 8601 calendar standard, widely used in Europe, defines the first week as the one with the majority of days (four or more) falling in the new year. This means that if **return_type** is any valid value other than 21, for any years in which there are three days or less in the first week of January, the WEEKNUM function returns week numbers that are different from the ISO 8601 definition.
- For **return_type**, except for 21, the following valid values may not be supported by some DirectQuery data sources:

| RETURN_TYPE | WEEK BEGINS ON | SYSTEM |
|--------------|----------------|--------|
| 1 or omitted | Sunday | 1 |
| 2 | Monday | 1 |

| RETURN_TYPE | WEEK BEGINS ON | SYSTEM |
|-------------|----------------|--------|
| 11 | Monday | 1 |
| 12 | Tuesday | 1 |
| 13 | Wednesday | 1 |
| 14 | Thursday | 1 |
| 15 | Friday | 1 |
| 16 | Saturday | 1 |
| 17 | Sunday | 1 |
| 21 | Monday | 2 |

Example 1

The following example returns the week number for February 14, 2010. This calculation assumes weeks begin on Monday.

```
= WEEKNUM("Feb 14, 2010", 2)
```

Example 2

The following example returns the week number of the date stored in the column, **HireDate**, from the table, **Employees**. This calculation assumes weeks begin on Sunday.

```
= WEEKNUM('Employees'[HireDate])
```

See also

[YEARFRAC function](#)

[WEEKDAY function](#)

YEAR

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the year of a date as a four digit integer in the range 1900-9999.

Syntax

```
YEAR(<date>)
```

Parameters

| TERM | DEFINITION |
|------|---|
| date | A date in datetime or text format, containing the year you want to find. |

Return value

An integer in the range 1900-9999.

Remarks

- In contrast to Microsoft Excel, which stores dates as serial numbers, DAX uses a **datetime** data type to work with dates and times.
- Dates should be entered by using the DATE function, or as results of other formulas or functions. You can also enter dates in accepted text representations of a date, such as March 3, 2007, or Mar-3-2003.
- Values returned by the YEAR, MONTH, and DAY functions will be Gregorian values regardless of the display format for the supplied date value. For example, if the display format of the supplied date uses the Hijri calendar, the returned values for the YEAR, MONTH, and DAY functions will be values associated with the equivalent Gregorian date.
- When the date argument is a text representation of the date, the function uses the locale and date time settings of the client computer to understand the text value in order to perform the conversion. Errors may arise if the format of strings is incompatible with the current locale settings. For example, if your locale defines dates to be formatted as month/day/year, and the date is provided as day/month/year, then 25/1/2009 will not be interpreted as January 25th of 2009 but as an invalid date.

Example

The following example returns 2007.

```
= YEAR("March 2007")
```

Example - Date as result of expression

Description

The following example returns the year for today's date.

```
= YEAR(TODAY())
```

See also

[Date and time functions](#)

[HOUR function](#)

[MINUTE function](#)

[YEAR function](#)

[SECOND function](#)

YEARFRAC

10/31/2022 • 2 minutes to read • [Edit Online](#)

Calculates the fraction of the year represented by the number of whole days between two dates. Use the YEARFRAC worksheet function to identify the proportion of a whole year's benefits or obligations to assign to a specific term.

Syntax

```
YEARFRAC(<start_date>, <end_date>, <basis>)
```

Parameters

| TERM | DEFINITION |
|------------|---|
| start_date | The start date in datetime format. |
| end_date | The end date in datetime format. |
| basis | <p>(Optional) The type of day count basis to use. All arguments are truncated to integers.</p> <p>Basis - Description</p> <p>0 - US (NASD) 30/360 (Default value)</p> <p>1 - Actual/actual</p> <p>2 - Actual/360</p> <p>3 - Actual/365</p> <p>4 - European 30/360</p> |

Return value

A decimal number. The internal data type is a signed IEEE 64-bit (8-byte) double-precision floating-point number.

Remarks

- In contrast to Microsoft Excel, which stores dates as serial numbers, DAX uses a **datetime** format to work with dates and times. If you need to view dates as serial numbers, you can use the formatting options in Excel.
- If **start_date** or **end_date** are not valid dates, YEARFRAC returns an error.
- If **basis** < 0 or if **basis** > 4, YEARFRAC returns an error.

Example 1

The following example returns the fraction of a year represented by the difference between the dates in the two

columns, `TransactionDate` and `ShippingDate` :

```
= YEARFRAC(Orders[TransactionDate],Orders[ShippingDate])
```

Example 2

The following example returns the fraction of a year represented by the difference between the dates, January 1 and March 1:

```
= YEARFRAC("Jan 1 2007","Mar 1 2007")
```

Use four-digit years whenever possible, to avoid getting unexpected results. When the year is truncated, the current year is assumed. When the date is or omitted, the first date of the month is assumed.

The second argument, **basis**, has also been omitted. Therefore, the year fraction is calculated according to the US (NASD) 30/360 standard.

See also

[Date and time functions](#)

[WEEKNUM function](#)

[YEARFRAC function](#)

[WEEKDAY function](#)

Filter functions

10/31/2022 • 2 minutes to read • [Edit Online](#)

The filter and value functions in DAX are some of the most complex and powerful, and differ greatly from Excel functions. The lookup functions work by using tables and relationships, like a database. The filtering functions let you manipulate data context to create dynamic calculations.

In this category

| FUNCTION | DESCRIPTION |
|----------------------------------|---|
| ALL | Returns all the rows in a table, or all the values in a column, ignoring any filters that might have been applied. |
| ALLCROSSFILTERED | Clear all filters which are applied to a table. |
| ALLEXCEPT | Removes all context filters in the table except filters that have been applied to the specified columns. |
| ALLNOBLANKROW | From the parent table of a relationship, returns all rows but the blank row, or all distinct values of a column but the blank row, and disregards any context filters that might exist. |
| ALLSELECTED | Removes context filters from columns and rows in the current query, while retaining all other context filters or explicit filters. |
| CALCULATE | Evaluates an expression in a modified filter context. |
| CALCULATETABLE | Evaluates a table expression in a modified filter context. |
| EARLIER | Returns the current value of the specified column in an outer evaluation pass of the mentioned column. |
| EARLIEST | Returns the current value of the specified column in an outer evaluation pass of the specified column. |
| FILTER | Returns a table that represents a subset of another table or expression. |
| KEEPFILTERS | Modifies how filters are applied while evaluating a CALCULATE or CALCULATETABLE function. |
| LOOKUPVALUE | Returns the value for the row that meets all criteria specified by search conditions. The function can apply one or more search conditions. |
| REMOVEFILTERS | Clears filters from the specified tables or columns. |

| FUNCTION | DESCRIPTION |
|---------------|---|
| SELECTEDVALUE | Returns the value when the context for columnName has been filtered down to one distinct value only. Otherwise returns alternateResult. |

ALL

10/31/2022 • 7 minutes to read • [Edit Online](#)

Returns all the rows in a table, or all the values in a column, ignoring any filters that might have been applied. This function is useful for clearing filters and creating calculations on all the rows in a table.

Syntax

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

Parameters

| TERM | DEFINITION |
|--------|---|
| table | The table that you want to clear filters on. |
| column | The column that you want to clear filters on. |

The argument to the ALL function must be either a reference to a base table or a reference to a base column. You cannot use table expressions or column expressions with the ALL function.

Return value

The table or column with filters removed.

Remarks

- This function is not used by itself, but serves as an intermediate function that can be used to change the set of results over which some other calculation is performed.
- The normal behavior for DAX expressions containing the ALL() function is that any filters applied will be ignored. However, there are some scenarios where this is not the case because of *auto-exist*, a DAX technology that optimizes filtering in order to reduce the amount of processing required for certain DAX queries. An example where auto-exist and ALL() provide unexpected results is when filtering on two or more columns of the same table (like when using slicers), and there is a measure on that same table that uses ALL(). In this case, auto-exist will *merge* the multiple filters into one and will only filter on existing combinations of values. Because of this merge, the measure will be calculated on the existing combinations of values and the result will be based on filtered values instead of all values as expected. To learn more about auto-exist and its effect on calculations, see Microsoft MVP Alberto Ferrari's [Understanding DAX Auto-Exist](#) article on sql.bi.com.
- The following table describes how you can use the ALL and ALLEXCEPT functions in different scenarios.

| FUNCTION AND USAGE | DESCRIPTION |
|--------------------|--|
| ALL() | Removes all filters everywhere. ALL() can only be used to clear filters but not to return a table. |

| FUNCTION AND USAGE | DESCRIPTION |
|---|---|
| ALL(Table) | Removes all filters from the specified table. In effect, ALL(Table) returns all of the values in the table, removing any filters from the context that otherwise might have been applied. This function is useful when you are working with many levels of grouping, and want to create a calculation that creates a ratio of an aggregated value to the total value. The first example demonstrates this scenario. |
| ALL (Column[, Column[, ...]]) | Removes all filters from the specified columns in the table; all other filters on other columns in the table still apply. All column arguments must come from the same table. The ALL(Column) variant is useful when you want to remove the context filters for one or more specific columns and to keep all other context filters. The second and third examples demonstrate this scenario. |
| ALLEXCEPT(Table, Column1 [,Column2]...) | Removes all context filters in the table except filters that are applied to the specified columns. This is a convenient shortcut for situations in which you want to remove the filters on many, but not all, columns in a table. |

- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example 1

Calculate ratio of Category Sales to Total Sales

Assume that you want to find the amount of sales for the current cell, in your PivotTable, divided by the total sales for all resellers. To ensure that the denominator is the same regardless of how the PivotTable user might be filtering or grouping the data, you define a formula that uses ALL to create the correct grand total.

The following table shows the results when a new measure, **All Reseller Sales Ratio**, is created using the formula shown in the code section. To see how this works, add the field, CalendarYear, to the **Row Labels** area of the PivotTable, and add the field, ProductCategoryName, to the **Column Labels** area. Then, drag the measure, **All Reseller Sales Ratio**, to the **Values** area of the Pivot Table. To view the results as percentages, use the formatting features of Excel to apply a percentage number formatting to the cells that contains the measure.

| ROW LABELS | ACCESSORIES | BIKES | CLOTHING | COMPONENTS | GRAND TOTAL |
|-------------|-------------|--------|----------|------------|-------------|
| 2005 | 0.02% | 9.10% | 0.04% | 0.75% | 9.91% |
| 2006 | 0.11% | 24.71% | 0.60% | 4.48% | 29.90% |
| 2007 | 0.36% | 31.71% | 1.07% | 6.79% | 39.93% |
| 2008 | 0.20% | 16.95% | 0.48% | 2.63% | 20.26% |
| Grand Total | 0.70% | 82.47% | 2.18% | 14.65% | 100.00% |

Formula

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

The formula is constructed as follows:

1. The numerator, `SUMX(ResellerSales_USD, ResellerSales_USD[SalesAmount_USD])`, is the sum of the values in `ResellerSales_USD[SalesAmount_USD]` for the current cell in the PivotTable, with context filters applied on `CalendarYear` and `ProductCategoryName`.
2. For the denominator, you start by specifying a table, `ResellerSales_USD`, and use the `ALL` function to remove all context filters on the table.
3. You then use the `SUMX` function to sum the values in the `ResellerSales_USD[SalesAmount_USD]` column. In other words, you get the sum of `ResellerSales_USD[SalesAmount_USD]` for all resellers sales.

Example 2

Calculate Ratio of Product Sales to Total Sales Through Current Year

Assume that you want to create a table showing the percentage of sales compared over the years for each product category (`ProductCategoryName`). To obtain the percentage for each year over each value of `ProductCategoryName`, you need to divide the sum of sales for that particular year and product category by the sum of sales for the same product category over all years. In other words, you want to keep the filter on `ProductCategoryName` but remove the filter on the year when calculating the denominator of the percentage.

The following table shows the results when a new measure, **Reseller Sales Year**, is created using the formula shown in the code section. To see how this works, add the field, `CalendarYear`, to the **Row Labels** area of a PivotTable, and add the field, `ProductCategoryName`, to the **Column Labels** area. To view the results as percentages, use Excel's formatting features to apply a percentage number format to the cells containing the measure, **Reseller Sales Year**.

| ROW LABELS | ACCESSORIES | BIKES | CLOTHING | COMPONENTS | GRAND TOTAL |
|-------------|-------------|---------|----------|------------|-------------|
| 2005 | 3.48% | 11.03% | 1.91% | 5.12% | 9.91% |
| 2006 | 16.21% | 29.96% | 27.29% | 30.59% | 29.90% |
| 2007 | 51.62% | 38.45% | 48.86% | 46.36% | 39.93% |
| 2008 | 28.69% | 20.56% | 21.95% | 17.92% | 20.26% |
| Grand Total | 100.00% | 100.00% | 100.00% | 100.00% | 100.00% |

Formula

```
= SUMX(ResellerSales_USD, ResellerSales_USD[SalesAmount_USD])/CALCULATE( SUM(  
ResellerSales_USD[SalesAmount_USD]), ALL(DateTime[CalendarYear]))
```

The formula is constructed as follows:

1. The numerator, `SUMX(ResellerSales_USD, ResellerSales_USD[SalesAmount_USD])`, is the sum of the values in `ResellerSales_USD[SalesAmount_USD]` for the current cell in the pivot table, with context filters applied on the columns `CalendarYear` and `ProductCategoryName`.
2. For the denominator, you remove the existing filter on `CalendarYear` by using the `ALL(Column)` function.

This calculates the sum over the remaining rows on the ResellerSales_USD table, after applying the existing context filters from the column labels. The net effect is that for the denominator the sum is calculated over the selected ProductCategoryName (the implied context filter) and for all values in Year.

Example 3

Calculate Contribution of Product Categories to Total Sales Per Year

Assume that you want to create a table that shows the percentage of sales for each product category, on a year-by-year basis. To obtain the percentage for each product category in a particular year, you need to calculate the sum of sales for that particular product category (ProductCategoryName) in year n, and then divide the resulting value by the sum of sales for the year n over all product categories. In other words, you want to keep the filter on year but remove the filter on ProductCategoryName when calculating the denominator of the percentage.

The following table shows the results when a new measure, **Reseller Sales CategoryName**, is created using the formula shown in the code section. To see how this works, add the field, CalendarYear to the **Row Labels** area of the PivotTable, and add the field, ProductCategoryName, to the **Column Labels** area. Then add the new measure to the **Values** area of the PivotTable. To view the results as percentages, use Excel's formatting features to apply a percentage number format to the cells that contain the new measure, **Reseller Sales CategoryName**.

| ROW LABELS | ACCESSORIES | BIKES | CLOTHING | COMPONENTS | GRAND TOTAL |
|-------------|-------------|--------|----------|------------|-------------|
| 2005 | 0.25% | 91.76% | 0.42% | 7.57% | 100.00% |
| 2006 | 0.38% | 82.64% | 1.99% | 14.99% | 100.00% |
| 2007 | 0.90% | 79.42% | 2.67% | 17.01% | 100.00% |
| 2008 | 0.99% | 83.69% | 2.37% | 12.96% | 100.00% |
| Grand Total | 0.70% | 82.47% | 2.18% | 14.65% | 100.00% |

Formula

```
= SUMX(ResellerSales_USD, ResellerSales_USD[SalesAmount_USD])/CALCULATE( SUM(
ResellerSales_USD[SalesAmount_USD]), ALL(ProductCategory[ProductCategoryName]))
```

The formula is constructed as follows:

1. The numerator, `SUMX(ResellerSales_USD, ResellerSales_USD[SalesAmount_USD])`, is the sum of the values in ResellerSales_USD[SalesAmount_USD] for the current cell in the PivotTable, with context filters applied on the fields, CalendarYear and ProductCategoryName.
2. For the denominator, you use the function, ALL(Column), to remove the filter on ProductCategoryName and calculate the sum over the remaining rows on the ResellerSales_USD table, after applying the existing context filters from the row labels. The net effect is that, for the denominator, the sum is calculated over the selected Year (the implied context filter) and for all values of ProductCategoryName.

See also

[Filter functions](#)

[ALL function](#)

ALLEXCEPT function

FILTER function

ALLCROSSFILTERED

10/31/2022 • 2 minutes to read • [Edit Online](#)

Clear all filters which are applied to a table.

Syntax

```
ALLCROSSFILTERED(<table>)
```

Parameters

| TERM | DEFINITION |
|-------|--|
| table | The table that you want to clear filters on. |

Return value

N/A. See remarks.

Remarks

- ALLCROSSFILTERED can only be used to clear filters but not to return a table.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

```
DEFINE
MEASURE FactInternetSales[TotalQuantity1] =
    CALCULATE(SUM(FactInternetSales[OrderQuantity]), ALLCROSSFILTERED(FactInternetSales))
MEASURE FactInternetSales[TotalQuantity2] =
    CALCULATE(SUM(FactInternetSales[OrderQuantity]), ALL(FactInternetSales))
EVALUATE
    SUMMARIZECOLUMNS(DimSalesReason[SalesReasonName],
        "TotalQuantity1", [TotalQuantity1],
        "TotalQuantity2", [TotalQuantity2])
    ORDER BY DimSalesReason[SalesReasonName]
```

Returns,

| DIMSALESREASON[SALESREASONNAME] | [TOTALQUANTITY1] | [TOTALQUANTITY2] |
|---------------------------------|------------------|------------------|
| Demo Event | 60398 | |
| Magazine Advertisement | 60398 | |
| Manufacturer | 60398 | 1818 |

| DIMSALESREASON[SALESREASONNAME] | [TOTALQUANTITY1] | [TOTALQUANTITY2] |
|---------------------------------|------------------|------------------|
| On Promotion | 60398 | 7390 |
| Other | 60398 | 3653 |
| Price | 60398 | 47733 |
| Quality | 60398 | 1551 |
| Review | 60398 | 1640 |
| Sponsorship | 60398 | |
| Television Advertisement | 60398 | 730 |
| | | |

NOTE

There is a direct or indirect many-to-many relationship between FactInternetSales table and DimSalesReason table.

ALLEXCEPT

10/31/2022 • 2 minutes to read • [Edit Online](#)

Removes all context filters in the table except filters that have been applied to the specified columns.

Syntax

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

Parameters

| TERM | DEFINITION |
|--------|---|
| table | The table over which all context filters are removed, except filters on those columns that are specified in subsequent arguments. |
| column | The column for which context filters must be preserved. |

The first argument to the ALLEXCEPT function must be a reference to a base table. All subsequent arguments must be references to base columns. You cannot use table expressions or column expressions with the ALLEXCEPT function.

Return value

A table with all filters removed except for the filters on the specified columns.

Remarks

- This function is not used by itself, but serves as an intermediate function that can be used to change the set of results over which some other calculation is performed.
- ALL and ALLEXCEPT can be used in different scenarios:

| FUNCTION AND USAGE | DESCRIPTION |
|-------------------------------|---|
| ALL(Table) | Removes all filters from the specified table. In effect, ALL(Table) returns all of the values in the table, removing any filters from the context that otherwise might have been applied. This function is useful when you are working with many levels of grouping, and want to create a calculation that creates a ratio of an aggregated value to the total value. |
| ALL (Column[, Column[, ...]]) | Removes all filters from the specified columns in the table; all other filters on other columns in the table still apply. All column arguments must come from the same table. The ALL(Column) variant is useful when you want to remove the context filters for one or more specific columns and to keep all other context filters. |

| FUNCTION AND USAGE | DESCRIPTION |
|---|---|
| ALLEXCEPT(Table, Column1 [,Column2]...) | Removes all context filters in the table except filters that are applied to the specified columns. This is a convenient shortcut for situations in which you want to remove the filters on many, but not all, columns in a table. |

- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following measure formula sums SalesAmount_USD and uses the ALLEXCEPT function to remove any context filters on the DateTime table except if the filter has been applied to the CalendarYear column.

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

Because the formula uses ALLEXCEPT, whenever any column but CalendarYear from the table DateTime is used to slice a visualization, the formula will remove any slicer filters, providing a value equal to the sum of SalesAmount_USD. However, if the column CalendarYear is used to slice the visualization, the results are different. Because CalendarYear is specified as the argument to ALLEXCEPT, when the data is sliced on the year, a filter will be applied on years at the row level

See also

[Filter functions](#)
[ALL function](#)
[FILTER function](#)

ALLNOBLANKROW

10/31/2022 • 4 minutes to read • [Edit Online](#)

From the parent table of a relationship, returns all rows but the blank row, or all distinct values of a column but the blank row, and disregards any context filters that might exist.

Syntax

```
ALLNOBLANKROW( {<table> | <column>[, <column>[, <column>[,...]]]} )
```

Parameters

| TERM | DEFINITION |
|--------|---|
| table | The table over which all context filters are removed. |
| column | A column over which all context filters are removed. |

Only one parameter must be passed; the parameter is either a table or a column.

Return value

A table, when the passed parameter was a table, or a column of values, when the passed parameter was a column.

Remarks

- The ALLNOBLANKROW function only filters the blank row that a parent table, in a relationship, will show when there are one or more rows in the child table that have non-matching values to the parent column. See the example below for a thorough explanation.
- The following table summarizes the variations of ALL that are provided in DAX, and their differences:

| FUNCTION AND USAGE | DESCRIPTION |
|-------------------------------|--|
| ALL(Column) | Removes all filters from the specified column in the table; all other filters in the table, over other columns, still apply. |
| ALL(Table) | Removes all filters from the specified table. |
| ALLEXCEPT(Table,Col1,Col2...) | Overrides all context filters in the table except over the specified columns. |
| ALLNOBLANK(table column) | From the parent table of a relationship, returns all rows but the blank row, or all distinct values of a column but the blank row, and disregards any context filters that might exist |

For a general description of how the ALL function works, together with step-by-step examples that use ALL(Table) and ALL(Column), see [ALL function](#).

- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

In the sample data, the ResellerSales_USD table contains one row that has no values and therefore cannot be related to any of the parent tables in the relationships within the workbook. You will use this table in a PivotTable so that you can see the blank row behavior and how to handle counts on unrelated data.

Step 1: Verify the unrelated data

Open the **Power Pivot window**, then select the ResellerSales_USD table. In the ProductKey column, filter for blank values. One row will remain. In that row, all column values should be blank except for SalesOrderLineNumber.

Step 2: Create a PivotTable

Create a new PivotTable, then drag the column, datetime.[Calendar Year], to the Row Labels pane. The following table shows the expected results:

| ROW LABELS |
|-------------|
| 2005 |
| 2006 |
| 2007 |
| 2008 |
| |
| Grand Total |

Note the blank label between **2008** and **Grand Total**. This blank label represents the Unknown member, which is a special group that is created to account for any values in the child table that have no matching value in the parent table, in this example the datetime.[Calendar Year] column.

When you see this blank label in the PivotTable, you know that in some of the tables that are related to the column, datetime.[Calendar Year], there are either blank values or non-matching values. The parent table is the one that shows the blank label, but the rows that do not match are in one or more of the child tables.

The rows that get added to this blank label group are either values that do not match any value in the parent table-- for example, a date that does not exist in the datetime table-- or null values, meaning no value for date at all. In this example we have placed a blank value in all columns of the child sales table. Having more values in the parent table than in the children tables does not cause a problem.

Step 3: Count rows using ALL and ALLNOBLANK

Add the following two measures to the datetime table, to count the table rows: **Countrows ALLNOBLANK of datetime**, **Countrows ALL of datetime**. The formulas that you can use to define these measures are:

```
// Countrows ALLNOBLANK of datetime
= COUNTROWS(ALLNOBLANKROW('DateTime'))

// Countrows ALL of datetime
= COUNTROWS(ALL('DateTime'))

// Countrows ALLNOBLANKROW of ResellerSales_USD
= COUNTROWS(ALLNOBLANKROW('ResellerSales_USD'))

// Countrows ALL of ResellerSales_USD
= COUNTROWS(ALL('ResellerSales_USD'))
```

On a blank PivotTable add datetime.[Calendar Year] column to the row labels, and then add the newly created measures. The results should look like the following table:

| ROW LABELS | COUNTROWS ALLNOBLANK OF DATETIME | COUNTROWS ALL OF DATETIME |
|-------------|----------------------------------|---------------------------|
| 2005 | 1280 | 1281 |
| 2006 | 1280 | 1281 |
| 2007 | 1280 | 1281 |
| 2008 | 1280 | 1281 |
| | 1280 | 1281 |
| Grand Total | 1280 | 1281 |

The results show a difference of 1 row in the table rows count. However, if you open the **Power Pivot window** and select the datetime table, you cannot find any blank row in the table because the special blank row mentioned here is the Unknown member.

Step 4: Verify that the count is accurate

In order to prove that the ALLNOBLANKROW does not count any truly blank rows, and only handles the special blank row on the parent table only, add the following two measures to the ResellerSales_USD table: **Countrows ALLNOBLANKROW of ResellerSales_USD**, **Countrows ALL of ResellerSales_USD**.

Create a new PivotTable, and drag the column, datetime.[Calendar Year], to the Row Labels pane. Now add the measures that you just created. The results should look like the following:

| ROW LABELS | COUNTROWS ALLNOBLANKROW OF RESELLERSALES_USD | COUNTROWS ALL OF RESELLERSALES_USD |
|------------|--|------------------------------------|
| 2005 | 60856 | 60856 |
| 2006 | 60856 | 60856 |
| 2007 | 60856 | 60856 |
| 2008 | 60856 | 60856 |
| | 60856 | 60856 |

| ROW LABELS | COUNTROWS ALLNOBLANKROW OF RESELLERSALES_USD | COUNTROWS ALL OF RESELLERSALES_USD |
|-------------|---|---------------------------------------|
| Grand Total | 60856 | 60856 |

Now the two measures have the same results. That is because the ALLNOBLANKROW function does not count truly blank rows in a table, but only handles the blank row that is a special case generated in a parent table, when one or more of the child tables in the relationship contain non-matching values or blank values.

See also

[Filter functions](#)

[ALL function](#)

[FILTER function](#)

ALLSELECTED

10/31/2022 • 4 minutes to read • [Edit Online](#)

Removes context filters from columns and rows in the current query, while retaining all other context filters or explicit filters.

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

Syntax

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

Parameters

| TERM | DEFINITION |
|------------|--|
| tableName | The name of an existing table, using standard DAX syntax. This parameter cannot be an expression. This parameter is optional. |
| columnName | The name of an existing column using standard DAX syntax, usually fully qualified. It cannot be an expression. This parameter is optional. |

Return value

The context of the query without any column and row filters.

Remarks

- If there is one argument, the argument is either *tableName* or *columnName*. If there is more than one argument, they must be columns from the same table.
- This function is different from ALL() because it retains all filters explicitly set within the query, and it retains all context filters other than row and column filters.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following example shows how to generate different levels of visual totals in a table report using DAX expressions. In the report two (2) previous filters have been applied to the Reseller Sales data; one on Sales Territory Group = *Europe* and the other on Promotion Type = *Volume Discount*. Once filters have been applied, visual totals can be calculated for the entire report, for All Years, or for All Product Categories. Also, for illustration purposes the grand total for All Reseller Sales is obtained too, removing all filters in the report. Evaluating the following DAX expression results in a table with all the information needed to build a table with Visual Totals.

```

define
measure 'Reseller Sales'[Reseller Sales Amount]=sum('Reseller Sales'[Sales Amount])
measure 'Reseller Sales'[Reseller Grand Total]=calculate(sum('Reseller Sales'[Sales Amount]), ALL('Reseller Sales'))
measure 'Reseller Sales'[Reseller Visual Total]=calculate(sum('Reseller Sales'[Sales Amount]), ALLSELECTED())
measure 'Reseller Sales'[Reseller Visual Total for All of Calendar Year]=calculate(sum('Reseller Sales'[Sales Amount]), ALLSELECTED('Date'[Calendar Year]))
measure 'Reseller Sales'[Reseller Visual Total for All of Product Category Name]=calculate(sum('Reseller Sales'[Sales Amount]), ALLSELECTED('Product Category'[Product Category Name]))
evaluate
CalculateTable(
    //CT table expression
    summarize(
//summarize table expression
crossjoin(distinct('Product Category'[Product Category Name]), distinct('Date'[Calendar Year]))
//First Group by expression
, 'Product Category'[Product Category Name]
//Second Group by expression
, 'Date'[Calendar Year]
//Summary expressions
, "Reseller Sales Amount", [Reseller Sales Amount]
, "Reseller Grand Total", [Reseller Grand Total]
, "Reseller Visual Total", [Reseller Visual Total]
, "Reseller Visual Total for All of Calendar Year", [Reseller Visual Total for All of Calendar Year]
, "Reseller Visual Total for All of Product Category Name", [Reseller Visual Total for All of Product Category Name]
)
//CT filters
, 'Sales Territory'[Sales Territory Group]="Europe", 'Promotion'[Promotion Type]="Volume Discount"
)
order by [Product Category Name], [Calendar Year]

```

After executing the above expression in SQL Server Management Studio against AdventureWorks DW Tabular Model, you obtain the following results:

| [PRODUCT CATEGORY NAME] | [CALENDAR YEAR] | [RESELLER SALES AMOUNT] | [RESELLER GRAND TOTAL] | [RESELLER VISUAL TOTAL] | [RESELLER VISUAL TOTAL FOR ALL OF CALENDAR YEAR] | [RESELLER VISUAL TOTAL FOR ALL OF PRODUCT CATEGORY NAME] |
|-------------------------|-----------------|-------------------------|------------------------|-------------------------|--|--|
| Accessories | 2000 | | 80450596.9823 | 877006.7987 | 38786.018 | |
| Accessories | 2001 | | 80450596.9823 | 877006.7987 | 38786.018 | |
| Accessories | 2002 | 625.7933 | 80450596.9823 | 877006.7987 | 38786.018 | 91495.3104 |
| Accessories | 2003 | 26037.3132 | 80450596.9823 | 877006.7987 | 38786.018 | 572927.0136 |
| Accessories | 2004 | 12122.9115 | 80450596.9823 | 877006.7987 | 38786.018 | 212584.4747 |
| Accessories | 2005 | | 80450596.9823 | 877006.7987 | 38786.018 | |

| [PRODUCT CATEGORY NAME] | [CALENDAR YEAR] | [RESELLER SALES AMOUNT] | [RESELLER GRAND TOTAL] | [RESELLER VISUAL TOTAL] | [RESELLER VISUAL TOTAL FOR ALL OF CALENDAR YEAR] | [RESELLER VISUAL TOTAL FOR ALL OF PRODUCT CATEGORY NAME] |
|-------------------------|-----------------|-------------------------|------------------------|-------------------------|--|--|
| Accessories | 2006 | | 80450596.9823 | 877006.7987 | 38786.018 | |
| Bikes | 2000 | | 80450596.9823 | 877006.7987 | 689287.7939 | |
| Bikes | 2001 | | 80450596.9823 | 877006.7987 | 689287.7939 | |
| Bikes | 2002 | 73778.938 | 80450596.9823 | 877006.7987 | 689287.7939 | 91495.3104 |
| Bikes | 2003 | 439771.4136 | 80450596.9823 | 877006.7987 | 689287.7939 | 572927.0136 |
| Bikes | 2004 | 175737.4423 | 80450596.9823 | 877006.7987 | 689287.7939 | 212584.4747 |
| Bikes | 2005 | | 80450596.9823 | 877006.7987 | 689287.7939 | |
| Bikes | 2006 | | 80450596.9823 | 877006.7987 | 689287.7939 | |
| Clothing | 2000 | | 80450596.9823 | 877006.7987 | 95090.7757 | |
| Clothing | 2001 | | 80450596.9823 | 877006.7987 | 95090.7757 | |
| Clothing | 2002 | 12132.4334 | 80450596.9823 | 877006.7987 | 95090.7757 | 91495.3104 |
| Clothing | 2003 | 58234.2214 | 80450596.9823 | 877006.7987 | 95090.7757 | 572927.0136 |
| Clothing | 2004 | 24724.1209 | 80450596.9823 | 877006.7987 | 95090.7757 | 212584.4747 |
| Clothing | 2005 | | 80450596.9823 | 877006.7987 | 95090.7757 | |
| Clothing | 2006 | | 80450596.9823 | 877006.7987 | 95090.7757 | |
| Components | 2000 | | 80450596.9823 | 877006.7987 | 53842.2111 | |
| Components | 2001 | | 80450596.9823 | 877006.7987 | 53842.2111 | |

| [PRODUCT CATEGORY NAME] | [CALENDAR YEAR] | [RESELLER SALES AMOUNT] | [RESELLER GRAND TOTAL] | [RESELLER VISUAL TOTAL] | [RESELLER VISUAL TOTAL FOR ALL OF CALENDAR YEAR] | [RESELLER VISUAL TOTAL FOR ALL OF PRODUCT CATEGORY NAME] |
|-------------------------|-----------------|-------------------------|------------------------|-------------------------|--|--|
| Components | 2002 | 4958.1457 | 80450596.9823 | 877006.7987 | 53842.2111 | 91495.3104 |
| Components | 2003 | 48884.0654 | 80450596.9823 | 877006.7987 | 53842.2111 | 572927.0136 |
| Components | 2004 | | 80450596.9823 | 877006.7987 | 53842.2111 | 212584.4747 |
| Components | 2005 | | 80450596.9823 | 877006.7987 | 53842.2111 | |
| Components | 2006 | | 80450596.9823 | 877006.7987 | 53842.2111 | |

The columns in the report are:

Reseller Sales Amount

The actual value of Reseller Sales for the year and product category. This value appears in a cell in the center of your report, at the intersection of year and category.

Reseller Visual Total for All of Calendar Year

The total value for a product category across all years. This value appears at the end of a column or row for a given product category and across all years in the report.

Reseller Visual Total for All of Product Category Name

The total value for a year across all product categories. This value appears at the end of a column or row for a given year and across all product categories in the report.

Reseller Visual Total

The total value for all years and product categories. This value usually appears in the bottom rightmost corner of the table.

Reseller Grand Total

This is the grand total for all reseller sales, before any filter has been applied; you should notice the difference with [Reseller Visual Total]. You do remember that this report includes two (2) filters, one on Product Category Group and the other in Promotion Type.

NOTE

if you have explicit filters in your expression, those filters are also applied to the expression.

CALCULATE

10/31/2022 • 4 minutes to read • [Edit Online](#)

Evaluates an expression in a modified filter context.

NOTE

There's also the [CALCULATETABLE](#) function. It performs exactly the same functionality, except it modifies the [filter context](#) applied to an expression that returns a *table object*.

Syntax

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

Parameters

| TERM | DEFINITION |
|----------------------|---|
| expression | The expression to be evaluated. |
| filter1, filter2,... | (Optional) Boolean expressions or table expressions that defines filters, or filter modifier functions. |

The expression used as the first parameter is essentially the same as a measure.

Filters can be:

- Boolean filter expressions
- Table filter expressions
- Filter modification functions

When there are multiple filters, they can be evaluated by using the AND (&&) [logical operator](#), meaning all conditions must be TRUE, or by the OR (||) logical operator, meaning either condition can be true.

Boolean filter expressions

A Boolean expression filter is an expression that evaluates to TRUE or FALSE. There are several rules that they must abide by:

- They can reference columns from a single table.
- They cannot reference measures.
- They cannot use a nested CALCULATE function.

Beginning with the September 2021 release of Power BI Desktop, the following also apply:

- They cannot use functions that scan or return a table unless they are passed as arguments to aggregation functions.
- They *can* contain an aggregation function that returns a scalar value. For example,

```
Total sales on the last selected date =
CALCULATE (
    SUM ( Sales[Sales Amount] ),
    'Sales'[OrderDateKey] = MAX ( 'Sales'[OrderDateKey] )
)
```

Table filter expression

A table expression filter applies a table object as a filter. It could be a reference to a model table, but more likely it's a function that returns a table object. You can use the [FILTER](#) function to apply complex filter conditions, including those that cannot be defined by a Boolean filter expression.

Filter modifier functions

Filter modifier functions allow you to do more than simply add filters. They provide you with additional control when modifying filter context.

| FUNCTION | PURPOSE |
|--|--|
| REMOVEFILTERS | Remove all filters, or filters from one or more columns of a table, or from all columns of a single table. |
| ALL ¹ , ALLEXCEPT , ALLNOBLANKROW | Remove filters from one or more columns, or from all columns of a single table. |
| KEEPFILTERS | Add filter without removing existing filters on the same columns. |
| USERELATIONSHIP | Engage an inactive relationship between related columns, in which case the active relationship will automatically become inactive. |
| CROSSFILTER | Modify filter direction (from both to single, or from single to both) or disable a relationship. |

¹ The ALL function and its variants behave as both filter modifiers and as functions that return table objects. If the REMOVEFILTERS function is supported by your tool, it's better to use it to remove filters.

Return value

The value that is the result of the expression.

Remarks

- When filter expressions are provided, the CALCULATE function modifies the filter context to evaluate the expression. For each filter expression, there are two possible standard outcomes when the filter expression is not wrapped in the KEEPFILTERS function:
 - If the columns (or tables) aren't in the filter context, then new filters will be added to the filter context to evaluate the expression.
 - If the columns (or tables) are already in the filter context, the existing filters will be overwritten by the new filters to evaluate the CALCULATE expression.
- The CALCULATE function used *without filters* achieves a specific requirement. It transitions row context to filter context. It's required when an expression (not a model measure) that summarizes model data needs to be evaluated in row context. This scenario can happen in a calculated column formula or when an expression in an iterator function is evaluated. Note that when a model measure is used in row context, context transition is automatic.

- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Examples

The following **Sales** table measure definition produces a revenue result, but only for products that have the color blue.

Examples in this article can be used with the sample Adventure Works DW 2020 Power BI Desktop model. To get the model, see [DAX sample model](#).

```
Blue Revenue =
CALCULATE(
    SUM(Sales[Sales Amount]),
    'Product'[Color] = "Blue"
)
```

| CATEGORY | SALES AMOUNT | BLUE REVENUE |
|--------------|-------------------------|-----------------------|
| Accessories | \$1,272,057.89 | \$165,406.62 |
| Bikes | \$94,620,526.21 | \$8,374,313.88 |
| Clothing | \$2,117,613.45 | \$259,488.37 |
| Components | \$11,799,076.66 | \$803,642.10 |
| Total | \$109,809,274.20 | \$9,602,850.97 |

The CALCULATE function evaluates the sum of the **Sales** table **Sales Amount** column in a modified filter context. A new filter is added to the **Product** table **Color** column—or, the filter overwrites any filter that's already applied to the column.

The following **Sales** table measure definition produces a ratio of sales over sales for all sales channels.

| CHANNEL | SALES AMOUNT | REVENUE % TOTAL CHANNEL |
|--------------|-------------------------|-------------------------|
| Internet | \$29,358,677.22 | 26.74% |
| Reseller | \$80,450,596.98 | 73.26% |
| Total | \$109,809,274.20 | 100.00% |

```
Revenue % Total Channel =
DIVIDE(
    SUM(Sales[Sales Amount]),
    CALCULATE(
        SUM(Sales[Sales Amount]),
        REMOVEFILTERS('Sales Order'[Channel])
    )
)
```

The [DIVIDE](#) function divides an expression that sums of the **Sales** table **Sales Amount** column value (in the filter context) by the same expression in a modified filter context. It's the CALCULATE function that modifies the filter context by using the REMOVEFILTERS function, which is a filter modifier function. It removes filters from

the **Sales Order** table **Channel** column.

The following **Customer** table calculated column definition classifies customers into a loyalty class. It's a very simple scenario: When the revenue produced by the customer is less than \$2500, they're classified as *Low*; otherwise they're *High*.

```
Customer Segment =  
IF(  
    CALCULATE(SUM(Sales[Sales Amount]), ALLEXCEPT(Customer, Customer[CustomerKey])) < 2500,  
    "Low",  
    "High"  
)
```

In this example, row context is converted to the filter context. It's known as *context transition*. The [ALLEXCEPT](#) function removes filters from all **Customer** table columns except the **CustomerKey** column.

See also

[Filter context](#)

[Row context](#)

[CALCULATETABLE function](#)

[Filter functions](#)

CALCULATETABLE

10/31/2022 • 3 minutes to read • [Edit Online](#)

Evaluates a table expression in a modified filter context.

NOTE

There's also the [CALCULATE](#) function. It performs exactly the same functionality, except it modifies the [filter context](#) applied to an expression that returns a *scalar value*.

Syntax

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

Parameters

| TERM | DEFINITION |
|----------------------|---|
| expression | The table expression to be evaluated. |
| filter1, filter2,... | (Optional) Boolean expressions or table expressions that defines filters, or filter modifier functions. |

The expression used as the first parameter must be a model table or a function that returns a table.

Filters can be:

- Boolean filter expressions
- Table filter expressions
- Filter modification functions

When there are multiple filters, they're evaluated by using the AND [logical operator](#). That means all conditions must be TRUE at the same time.

Boolean filter expressions

A Boolean expression filter is an expression that evaluates to TRUE or FALSE. There are several rules that they must abide by:

- They can reference only a single column.
- They cannot reference measures.
- They cannot use a nested CALCULATE function.

Beginning with the September 2021 release of Power BI Desktop, the following also apply:

- They cannot use functions that scan or return a table unless they are passed as arguments to aggregation functions.
- They *can* contain an aggregation function that returns a scalar value.

Table filter expression

A table expression filter applies a table object as a filter. It could be a reference to a model table, but more likely it's a function that returns a table object. You can use the [FILTER](#) function to apply complex filter conditions,

including those that cannot be defined by a Boolean filter expression.

Filter modifier functions

Filter modifier functions allow you to do more than simply add filters. They provide you with additional control when modifying filter context.

| FUNCTION | PURPOSE |
|--|--|
| REMOVEFILTERS | Remove all filters, or filters from one or more columns of a table, or from all columns of a single table. |
| ALL ¹ , ALLEXCEPT , ALLNOBLANKROW | Remove filters from one or more columns, or from all columns of a single table. |
| KEEPFILTERS | Add filter without removing existing filters on the same columns. |
| USERELATIONSHIP | Engage an inactive relationship between related columns, in which case the active relationship will automatically become inactive. |
| CROSSFILTER | Modify filter direction (from both to single, or from single to both) or disable a relationship. |

¹ The ALL function and its variants behave as both filter modifiers and as functions that return table objects. If the REMOVEFILTERS function is supported by your tool, it's better to use it to remove filters.

Return value

A table of values.

Remarks

- When filter expressions are provided, the CALCULATETABLE function modifies the filter context to evaluate the expression. For each filter expression, there are two possible standard outcomes when the filter expression is not wrapped in the KEEPFILTERS function:
 - If the columns (or tables) aren't in the filter context, then new filters will be added to the filter context to evaluate the expression.
 - If the columns (or tables) are already in the filter context, the existing filters will be overwritten by the new filters to evaluate the CALCULATETABLE expression.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following example uses the CALCULATETABLE function to get the sum of Internet sales for 2006. This value is later used to calculate the ratio of Internet sales compared to all sales for the year 2006.

The following formula:

```
= SUMX(  
    CALCULATETABLE(  
        'InternetSales_USD',  
        'DateTime'[CalendarYear] = 2006  
    ),  
    [SalesAmount_USD]  
)
```

It results in the following table:

| ROW LABELS | INTERNET SALESAMOUNT_USD | CALCULATETABLE 2006 INTERNET SALES | INTERNET SALES TO 2006 RATIO |
|-------------|-----------------------------|---------------------------------------|---------------------------------|
| 2005 | \$2,627,031.40 | \$5,681,440.58 | 0.46 |
| 2006 | \$5,681,440.58 | \$5,681,440.58 | 1.00 |
| 2007 | \$8,705,066.67 | \$5,681,440.58 | 1.53 |
| 2008 | \$9,041,288.80 | \$5,681,440.58 | 1.59 |
| Grand Total | \$26,054,827.45 | \$5,681,440.58 | 4.59 |

See also

- [Filter context](#)
- [CALCULATE function \(DAX\)](#)
- [Filter functions \(DAX\)](#)

EARLIER

10/31/2022 • 4 minutes to read • [Edit Online](#)

Returns the current value of the specified column in an outer evaluation pass of the mentioned column.

EARLIER is useful for nested calculations where you want to use a certain value as an input and produce calculations based on that input. In Microsoft Excel, you can do such calculations only within the context of the current row; however, in DAX you can store the value of the input and then make calculation using data from the entire table.

EARLIER is mostly used in the context of calculated columns.

Syntax

```
EARLIER(<column>, <number>)
```

Parameters

| TERM | DEFINITION |
|--------|--|
| column | A column or expression that resolves to a column. |
| num | (Optional) A positive number to the outer evaluation pass. The next evaluation level out is represented by 1; two levels out is represented by 2 and so on. When omitted default value is 1. |

Return value

The current value of row, from **column**, at **number** of outer evaluation passes.

Exceptions

Description of errors

Remarks

- **EARLIER** succeeds if there is a row context prior to the beginning of the table scan. Otherwise it returns an error.
- The performance of **EARLIER** might be slow because theoretically, it might have to perform a number of operations that is close to the total number of rows (in the column) times the same number (depending on the syntax of the expression). For example if you have 10 rows in the column, approximately a 100 operations could be required; if you have 100 rows then close to 10,000 operations might be performed.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

NOTE

In practice, the VertiPaq in-memory analytics engine performs optimizations to reduce the actual number of calculations, but you should be cautious when creating formulas that involve recursion.

Example

To illustrate the use of EARLIER, it is necessary to build a scenario that calculates a rank value and then uses that rank value in other calculations.

The following example is based on this simple table, **ProductSubcategory**, which shows the total sales for each ProductSubcategory.

The final table, including the ranking column is shown here.

| PRODUCTSUBCATEGORYKEY | ENGLISHPRODUCTSUBCATEGORYNAME | TOTALSUBCATEGORYSALES | SUBCATEGORYRANKING |
|-----------------------|-------------------------------|-----------------------|--------------------|
| 18 | Bib-Shorts | \$156,167.88 | 18 |
| 26 | Bike Racks | \$220,720.70 | 14 |
| 27 | Bike Stands | \$35,628.69 | 30 |
| 28 | Bottles and Cages | \$59,342.43 | 24 |
| 5 | Bottom Brackets | \$48,643.47 | 27 |
| 6 | Brakes | \$62,113.16 | 23 |
| 19 | Caps | \$47,934.54 | 28 |
| 7 | Chains | \$8,847.08 | 35 |
| 29 | Cleaners | \$16,882.62 | 32 |
| 8 | Cranksets | \$191,522.09 | 15 |
| 9 | Derailleurs | \$64,965.33 | 22 |
| 30 | Fenders | \$41,974.10 | 29 |
| 10 | Forks | \$74,727.66 | 21 |
| 20 | Gloves | \$228,353.58 | 12 |
| 4 | Handlebars | \$163,257.06 | 17 |
| 11 | Headsets | \$57,659.99 | 25 |
| 31 | Helmets | \$451,192.31 | 9 |
| 32 | Hydration Packs | \$96,893.78 | 20 |

| PRODUCTSUBCATEGORYKEY | ENGLISHPRODUCTSUBCATEGORYNAME | TOTALSUBCATEGORYSALES | SUBCATEGORYRANKING |
|-----------------------|-------------------------------|-----------------------|--------------------|
| 21 | Jerseys | \$699,429.78 | 7 |
| 33 | Lights | | 36 |
| 34 | Locks | \$15,059.47 | 33 |
| 1 | Mountain Bikes | \$34,305,864.29 | 2 |
| 12 | Mountain Frames | \$4,511,170.68 | 4 |
| 35 | Panniers | | 36 |
| 13 | Pedals | \$140,422.20 | 19 |
| 36 | Pumps | \$12,695.18 | 34 |
| 2 | Road Bikes | \$40,551,696.34 | 1 |
| 14 | Road Frames | \$3,636,398.71 | 5 |
| 15 | Saddles | \$52,526.47 | 26 |
| 22 | Shorts | \$385,707.80 | 10 |
| 23 | Socks | \$28,337.85 | 31 |
| 24 | Tights | \$189,179.37 | 16 |
| 37 | Tires and Tubes | \$224,832.81 | 13 |
| 3 | Touring Bikes | \$13,334,864.18 | 3 |
| 16 | Touring Frames | \$1,545,344.02 | 6 |
| 25 | Vests | \$240,990.04 | 11 |
| 17 | Wheels | \$648,240.04 | 8 |

Creating a Rank Value

One way to obtain a rank value for a given value in a row is to count the number of rows, in the same table, that have a value larger (or smaller) than the one that is being compared. This technique returns a blank or zero value for the highest value in the table, whereas equal values will have the same rank value and next value (after the equal values) will have a non consecutive rank value. See the sample below.

A new calculated column, **SubCategorySalesRanking**, is created by using the following formula.

```
= COUNTROWS(FILTER(ProductSubcategory, EARLIER(ProductSubcategory[TotalSubcategorySales])
<ProductSubcategory[TotalSubcategorySales]))+1
```

The following steps describe the method of calculation in more detail.

1. The **EARLIER** function gets the value of *TotalSubcategorySales* for the current row in the table. In this case, because the process is starting, it is the first row in the table
2. **EARLIER**(*TotalSubcategorySales*) evaluates to \$156,167.88, the current row in the outer loop.
3. The **FILTER** function now returns a table where all rows have a value of *TotalSubcategorySales* larger than \$156,167.88 (which is the current value for **EARLIER**).
4. The **COUNTROWS** function counts the rows of the filtered table and assigns that value to the new calculated column in the current row plus 1. Adding 1 is needed to prevent the top ranked value from become a Blank.
5. The calculated column formula moves to the next row and repeats steps 1 to 4. These steps are repeated until the end of the table is reached.

The **EARLIER** function will always get the value of the column prior to the current table operation. If you need to get a value from the loop before that, set the second argument to 2.

See also

[EARLIEST function](#)

[Filter functions](#)

EARLIEST

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the current value of the specified column in an outer evaluation pass of the specified column.

Syntax

```
EARLIEST(<column>)
```

Parameters

| TERM | DEFINITION |
|--------|--------------------------|
| column | A reference to a column. |

Return value

A column with filters removed.

Remarks

- The EARLIEST function is similar to EARLIER, but lets you specify one additional level of recursion.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The current sample data does not support this scenario.

```
= EARLIEST(<column>)
```

See also

[EARLIER function](#)

[Filter functions](#)

FILTER

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns a table that represents a subset of another table or expression.

Syntax

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

Parameters

| TERM | DEFINITION |
|--------|---|
| table | The table to be filtered. The table can also be an expression that results in a table. |
| filter | A Boolean expression that is to be evaluated for each row of the table. For example, <code>[Amount] > 0</code> or <code>[Region] = "France"</code> |

Return value

A table containing only the filtered rows.

Remarks

- You can use FILTER to reduce the number of rows in the table that you are working with, and use only specific data in calculations. FILTER is not used independently, but as a function that is embedded in other functions that require a table as an argument.
- For best practices when using FILTER, see [Avoid using FILTER as a filter argument](#).

Use COUNTROWS instead of COUNT in DAX

- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following example creates a report of Internet sales outside the United States by using a measure that filters out sales in the United States, and then slicing by calendar year and product categories. To create this measure, you filter the table, Internet Sales USD, by using Sales Territory, and then use the filtered table in a SUMX function.

In this example, the expression:

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

Returns a table that is a subset of Internet Sales minus all rows that belong to the United States sales territory. The RELATED function is what links the Territory key in the Internet Sales table to SalesTerritoryCountry in the

SalesTerritory table.

The following table demonstrates the proof of concept for the measure, NON USA Internet Sales, the formula for which is provided in the code section below. The table compares all Internet sales with non- USA Internet sales, to show that the filter expression works, by excluding United States sales from the computation.

To re-create this table, add the field, SalesTerritoryCountry, to the **Row Labels** area of a report or PivotTable.

Table 1. Comparing total sales for U.S. vs. all other regions

| ROW LABELS | INTERNET SALES | NON USA INTERNET SALES |
|----------------|-----------------|------------------------|
| Australia | \$4,999,021.84 | \$4,999,021.84 |
| Canada | \$1,343,109.10 | \$1,343,109.10 |
| France | \$2,490,944.57 | \$2,490,944.57 |
| Germany | \$2,775,195.60 | \$2,775,195.60 |
| United Kingdom | \$5,057,076.55 | \$5,057,076.55 |
| United States | \$9,389,479.79 | |
| Grand Total | \$26,054,827.45 | \$16,665,347.67 |

The final report table shows the results when you create a PivotTable by using the measure, NON USA Internet Sales. Add the field, CalendarYear, to the **Row Labels** area of the PivotTable and add the field, ProductCategoryName, to the **Column Labels** area.

Table 2. Comparing non- U.S. sales by product categories

| ROW LABELS | ACCESSORIES | BIKES | CLOTHING | GRAND TOTAL |
|-------------|--------------|-----------------|--------------|-----------------|
| 2005 | | \$1,526,481.95 | | \$1,526,481.95 |
| 2006 | | \$3,554,744.04 | | \$3,554,744.04 |
| 2007 | \$156,480.18 | \$5,640,106.05 | \$70,142.77 | \$5,866,729.00 |
| 2008 | \$228,159.45 | \$5,386,558.19 | \$102,675.04 | \$5,717,392.68 |
| Grand Total | \$384,639.63 | \$16,107,890.23 | \$172,817.81 | \$16,665,347.67 |

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

See also

[Filter functions](#)

[ALL function](#)

[ALLEXCEPT function](#)

KEEPFILTERS

10/31/2022 • 3 minutes to read • [Edit Online](#)

Modifies how filters are applied while evaluating a CALCULATE or CALCULATETABLE function.

Syntax

```
KEEPFILTERS(<expression>)
```

Parameters

| TERM | DEFINITION |
|------------|-----------------|
| expression | Any expression. |

Return value

A table of values.

Remarks

- You use KEEPFILTERS within the context CALCULATE and CALCULATETABLE functions, to override the standard behavior of those functions.
- By default, filter arguments in functions such as CALCULATE are used as the context for evaluating the expression, and as such filter arguments for CALCULATE replace all existing filters over the same columns. The new context effected by the filter argument for CALCULATE affects only existing filters on columns mentioned as part of the filter argument. Filters on columns other than those mentioned in the arguments of CALCULATE or other related functions remain in effect and unaltered.
- The KEEPFILTERS function allows you to modify this behavior. When you use KEEPFILTERS, any existing filters in the current context are compared with the columns in the filter arguments, and the intersection of those arguments is used as the context for evaluating the expression. The net effect over any one column is that both sets of arguments apply: both the filter arguments used in CALCULATE and the filters in the arguments of the KEEPFILTER function. In other words, whereas CALCULATE filters replace the current context, KEEPFILTERS adds filters to the current context.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following example takes you through some common scenarios that demonstrate use of the KEEPFILTERS function as part of a CALCULATE or CALCULATETABLE formula.

The first three expressions obtain simple data to be used for comparisons:

- Internet Sales for the state of Washington.
- Internet Sales for the states of Washington and Oregon (both states combined).

- Internet Sales for the state of Washington and the province of British Columbia (both regions combined).

The fourth expression calculates Internet Sales for Washington and Oregon, while the filter for Washington and British Columbia is applied.

The next expression calculates Internet Sales for Washington and Oregon but uses KEEPFILTERS; the filter for Washington and British Columbia is part of the prior context.

```
EVALUATE ROW(
  "$$ in WA"
  , CALCULATE('Internet Sales'[Internet Total Sales]
    , 'Geography'[State Province Code]="WA"
  )
  , "$$ in WA and OR"
  , CALCULATE('Internet Sales'[Internet Total Sales]
    , 'Geography'[State Province Code]="WA"
    || 'Geography'[State Province Code]="OR"
  )
  , "$$ in WA and BC"
  , CALCULATE('Internet Sales'[Internet Total Sales]
    , 'Geography'[State Province Code]="WA"
    || 'Geography'[State Province Code]="BC"
  )
  , "$$ in WA and OR ??"
  , CALCULATE(
    CALCULATE('Internet Sales'[Internet Total Sales]
      , 'Geography'[State Province Code]="WA"
      || 'Geography'[State Province Code]="OR"
    )
    , 'Geography'[State Province Code]="WA"
    || 'Geography'[State Province Code]="BC"
  )
  , "$$ in WA !!"
  , CALCULATE(
    CALCULATE('Internet Sales'[Internet Total Sales]
      , KEEPFILTERS('Geography'[State Province Code]="WA"
        || 'Geography'[State Province Code]="OR"
      )
    )
    , 'Geography'[State Province Code]="WA"
    || 'Geography'[State Province Code]="BC"
  )
)
```

When this expression is evaluated against the sample database AdventureWorks DW, the following results are obtained.

| COLUMN | VALUE |
|------------------------|-----------------|
| [\$\$ in WA] | \$ 2,467,248.34 |
| [\$\$ in WA and OR] | \$ 3,638,239.88 |
| [\$\$ in WA and BC] | \$ 4,422,588.44 |
| [\$\$ in WA and OR ??] | \$ 3,638,239.88 |
| [\$\$ in WA !!] | \$ 2,467,248.34 |

NOTE

The above results were formatted to a table, instead of a single row, for educational purposes.

First, examine the expression, `[$$ in WA and OR ??]`. You might wonder how this formula could return the value for sales in Washington and Oregon, since the outer CALCULATE expression includes a filter for Washington and British Columbia. The answer is that the default behavior of CALCULATE overrides the outer filters in 'Geography'[State Province Code] and substitutes its own filter arguments, because the filters apply to the same column.

Next, examine the expression, `[$$ in WA !!]`. You might wonder how this formula could return the value for sales in Washington and nothing else, since the argument filter includes Oregon and the outer CALCULATE expression includes a filter in Washington and British Columbia. The answer is that KEEPFILTERS modifies the default behavior of CALCULATE and adds an additional filter. Because the intersection of filters is used, now the outer filter 'Geography'[State Province Code]="WA" || 'Geography'[State Province Code]="BC" is added to the filter argument 'Geography'[State Province Code]="WA" || 'Geography'[State Province Code]="OR",. Because both filters apply to the same column, the resulting filter 'Geography'[State Province Code]="WA" is the filter that is applied when evaluating the expression.

See also

[Filter functions](#)

[CALCULATE function](#)

[CALCULATETABLE function](#)

LOOKUPVALUE

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the value for the row that meets all criteria specified by one or more search conditions.

Syntax

```
LOOKUPVALUE(  
    <result_columnName>,  
    <search_columnName>,  
    <search_value>  
    [, <search2_columnName>, <search2_value>]...  
    [, <alternateResult>]  
)
```

Parameters

| TERM | DEFINITION |
|-------------------|---|
| result_columnName | The name of an existing column that contains the value you want to return. It cannot be an expression. |
| search_columnName | The name of an existing column. It can be in the same table as result_columnName or in a related table. It cannot be an expression. |
| search_value | The value to search for in search_columnName. |
| alternateResult | (Optional) The value returned when the context for result_columnName has been filtered down to zero or more than one distinct value. When not provided, the function returns BLANK when result_columnName is filtered down to zero value or an error when more than one distinct value. |

Return value

The value of **result_column** at the row where all pairs of **search_column** and **search_value** have an exact match.

If there's no match that satisfies all the search values, BLANK or **alternateResult** (if supplied) is returned. In other words, the function won't return a lookup value if only some of the criteria match.

If multiple rows match the search values and in all cases **result_column** values are identical, then that value is returned. However, if **result_column** returns different values, an error or **alternateResult** (if supplied) is returned.

Remarks

- If there is a relationship between the result and search tables, in most cases, using [RELATED](#) function instead of LOOKUPVALUE is more efficient and provides better performance.
- The **search_value** and **alternateResult** parameters are evaluated before the function iterates through

the rows of the search table.

- Avoid using ISERROR or IFERROR functions to capture an error returned by LOOKUPVALUE. If some inputs to the function will result in an error when a single output value cannot be determined, providing an alternateResult parameter is the most reliable and highest performing way to handle the error.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

Examples in this article can be used with the sample Adventure Works DW 2020 Power BI Desktop model. To get the model, see [DAX sample model](#).

The following calculated column defined in the **Sales** table uses the LOOKUPVALUE function to return channel values from the **Sales Order** table.

```
CHANNEL = LOOKUPVALUE('Sales Order'[Channel], 'Sales Order'[SalesOrderLineKey], [SalesOrderLineKey])
```

However, in this case, because there is a relationship between the **Sales Order** and **Sales** tables, it's more efficient to use the [RELATED](#) function.

```
CHANNEL = RELATED('Sales Order'[Channel])
```

See also

[RELATED function \(DAX\)](#)

[Information functions](#)

REMOVEFILTERS

10/31/2022 • 2 minutes to read • [Edit Online](#)

Clear filters from the specified tables or columns.

Syntax

```
REMOVEFILTERS([<table> | <column>[, <column>[, <column>[,...]]]])
```

Parameters

| TERM | DEFINITION |
|--------|---|
| table | The table that you want to clear filters on. |
| column | The column that you want to clear filters on. |

Return value

N/A. See remarks.

Remarks

- REMOVEFILTERS can only be used to clear filters but not to return a table.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example 1

DAX query

```
DEFINE
MEASURE FactInternetSales[TotalSales] = SUM(FactInternetSales[SalesAmount])
MEASURE FactInternetSales[%Sales] = DIVIDE([TotalSales], CALCULATE([TotalSales], REMOVEFILTERS()))

EVALUATE
    SUMMARIZECOLUMNS(
        ROLLUPADDISSUBTOTAL(DimProductCategory[EnglishProductCategoryName], "IsGrandTotal"),
        "TotalSales", [TotalSales],
        "%Sales", [%Sales]
    )
ORDER BY
    [IsGrandTotal] DESC, [TotalSales] DESC
```

Returns

| DIMPRODUCTCATEGORY[ENGLISHPRODUCTCATEGORYNAME] | [ISGRANDTOTAL] | [TOTALSALES] | [%SALES] |
|--|----------------|---------------|--------------------|
| Row1 | True | 29358677.2207 | 1 |
| Bikes | False | 28318144.6507 | 0.964557920570538 |
| Accessories | False | 700759.96 | 0.023868921434441 |
| Clothing | False | 339772.61 | 0.0115731579950215 |

Example 2

DAX query

```

DEFINE
MEASURE FactInternetSales[TotalSales] = SUM(FactInternetSales[SalesAmount])
MEASURE FactInternetSales[%Sales] = DIVIDE([TotalSales],
CALCULATE([TotalSales],REMOVEFILTERS(DimProductSubcategory[EnglishProductSubcategoryName])))

EVALUATE
SUMMARIZECOLUMNS(
DimProductCategory[EnglishProductCategoryName],
DimProductSubcategory[EnglishProductSubcategoryName],
"TotalSales", [TotalSales],
"%Sales", [%Sales]
)
ORDER BY
DimProductCategory[EnglishProductCategoryName] ASC,
DimProductSubcategory[EnglishProductSubcategoryName] ASC

```

Returns

| DIMPRODUCTCATEGORY[ENGLISHPRODUCTCATEGORYNAME] | DIMPRODUCTSUBCATEGORY[ENGLISHPRODUCTSUBCATEGORYNAME] | [TOTALSALES] | [%SALES] |
|--|--|--------------|------------|
| Accessories | Bike Racks | 39360 | 0.05616759 |
| Accessories | Bike Stands | 39591 | 0.05649723 |
| Accessories | Bottles and Cages | 56798.19 | 0.08105228 |
| Accessories | Cleaners | 7218.6 | 0.0103011 |
| Accessories | Fenders | 46619.58 | 0.06652717 |
| Accessories | Helmets | 225335.6 | 0.3215589 |
| Accessories | Hydration Packs | 40307.67 | 0.05751994 |
| Accessories | Tires and Tubes | 245529.32 | 0.35037578 |
| Bikes | Mountain Bikes | 9952759.564 | 0.35146228 |

| DIMPRODUCTCATEGORY [ENGLISHPRODUCTCATEGORYNAME] | DIMPRODUCTSUBCATEGORY [ENGLISHPRODUCTSUBCATEGORYNAME] | [TOTALSALES] | [%SALES] |
|--|--|--------------|------------|
| Bikes | Road Bikes | 14520584.04 | 0.51276608 |
| Bikes | Touring Bikes | 3844801.05 | 0.13577164 |
| Clothing | Caps | 19688.1 | 0.05794493 |
| Clothing | Gloves | 35020.7 | 0.10307099 |
| Clothing | Jerseys | 172950.68 | 0.5090189 |
| Clothing | Shorts | 71319.81 | 0.20990453 |
| Clothing | Socks | 5106.32 | 0.01502864 |
| Clothing | Vests | 35687 | 0.10503201 |

SELECTEDVALUE

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the value when the context for columnName has been filtered down to one distinct value only. Otherwise returns alternateResult.

Syntax

```
SELECTEDVALUE(<columnName>[, <alternateResult>])
```

Parameters

| TERM | DEFINITION |
|-----------------|--|
| columnName | The name of an existing column, using standard DAX syntax. It cannot be an expression. |
| alternateResult | (Optional) The value returned when the context for columnName has been filtered down to zero or more than one distinct value. When not provided, the default value is BLANK(). |

Return value

The value when the context for columnName has been filtered down to one distinct value only. Else, alternateResult.

Remarks

- An equivalent expression for `SELECTEDVALUE(<columnName>, <alternateResult>)` is `IF(HASONEVALUE(<columnName>), VALUES(<columnName>), <alternateResult>)`.
- To learn more about best practices when using SELECTEDVALUE, see [Use SELECTEDVALUE instead of VALUES in DAX](#).
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following DAX query:

```
DEFINE
    MEASURE DimProduct[Selected Color] = SELECTEDVALUE(DimProduct[Color], "No Single Selection")
EVALUATE
    SUMMARIZECOLUMNS
        (ROLLUPADDISSUBTOTAL(DimProduct[Color], "Is Total"),
        "Selected Color", [Selected Color])ORDER BY [Is Total] ASC,
        [Color] ASC
```

Returns the following:

| DIMPRODUCT[COLOR] | [IS TOTAL] | [SELECTED COLOR] |
|-------------------|------------|---------------------|
| Black | FALSE | Black |
| Blue | FALSE | Blue |
| Grey | FALSE | Grey |
| Multi | FALSE | Multi |
| NA | FALSE | NA |
| Red | FALSE | Red |
| Silver | FALSE | Silver |
| Silver/Black | FALSE | Silver/Black |
| White | FALSE | White |
| Yellow | FALSE | Yellow |
| | TRUE | No Single Selection |

Financial functions

10/31/2022 • 4 minutes to read • [Edit Online](#)

Financial functions in DAX are used in formulas that perform financial calculations, such as net present value and rate of return. These functions are similar to financial functions used in Microsoft Excel.

In this category

| FUNCTION | DESCRIPTION |
|----------------------------|---|
| ACCRINT | Returns the accrued interest for a security that pays periodic interest. |
| ACCRINTM | Returns the accrued interest for a security that pays interest at maturity. |
| AMORDEGRC | Returns the depreciation for each accounting period. Similar to AMORLINC, except a depreciation coefficient is applied depending on the life of the assets. |
| AMORLINC | Returns the depreciation for each accounting period. |
| COUPDAYBS | Returns the number of days from the beginning of a coupon period until its settlement date. |
| COUPDAYS | Returns the number of days in the coupon period that contains the settlement date. |
| COUPDAYSNC | Returns the number of days from the settlement date to the next coupon date. |
| COUPNCD | Returns the next coupon date after the settlement date. |
| COUPNUM | Returns the number of coupons payable between the settlement date and maturity date, rounded up to the nearest whole coupon. |
| COUPPCD | Returns the previous coupon date before the settlement date. |
| CUMIPMT | Returns the cumulative interest paid on a loan between start_period and end_period. |
| CUMPRINC | Returns the cumulative principal paid on a loan between start_period and end_period. |
| DB | Returns the depreciation of an asset for a specified period using the fixed-declining balance method. |

| FUNCTION | DESCRIPTION |
|-----------|--|
| DDB | Returns the depreciation of an asset for a specified period using the double-declining balance method or some other method you specify. |
| DISC | Returns the discount rate for a security. |
| DOLLARDE | Converts a dollar price expressed as an integer part and a fraction part, such as 1.02, into a dollar price expressed as a decimal number. |
| DOLLARFR | Converts a dollar price expressed as an integer part and a fraction part, such as 1.02, into a dollar price expressed as a decimal number. |
| DURATION | Returns the Macauley duration for an assumed par value of \$100. |
| EFFECT | Returns the effective annual interest rate, given the nominal annual interest rate and the number of compounding periods per year. |
| FV | Calculates the future value of an investment based on a constant interest rate. |
| INTRATE | Returns the interest rate for a fully invested security. |
| IPMT | Returns the interest payment for a given period for an investment based on periodic, constant payments and a constant interest rate. |
| ISPMT | Calculates the interest paid (or received) for the specified period of a loan (or investment) with even principal payments. |
| MDURATION | Returns the modified Macauley duration for a security with an assumed par value of \$100. |
| NOMINAL | Returns the nominal annual interest rate, given the effective rate and the number of compounding periods per year. |
| NPER | Returns the number of periods for an investment based on periodic, constant payments and a constant interest rate. |
| ODDFPRICE | Returns the price per \$100 face value of a security having an odd (short or long) first period. |
| ODDFYIELD | Returns the yield of a security that has an odd (short or long) first period. |
| ODDLPRICE | Returns the price per \$100 face value of a security having an odd (short or long) last coupon period. |
| ODDLYIELD | Returns the yield of a security that has an odd (short or long) last period. |

| FUNCTION | DESCRIPTION |
|------------|---|
| PDURATION | Returns the number of periods required by an investment to reach a specified value. |
| PMT | Calculates the payment for a loan based on constant payments and a constant interest rate. |
| PPMT | Returns the payment on the principal for a given period for an investment based on periodic, constant payments and a constant interest rate. |
| PRICE | Returns the price per \$100 face value of a security that pays periodic interest. |
| PRICEDISC | Returns the price per \$100 face value of a discounted security. |
| PRICEMAT | Returns the price per \$100 face value of a security that pays interest at maturity. |
| PV | Calculates the present value of a loan or an investment, based on a constant interest rate. |
| RATE | Returns the interest rate per period of an annuity. |
| RECEIVED | Returns the amount received at maturity for a fully invested security. |
| RRI | Returns an equivalent interest rate for the growth of an investment. |
| SLN | Returns the straight-line depreciation of an asset for one period. |
| SYD | Returns the sum-of-years' digits depreciation of an asset for a specified period. |
| TBILLEQ | Returns the bond-equivalent yield for a Treasury bill. |
| TBILLPRICE | Returns the price per \$100 face value for a Treasury bill. |
| TBILLYIELD | Returns the yield for a Treasury bill. |
| VDB | Returns the depreciation of an asset for any period you specify, including partial periods, using the double-declining balance method or some other method you specify. |
| XIRR | Returns the internal rate of return for a schedule of cash flows that is not necessarily periodic. |
| XNPV | Returns the present value for a schedule of cash flows that is not necessarily periodic. |
| YIELD | Returns the yield on a security that pays periodic interest. |

| FUNCTION | DESCRIPTION |
|---------------------------|--|
| YIELDDISC | Returns the annual yield for a discounted security. |
| YIELDMAT | Returns the annual yield of a security that pays interest at maturity. |

ACCRINT

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the accrued interest for a security that pays periodic interest.

Syntax

```
ACCRINT(<issue>, <first_interest>, <settlement>, <rate>, <par>, <frequency>[, <basis>[, <calc_method>]])
```

Parameters

| TERM | DEFINITION |
|----------------|--|
| issue | The security's issue date. |
| first_interest | The security's first interest date. |
| settlement | The security's settlement date. The security settlement date is the date after the issue date when the security is traded to the buyer. |
| rate | The security's annual coupon rate. |
| par | The security's par value. |
| frequency | The number of coupon payments per year. For annual payments, frequency = 1; for semiannual, frequency = 2; for quarterly, frequency = 4. |
| basis | (Optional) The type of day count basis to use. If basis is omitted, it is assumed to be 0. The accepted values are listed below this table. |
| calc_method | (Optional) A logical value that specifies the way to calculate the total accrued interest when the date of settlement is later than the date of first_interest. If calc_method is omitted, it is assumed to be TRUE. - If calc_method evaluates to TRUE or is omitted, ACCRINT returns the total accrued interest from issue to settlement. - If calc_method evaluates to FALSE, ACCRINT returns the accrued interest from first_interest to settlement. |

The **basis** parameter accepts the following values:

| BASIS | DAY COUNT BASIS |
|--------------|------------------|
| 0 or omitted | US (NASD) 30/360 |
| 1 | Actual/actual |
| 2 | Actual/360 |

| BASIS | DAY COUNT BASIS |
|-------|-----------------|
| 3 | Actual/365 |
| 4 | European 30/360 |

Return Value

The accrued interest.

Remarks

- Dates are stored as sequential serial numbers so they can be used in calculations. In DAX, December 30, 1899 is day 0, and January 1, 2008 is 39448 because it is 39,448 days after December 30, 1899.

- ACCRINT is calculated as follows:

$$\text{ACCRINT} = \text{par} \times \frac{\text{rate}}{\text{frequency}} \times \sum_{i=1}^{\text{NC}} \frac{\text{A}_i}{\text{NL}_i}$$

where:

- A_i = number of accrued days for the i^{th} quasi-coupon period within odd period.
 - NC = number of quasi-coupon periods that fit in odd period. If this number contains a fraction, raise it to the next whole number.
 - NL_i = normal length in days of the quasi-coupon period within odd period.
- issue, first_interest, and settlement are truncated to integers.
- frequency and basis are rounded to the nearest integer.
- An error is returned if:
 - issue, first_interest, or settlement is not a valid date.
 - issue \geq settlement.
 - rate \leq 0.
 - par \leq 0.
 - frequency is any number other than 1, 2, or 4.
 - basis $<$ 0 or basis $>$ 4.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Examples

| DATA | DESCRIPTION |
|----------------|---------------------|
| 1-March-2007 | Issue date |
| 31-August-2008 | First interest date |
| 1-May-2008 | Settlement date |
| 10% | Coupon rate |

| DATA | DESCRIPTION |
|------|-------------------------------------|
| 1000 | Par value |
| 2 | Frequency is semiannual (see above) |
| 0 | 30/360 basis (see above) |

Example 1

The following DAX query:

```
EVALUATE
{
    ACCRINT(DATE(2007,3,1), DATE(2008,8,31), DATE(2008,5,1), 0.1, 1000, 2, 0)
}
```

Returns the accrued interest from issue to settlement, for a security with the terms specified above.

| [VALUE] |
|------------------|
| 116.944444444444 |

Example 2

The following DAX query:

```
EVALUATE
{
    ACCRINT(DATE(2007,3,1), DATE(2008,8,31), DATE(2008,5,1), 0.1, 1000, 2, 0, FALSE)
}
```

Returns the accrued interest from first_interest to settlement, for a security with the terms specified above.

| [VALUE] |
|------------------|
| 66.9444444444445 |

ACCRINTM

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the accrued interest for a security that pays interest at maturity.

Syntax

```
ACCRINTM(<issue>, <maturity>, <rate>, <par>[, <basis>])
```

Parameters

| TERM | DEFINITION |
|----------|---|
| issue | The security's issue date. |
| maturity | The security's maturity date. |
| rate | The security's annual coupon rate. |
| par | The security's par value. |
| basis | (Optional) The type of day count basis to use. If basis is omitted, it is assumed to be 0. The accepted values are listed below this table. |

The **basis** parameter accepts the following values:

| BASIS | DAY COUNT BASIS |
|--------------|------------------|
| 0 or omitted | US (NASD) 30/360 |
| 1 | Actual/actual |
| 2 | Actual/360 |
| 3 | Actual/365 |
| 4 | European 30/360 |

Return Value

The accrued interest.

Remarks

- Dates are stored as sequential serial numbers so they can be used in calculations. In DAX, December 30, 1899 is day 0, and January 1, 2008 is 39448 because it is 39,448 days after December 30, 1899.
- ACCRINTM is calculated as follows:

$$\text{ACCINTM} = \text{par} \times \text{rate} \times \frac{\text{A}}{\text{D}}$$

where:

- A = Number of accrued days counted according to a monthly basis. For interest at maturity items, the number of days from the issue date to the maturity date is used.
- D = Annual Year Basis.
- issue and maturity are truncated to integers.
- basis is rounded to the nearest integer.
- An error is returned if:
 - issue or maturity is not a valid date.
 - $\text{issue} \geq \text{maturity}$.
 - $\text{rate} \leq 0$.
 - $\text{par} \leq 0$.
 - $\text{basis} < 0$ or $\text{basis} > 4$.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

| DATA | DESCRIPTION |
|--------------|------------------------------|
| 1-April-2008 | Issue date |
| 15-June-2008 | Maturity date |
| 10% | Percent coupon |
| 1000 | Par value |
| 3 | Actual/365 basis (see above) |

The following DAX query:

```
EVALUATE
{
    ACCINTM(DATE(2008,4,1), DATE(2008,6,15), 0.1, 1000, 3)
}
```

Returns the accrued interest for a security with the terms specified above.

| [VALUE] |
|------------------|
| 20.5479452054795 |

AMORDEGRC

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the depreciation for each accounting period. This function is provided for the French accounting system. If an asset is purchased in the middle of the accounting period, the prorated depreciation is taken into account. The function is similar to AMORLINC, except that a depreciation coefficient is applied in the calculation depending on the life of the assets.

Syntax

```
AMORDEGRC(<cost>, <date_purchased>, <first_period>, <salvage>, <period>, <rate>[, <basis>])
```

Parameters

| TERM | DEFINITION |
|----------------|---|
| cost | The cost of the asset. |
| date_purchased | The date of the purchase of the asset. |
| first_period | The date of the end of the first period. |
| salvage | The salvage value at the end of the life of the asset. |
| period | The period. |
| rate | The rate of depreciation. |
| basis | (Optional) The type of day count basis to use. If basis is omitted, it is assumed to be 0. The accepted values are listed below this table. |

The **basis** parameter accepts the following values:

| BASIS | DATE SYSTEM |
|--------------|--------------------------------------|
| 0 or omitted | 360 days (NASD method) |
| 1 | Actual |
| 3 | 365 days in a year |
| 4 | 360 days in a year (European method) |

Return Value

The depreciation for each accounting period.

Remarks

- Dates are stored as sequential serial numbers so they can be used in calculations. In DAX, December 30, 1899 is day 0, and January 1, 2008 is 39448 because it is 39,448 days after December 30, 1899.
- This function will return the depreciation until the last period of the life of the assets or until the cumulated value of depreciation is greater than the cost of the assets minus the salvage value.
- The depreciation coefficients are:

| LIFE OF ASSETS (1/RATE) | DEPRECIATION COEFFICIENT |
|-------------------------|--------------------------|
| Between 3 and 4 years | 1.5 |
| Between 5 and 6 years | 2 |
| More than 6 years | 2.5 |

- The depreciation rate will grow to 50 percent for the period preceding the last period and will grow to 100 percent for the last period.
- period and basis are rounded to the nearest integer.
- An error is returned if:
 - cost < 0.
 - first_period or date_purchased is not a valid date.
 - date_purchased > first_period.
 - salvage < 0 or salvage > cost.
 - period < 0.
 - rate ≤ 0.
 - The life of assets is between 0 (zero) and 1, 1 and 2, 2 and 3, or 4 and 5.
 - basis is any number other than 0, 1, 3, or 4.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

| DATA | DESCRIPTION |
|------------------|--------------------------|
| 2400 | Cost |
| 19-August-2008 | Date purchased |
| 31-December-2008 | End of the first period |
| 300 | Salvage value |
| 1 | Period |
| 15% | Depreciation rate |
| 1 | Actual basis (see above) |

The following DAX query:

```
EVALUATE
{
    AMORDEGRC(2400, DATE(2008,8,19), DATE(2008,12,31), 300, 1, 0.15, 1)
}
```

Returns the first period's depreciation, given the terms specified above.

| [VALUE] |
|---------|
| 776 |

AMORLINC

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the depreciation for each accounting period. This function is provided for the French accounting system. If an asset is purchased in the middle of the accounting period, the prorated depreciation is taken into account.

Syntax

```
AMORLINC(<cost>, <date_purchased>, <first_period>, <salvage>, <period>, <rate>[, <basis>])
```

Parameters

| TERM | DEFINITION |
|----------------|---|
| cost | The cost of the asset. |
| date_purchased | The date of the purchase of the asset. |
| first_period | The date of the end of the first period. |
| salvage | The salvage value at the end of the life of the asset. |
| period | The period. |
| rate | The rate of depreciation. |
| basis | (Optional) The type of day count basis to use. If basis is omitted, it is assumed to be 0. The accepted values are listed below this table. |

The **basis** parameter accepts the following values:

| BASIS | DATE SYSTEM |
|--------------|--------------------------------------|
| 0 or omitted | 360 days (NASD method) |
| 1 | Actual |
| 3 | 365 days in a year |
| 4 | 360 days in a year (European method) |

Return Value

The depreciation for each accounting period.

Remarks

- Dates are stored as sequential serial numbers so they can be used in calculations. In DAX, December 30,

1899 is day 0, and January 1, 2008 is 39448 because it is 39,448 days after December 30, 1899.

- period and basis are rounded to the nearest integer.
- An error is returned if:
 - $\text{cost} < 0$.
 - first_period or date_purchased is not a valid date.
 - $\text{date_purchased} > \text{first_period}$.
 - $\text{salvage} < 0$ or $\text{salvage} > \text{cost}$.
 - $\text{period} < 0$.
 - $\text{rate} \leq 0$.
 - basis is any number other than 0, 1, 3, or 4.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

| DATA | DESCRIPTION |
|------------------|--------------------------|
| 2400 | Cost |
| 19-August-2008 | Date purchased |
| 31-December-2008 | End of the first period |
| 300 | Salvage value |
| 1 | Period |
| 15% | Depreciation rate |
| 1 | Actual basis (see above) |

The following DAX query:

```
EVALUATE
{
    AMORLINC(2400, DATE(2008,8,19), DATE(2008,12,31), 300, 1, 0.15, 1)
}
```

Returns the first period's depreciation, given the terms specified above.

| [VALUE] |
|---------|
| 360 |

COUPDAYBS

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the number of days from the beginning of a coupon period until its settlement date.

Syntax

```
COUPDAYBS(<settlement>, <maturity>, <frequency>[, <basis>])
```

Parameters

| TERM | DEFINITION |
|------------|---|
| settlement | The security's settlement date. The security settlement date is the date after the issue date when the security is traded to the buyer. |
| maturity | The security's maturity date. The maturity date is the date when the security expires. |
| frequency | The number of coupon payments per year. For annual payments, frequency = 1; for semiannual, frequency = 2; for quarterly, frequency = 4. |
| basis | (Optional) The type of day count basis to use. If basis is omitted, it is assumed to be 0. The accepted values are listed below this table. |

The **basis** parameter accepts the following values:

| BASIS | DAY COUNT BASIS |
|--------------|------------------|
| 0 or omitted | US (NASD) 30/360 |
| 1 | Actual/actual |
| 2 | Actual/360 |
| 3 | Actual/365 |
| 4 | European 30/360 |

Return Value

The number of days from the beginning of a coupon period until its settlement date.

Remarks

- Dates are stored as sequential serial numbers so they can be used in calculations. In DAX, December 30, 1899 is day 0, and January 1, 2008 is 39448 because it is 39,448 days after December 30, 1899.

- The settlement date is the date a buyer purchases a coupon, such as a bond. The maturity date is the date when a coupon expires. For example, suppose a 30-year bond is issued on January 1, 2008, and is purchased by a buyer six months later. The issue date would be January 1, 2008, the settlement date would be July 1, 2008, and the maturity date would be January 1, 2038, 30 years after the January 1, 2008, issue date.
- settlement and maturity are truncated to integers.
- frequency and basis are rounded to the nearest integer.
- An error is returned if:
 - settlement or maturity is not a valid date.
 - settlement \geq maturity.
 - frequency is any number other than 1, 2, or 4.
 - basis < 0 or basis > 4.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

| DATA | DESCRIPTION |
|-----------|---------------------------------|
| 25-Jan-11 | Settlement date |
| 15-Nov-11 | Maturity date |
| 2 | Semiannual coupon (see above) |
| 1 | Actual/actual basis (see above) |

The following DAX query:

```
EVALUATE
{
    COUPDAYBS(DATE(2011,1,25), DATE(2011,11,15), 2, 1)
}
```

Returns the number of days from the beginning of the coupon period until the settlement date, for a bond with terms above.

| [VALUE] |
|---------|
| 71 |

COUPDAYS

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the number of days in the coupon period that contains the settlement date.

Syntax

```
COUPDAYS(<settlement>, <maturity>, <frequency>[, <basis>])
```

Parameters

| TERM | DEFINITION |
|------------|---|
| settlement | The security's settlement date. The security settlement date is the date after the issue date when the security is traded to the buyer. |
| maturity | The security's maturity date. The maturity date is the date when the security expires. |
| frequency | The number of coupon payments per year. For annual payments, frequency = 1; for semiannual, frequency = 2; for quarterly, frequency = 4. |
| basis | (Optional) The type of day count basis to use. If basis is omitted, it is assumed to be 0. The accepted values are listed below this table. |

The **basis** parameter accepts the following values:

| BASIS | DAY COUNT BASIS |
|--------------|------------------|
| 0 or omitted | US (NASD) 30/360 |
| 1 | Actual/actual |
| 2 | Actual/360 |
| 3 | Actual/365 |
| 4 | European 30/360 |

Return Value

The number of days in the coupon period that contains the settlement date.

Remarks

- Dates are stored as sequential serial numbers so they can be used in calculations. In DAX, December 30, 1899 is day 0, and January 1, 2008 is 39448 because it is 39,448 days after December 30, 1899.

- The settlement date is the date a buyer purchases a coupon, such as a bond. The maturity date is the date when a coupon expires. For example, suppose a 30-year bond is issued on January 1, 2008, and is purchased by a buyer six months later. The issue date would be January 1, 2008, the settlement date would be July 1, 2008, and the maturity date is January 1, 2038, 30 years after the January 1, 2008 issue date.
- settlement and maturity are truncated to integers.
- frequency and basis are rounded to the nearest integer.
- An error is returned if:
 - settlement or maturity is not a valid date.
 - settlement \geq maturity.
 - frequency is any number other than 1, 2, or 4.
 - basis < 0 or basis > 4.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

| DATA | DESCRIPTION |
|-----------|---------------------------------|
| 25-Jan-11 | Settlement date |
| 15-Nov-11 | Maturity date |
| 2 | Semiannual coupon (see above) |
| 1 | Actual/actual basis (see above) |

The following DAX query:

```
EVALUATE
{
    COUPDAYS(DATE(2011,1,25), DATE(2011,11,15), 2, 1)
}
```

Returns the number of days in the coupon period that contains the settlement date, for a bond with the terms specified above.

| [VALUE] |
|---------|
| 181 |

COUPDAYSNC

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the number of days from the settlement date to the next coupon date.

Syntax

```
COUPDAYSNC(<settlement>, <maturity>, <frequency>[, <basis>])
```

Parameters

| TERM | DEFINITION |
|------------|---|
| settlement | The security's settlement date. The security settlement date is the date after the issue date when the security is traded to the buyer. |
| maturity | The security's maturity date. The maturity date is the date when the security expires. |
| frequency | The number of coupon payments per year. For annual payments, frequency = 1; for semiannual, frequency = 2; for quarterly, frequency = 4. |
| basis | (Optional) The type of day count basis to use. If basis is omitted, it is assumed to be 0. The accepted values are listed below this table. |

The **basis** parameter accepts the following values:

| BASIS | DAY COUNT BASIS |
|--------------|------------------|
| 0 or omitted | US (NASD) 30/360 |
| 1 | Actual/actual |
| 2 | Actual/360 |
| 3 | Actual/365 |
| 4 | European 30/360 |

Return Value

The number of days from the settlement date to the next coupon date.

Remarks

- Dates are stored as sequential serial numbers so they can be used in calculations. In DAX, December 30, 1899 is day 0, and January 1, 2008 is 39448 because it is 39,448 days after December 30, 1899.

- The settlement date is the date a buyer purchases a coupon, such as a bond. The maturity date is the date when a coupon expires. For example, suppose a 30-year bond is issued on January 1, 2008, and is purchased by a buyer six months later. The issue date would be January 1, 2008, the settlement date would be July 1, 2008, and the maturity date would be January 1, 2038, which is 30 years after the January 1, 2008, issue date.
- settlement and maturity are truncated to integers.
- frequency and basis are rounded to the nearest integer.
- An error is returned if:
 - settlement or maturity is not a valid date.
 - settlement \geq maturity.
 - frequency is any number other than 1, 2, or 4.
 - basis < 0 or basis > 4.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

| DATA | DESCRIPTION |
|-----------|---------------------------------|
| 25-Jan-11 | Settlement date |
| 15-Nov-11 | Maturity date |
| 2 | Semiannual coupon (see above) |
| 1 | Actual/actual basis (see above) |

The following DAX query:

```
EVALUATE
{
    COUPDAYSNC(DATE(2011,1,25), DATE(2011,11,15), 2, 1)
}
```

Returns the number of days from the settlement date to the next coupon date, for a bond with the terms specified above.

| [VALUE] |
|---------|
| 110 |

COUPNCD

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the next coupon date after the settlement date.

Syntax

```
COUPNCD(<settlement>, <maturity>, <frequency>[, <basis>])
```

Parameters

| TERM | DEFINITION |
|------------|---|
| settlement | The security's settlement date. The security settlement date is the date after the issue date when the security is traded to the buyer. |
| maturity | The security's maturity date. The maturity date is the date when the security expires. |
| frequency | The number of coupon payments per year. For annual payments, frequency = 1; for semiannual, frequency = 2; for quarterly, frequency = 4. |
| basis | (Optional) The type of day count basis to use. If basis is omitted, it is assumed to be 0. The accepted values are listed below this table. |

The **basis** parameter accepts the following values:

| BASIS | DAY COUNT BASIS |
|--------------|------------------|
| 0 or omitted | US (NASD) 30/360 |
| 1 | Actual/actual |
| 2 | Actual/360 |
| 3 | Actual/365 |
| 4 | European 30/360 |

Return Value

The next coupon date after the settlement date.

Remarks

- Dates are stored as sequential serial numbers so they can be used in calculations. In DAX, December 30, 1899 is day 0, and January 1, 2008 is 39448 because it is 39,448 days after December 30, 1899.

- The settlement date is the date a buyer purchases a coupon, such as a bond. The maturity date is the date when a coupon expires. For example, suppose a 30-year bond is issued on January 1, 2008, and is purchased by a buyer six months later. The issue date would be January 1, 2008, the settlement date would be July 1, 2008, and the maturity date is January 1, 2038, 30 years after the January 1, 2008 issue date.
- settlement and maturity are truncated to integers.
- frequency and basis are rounded to the nearest integer.
- An error is returned if:
 - settlement or maturity is not a valid date.
 - settlement \geq maturity.
 - frequency is any number other than 1, 2, or 4.
 - basis < 0 or basis > 4.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

| DATA | DESCRIPTION |
|-----------|---------------------------------|
| 25-Jan-11 | Settlement date |
| 15-Nov-11 | Maturity date |
| 2 | Semiannual coupon (see above) |
| 1 | Actual/actual basis (see above) |

The following DAX query:

```
EVALUATE
{
    COUPNCD(DATE(2011,1,25), DATE(2011,11,15), 2, 1)
}
```

Returns the next coupon date after the settlement date, for a bond with the terms specified above.

| [VALUE] |
|-----------------------|
| 5/15/2011 12:00:00 AM |

COUPNUM

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the number of coupons payable between the settlement date and maturity date, rounded up to the nearest whole coupon.

Syntax

```
COUPNUM(<settlement>, <maturity>, <frequency>[, <basis>])
```

Parameters

| TERM | DEFINITION |
|------------|---|
| settlement | The security's settlement date. The security settlement date is the date after the issue date when the security is traded to the buyer. |
| maturity | The security's maturity date. The maturity date is the date when the security expires. |
| frequency | The number of coupon payments per year. For annual payments, frequency = 1; for semiannual, frequency = 2; for quarterly, frequency = 4. |
| basis | (Optional) The type of day count basis to use. If basis is omitted, it is assumed to be 0. The accepted values are listed below this table. |

The **basis** parameter accepts the following values:

| BASIS | DAY COUNT BASIS |
|--------------|------------------|
| 0 or omitted | US (NASD) 30/360 |
| 1 | Actual/actual |
| 2 | Actual/360 |
| 3 | Actual/365 |
| 4 | European 30/360 |

Return Value

The number of coupons payable between the settlement date and maturity date.

Remarks

- Dates are stored as sequential serial numbers so they can be used in calculations. In DAX, December 30,

1899 is day 0, and January 1, 2008 is 39448 because it is 39,448 days after December 30, 1899.

- The settlement date is the date a buyer purchases a coupon, such as a bond. The maturity date is the date when a coupon expires. For example, suppose a 30-year bond is issued on January 1, 2008, and is purchased by a buyer six months later. The issue date would be January 1, 2008, the settlement date would be July 1, 2008, and the maturity date would be January 1, 2038, which is 30 years after the January 1, 2008, issue date.
- settlement and maturity are truncated to integers.
- frequency and basis are rounded to the nearest integer.
- An error is returned if:
 - settlement or maturity is not a valid date.
 - settlement \geq maturity.
 - frequency is any number other than 1, 2, or 4.
 - basis < 0 or basis > 4 .
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

| DATA | DESCRIPTION |
|-----------|---------------------------------|
| 25-Jan-07 | Settlement date |
| 15-Nov-08 | Maturity date |
| 2 | Semiannual coupon (see above) |
| 1 | Actual/actual basis (see above) |

The following DAX query:

```
EVALUATE
{
    COUPNUM(DATE(2007,1,25), DATE(2008,11,15), 2, 1)
}
```

Returns the number of coupon payments for a bond with the terms specified above.

| [VALUE] |
|---------|
| 4 |

COUPPCD

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the previous coupon date before the settlement date.

Syntax

```
COUPPCD(<settlement>, <maturity>, <frequency>[, <basis>])
```

Parameters

| TERM | DEFINITION |
|------------|---|
| settlement | The security's settlement date. The security settlement date is the date after the issue date when the security is traded to the buyer. |
| maturity | The security's maturity date. The maturity date is the date when the security expires. |
| frequency | The number of coupon payments per year. For annual payments, frequency = 1; for semiannual, frequency = 2; for quarterly, frequency = 4. |
| basis | (Optional) The type of day count basis to use. If basis is omitted, it is assumed to be 0. The accepted values are listed below this table. |

The **basis** parameter accepts the following values:

| BASIS | DAY COUNT BASIS |
|--------------|------------------|
| 0 or omitted | US (NASD) 30/360 |
| 1 | Actual/actual |
| 2 | Actual/360 |
| 3 | Actual/365 |
| 4 | European 30/360 |

Return Value

The previous coupon date before the settlement date.

Remarks

- Dates are stored as sequential serial numbers so they can be used in calculations. In DAX, December 30, 1899 is day 0, and January 1, 2008 is 39448 because it is 39,448 days after December 30, 1899.

- The settlement date is the date a buyer purchases a coupon, such as a bond. The maturity date is the date when a coupon expires. For example, suppose a 30-year bond is issued on January 1, 2008, and is purchased by a buyer six months later. The issue date would be January 1, 2008, the settlement date would be July 1, 2008, and the maturity date would be January 1, 2038, which is 30 years after the January 1, 2008, issue date.
- settlement and maturity are truncated to integers.
- frequency and basis are rounded to the nearest integer.
- An error is returned if:
 - settlement or maturity is not a valid date.
 - settlement \geq maturity.
 - frequency is any number other than 1, 2, or 4.
 - basis < 0 or basis > 4.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

| DATA | DESCRIPTION |
|-----------|---------------------------------|
| 25-Jan-11 | Settlement date |
| 15-Nov-11 | Maturity date |
| 2 | Semiannual coupon (see above) |
| 1 | Actual/actual basis (see above) |

The following DAX query:

```
EVALUATE
{
    COUPPCD(DATE(2011,1,25), DATE(2011,11,15), 2, 1)
}
```

Returns the previous coupon date before the settlement date, for a bond using the terms specified above.

| [VALUE] |
|------------------------|
| 11/15/2010 12:00:00 AM |

CUMIPMT

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the cumulative interest paid on a loan between start_period and end_period.

Syntax

```
CUMIPMT(<rate>, <nper>, <pv>, <start_period>, <end_period>, <type>)
```

Parameters

| TERM | DEFINITION |
|--------------|--|
| rate | The interest rate. |
| nper | The total number of payment periods. |
| pv | The present value. |
| start_period | The first period in the calculation. Must be between 1 and end_period (inclusive). |
| end_period | The last period in the calculation. Must be between start_period and nper (inclusive). |
| type | The timing of the payment. The accepted values are listed below this table. |

The **type** parameter accepts the following values:

| TYPE | TIMING |
|----------|--|
| 0 (zero) | Payment at the end of the period |
| 1 | Payment at the beginning of the period |

Return Value

The cumulative interest paid in the specified period.

Remarks

- Make sure that you are consistent about the units you use for specifying rate and nper. If you make monthly payments on a four-year loan at an annual interest rate of 10 percent, use 0.1/12 for rate and 4*12 for nper. If you make annual payments on the same loan, use 0.1 for rate and 4 for nper.
- start_period, end_period, and type are rounded to the nearest integer.
- An error is returned if:

- $\text{rate} \leq 0$.
- $\text{nper} < 1$.
- $\text{pv} \leq 0$.
- $\text{start_period} < 1$ or $\text{start_period} > \text{end_period}$.
- $\text{end_period} < \text{start_period}$ or $\text{end_period} > \text{nper}$.
- type is any number other than 0 or 1.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Examples

| DATA | DESCRIPTION |
|--------|----------------------|
| 9% | Annual interest rate |
| 30 | Years of the loan |
| 125000 | Present value |

Example 1

The following DAX query:

```
EVALUATE
{
    CUMIPMT(0.09/12, 30*12, 125000, 13, 24, 1)
}
```

Returns the total interest paid in the second year of payments, periods 13 through 24, assuming that the payments are made at the beginning of each month.

| [VALUE] |
|-------------------|
| -11052.3395838718 |

Example 2

The following DAX query:

```
EVALUATE
{
    CUMIPMT(0.09/12, 30*12, 125000, 1, 1, 0)
}
```

Returns the interest paid in a single payment in the first month, assuming that the payment is made at the end of the month.

| [VALUE] |
|---------|
| -937.5 |

CUMPRINC

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the cumulative principal paid on a loan between start_period and end_period.

Syntax

```
CUMPRINC(<rate>, <nper>, <pv>, <start_period>, <end_period>, <type>)
```

Parameters

| TERM | DEFINITION |
|--------------|--|
| rate | The interest rate. |
| nper | The total number of payment periods. |
| pv | The present value. |
| start_period | The first period in the calculation. Must be between 1 and end_period (inclusive). |
| end_period | The last period in the calculation. Must be between start_period and nper (inclusive). |
| type | The timing of the payment. The accepted values are listed below this table. |

The **type** parameter accepts the following values:

| TYPE | TIMING |
|----------|--|
| 0 (zero) | Payment at the end of the period |
| 1 | Payment at the beginning of the period |

Return Value

The cumulative principal paid in the specified period.

Remarks

- Make sure that you are consistent about the units you use for specifying rate and nper. If you make monthly payments on a four-year loan at an annual interest rate of 10 percent, use 0.1/12 for rate and 4*12 for nper. If you make annual payments on the same loan, use 0.1 for rate and 4 for nper.
- start_period, end_period, and type are rounded to the nearest integer.
- An error is returned if:

- $\text{rate} \leq 0$.
- $\text{nper} < 1$.
- $\text{pv} \leq 0$.
- $\text{start_period} < 1$ or $\text{start_period} > \text{end_period}$.
- $\text{end_period} < \text{start_period}$ or $\text{end_period} > \text{nper}$.
- type is any number other than 0 or 1.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Examples

| DATA | DESCRIPTION |
|--------|----------------------|
| 9% | Annual interest rate |
| 30 | Term in years |
| 125000 | Present value |

Example 1

The following DAX query:

```
EVALUATE
{
    CUMPRINC(0.09/12, 30*12, 125000, 13, 24, 1)
}
```

Returns the total principal paid in the second year of payments, periods 13 through 24, assuming the payments are made at the beginning of each month.

| [VALUE] |
|-------------------|
| -927.153472378062 |

Example 2

The following DAX query:

```
EVALUATE
{
    CUMPRINC(0.09/12, 30*12, 125000, 1, 1, 0)
}
```

Returns the principal paid in a single payment in the first month, assuming the payment is made at the end of the month.

| [VALUE] |
|-------------------|
| -68.2782711809784 |

DB

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the depreciation of an asset for a specified period using the fixed-declining balance method.

Syntax

```
DB(<cost>, <salvage>, <life>, <period>[, <month>])
```

Parameters

| TERM | DEFINITION |
|---------|--|
| cost | The initial cost of the asset. |
| salvage | The value at the end of the depreciation (sometimes called the salvage value of the asset). This value can be 0. |
| life | The number of periods over which the asset is being depreciated (sometimes called the useful life of the asset). |
| period | The period for which you want to calculate the depreciation. Period must use the same units as life. Must be between 1 and life (inclusive). |
| month | (Optional) The number of months in the first year. If month is omitted, it is assumed to be 12. |

Return Value

The depreciation over the specified period.

Remarks

- The fixed-declining balance method computes depreciation at a fixed rate. DB uses the following formulas to calculate depreciation for a period:

$$\text{DB}(\text{cost} - \text{total depreciation from prior periods}) \times \text{rate}$$

where:

- $\text{rate} = 1 - (\frac{\text{salvage}}{\text{cost}})^{\frac{1}{\text{life}}}$, rounded to three decimal places
- Depreciation for the first and last periods is a special case.
 - For the first period, DB uses this formula:
$$\frac{\text{cost} \times \text{rate} \times \text{month}}{12}$$
 - For the last period, DB uses this formula:
$$\frac{\text{cost} - \text{total depreciation from prior periods}}{\text{rate} \times (12 -$$

`\text{month}}){12}$$`

- period and month are rounded to the nearest integer.
- An error is returned if:
 - `cost < 0`.
 - `salvage < 0`.
 - `life < 1`.
 - `period < 1` or `period > life`.
 - `month < 1` or `month > 12`.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Examples

Example 1

The following DAX query:

```
EVALUATE
{
    DB(1000000, 0, 6, 1, 2)
}
```

Returns an asset's depreciation in the last two months of the first year, assuming it will be worth `\$0` after 6 years.

| [VALUE] |
|------------------|
| 166666.666666667 |

Example 2

The following calculates the total depreciation of all assets in different years over their lifetimes. Here, the first year only includes 7 months of depreciation, and the last year only includes 5 months.

```
DEFINE
VAR NumDepreciationPeriods = MAX(Asset[LifetimeYears])+1
VAR DepreciationPeriods = GENERATESERIES(1, NumDepreciationPeriods)
EVALUATE
ADDCOLUMNS (
    DepreciationPeriods,
    "Current Period Total Depreciation",
    SUMX (
        FILTER (
            Asset,
            [Value] <= [LifetimeYears]+1
        ),
        DB([InitialCost], [SalvageValue], [LifetimeYears], [Value], 7)
    )
)
```


DDB

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the depreciation of an asset for a specified period using the double-declining balance method or some other method you specify.

Syntax

```
DDB(<cost>, <salvage>, <life>, <period>[, <factor>])
```

Parameters

| TERM | DEFINITION |
|---------|--|
| cost | The initial cost of the asset. |
| salvage | The value at the end of the depreciation (sometimes called the salvage value of the asset). This value can be 0. |
| life | The number of periods over which the asset is being depreciated (sometimes called the useful life of the asset). |
| period | The period for which you want to calculate the depreciation. Period must use the same units as life. Must be between 1 and life (inclusive). |
| factor | (Optional) The rate at which the balance declines. If factor is omitted, it is assumed to be 2 (the double-declining balance method). |

Return Value

The depreciation over the specified period.

Remarks

- The double-declining balance method computes depreciation at an accelerated rate. Depreciation is highest in the first period and decreases in successive periods. DDB uses the following formula to calculate depreciation for a period:

$$\text{Min}((\text{cost} - \text{total depreciation from prior periods}) \times \frac{\text{factor}}{\text{life}}), (\text{cost} - \text{salvage} - \text{total depreciation from prior periods}))$$

- Change factor if you do not want to use the double-declining balance method.
- Use the VDB function if you want to switch to the straight-line depreciation method when depreciation is greater than the declining balance calculation.
- period is rounded to the nearest integer.
- An error is returned if:

- $\text{cost} < 0$.
- $\text{salvage} < 0$.
- $\text{life} < 1$.
- $\text{period} < 1$ or $\text{period} > \text{life}$.
- $\text{factor} \leq 0$.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Examples

Example 1

The following DAX query:

```
EVALUATE
{
    DDB(1000000, 0, 10, 5, 1.5)
}
```

Returns an asset's depreciation in the 5th year, assuming it will be worth \$0 after 10 years. This calculation uses a factor of 1.5.

[VALUE]

78300.9375

Example 2

The following calculates the total depreciation of all assets in different years over their lifetimes. This calculation uses the default factor of 2 (the double-declining balance method).

```
DEFINE
VAR NumDepreciationPeriods = MAX(Asset[LifetimeYears])
VAR DepreciationPeriods = GENERATESERIES(1, NumDepreciationPeriods)
EVALUATE
    ADDCOLUMNS (
        DepreciationPeriods,
        "Current Period Total Depreciation",
        SUMX (
            FILTER (
                Asset,
                [Value] <= [LifetimeYears]
            ),
            DDB([InitialCost], [SalvageValue], [LifetimeYears], [Value])
        )
    )
```

DISC

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the discount rate for a security.

Syntax

```
DISC(<settlement>, <maturity>, <pr>, <redemption>[, <basis>])
```

Parameters

| TERM | DEFINITION |
|------------|---|
| settlement | The security's settlement date. The security settlement date is the date after the issue date when the security is traded to the buyer. |
| maturity | The security's maturity date. The maturity date is the date when the security expires. |
| pr | The security's price per \ \$100 face value. |
| redemption | The security's redemption value per \ \$100 face value. |
| basis | (Optional) The type of day count basis to use. If basis is omitted, it is assumed to be 0. The accepted values are listed below this table. |

The **basis** parameter accepts the following values:

| BASIS | DAY COUNT BASIS |
|--------------|------------------|
| 0 or omitted | US (NASD) 30/360 |
| 1 | Actual/actual |
| 2 | Actual/360 |
| 3 | Actual/365 |
| 4 | European 30/360 |

Return Value

The discount rate.

Remarks

- Dates are stored as sequential serial numbers so they can be used in calculations. In DAX, December 30, 1899 is day 0, and January 1, 2008 is 39448 because it is 39,448 days after December 30, 1899.

- The settlement date is the date a buyer purchases a coupon, such as a bond. The maturity date is the date when a coupon expires. For example, suppose a 30-year bond is issued on January 1, 2018, and is purchased by a buyer six months later. The issue date would be January 1, 2018, the settlement date would be July 1, 2018, and the maturity date would be January 1, 2048, 30 years after the January 1, 2018, issue date.

- DISC is calculated as follows:

$$\text{DISC} = \frac{\text{redemption} - \text{par}}{\text{redemption}} \times \frac{\text{B}}{\text{DSM}}$$

where:

- B = number of days in a year, depending on the year basis.
 - DSM = number of days between settlement and maturity.
- settlement and maturity are truncated to integers.
- basis is rounded to the nearest integer.
- An error is returned if:
 - settlement or maturity is not a valid date.
 - settlement \geq maturity.
 - $\text{pr} \leq 0$.
 - redemption ≤ 0 .
 - basis < 0 or basis > 4 .
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

| DATA | DESCRIPTION |
|------------|---------------------------------|
| 07/01/2018 | Settlement date |
| 01/01/2048 | Maturity date |
| 97.975 | Price |
| 100 | Redemption value |
| 1 | Actual/actual basis (see above) |

The following DAX query:

```
EVALUATE
{
    DISC(DATE(2018,7,1), DATE(2048,1,1), 97.975, 100, 1)
}
```

Returns the bond discount rate, for a bond with the terms specified above.

[VALUE]

0.000686384169121348

DOLLARDE

10/31/2022 • 2 minutes to read • [Edit Online](#)

Converts a dollar price expressed as an integer part and a fraction part, such as 1.02, into a dollar price expressed as a decimal number. Fractional dollar numbers are sometimes used for security prices.

Syntax

```
DOLLARDE(<fractional_dollar>, <fraction>)
```

Parameters

| TERM | DEFINITION |
|-------------------|---|
| fractional_dollar | A number expressed as an integer part and a fraction part, separated by a decimal symbol. |
| fraction | The integer to use in the denominator of the fraction. |

Return Value

The decimal value of *fractional_dollar*.

Remarks

- The fraction part of the value is divided by an integer that you specify. For example, if you want your price to be expressed to a precision of 1/16 of a dollar, you divide the fraction part by 16. In this case, 1.02 represents $\$1.125$ ($\$1 + 2/16 = \1.125).
- fraction is rounded to the nearest integer.
- An error is returned if:
 - fraction < 1.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following DAX query:

```
EVALUATE
{
    DOLLARDE(1.02, 16)
}
```

Returns 1.125, the decimal price of the original fractional price, 1.02, read as 1 and 2/16. Since the fraction value is 16, the price has a precision of 1/16 of a dollar.

DOLLARFR

10/31/2022 • 2 minutes to read • [Edit Online](#)

Converts a dollar price expressed as a decimal number into a dollar price expressed as an integer part and a fraction part, such as 1.02. Fractional dollar numbers are sometimes used for security prices.

Syntax

```
DOLLARFR(<decimal_dollar>, <fraction>)
```

Parameters

| TERM | DEFINITION |
|----------------|--|
| decimal_dollar | A decimal number. |
| fraction | The integer to use in the denominator of the fraction. |

Return Value

The fractional value of *decimal_dollar*, expressed as an integer part and a fraction part.

Remarks

- fraction is rounded to the nearest integer.
- An error is returned if:
 - fraction < 1.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following DAX query:

```
EVALUATE
{
    DOLLARFR(1.125, 16)
}
```

Returns 1.02, read as 1 and 2/16, which is the corresponding fraction price of the original decimal price, 1.125. Since the fraction value is 16, the price has a precision of 1/16 of a dollar.

DURATION

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the Macauley duration for an assumed par value of \ \$100. Duration is defined as the weighted average of the present value of cash flows, and is used as a measure of a bond price's response to changes in yield.

Syntax

```
DURATION(<settlement>, <maturity>, <coupon>, <yld>, <frequency>[, <basis>])
```

Parameters

| TERM | DEFINITION |
|------------|---|
| settlement | The security's settlement date. The security settlement date is the date after the issue date when the security is traded to the buyer. |
| maturity | The security's maturity date. The maturity date is the date when the security expires. |
| coupon | The security's annual coupon rate. |
| yld | The security's annual yield. |
| frequency | The number of coupon payments per year. For annual payments, frequency = 1; for semiannual, frequency = 2; for quarterly, frequency = 4. |
| basis | (Optional) The type of day count basis to use. If basis is omitted, it is assumed to be 0. The accepted values are listed below this table. |

The **basis** parameter accepts the following values:

| BASIS | DAY COUNT BASIS |
|--------------|------------------|
| 0 or omitted | US (NASD) 30/360 |
| 1 | Actual/actual |
| 2 | Actual/360 |
| 3 | Actual/365 |
| 4 | European 30/360 |

Return Value

The Macauley duration.

Remarks

- Dates are stored as sequential serial numbers so they can be used in calculations. In DAX, December 30, 1899 is day 0, and January 1, 2008 is 39448 because it is 39,448 days after December 30, 1899.
- The settlement date is the date a buyer purchases a coupon, such as a bond. The maturity date is the date when a coupon expires. For example, suppose a 30-year bond is issued on January 1, 2008, and is purchased by a buyer six months later. The issue date would be January 1, 2008, the settlement date would be July 1, 2008, and the maturity date is January 1, 2038, which is 30 years after the January 1, 2008, issue date.
- settlement and maturity are truncated to integers.
- frequency, and basis are rounded to the nearest integer.
- An error is returned if:
 - settlement or maturity is not a valid date.
 - settlement \geq maturity.
 - coupon < 0 .
 - yld < 0
 - frequency is any number other than 1, 2, or 4.
 - basis < 0 or basis > 4 .
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

| DATA | DESCRIPTION |
|------------|-------------------------------------|
| 07/01/2018 | Settlement date |
| 01/01/2048 | Maturity date |
| 8.0% | Percent coupon |
| 9.0% | Percent yield |
| 2 | Frequency is semiannual (see above) |
| 1 | Actual/actual basis (see above) |

The following DAX query:

```
EVALUATE
{
    DURATION(DATE(2018,7,1), DATE(2048,1,1), 0.08, 0.09, 2, 1)
}
```

Returns the Macauley duration for a bond with the terms specified above.

| [VALUE] |
|------------------|
| 10.9191452815919 |

EFFECT

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the effective annual interest rate, given the nominal annual interest rate and the number of compounding periods per year.

Syntax

```
EFFECT(<nominal_rate>, <npery>)
```

Parameters

| TERM | DEFINITION |
|--------------|---|
| nominal_rate | The nominal interest rate. |
| npery | The number of compounding periods per year. |

Return Value

The effective annual interest rate.

Remarks

- EFFECT is calculated as follows:

$$\text{EFFECT} = \left(1 + \frac{\text{nominal_rate}}{\text{npery}} \right)^{\text{npery}} - 1$$

- npery is rounded to the nearest integer.
- An error is returned if:
 - nominal_rate ≤ 0 .
 - npery < 1 .
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

| DATA | DESCRIPTION |
|-------|--|
| 5.25% | Nominal interest rate |
| 4 | Number of compounding periods per year |

The following DAX query:

```
EVALUATE
{
  EFFECT(0.0525, 4)
}
```

Returns the effective interest rate using the terms specified above.

[VALUE]

0.0535426673707584

FV

10/31/2022 • 2 minutes to read • [Edit Online](#)

Calculates the future value of an investment based on a constant interest rate. You can use FV with either periodic, constant payments, and/or a single lump sum payment.

Syntax

```
FV(<rate>, <nper>, <pmt>[, <pv>[, <type>]])
```

Parameters

| TERM | DEFINITION |
|------|---|
| rate | The interest rate per period. |
| nper | The total number of payment periods in an annuity. |
| pmt | The payment made each period; it cannot change over the life of the annuity. Typically, pmt contains principal and interest but no other fees or taxes. |
| pv | (Optional) The present value, or the lump-sum amount that a series of future payments is worth right now. If pv is omitted, it is assumed to be BLANK. |
| type | (Optional) The number 0 or 1 which indicates when payments are due. If type is omitted, it is assumed to be 0. The accepted values are listed below this table. |

The **type** parameter accepts the following values:

| SET TYPE EQUAL TO | IF PAYMENTS ARE DUE |
|-------------------|--------------------------------|
| 0 or omitted | At the end of the period |
| 1 | At the beginning of the period |

Note: For a more complete description of the arguments in FV and for more information on annuity functions, see the PV function.

Return Value

The future value of an investment.

Remarks

- Make sure that you are consistent about the units you use for specifying rate and nper. If you make monthly payments on a four-year loan at 12 percent annual interest, use 0.12/12 for rate and 4*12 for nper. If you make annual payments on the same loan, use 0.12 for rate and 4 for nper.

- For all the arguments, cash you pay out, such as deposits to savings, is represented by negative numbers; cash you receive, such as dividend checks, is represented by positive numbers.
- type is rounded to the nearest integer.
- An error is returned if:
 - nper < 1
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

| DATA | DESCRIPTION |
|------|---|
| 6% | Annual interest rate |
| 10 | Number of payments |
| -200 | Amount of the payment |
| -500 | Present value |
| 1 | Payment is due at the beginning of the period (0 indicates payment is due at end of period) |

The following DAX query:

```
EVALUATE
{
    FV(0.06/12, 10, -200, -500, 1)
}
```

Returns the future value of an investment using the terms specified above.

| [VALUE] |
|------------------|
| 2581.40337406012 |

INTRATE

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the interest rate for a fully invested security.

Syntax

```
INTRATE(<settlement>, <maturity>, <investment>, <redemption>[, <basis>])
```

Parameters

| TERM | DEFINITION |
|------------|---|
| settlement | The security's settlement date. The security settlement date is the date after the issue date when the security is traded to the buyer. |
| maturity | The security's maturity date. The maturity date is the date when the security expires. |
| investment | The amount invested in the security. |
| redemption | The amount to be received at maturity. |
| basis | (Optional) The type of day count basis to use. If basis is omitted, it is assumed to be 0. The accepted values are listed below this table. |

The **basis** parameter accepts the following values:

| BASIS | DAY COUNT BASIS |
|--------------|------------------|
| 0 or omitted | US (NASD) 30/360 |
| 1 | Actual/actual |
| 2 | Actual/360 |
| 3 | Actual/365 |
| 4 | European 30/360 |

Return Value

The interest rate.

Remarks

- Dates are stored as sequential serial numbers so they can be used in calculations. In DAX, December 30, 1899 is day 0, and January 1, 2008 is 39448 because it is 39,448 days after December 30, 1899.

- The settlement date is the date a buyer purchases a coupon, such as a bond. The maturity date is the date when a coupon expires. For example, suppose a 30-year bond is issued on January 1, 2008, and is purchased by a buyer six months later. The issue date would be January 1, 2008, the settlement date would be July 1, 2008, and the maturity date is January 1, 2038, which is 30 years after the January 1, 2008, issue date.

- INTRATE is calculated as follows:

$$\text{INTRATE} = \frac{\text{redemption} - \text{investment}}{\text{investment}} \times \frac{\text{B}}{\text{DIM}}$$

where:

- B = number of days in a year, depending on the year basis.
 - DIM = number of days from settlement to maturity.
- settlement and maturity are truncated to integers.
- basis is rounded to the nearest integer.
- An error is returned if:
 - settlement or maturity is not a valid date.
 - settlement \geq maturity.
 - investment \leq 0.
 - redemption \leq 0.
 - basis < 0 or basis > 4.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

| DATA | DESCRIPTION |
|--------------|------------------|
| 2/15/2008 | Settlement date |
| 5/15/2008 | Maturity date |
| \\$1,000,000 | Investment |
| \\$1,014,420 | Redemption value |
| 2 | Actual/360 basis |

The following DAX query:

```
EVALUATE
{
    INTRATE(DATE(2008,2,15), DATE(2008,5,15), 1000000, 1014420, 2)
}
```

Returns the discount rate for a bond using the terms specified above.

| |
|---------|
| [VALUE] |
| 0.05768 |

IPMT

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the interest payment for a given period for an investment based on periodic, constant payments and a constant interest rate.

Syntax

```
IPMT(<rate>, <per>, <nper>, <pv>[, <fv>[, <type>]])
```

Parameters

| TERM | DEFINITION |
|------|---|
| rate | The interest rate per period. |
| per | The period for which you want to find the interest. Must be between 1 and nper (inclusive). |
| nper | The total number of payment periods in an annuity. |
| pv | The present value, or the lump-sum amount that a series of future payments is worth right now. |
| fv | (Optional) The future value, or a cash balance you want to attain after the last payment is made. If fv is omitted, it is assumed to be BLANK. |
| type | (Optional) The number 0 or 1 which indicates when payments are due. If type is omitted, it is assumed to be 0. The accepted values are listed below this table. |

The **type** parameter accepts the following values:

| SET TYPE EQUAL TO | IF PAYMENTS ARE DUE |
|-------------------|--------------------------------|
| 0 or omitted | At the end of the period |
| 1 | At the beginning of the period |

Return Value

The interest payment for the given period.

Remarks

- Make sure that you are consistent about the units you use for specifying rate and nper. If you make monthly payments on a four-year loan at 12 percent annual interest, use 0.12/12 for rate and 4*12 for nper. If you make annual payments on the same loan, use 0.12 for rate and 4 for nper.

- For all the arguments, cash you pay out, such as deposits to savings, is represented by negative numbers; cash you receive, such as dividend checks, is represented by positive numbers.
- type is rounded to the nearest integer.
- An error is returned if:
 - $\text{per} < 1$ or $\text{per} > \text{nper}$
 - $\text{nper} < 1$
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Examples

| DATA | DESCRIPTION |
|----------|-----------------------|
| 10.00% | Annual interest |
| 3 | Years of loan |
| \\$8,000 | Present value of loan |

Example 1

The following DAX query:

```
EVALUATE
{
    IPMT(0.1/12, 1, 3*12, 8000)
}
```

Returns the monthly interest due in the first month for a loan with the terms specified above.

| [VALUE] |
|-------------------|
| -66.6666666666667 |

Example 2

The following DAX query:

```
EVALUATE
{
    IPMT(0.1, 3, 3, 8000)
}
```

Returns the yearly interest due in the last year for a loan with the terms specified above, where payments are made yearly.

| [VALUE] |
|-------------------|
| -292.447129909366 |

ISPMT

10/31/2022 • 2 minutes to read • [Edit Online](#)

Calculates the interest paid (or received) for the specified period of a loan (or investment) with even principal payments.

Syntax

```
ISPMT(<rate>, <per>, <nper>, <pv>)
```

Parameters

| TERM | DEFINITION |
|------|---|
| rate | The interest rate for the investment. |
| per | The period for which you want to find the interest. Must be between 0 and nper-1 (inclusive). |
| nper | The total number of payment periods for the investment. |
| pv | The present value of the investment. For a loan, pv is the loan amount. |

Return Value

The interest paid (or received) for the specified period.

Remarks

- Make sure that you are consistent about the units you use for specifying rate and nper. If you make monthly payments on a four-year loan at an annual interest rate of 12 percent, use 0.12/12 for rate and 4*12 for nper. If you make annual payments on the same loan, use 0.12 for rate and 4 for nper.
- For all the arguments, the cash you pay out, such as deposits to savings or other withdrawals, is represented by negative numbers; the cash you receive, such as dividend checks and other deposits, is represented by positive numbers.
- ISPMT counts each period beginning with zero, not one.
- Most loans use a repayment schedule with even periodic payments. The IPMT function returns the interest payment for a given period for this type of loan.
- Some loans use a repayment schedule with even principal payments. The ISPMT function returns the interest payment for a given period for this type of loan.
- An error is returned if:
 - nper = 0.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

| DATA | DESCRIPTION |
|----------|-------------------|
| \\$4,000 | Present value |
| 4 | Number of periods |
| 10% | Rate |

To illustrate when to use ISPMT, the amortization table below uses an even-principal repayment schedule with the terms specified above. The interest charge each period is equal to the rate times the unpaid balance for the previous period. And the payment each period is equal to the even principal plus the interest for the period.

| PERIOD | PRINCIPAL PAYMENT | INTEREST PAYMENT | TOTAL PAYMENT | BALANCE |
|--------|-------------------|------------------|---------------|----------|
| | | | | 4,000.00 |
| 1 | 1,000.00 | 400.00 | 1,400.00 | 3,000.00 |
| 2 | 1,000.00 | 300.00 | 1,300.00 | 2,000.00 |
| 3 | 1,000.00 | 200.00 | 1,200.00 | 1,000.00 |
| 4 | 1,000.00 | 100.00 | 1,100.00 | 0.00 |

The following DAX query:

```
DEFINE
VAR NumPaymentPeriods = 4
VAR PaymentPeriods = GENERATESERIES(0, NumPaymentPeriods-1)
EVALUATE
ADDCOLUMNS (
    PaymentPeriods,
    "Interest Payment",
    ISPMT(0.1, [Value], NumPaymentPeriods, 4000)
)
```

Returns the interest paid during each period, using the even-principal repayment schedule and terms specified above. The values are negative to indicate that it is interest paid, not received.

| [VALUE] | [INTEREST PAYMENT] |
|---------|--------------------|
| 0 | -400 |
| 1 | -300 |
| 2 | -200 |
| 3 | -100 |

MDURATION

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the modified Macauley duration for a security with an assumed par value of \\$.100.

Syntax

```
MDURATION(<settlement>, <maturity>, <coupon>, <yld>, <frequency>[, <basis>])
```

Parameters

| TERM | DEFINITION |
|------------|---|
| settlement | The security's settlement date. The security settlement date is the date after the issue date when the security is traded to the buyer. |
| maturity | The security's maturity date. The maturity date is the date when the security expires. |
| coupon | The security's annual coupon rate. |
| yld | The security's annual yield. |
| frequency | The number of coupon payments per year. For annual payments, frequency = 1; for semiannual, frequency = 2; for quarterly, frequency = 4. |
| basis | (Optional) The type of day count basis to use. If basis is omitted, it is assumed to be 0. The accepted values are listed below this table. |

The **basis** parameter accepts the following values:

| BASIS | DAY COUNT BASIS |
|--------------|------------------|
| 0 or omitted | US (NASD) 30/360 |
| 1 | Actual/actual |
| 2 | Actual/360 |
| 3 | Actual/365 |
| 4 | European 30/360 |

Return Value

The modified Macauley duration.

Remarks

- Dates are stored as sequential serial numbers so they can be used in calculations. In DAX, December 30, 1899 is day 0, and January 1, 2008 is 39448 because it is 39,448 days after December 30, 1899.
- The settlement date is the date a buyer purchases a coupon, such as a bond. The maturity date is the date when a coupon expires. For example, suppose a 30-year bond is issued on January 1, 2008, and is purchased by a buyer six months later. The issue date would be January 1, 2008, the settlement date would be July 1, 2008, and the maturity date is January 1, 2038, which is 30 years after the January 1, 2008, issue date.
- Modified duration is defined as follows:

$$\text{MDURATION} = \frac{\text{DURATION}}{1 + (\frac{\text{Market yield}}{\text{Coupon payments per year}})}$$

- settlement and maturity are truncated to integers.
- frequency, and basis are rounded to the nearest integer.
- An error is returned if:
 - settlement or maturity is not a valid date.
 - settlement \geq maturity.
 - coupon < 0 .
 - yld < 0
 - frequency is any number other than 1, 2, or 4.
 - basis < 0 or basis > 4 .
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

| DATA | DESCRIPTION |
|----------|-------------------------------------|
| 1/1/2008 | Settlement date |
| 1/1/2016 | Maturity date |
| 8% | Percent coupon |
| 9% | Percent yield |
| 2 | Frequency is semiannual (see above) |
| 1 | Actual/actual basis (see above) |

The following DAX query:

```
EVALUATE
{
    MDURATION(DATE(2008,1,1), DATE(2016,1,1), 0.08, 0.09, 2, 1)
}
```

Returns the modified Macauley duration of a bond using the terms specified above.

| |
|------------------|
| [VALUE] |
| 5.73566981391884 |

NOMINAL

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the nominal annual interest rate, given the effective rate and the number of compounding periods per year.

Syntax

```
NOMINAL(<effect_rate>, <npery>)
```

Parameters

| TERM | DEFINITION |
|-------------|---|
| effect_rate | The effective interest rate. |
| npery | The number of compounding periods per year. |

Return Value

The nominal annual interest rate.

Remarks

- The relationship between NOMINAL and EFFECT is shown in the following equation:
$$\text{EFFECT} = \left(1 + \frac{\text{nominal_rate}}{\text{npery}} \right)^{\text{npery}} - 1$$
- npery is rounded to the nearest integer.
- An error is returned if:
 - effect_rate \leq 0.
 - npery < 1.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

| DATA | DESCRIPTION |
|---------|--|
| 5.3543% | Effective interest rate |
| 4 | Number of compounding periods per year |

The following DAX query:


```
EVALUATE
{
  NOMINAL(0.053543, 4)
}
```

Returns the nominal interest rate, using the terms specified above.

[VALUE]

0.052500319868356

NPER

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the number of periods for an investment based on periodic, constant payments and a constant interest rate.

Syntax

```
NPER(<rate>, <pmt>, <pv>[, <fv>[, <type>]])
```

Parameters

| TERM | DEFINITION |
|------|---|
| rate | The interest rate per period. |
| pmt | The payment made each period; it cannot change over the life of the annuity. Typically, pmt contains principal and interest but no other fees or taxes. |
| pv | The present value, or the lump-sum amount that a series of future payments is worth right now. |
| fv | (Optional) The future value, or a cash balance you want to attain after the last payment is made. If fv is omitted, it is assumed to be BLANK. |
| type | (Optional) The number 0 or 1 and indicates when payments are due. If type is omitted, it is assumed to be 0. The accepted values are listed below this table. |

The **type** parameter accepts the following values:

| SET TYPE EQUAL TO | IF PAYMENTS ARE DUE |
|-------------------|--------------------------------|
| 0 or omitted | At the end of the period |
| 1 | At the beginning of the period |

Return Value

The number of periods for an investment.

Remarks

- type is rounded to the nearest integer.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

| DATA | DESCRIPTION |
|-------|---|
| 12% | Annual interest rate |
| -100 | Payment made each period |
| -1000 | Present value |
| 10000 | Future value |
| 1 | Payment is due at the beginning of the period (see above) |

The following DAX query:

```
EVALUATE
{
    NPER(0.12/12, -100, -1000, 10000, 1)
}
```

Returns the number of periods for the investment described by the terms specified above.

| [VALUE] |
|------------------|
| 59.6738656742946 |

ODDFPRICE

10/31/2022 • 4 minutes to read • [Edit Online](#)

Returns the price per \$100 face value of a security having an odd (short or long) first period.

Syntax

```
ODDFPRICE(<settlement>, <maturity>, <issue>, <first_coupon>, <rate>, <yld>, <redemption>, <frequency>[,  
<basis>])
```

Parameters

| TERM | DEFINITION |
|--------------|---|
| settlement | The security's settlement date. The security settlement date is the date after the issue date when the security is traded to the buyer. |
| maturity | The security's maturity date. The maturity date is the date when the security expires. |
| issue | The security's issue date. |
| first_coupon | The security's first coupon date. |
| rate | The security's interest rate. |
| yld | The security's annual yield. |
| redemption | The security's redemption value per \$100 face value. |
| frequency | The number of coupon payments per year. For annual payments, frequency = 1; for semiannual, frequency = 2; for quarterly, frequency = 4. |
| basis | (Optional) The type of day count basis to use. If basis is omitted, it is assumed to be 0. The accepted values are listed below this table. |

The **basis** parameter accepts the following values:

| BASIS | DAY COUNT BASIS |
|--------------|------------------|
| 0 or omitted | US (NASD) 30/360 |
| 1 | Actual/actual |
| 2 | Actual/360 |
| 3 | Actual/365 |

| BASIS | DAY COUNT BASIS |
|-------|-----------------|
| 4 | European 30/360 |

Return Value

The price per \\$100 face value.

Remarks

- Dates are stored as sequential serial numbers so they can be used in calculations. In DAX, December 30, 1899 is day 0, and January 1, 2008 is 39448 because it is 39,448 days after December 30, 1899.
- The settlement date is the date a buyer purchases a coupon, such as a bond. The maturity date is the date when a coupon expires. For example, suppose a 30-year bond is issued on January 1, 2008, and is purchased by a buyer six months later. The issue date would be January 1, 2008, the settlement date would be July 1, 2008, and the maturity date would be January 1, 2038, which is 30 years after the January 1, 2008, issue date.
- ODDFPRICE is calculated as follows:

Odd short first coupon:

$$\text{ODDFPRICE} = \frac{\text{redemption}}{(1 + \frac{\text{yld}}{\text{frequency}})^{(N - 1 + \frac{\text{DSC}}{\text{E}})}} + \frac{100 \times \frac{\text{rate}}{\text{frequency}} \times \frac{\text{DFC}}{\text{E}}}{(1 + \frac{\text{yld}}{\text{frequency}})^{(\frac{\text{DSC}}{\text{E}})}} + \sum_{k=2}^N \frac{100 \times \frac{\text{rate}}{\text{frequency}}}{(1 + \frac{\text{yld}}{\text{frequency}})^{(k - 1 + \frac{\text{DSC}}{\text{E}})}} - \frac{100 \times \frac{\text{rate}}{\text{frequency}} \times \frac{\text{A}}{\text{E}}}{\text{Big}}$$

where:

- A = number of days from the beginning of the coupon period to the settlement date (accrued days).
- DSC = number of days from the settlement to the next coupon date.
- DFC = number of days from the beginning of the odd first coupon to the first coupon date.
- E = number of days in the coupon period.
- N = number of coupons payable between the settlement date and the redemption date. (If this number contains a fraction, it is raised to the next whole number.)

Odd long first coupon:

$$\text{ODDFPRICE} = \frac{\text{redemption}}{(1 + \frac{\text{yld}}{\text{frequency}})^{(\text{N} + \text{N}_q + \frac{\text{DSC}}{\text{E}})}} + \frac{100 \times \frac{\text{rate}}{\text{frequency}} \times \text{Big} \times \sum_{i=1}^{\text{NC}} \frac{\text{DC}_i}{\text{NL}_i}}{(1 + \frac{\text{yld}}{\text{frequency}})^{(\text{N}_q + \frac{\text{DSC}}{\text{E}})}} + \sum_{k=1}^{\text{N}} \frac{100 \times \frac{\text{rate}}{\text{frequency}}}{(1 + \frac{\text{yld}}{\text{frequency}})^{(k - \text{N}_q + \frac{\text{DSC}}{\text{E}})}} - \frac{100 \times \frac{\text{rate}}{\text{frequency}} \times \sum_{i=1}^{\text{NC}} \frac{\text{A}_i}{\text{NL}_i}}{\text{Big}}$$

where:

- A_i = number of days from the beginning of the i^{th} , or last, quasi-coupon period within odd period.
- DC_i = number of days from dated date (or issue date) to first quasi-coupon ($i = 1$) or number of days in quasi-coupon ($i = 2, \dots, i = \text{NC}$).

- DSC = number of days from settlement to next coupon date.
- E = number of days in coupon period.
- N = number of coupons payable between the first real coupon date and redemption date. (If this number contains a fraction, it is raised to the next whole number.)
- NC = number of quasi-coupon periods that fit in odd period. (If this number contains a fraction, it is raised to the next whole number.)
- NL_i = normal length in days of the full i^{th} , or last, quasi-coupon period within odd period.
- N_q = number of whole quasi-coupon periods between settlement date and first coupon.
- settlement, maturity, issue, and first_coupon are truncated to integers.
- basis and frequency are rounded to the nearest integer.
- An error is returned if:
 - settlement, maturity, issue, or first_coupon is not a valid date.
 - maturity > first_coupon > settlement > issue is not satisfied.
 - rate < 0.
 - yld < 0.
 - redemption ≤ 0.
 - frequency is any number other than 1, 2, or 4.
 - basis < 0 or basis > 4.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

| DATA | ARGUMENT DESCRIPTION |
|------------|-------------------------|
| 11/11/2008 | Settlement date |
| 3/1/2021 | Maturity date |
| 10/15/2008 | Issue date |
| 3/1/2009 | First coupon date |
| 7.85% | Percent coupon |
| 6.25% | Percent yield |
| \\$100.00 | Redemptive value |
| 2 | Frequency is semiannual |
| 1 | Actual/actual basis |

The following DAX query:

EVALUATE

```
{  
  ODDFPRICE(DATE(2008,11,11), DATE(2021,3,1), DATE(2008,10,15), DATE(2009,3,1), 0.0785, 0.0625, 100.00, 2,  
  1)  
}
```

Returns the price per \$100 face value of a security having an odd (short or long) first period, using the terms specified above.

[VALUE]

113.597717474079

ODDFYIELD

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the yield of a security that has an odd (short or long) first period.

Syntax

```
ODDFYIELD(<settlement>, <maturity>, <issue>, <first_coupon>, <rate>, <pr>, <redemption>, <frequency>[,  
<basis>])
```

Parameters

| TERM | DEFINITION |
|--------------|---|
| settlement | The security's settlement date. The security settlement date is the date after the issue date when the security is traded to the buyer. |
| maturity | The security's maturity date. The maturity date is the date when the security expires. |
| issue | The security's issue date. |
| first_coupon | The security's first coupon date. |
| rate | The security's interest rate. |
| pr | The security's price. |
| redemption | The security's redemption value per \$100 face value. |
| frequency | The number of coupon payments per year. For annual payments, frequency = 1; for semiannual, frequency = 2; for quarterly, frequency = 4. |
| basis | (Optional) The type of day count basis to use. If basis is omitted, it is assumed to be 0. The accepted values are listed below this table. |

The **basis** parameter accepts the following values:

| BASIS | DAY COUNT BASIS |
|--------------|------------------|
| 0 or omitted | US (NASD) 30/360 |
| 1 | Actual/actual |
| 2 | Actual/360 |
| 3 | Actual/365 |

| BASIS | DAY COUNT BASIS |
|-------|-----------------|
| 4 | European 30/360 |

Return Value

The security's yield.

Remarks

- Dates are stored as sequential serial numbers so they can be used in calculations. In DAX, December 30, 1899 is day 0, and January 1, 2008 is 39448 because it is 39,448 days after December 30, 1899.
- The settlement date is the date a buyer purchases a coupon, such as a bond. The maturity date is the date when a coupon expires. For example, suppose a 30-year bond is issued on January 1, 2008, and is purchased by a buyer six months later. The issue date would be January 1, 2008, the settlement date would be July 1, 2008, and the maturity date would be January 1, 2038, which is 30 years after the January 1, 2008, issue date.
- ODDFYIELD is calculated using an iterative method. It uses the Newton method based on the formula used for the function ODDFPRICE. The yield is changed through 100 iterations until the estimated price with the given yield is close to the price. See ODDFPRICE for the formula that ODDFYIELD uses.
- settlement, maturity, issue, and first_coupon are truncated to integers.
- basis and frequency are rounded to the nearest integer.
- An error is returned if:
 - settlement, maturity, issue, or first_coupon is not a valid date.
 - maturity > first_coupon > settlement > issue is not satisfied.
 - rate < 0.
 - pr ≤ 0.
 - redemption ≤ 0.
 - frequency is any number other than 1, 2, or 4.
 - basis < 0 or basis > 4.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

| DATA | ARGUMENT DESCRIPTION |
|-------------------|----------------------|
| November 11, 2008 | Settlement date |
| March 1, 2021 | Maturity date |
| October 15, 2008 | Issue date |
| March 1, 2009 | First coupon date |
| 5.75% | Percent coupon |

| DATA | ARGUMENT DESCRIPTION |
|-------|-------------------------|
| 84.50 | Price |
| 100 | Redemptive value |
| 2 | Frequency is semiannual |
| 0 | 30/360 basis |

The following DAX query:

```
EVALUATE
{
    ODDFYIELD(
        DATE(2008,11,11),
        DATE(2021,3,1),
        DATE(2008,10,15),
        DATE(2009,3,1),
        0.0575,
        84.50,
        100,
        2,
        0
    )
}
```

Returns the yield of a security that has an odd (short or long) first period, using the terms specified above.

| [VALUE] |
|--------------------|
| 0.0772455415972989 |

ODDLPRICE

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the price per \$100 face value of a security having an odd (short or long) last coupon period.

Syntax

```
ODDLPRICE(<settlement>, <maturity>, <last_interest>, <rate>, <yld>, <redemption>, <frequency>[, <basis>])
```

Parameters

| TERM | DEFINITION |
|---------------|---|
| settlement | The security's settlement date. The security settlement date is the date after the issue date when the security is traded to the buyer. |
| maturity | The security's maturity date. The maturity date is the date when the security expires. |
| last_interest | The security's last coupon date. |
| rate | The security's interest rate. |
| yld | The security's annual yield. |
| redemption | The security's redemption value per \$100 face value. |
| frequency | The number of coupon payments per year. For annual payments, frequency = 1; for semiannual, frequency = 2; for quarterly, frequency = 4. |
| basis | (Optional) The type of day count basis to use. If basis is omitted, it is assumed to be 0. The accepted values are listed below this table. |

The **basis** parameter accepts the following values:

| BASIS | DAY COUNT BASIS |
|--------------|------------------|
| 0 or omitted | US (NASD) 30/360 |
| 1 | Actual/actual |
| 2 | Actual/360 |
| 3 | Actual/365 |
| 4 | European 30/360 |

Return Value

The price per \ \$100 face value.

Remarks

- Dates are stored as sequential serial numbers so they can be used in calculations. In DAX, December 30, 1899 is day 0, and January 1, 2008 is 39448 because it is 39,448 days after December 30, 1899.
- The settlement date is the date a buyer purchases a coupon, such as a bond. The maturity date is the date when a coupon expires. For example, suppose a 30-year bond is issued on January 1, 2008, and is purchased by a buyer six months later. The issue date would be January 1, 2008, the settlement date would be July 1, 2008, and the maturity date would be January 1, 2038, which is 30 years after the January 1, 2008, issue date.
- settlement, maturity, and last_interest are truncated to integers.
- basis and frequency are rounded to the nearest integer.
- An error is returned if:
 - settlement, maturity, or last_interest is not a valid date.
 - maturity > settlement > last_interest is not satisfied.
 - rate < 0.
 - yld < 0.
 - redemption ≤ 0.
 - frequency is any number other than 1, 2, or 4.
 - basis < 0 or basis > 4.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following DAX query:

| DATA | ARGUMENT DESCRIPTION |
|------------------|-------------------------|
| February 7, 2008 | Settlement date |
| June 15, 2008 | Maturity date |
| October 15, 2007 | Last interest date |
| 3.75% | Percent coupon |
| 4.05% | Percent yield |
| \ \$100 | Redemptive value |
| 2 | Frequency is semiannual |
| 0 | 30/360 basis |

EVALUATE

```
{  
    ODDLPRICE(DATE(2008,2,7), DATE(2008,6,15), DATE(2007,10,15), 0.0375, 0.0405, 100, 2, 0)  
}
```

Returns the price per \$100 face value of a security that has an odd (short or long) last coupon period, using the terms specified above.

[VALUE]

99.8782860147213

ODDLYIELD

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the yield of a security that has an odd (short or long) last period.

Syntax

```
ODDLYIELD(<settlement>, <maturity>, <last_interest>, <rate>, <pr>, <redemption>, <frequency>[, <basis>])
```

Parameters

| TERM | DEFINITION |
|---------------|---|
| settlement | The security's settlement date. The security settlement date is the date after the issue date when the security is traded to the buyer. |
| maturity | The security's maturity date. The maturity date is the date when the security expires. |
| last_interest | The security's last coupon date. |
| rate | The security's interest rate. |
| pr | The security's price. |
| redemption | The security's redemption value per \$100 face value. |
| frequency | The number of coupon payments per year. For annual payments, frequency = 1; for semiannual, frequency = 2; for quarterly, frequency = 4. |
| basis | (Optional) The type of day count basis to use. If basis is omitted, it is assumed to be 0. The accepted values are listed below this table. |

The **basis** parameter accepts the following values:

| BASIS | DAY COUNT BASIS |
|--------------|------------------|
| 0 or omitted | US (NASD) 30/360 |
| 1 | Actual/actual |
| 2 | Actual/360 |
| 3 | Actual/365 |
| 4 | European 30/360 |

Return Value

The security's yield.

Remarks

- Dates are stored as sequential serial numbers so they can be used in calculations. In DAX, December 30, 1899 is day 0, and January 1, 2008 is 39448 because it is 39,448 days after December 30, 1899.
- The settlement date is the date a buyer purchases a coupon, such as a bond. The maturity date is the date when a coupon expires. For example, suppose a 30-year bond is issued on January 1, 2008, and is purchased by a buyer six months later. The issue date would be January 1, 2008, the settlement date would be July 1, 2008, and the maturity date would be January 1, 2038, which is 30 years after the January 1, 2008, issue date.
- ODDLYIELD is calculated as follows:

$$\text{ODDLYIELD} = \left[\frac{\text{redemption} + \left(\sum_{i=1}^{\text{NC}} \frac{\text{DC}_i}{\text{NL}_i} \right) \times \frac{100 \times \text{rate}}{\text{frequency}}} - \text{par} + \left(\sum_{i=1}^{\text{NC}} \frac{\text{A}_i}{\text{NL}_i} \right) \times \frac{100 \times \text{rate}}{\text{frequency}} \right] \times \left[\frac{\text{frequency}}{\text{par} + \left(\sum_{i=1}^{\text{NC}} \frac{\text{A}_i}{\text{NL}_i} \right) \times \frac{100 \times \text{rate}}{\text{frequency}}} \right] \times \left[\frac{\text{frequency}}{\sum_{i=1}^{\text{NC}} \frac{\text{DSC}_i}{\text{NL}_i}} \right]$$

where:

- A_i = number of accrued days for the i^{th} , or last, quasi-coupon period within odd period counting forward from last interest date before redemption.
- DC_i = number of days counted in the i^{th} , or last, quasi-coupon period as delimited by the length of the actual coupon period.
- NC = number of quasi-coupon periods that fit in odd period; if this number contains a fraction it will be raised to the next whole number.
- NL_i = normal length in days of the i^{th} , or last, quasi-coupon period within odd coupon period.
- settlement, maturity, last_interest are truncated to integers.
- basis and frequency are rounded to the nearest integer.
- An error is returned if:
 - settlement, maturity, last_interest is not a valid date.
 - maturity > settlement > last_interest is not satisfied.
 - rate < 0.
 - pr ≤ 0.
 - redemption ≤ 0.
 - frequency is any number other than 1, 2, or 4.
 - basis < 0 or basis > 4.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following DAX query:

| DATA | ARGUMENT DESCRIPTION |
|------------|-------------------------|
| 4/20/2008 | Settlement date |
| 6/15/2008 | Maturity date |
| 12/24/2007 | Last interest date |
| 3.75% | Percent coupon |
| \\$99.875 | Price |
| \\$100 | Redemption value |
| 2 | Frequency is semiannual |
| 0 | 30/360 basis |

```
EVALUATE
{
  ODDLYIELD(DATE(2008,4,20), DATE(2008,6,15), DATE(2007,12,24), 0.0375, 99.875, 100, 2, 0)
}
```

Returns the yield of a security that has an odd (short of long) last period, using the terms specified above.

| [VALUE] |
|--------------------|
| 0.0451922356291692 |

PDURATION

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the number of periods required by an investment to reach a specified value.

Syntax

```
PDURATION(<rate>, <pv>, <fv>)
```

Parameters

| TERM | DEFINITION |
|------|---|
| rate | The interest rate per period. |
| pv | The present value of the investment. |
| fv | The desired future value of the investment. |

Return Value

The number of periods.

Remarks

- PDURATION uses the following equation:

$$\text{PDURATION} = \frac{\log(\text{fv}) - \log(\text{pv})}{\log(1 + \text{rate})}$$

- An error is returned if:
 - rate ≤ 0 .
 - pv ≤ 0 .
 - fv ≤ 0 .
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example 1

The following DAX query:

```
EVALUATE
{
    PDURATION(0.025, 2000, 2200)
}
```

Returns the number of years required for an investment of \$2000, earning 2.5% annually, to reach \$2200.

| |
|------------------|
| [VALUE] |
| 3.85986616262266 |

Example 2

The following DAX query:

```
EVALUATE
{
    PDURATION(0.025/12, 1000, 1200)
}
```

Returns the number of months required for an investment of \ \$1000, earning 2.5% annually, to reach \ \$1200.

| |
|------------------|
| [VALUE] |
| 87.6054764193714 |

PMT

10/31/2022 • 2 minutes to read • [Edit Online](#)

Calculates the payment for a loan based on constant payments and a constant interest rate.

Syntax

```
PMT(<rate>, <nper>, <pv>[, <fv>[, <type>]])
```

Parameters

| TERM | DEFINITION |
|------|---|
| rate | The interest rate for the loan. |
| nper | The total number of payments for the loan. |
| pv | The present value, or the total amount that a series of future payments is worth now; also known as the principal. |
| fv | (Optional) The future value, or a cash balance you want to attain after the last payment is made. If fv is omitted, it is assumed to be BLANK. |
| type | (Optional) The number 0 or 1 which indicates when payments are due. If type is omitted, it is assumed to be 0. The accepted values are listed below this table. |

The **type** parameter accepts the following values:

| SET TYPE EQUAL TO | IF PAYMENTS ARE DUE |
|-------------------|--------------------------------|
| 0 or omitted | At the end of the period |
| 1 | At the beginning of the period |

Note: For a more complete description of the arguments in PMT, see the PV function.

Return Value

The amount of a single loan payment.

Remarks

- The payment returned by PMT includes principal and interest but no taxes, reserve payments, or fees sometimes associated with loans.
- Make sure that you are consistent about the units you use for specifying rate and nper. If you make monthly payments on a four-year loan at an annual interest rate of 12 percent, use 0.12/12 for rate and 4*12 for nper. If you make annual payments on the same loan, use 0.12 for rate and 4 for nper.

- type is rounded to the nearest integer.
- An error is returned if:
 - $nper < 1$

Tip: To find the total amount paid over the duration of the loan, multiply the returned PMT value by nper.

- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Examples

Example 1

| DATA | DESCRIPTION |
|-----------|------------------------------|
| 8% | Annual interest rate |
| 10 | Number of months of payments |
| \\$10,000 | Amount of loan |

The following DAX query:

```
EVALUATE
{
    PMT(0.08/12, 10, 10000, 0, 1)
}
```

Returns the monthly payment amount, paid at the beginning of the month, for a loan with the terms specified above.

| [VALUE] |
|-------------------|
| -1030.16432717797 |

Note: 1030.16432717797 is the payment per period. As a result, the total amount paid over the duration of the loan is approximately $1030.16 * 10 = \$10,301.60$. In other words, approximately \\$301.60 of interest is paid.

Example 2

| DATA | DESCRIPTION |
|-----------|-----------------------------|
| 6% | Annual interest rate |
| 18 | Number of years of payments |
| \\$50,000 | Amount of loan |

The following DAX query:

EVALUATE

```
{  
  PMT(0.06/12, 18*12, 0, 50000)  
}
```

[VALUE]

-129.081160867991

Returns the amount to save each month to have \ \$50,000 at the end of 18 years, using the terms specified above.

PPMT

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the payment on the principal for a given period for an investment based on periodic, constant payments and a constant interest rate.

Syntax

```
PPMT(<rate>, <per>, <nper>, <pv>[, <fv>[, <type>]])
```

Parameters

| TERM | DEFINITION |
|------|---|
| rate | The interest rate for the loan. |
| per | Specifies the period. Must be between 1 and nper (inclusive). |
| nper | The total number of payment periods in an annuity. |
| pv | The present value — the total amount that a series of future payments is worth now. |
| fv | (Optional) The future value, or a cash balance you want to attain after the last payment is made. If fv is omitted, it is assumed to be BLANK. |
| type | (Optional) The number 0 or 1 which indicates when payments are due. If type is omitted, it is assumed to be 0. The accepted values are listed below this table. |

The **type** parameter accepts the following values:

| SET TYPE EQUAL TO | IF PAYMENTS ARE DUE |
|-------------------|--------------------------------|
| 0 or omitted | At the end of the period |
| 1 | At the beginning of the period |

Note: For a more complete description of the arguments in PPMT, see the PV function.

Return Value

The payment on the principal for a given period.

Remarks

- Make sure that you are consistent about the units you use for specifying rate and nper. If you make monthly payments on a four-year loan at an annual interest rate of 12 percent, use 0.12/12 for rate and 4*12 for nper. If you make annual payments on the same loan, use 0.12 for rate and 4 for nper.

- type is rounded to the nearest integer.
- An error is returned if:
 - $\text{per} < 1$ or $\text{per} > \text{nper}$
 - $\text{nper} < 1$
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example 1

| DATA | ARGUMENT DESCRIPTION |
|-------------|------------------------------|
| 10% | Annual interest rate |
| 2 | Number of years for the loan |
| \\$2,000.00 | Amount of loan |

The following DAX query:

```
EVALUATE
{
    PPMT(0.1/12, 1, 2*12, 2000.00)
}
```

Returns the principal payment made in the first month for a loan with the terms specified above.

| [VALUE] |
|-------------------|
| -75.6231860083663 |

Example 2

| DATA | ARGUMENT DESCRIPTION |
|---------------|------------------------------|
| 8% | Annual interest rate |
| 10 | Number of years for the loan |
| \\$200,000.00 | Amount of loan |

The following DAX query:

```
EVALUATE
{
    PPMT(0.08, 10, 10, 200000.00)
}
```

Returns the principal payment made in the 10th year for a loan with the terms specified above.

[VALUE]

-27598.0534624214

PRICE

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the price per \ \$100 face value of a security that pays periodic interest.

Syntax

```
PRICE(<settlement>, <maturity>, <rate>, <yld>, <redemption>, <frequency>[, <basis>])
```

Parameters

| TERM | DEFINITION |
|------------|---|
| settlement | The security's settlement date. The security settlement date is the date after the issue date when the security is traded to the buyer. |
| maturity | The security's maturity date. The maturity date is the date when the security expires. |
| rate | The security's annual coupon rate. |
| yld | The security's annual yield. |
| redemption | The security's redemption value per \ \$100 face value. |
| frequency | The number of coupon payments per year. For annual payments, frequency = 1; for semiannual, frequency = 2; for quarterly, frequency = 4. |
| basis | (Optional) The type of day count basis to use. If basis is omitted, it is assumed to be 0. The accepted values are listed below this table. |

The **basis** parameter accepts the following values:

| BASIS | DAY COUNT BASIS |
|--------------|------------------|
| 0 or omitted | US (NASD) 30/360 |
| 1 | Actual/actual |
| 2 | Actual/360 |
| 3 | Actual/365 |
| 4 | European 30/360 |

Return Value

The price per \$100 face value.

Remarks

- Dates are stored as sequential serial numbers so they can be used in calculations. In DAX, December 30, 1899 is day 0, and January 1, 2008 is 39448 because it is 39,448 days after December 30, 1899.
- The settlement date is the date a buyer purchases a coupon, such as a bond. The maturity date is the date when a coupon expires. For example, suppose a 30-year bond is issued on January 1, 2008, and is purchased by a buyer six months later. The issue date would be January 1, 2008, the settlement date would be July 1, 2008, and the maturity date would be January 1, 2038, which is 30 years after the January 1, 2008, issue date.
- settlement and maturity are truncated to integers.
- basis and frequency are rounded to the nearest integer.
- An error is returned if:
 - settlement or maturity is not a valid date.
 - settlement \geq maturity.
 - rate < 0 .
 - yld < 0 .
 - redemption ≤ 0 .
 - frequency is any number other than 1, 2, or 4.
 - basis < 0 or basis > 4 .
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Important:

- When $N > 1$ (N is the number of coupons payable between the settlement date and redemption date), **PRICE** is calculated as follows:

$$\text{PRICE} = \left[\frac{\text{redemption}}{(1 + \frac{\text{yld}}{\text{frequency}})^{(N - 1 + \frac{\text{DSC}}{\text{E}})}} \right] + \left[\sum_{k=1}^N \frac{100 \times \text{rate}}{\text{frequency} \times (1 + \frac{\text{yld}}{\text{frequency}})^{(k - 1 + \frac{\text{DSC}}{\text{E}})}} \right] - \left[100 \times \frac{\text{rate}}{\text{frequency}} \times \frac{\text{A}}{\text{E}} \right]$$

- When $N = 1$ (N is the number of coupons payable between the settlement date and redemption date), **PRICE** is calculated as follows:

$$\text{DSR} = \text{E} - \text{A}$$

$$\text{T1} = 100 \times \frac{\text{rate}}{\text{frequency}} + \text{redemption}$$

$$\text{T2} = \frac{\text{yld}}{\text{frequency}} \times \frac{\text{DSR}}{\text{E}} + 1$$

$$\text{T3} = 100 \times \frac{\text{rate}}{\text{frequency}} \times \frac{\text{A}}{\text{E}}$$

$$\text{PRICE} = \frac{\text{T1}}{\text{T2}} - \text{T3}$$

where:

- DSC = number of days from settlement to next coupon date.
- E = number of days in coupon period in which the settlement date falls.
- A = number of days from beginning of coupon period to settlement date.

Example

| DATA | ARGUMENT DESCRIPTION |
|------------|---------------------------|
| 2/15/2008 | Settlement date |
| 11/15/2017 | Maturity date |
| 5.75% | Percent semiannual coupon |
| 6.50% | Percent yield |
| \\$100 | Redemption value |
| 2 | Frequency is semiannual |
| 0 | 30/360 basis |

The following DAX query:

```
EVALUATE
{
    PRICE (DATE (2008,2,15), DATE (2017,11,15), 0.0575, 0.065, 100, 2, 0)
}
```

Returns the bond price, for a bond using the terms specified above.

| [VALUE] |
|------------------|
| 94.6343616213221 |

PRICEDISC

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the price per \ \$100 face value of a discounted security.

Syntax

```
PRICEDISC(<settlement>, <maturity>, <discount>, <redemption>[, <basis>])
```

Parameters

| TERM | DEFINITION |
|------------|---|
| settlement | The security's settlement date. The security settlement date is the date after the issue date when the security is traded to the buyer. |
| maturity | The security's maturity date. The maturity date is the date when the security expires. |
| discount | The security's discount rate. |
| redemption | The security's redemption value per \ \$100 face value. |
| basis | (Optional) The type of day count basis to use. If basis is omitted, it is assumed to be 0. The accepted values are listed below this table. |

The **basis** parameter accepts the following values:

| BASIS | DAY COUNT BASIS |
|--------------|------------------|
| 0 or omitted | US (NASD) 30/360 |
| 1 | Actual/actual |
| 2 | Actual/360 |
| 3 | Actual/365 |
| 4 | European 30/360 |

Return Value

The price per \ \$100 face value.

Remarks

- Dates are stored as sequential serial numbers so they can be used in calculations. In DAX, December 30, 1899 is day 0, and January 1, 2008 is 39448 because it is 39,448 days after December 30, 1899.

- The settlement date is the date a buyer purchases a coupon, such as a bond. The maturity date is the date when a coupon expires. For example, suppose a 30-year bond is issued on January 1, 2018, and is purchased by a buyer six months later. The issue date would be January 1, 2018, the settlement date would be July 1, 2018, and the maturity date would be January 1, 2048, 30 years after the January 1, 2018, issue date.
- PRICEDISC is calculated as follows:
$$\text{PRICEDISC} = \text{redemption} - \text{discount} \times \text{redemption} \times \frac{\text{DSM}}{\text{B}}$$
where:
 - B = number of days in year, depending on year basis.
 - DSM = number of days from settlement to maturity.
- settlement and maturity are truncated to integers.
- basis is rounded to the nearest integer.
- An error is returned if:
 - settlement or maturity is not a valid date.
 - settlement \geq maturity.
 - discount \leq 0.
 - redemption \leq 0.
 - basis < 0 or basis > 4.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

| DATA | ARGUMENT DESCRIPTION |
|-----------|-----------------------|
| 2/16/2008 | Settlement date |
| 3/1/2008 | Maturity date |
| 5.25% | Percent discount rate |
| \\$100 | Redemption value |
| 2 | Actual/360 basis |

The following DAX query:

```
EVALUATE
{
    PRICEDISC(DATE(2008,2,16), DATE(2008,3,1), 0.0525, 100, 2)
}
```

Returns the bond price per \\$100 face value, for a bond with the terms specified above.

[VALUE]

99.79583333333333

PRICEMAT

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the price per \ \$100 face value of a security that pays interest at maturity.

Syntax

```
PRICEMAT(<settlement>, <maturity>, <issue>, <rate>, <yld>[, <basis>])
```

Parameters

| TERM | DEFINITION |
|------------|---|
| settlement | The security's settlement date. The security settlement date is the date after the issue date when the security is traded to the buyer. |
| maturity | The security's maturity date. The maturity date is the date when the security expires. |
| issue | The security's issue date. |
| rate | The security's interest rate at date of issue. |
| yld | The security's annual yield. |
| basis | (Optional) The type of day count basis to use. If basis is omitted, it is assumed to be 0. The accepted values are listed below this table. |

The **basis** parameter accepts the following values:

| BASIS | DAY COUNT BASIS |
|--------------|------------------|
| 0 or omitted | US (NASD) 30/360 |
| 1 | Actual/actual |
| 2 | Actual/360 |
| 3 | Actual/365 |
| 4 | European 30/360 |

Return Value

The price per \ \$100 face value.

Remarks

- Dates are stored as sequential serial numbers so they can be used in calculations. In DAX, December 30, 1899 is day 0, and January 1, 2008 is 39448 because it is 39,448 days after December 30, 1899.
- The settlement date is the date a buyer purchases a coupon, such as a bond. The maturity date is the date when a coupon expires. For example, suppose a 30-year bond is issued on January 1, 2008, and is purchased by a buyer six months later. The issue date would be January 1, 2008, the settlement date would be July 1, 2008, and the maturity date would be January 1, 2038, which is 30 years after the January 1, 2008, issue date.
- PRICEMAT is calculated as follows:

$$\text{PRICEMAT} = \frac{100 + (\frac{\text{DIM}}{\text{B}} \times \text{rate} \times 100)}{1 + (\frac{\text{DSM}}{\text{B}} \times \text{yld})} - (\frac{\text{A}}{\text{B}} \times \text{rate} \times 100)$$

where:

- B = number of days in year, depending on year basis.
- DSM = number of days from settlement to maturity.
- DIM = number of days from issue to maturity.
- A = number of days from issue to settlement.
- settlement, maturity, and issue are truncated to integers.
- basis is rounded to the nearest integer.
- An error is returned if:
 - settlement, maturity, or issue is not a valid date.
 - maturity > settlement > issue is not satisfied.
 - rate < 0.
 - yld < 0.
 - basis < 0 or basis > 4.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following DAX query:

| DATA | DESCRIPTION |
|------------|---------------------------|
| 2/15/2008 | Settlement date |
| 4/13/2008 | Maturity date |
| 11/11/2007 | Issue date |
| 6.10% | Percent semiannual coupon |
| 6.10% | Percent yield |
| 0 | 30/360 basis |

EVALUATE

```
{  
  PRICEMAT(DATE(2008,2,15), DATE(2008,4,13), DATE(2007,11,11), 0.061, 0.061, 0)  
}
```

Returns the price per \ \$100 face value of a security with the terms specified above.

[VALUE]

99.9844988755569

PV

10/31/2022 • 3 minutes to read • [Edit Online](#)

Calculates the present value of a loan or an investment, based on a constant interest rate. You can use PV with either periodic, constant payments (such as a mortgage or other loan), and/or a future value that's your investment goal.

Syntax

```
PV(<rate>, <nper>, <pmt>[, <fv>[, <type>]])
```

Parameters

| TERM | DEFINITION |
|------|---|
| rate | The interest rate per period. For example, if you obtain an automobile loan at a 10 percent annual interest rate and make monthly payments, your interest rate per month is 0.1/12, or 0.0083. You would enter 0.1/12, or 0.0083, into the formula as the rate. |
| nper | The total number of payment periods in an annuity. For example, if you get a four-year car loan and make monthly payments, your loan has 4*12 (or 48) periods. You would enter 48 into the formula for nper. |
| pmt | The payment made each period that cannot change over the life of the annuity. Typically, pmt includes principal and interest but no other fees or taxes. For example, the monthly payments on a \$10,000, four-year car loan at 12 percent are \$263.33. You would enter -263.33 into the formula as the pmt. |
| fv | (Optional) The future value, or a cash balance you want to attain after the last payment is made. If fv is omitted, it is assumed to be BLANK. For example, if you want to save \$50,000 to pay for a special project in 18 years, then \$50,000 is the future value. You could then make a conservative guess at an interest rate and determine how much you must save each month. |
| type | (Optional) The number 0 or 1 which indicates when payments are due. If type is omitted, it is assumed to be 0. The accepted values are listed below this table. |

The **type** parameter accepts the following values:

| SET TYPE EQUAL TO | IF PAYMENTS ARE DUE |
|-------------------|--------------------------------|
| 0 or omitted | At the end of the period |
| 1 | At the beginning of the period |

Return Value

The present value of a loan or investment.

Remarks

- Make sure that you are consistent about the units you use for specifying rate and nper. If you make monthly payments on a four-year loan at 12 percent annual interest, use 0.12/12 for rate and 4*12 for nper. If you make annual payments on the same loan, use 0.12 for rate and 4 for nper.
- The following functions apply to annuities:
 - CUMIPMT
 - CUMPRINC
 - FV
 - IPMT
 - PMT
 - PPMT
 - PV
 - RATE
 - XIRR
 - XNPV
- An annuity is a series of constant cash payments made over a continuous period. For example, a car loan or a mortgage is an annuity. For more information, see the description for each annuity function.
- In annuity functions, cash you pay out, such as a deposit to savings, is represented by a negative number; cash you receive, such as a dividend check, is represented by a positive number. For example, a \$1,000 deposit to the bank would be represented by the argument -1000 if you are the depositor and by the argument 1000 if you are the bank.
- One financial argument is solved in terms of the others.
 - If rate is not 0, then:
$$\text{pv} \times (1 + \text{rate})^{\text{nper}} + \text{pmt}(1 + \text{rate} \times \text{type}) \times \text{bigg}(\frac{(1 + \text{rate})^{\text{nper}} - 1}{\text{rate}}) + \text{fv} = 0$$
 - If rate is 0, then:
$$\text{pmt} \times \text{nper} + \text{pv} + \text{fv} = 0$$
- type is rounded to the nearest integer.
- An error is returned if:
 - nper < 1 or blank
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

| DATA | DESCRIPTION |
|-----------|---|
| \\$500.00 | Money paid out of an insurance annuity at the end of every month. |

| DATA | DESCRIPTION |
|------|---|
| 8% | Interest rate earned on the money paid out. |
| 20 | Years the money will be paid out. |

The following DAX query:

```
EVALUATE
{
    PV(0.08/12, 12*20, 500.00, 0, 0)
}
```

Returns the present value of an annuity using the terms specified above.

| [VALUE] |
|-------------------|
| -59777.1458511878 |

RATE

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the interest rate per period of an annuity. RATE is calculated by iteration and can have zero or more solutions. If the successive results of RATE do not converge to within 0.0000001 after 20 iterations, an error is returned.

Syntax

```
RATE(<nper>, <pmt>, <pv>[, <fv>[, <type>[, <guess>]]])
```

Parameters

| TERM | DEFINITION |
|-------|--|
| nper | The total number of payment periods in an annuity. |
| pmt | The payment made each period and cannot change over the life of the annuity. Typically, pmt includes principal and interest but no other fees or taxes. |
| pv | The present value — the total amount that a series of future payments is worth now. |
| fv | (Optional) The future value, or a cash balance you want to attain after the last payment is made. If fv is omitted, it is assumed to be 0 (the future value of a loan, for example, is 0). |
| type | (Optional) The number 0 or 1 which indicates when payments are due. If type is omitted, it is assumed to be 0. The accepted values are listed below this table. |
| guess | (Optional) Your guess for what the rate will be. <ul style="list-style-type: none">- If omitted, it is assumed to be 10%.- If RATE does not converge, try different values for guess. RATE usually converges if guess is between 0 and 1. |

The **type** parameter accepts the following values:

| SET TYPE EQUAL TO | IF PAYMENTS ARE DUE |
|-------------------|--------------------------------|
| 0 or omitted | At the end of the period |
| 1 | At the beginning of the period |

Return Value

The interest rate per period.

Remarks

- Make sure that you are consistent about the units you use for specifying guess and nper. If you make monthly payments on a four-year loan at 12 percent annual interest, use 0.12/12 for guess and 4*12 for nper. If you make annual payments on the same loan, use 0.12 for guess and 4 for nper.
- type is rounded to the nearest integer.
- An error is returned if:
 - $nper \leq 0$.
 - RATE does not converge to within 0.0000001 after 20 iterations
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Examples

| DATA | DESCRIPTION |
|------|--------------------|
| 4 | Years of the loan |
| -200 | Monthly payment |
| 8000 | Amount of the loan |

Example 1

The following DAX query:

```
EVALUATE
{
    RATE(4*12, -200, 8000)
}
```

Returns the monthly rate of the loan using the terms specified above.

| [VALUE] |
|---------------------|
| 0.00770147248820137 |

Example 2

The following DAX query:

```
EVALUATE
{
    RATE(4*12, -200, 8000) * 12
}
```

Returns the annual rate of the loan using the terms specified above.

| [VALUE] |
|--------------------|
| 0.0924176698584164 |

RECEIVED

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the amount received at maturity for a fully invested security.

Syntax

```
RECEIVED(<settlement>, <maturity>, <investment>, <discount>[, <basis>])
```

Parameters

| TERM | DEFINITION |
|------------|---|
| settlement | The security's settlement date. The security settlement date is the date after the issue date when the security is traded to the buyer. |
| maturity | The security's maturity date. The maturity date is the date when the security expires. |
| investment | The amount invested in the security. |
| discount | The security's discount rate. |
| basis | (Optional) The type of day count basis to use. If basis is omitted, it is assumed to be 0. The accepted values are listed below this table. |

The **basis** parameter accepts the following values:

| BASIS | DAY COUNT BASIS |
|--------------|------------------|
| 0 or omitted | US (NASD) 30/360 |
| 1 | Actual/actual |
| 2 | Actual/360 |
| 3 | Actual/365 |
| 4 | European 30/360 |

Return Value

The amount received at maturity.

Remarks

- Dates are stored as sequential serial numbers so they can be used in calculations. In DAX, December 30, 1899 is day 0, and January 1, 2008 is 39448 because it is 39,448 days after December 30, 1899.

- The settlement date is the date a buyer purchases a coupon, such as a bond. The maturity date is the date when a coupon expires. For example, suppose a 30-year bond is issued on January 1, 2008, and is purchased by a buyer six months later. The issue date would be January 1, 2008, the settlement date would be July 1, 2008, and the maturity date would be January 1, 2038, which is 30 years after the January 1, 2008, issue date.

- RECEIVED is calculated as follows:

$$\text{RECEIVED} = \frac{\text{investment}}{1 - (\text{discount} \times \frac{\text{DIM}}{\text{B}})}$$

where:

- B = number of days in a year, depending on the year basis.
 - DIM = number of days from issue to maturity.
- settlement and maturity are truncated to integers.
- basis is rounded to the nearest integer.
- An error is returned if:
 - settlement or maturity is not a valid date.
 - settlement \geq maturity.
 - investment \leq 0.
 - discount \leq 0.
 - basis < 0 or basis > 4.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following DAX query:

| DATA | DESCRIPTION |
|-----------------|-------------------------|
| 15-Feb-08 | Settlement (issue) date |
| 15-May-08 | Maturity date |
| \\$1,000,000.00 | Investment |
| 5.75% | Percent discount rate |
| 2 | Actual/360 basis |

```
EVALUATE
{
    RECEIVED(DATE(2008,2,15), DATE(2008,5,15), 1000000.00, 0.0575, 2)
}
```

Returns the total amount to be received at maturity, for a bond with the terms specified above.

| [VALUE] |
|-----------------|
| 1014584.6544071 |

RRI

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns an equivalent interest rate for the growth of an investment.

Syntax

```
RRI(<nper>, <pv>, <fv>)
```

Parameters

| TERM | DEFINITION |
|------|---|
| nper | The number of periods for the investment. |
| pv | The present value of the investment. |
| fv | The future value of the investment. |

Return Value

The equivalent interest rate.

Remarks

- RRI returns the interest rate given $\text{\$}\text{\textit{nper}}$ (the number of periods), $\text{\$}\text{\textit{pv}}$ (present value), and $\text{\$}\text{\textit{fv}}$ (future value), calculated by using the following equation:

$$\text{\$}\text{\textit{bigg}}\left(\frac{\text{\$}\text{\textit{fv}}}{\text{\$}\text{\textit{pv}}}\right)^{\frac{1}{\text{\textit{nper}}}} - 1 \text{\$}$$

- An error is returned if:
 - $\text{nper} \leq 0$.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

| DATA | DESCRIPTION |
|-----------|----------------|
| \\$10,000 | Present value |
| \\$21,000 | Future value |
| 4 | Years invested |

The following DAX query:

EVALUATE

```
{  
  RRI(4*12, 10000, 21000)  
}
```

Returns an equivalent interest rate for the growth of an investment with the terms specified above.

[VALUE]

0.0155771057566627

SLN

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the straight-line depreciation of an asset for one period.

Syntax

```
SLN(<cost>, <salvage>, <life>)
```

Parameters

| TERM | DEFINITION |
|---------|--|
| cost | The initial cost of the asset. |
| salvage | The value at the end of the depreciation (sometimes called the salvage value of the asset). |
| life | The number of periods over which the asset is depreciated (sometimes called the useful life of the asset). |

Return Value

The straight-line depreciation for one period.

Remarks

- An error is returned if:
life = 0.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

| DATA | DESCRIPTION |
|-----------|----------------------|
| \\$30,000 | Cost |
| \\$7,500 | Salvage value |
| 10 | Years of useful life |

The following DAX query:

```
EVALUATE
{
    SLN(30000, 7500, 10)
}
```

Returns the yearly depreciation allowance using the terms specified above.

| |
|---------|
| [VALUE] |
| 2250 |

SYD

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the sum-of-years' digits depreciation of an asset for a specified period.

Syntax

```
SYD(<cost>, <salvage>, <life>, <per>)
```

Parameters

| TERM | DEFINITION |
|---------|--|
| cost | The initial cost of the asset. |
| salvage | The value at the end of the depreciation (sometimes called the salvage value of the asset). |
| life | The number of periods over which the asset is depreciated (sometimes called the useful life of the asset). |
| per | The period. Must use the same units as life. Must be between 1 and life (inclusive). |

Return Value

The sum-of-years' digits depreciation for the specified period.

Remarks

- SYD is calculated as follows:

$$\text{SYD} = \frac{(\text{cost} - \text{salvage}) \times (\text{life} - \text{per} + 1) \times 2}{(\text{life} + 1)}$$

- An error is returned if:
 - life < 1.
 - per < 1 or per > life.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Examples

| DATA | DESCRIPTION |
|--------------|---------------|
| \\$30,000.00 | Initial cost |
| \\$7,500.00 | Salvage value |

| DATA | DESCRIPTION |
|------|-------------------|
| 10 | Lifespan in years |

Example 1

The following DAX query:

```
EVALUATE
{
    SYD(30000.00, 7500.00, 10, 1)
}
```

Returns an asset's sum-of-years' digits depreciation allowance for the first year, given the terms specified above.

| [VALUE] |
|------------------|
| 4090.90909090909 |

Example 2

The following DAX query:

```
EVALUATE
{
    SYD(30000.00, 7500.00, 10, 10)
}
```

Returns an asset's sum-of-years' digits depreciation allowance for the tenth (final) year, given the terms specified above.

| [VALUE] |
|------------------|
| 409.090909090909 |

TBILLEQ

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the bond-equivalent yield for a Treasury bill.

Syntax

```
TBILLEQ(<settlement>, <maturity>, <discount>)
```

Parameters

| TERM | DEFINITION |
|------------|---|
| settlement | The Treasury bill's settlement date. The security settlement date is the date after the issue date when the Treasury bill is traded to the buyer. |
| maturity | The Treasury bill's maturity date. The maturity date is the date when the Treasury bill expires. |
| discount | The Treasury bill's discount rate. |

Return Value

The Treasury Bill's bond-equivalent yield.

Remarks

- Dates are stored as sequential serial numbers so they can be used in calculations. In DAX, December 30, 1899 is day 0, and January 1, 2008 is 39448 because it is 39,448 days after December 30, 1899.

- TBILLEQ is calculated as:

$$\text{TBILLEQ} = \frac{365 \times \text{discount}}{360 - (\text{discount} \times \text{DSM})}$$

where:

- DSM is the number of days between settlement and maturity computed according to the 360 days per year basis.
- settlement and maturity are truncated to integers.
- An error is returned if:
 - settlement or maturity is not a valid date.
 - $\text{settlement} \geq \text{maturity}$ or maturity is more than one year after settlement.
 - $\text{discount} \leq 0$.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

| DATA | DESCRIPTION |
|-----------|-----------------------|
| 3/31/2008 | Settlement date |
| 6/1/2008 | Maturity date |
| 9.14% | Percent discount rate |

The following DAX query:

```
EVALUATE
{
    TBILLEQ(DATE(2008,3,31), DATE(2008,6,1), 0.0914)
}
```

Returns the bond-equivalent yield for a Treasury bill using the terms specified above.

| [VALUE] |
|-------------------|
| 0.094151493565943 |

TBILLPRICE

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the price per \ \$100 face value for a Treasury bill.

Syntax

```
TBILLPRICE(<settlement>, <maturity>, <discount>)
```

Parameters

| TERM | DEFINITION |
|------------|---|
| settlement | The Treasury bill's settlement date. The security settlement date is the date after the issue date when the Treasury bill is traded to the buyer. |
| maturity | The Treasury bill's maturity date. The maturity date is the date when the Treasury bill expires. |
| discount | The Treasury bill's discount rate. |

Return Value

The Treasury Bill's price per \ \$100 face value.

Remarks

- Dates are stored as sequential serial numbers so they can be used in calculations. In DAX, December 30, 1899 is day 0, and January 1, 2008 is 39448 because it is 39,448 days after December 30, 1899.

- TBILLPRICE is calculated as follows:

$$\text{\$}\text{\text{TBILLPRICE}} = 100 \times (1 - \frac{\text{\text{discount}}}{360})$$

where:

- $\text{\text{DSM}}$ = number of days from settlement to maturity, excluding any maturity date that is more than one calendar year after the settlement date.
- settlement and maturity are truncated to integers.
- An error is returned if:
 - settlement or maturity is not a valid date.
 - $\text{settlement} \geq \text{maturity}$ or maturity is more than one year after settlement.
 - $\text{discount} \leq 0$.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

| DATA | DESCRIPTION |
|-----------|-----------------------|
| 3/31/2008 | Settlement date |
| 6/1/2008 | Maturity date |
| 9.0% | Percent discount rate |

The following DAX query:

```
EVALUATE
{
    TBILLPRICE (DATE(2008,3,31), DATE(2008,6,1), 0.09)
}
```

Returns the Treasury Bill's price per \ \$100 face value, given the terms specified above.

| [VALUE] |
|---------|
| 98.45 |

TBILLYIELD

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the yield for a Treasury bill.

Syntax

```
TBILLYIELD(<settlement>, <maturity>, <pr>)
```

Parameters

| TERM | DEFINITION |
|------------|---|
| settlement | The Treasury bill's settlement date. The security settlement date is the date after the issue date when the Treasury bill is traded to the buyer. |
| maturity | The Treasury bill's maturity date. The maturity date is the date when the Treasury bill expires. |
| pr | The Treasury bill's price per \$100 face value. |

Return Value

The Treasury bill's yield.

Remarks

- Dates are stored as sequential serial numbers so they can be used in calculations. In DAX, December 30, 1899 is day 0, and January 1, 2008 is 39448 because it is 39,448 days after December 30, 1899.

- TBILLYIELD is calculated as follows:

$$\text{TBILLYIELD} = \frac{100 - \text{pr}}{\text{pr}} \times \frac{360}{\text{DSM}}$$

where:

- DSM = number of days from settlement to maturity, excluding any maturity date that is more than one calendar year after the settlement date.
- settlement and maturity are truncated to integers.
- An error is returned if:
 - settlement or maturity is not a valid date.
 - settlement \geq maturity or maturity is more than one year after settlement.
 - pr ≤ 0 .
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following DAX query:

| DATA | DESCRIPTION |
|-----------|-----------------------------|
| 3/31/2008 | Settlement date |
| 6/1/2008 | Maturity date |
| \\$98.45 | Price per \\$100 face value |

```
EVALUATE
{
  TBILLYIELD(DATE(2008,3,31), DATE(2008,6,1), 98.45)
}
```

Returns the yield of a Treasury bill using the terms specified above.

| [VALUE] |
|--------------------|
| 0.0914169629253426 |

VDB

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the depreciation of an asset for any period you specify, including partial periods, using the double-declining balance method or some other method you specify. VDB stands for variable declining balance.

Syntax

```
VDB(<cost>, <salvage>, <life>, <start_period>, <end_period>[, <factor>[, <no_switch>]])
```

Parameters

| TERM | DEFINITION |
|--------------|---|
| cost | The initial cost of the asset. |
| salvage | The value at the end of the depreciation (sometimes called the salvage value of the asset). This value can be 0. |
| life | The number of periods over which the asset is being depreciated (sometimes called the useful life of the asset). |
| start_period | The starting period for which you want to calculate the depreciation. Start_period must use the same units as life. Must be between 1 and life (inclusive). |
| end_period | The ending period for which you want to calculate the depreciation. End_period must use the same units as life. Must be between start_period and life (inclusive). |
| factor | (Optional) The rate at which the balance declines. If factor is omitted, it is assumed to be 2 (the double-declining balance method). Change factor if you do not want to use the double-declining balance method. For a description of the double-declining balance method, see DDB. |
| no_switch | (Optional) A logical value specifying whether to switch to straight-line depreciation when depreciation is greater than the declining balance calculation. If omitted, it is assumed to be FALSE. <ul style="list-style-type: none">- If no_switch evaluates to TRUE, VDB does not switch to straight-line depreciation, even when the depreciation is greater than the declining balance calculation.- If no_switch evaluates to FALSE or is omitted, VDB switches to straight-line depreciation when depreciation is greater than the declining balance calculation. |

Return Value

The depreciation over the specified period.

Remarks

- An error is returned if:
 - `cost < 0`.
 - `salvage < 0`.
 - `life < 1`.
 - `start_period < 1` or `start_period > end_period`.
 - `end_period < start_period` or `end_period > life`.
 - `factor < 0`.
 - `no_switch` does not evaluate to either `TRUE` or `FALSE`.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Examples

| DATA | DESCRIPTION |
|------|-------------------|
| 2400 | Initial cost |
| 300 | Salvage value |
| 10 | Lifetime in years |

Example 1

The following DAX query:

```
EVALUATE
{
    VDB(2400, 300, 10*365, 0, 1)
}
```

Returns an asset's first day's depreciation using a factor of 2.

| [VALUE] |
|------------------|
| 1.31506849315068 |

Example 2

The following DAX query:

```
EVALUATE
{
    VDB(2400, 300, 10*12, 6, 18, 3)
}
```

Returns an asset's depreciation between the 6th month and the 18th month. This calculation uses a factor of 3.

| [VALUE] |
|------------------|
| 540.185558199698 |

Example 3

The following DAX query:

```
EVALUATE
{
    VDB(2400, 300, 10, 0, 0.875, 1.5)
}
```

Returns an asset's depreciation in the first fiscal year that you own it, assuming that tax laws limit you to 150% depreciation of the declining balance. The asset is purchased in the middle of the first quarter of the fiscal year.

| [VALUE] |
|---------|
| 315 |

XIRR

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the internal rate of return for a schedule of cash flows that is not necessarily periodic.

Syntax

```
XIRR(<table>, <values>, <dates>, [, <guess>[, <alternateResult>]])
```

Parameters

| TERM | DEFINITION |
|-----------------|--|
| table | A table for which the values and dates expressions should be calculated. |
| values | An expression that returns the cash flow value for each row of the table. |
| dates | An expression that returns the cash flow date for each row of the table. |
| guess | (Optional) An initial guess for the internal rate of return. If omitted, the default guess of 0.1 is used. |
| alternateResult | (Optional) A value returned in place of an error when a solution cannot be determined. |

Return value

Internal rate of return for the given inputs. If the calculation fails to return a valid result, an error or value specified as alternateResult is returned.

Remarks

- The value is calculated as the rate that satisfies the following function:

$$\sum_{j=1}^N \frac{P_j}{(1 + \text{rate})^{\frac{d_j}{365}}} = 0$$

Where:

- P_j is the j^{th} payment
- d_j is the j^{th} payment date
- d_1 is the first payment date
- The series of cash flow values must contain at least one positive number and one negative number.
- Avoid using ISERROR or IFERROR functions to capture an error returned by XIRR. If some inputs to the function may result in a no solution error, providing an alternateResult parameter is the most reliable and highest performing way to handle the error.
- To learn more about using the alternateResult parameter, be to check out this [video](#).

- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following formula calculates the internal rate of return of the CashFlows table:

```
= XIRR( CashFlows, [Payment], [Date] )
```

| DATE | PAYMENT |
|------------|---------|
| 1/1/2014 | -10000 |
| 3/1/2014 | 2750 |
| 10/30/2014 | 4250 |
| 2/15/2015 | 3250 |
| 4/1/2015 | 2750 |

Rate of return = 37.49%

XNPV

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the present value for a schedule of cash flows that is not necessarily periodic.

Syntax

```
XNPV(<table>, <values>, <dates>, <rate>)
```

Parameters

| TERM | DEFINITION |
|--------|---|
| table | A table for which the values and dates expressions should be calculated. |
| values | An expression that returns the cash flow value for each row of the table. |
| dates | An expression that returns the cash flow date for each row of the table. |
| rate | The discount rate to apply to the cash flow for each row of the table. |

Return value

Net present value.

Remarks

- The value is calculated as the following summation:

$$\sum_{j=1}^N \frac{P_j}{(1 + \text{rate})^{\frac{d_j - d_1}{365}}}$$

Where:

- P_j is the j^{th} payment
- d_j is the j^{th} payment date
- d_1 is the first payment date
- The series of cash flow values must contain at least one positive number and one negative number.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following calculates the present value of the CashFlows table:

= XNPV(CashFlows, [Payment], [Date], 0.09)

| DATE | PAYMENT |
|------------|---------|
| 1/1/2014 | -10000 |
| 3/1/2014 | 2750 |
| 10/30/2014 | 4250 |
| 2/15/2015 | 3250 |
| 4/1/2015 | 2750 |

Present value = 2086.65

YIELD

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the yield on a security that pays periodic interest. Use YIELD to calculate bond yield.

Syntax

```
YIELD(<settlement>, <maturity>, <rate>, <pr>, <redemption>, <frequency>[, <basis>])
```

Parameters

| TERM | DEFINITION |
|------------|---|
| settlement | The security's settlement date. The security settlement date is the date after the issue date when the security is traded to the buyer. |
| maturity | The security's maturity date. The maturity date is the date when the security expires. |
| rate | The security's annual coupon rate. |
| pr | The security's price per \ \$100 face value. |
| redemption | The security's redemption value per \ \$100 face value. |
| frequency | The number of coupon payments per year. For annual payments, frequency = 1; for semiannual, frequency = 2; for quarterly, frequency = 4. |
| basis | (Optional) The type of day count basis to use. If basis is omitted, it is assumed to be 0. The accepted values are listed below this table. |

The **basis** parameter accepts the following values:

| BASIS | DAY COUNT BASIS |
|--------------|------------------|
| 0 or omitted | US (NASD) 30/360 |
| 1 | Actual/actual |
| 2 | Actual/360 |
| 3 | Actual/365 |
| 4 | European 30/360 |

Return Value

The yield on the security.

Remarks

- Dates are stored as sequential serial numbers so they can be used in calculations. In DAX, December 30, 1899 is day 0, and January 1, 2008 is 39448 because it is 39,448 days after December 30, 1899.
- The settlement date is the date a buyer purchases a coupon, such as a bond. The maturity date is the date when a coupon expires. For example, suppose a 30-year bond is issued on January 1, 2008, and is purchased by a buyer six months later. The issue date would be January 1, 2008, the settlement date would be July 1, 2008, and the maturity date would be January 1, 2038, which is 30 years after the January 1, 2008, issue date.

- If there is one coupon period or less until redemption, YIELD is calculated as follows:

$$\text{YIELD} = \frac{\left(\frac{\text{redemption}}{\text{frequency}} + \frac{\text{rate}}{\text{frequency}} \right) - \left(\frac{\text{par}}{100} + \frac{\text{A}}{\text{E}} \times \frac{\text{rate}}{\text{frequency}} \right)}{\left(\frac{\text{par}}{100} + \frac{\text{A}}{\text{E}} \times \frac{\text{rate}}{\text{frequency}} \right) \times \frac{\text{frequency}}{\text{E}} + \text{DSR}}$$

where:

- A = number of days from the beginning of the coupon period to the settlement date (accrued days).
- DSR = number of days from the settlement date to the redemption date.
- E = number of days in the coupon period.
- If there is more than one coupon period until redemption, YIELD is calculated through a hundred iterations. The resolution uses the Newton method, based on the formula used for the function PRICE. The yield is changed until the estimated price given the yield is close to price.
- settlement and maturity are truncated to integers.
- frequency, and basis are rounded to the nearest integer.
- An error is returned if:
 - settlement or maturity is not a valid date.
 - settlement \geq maturity.
 - rate < 0 .
 - pr ≤ 0 .
 - redemption ≤ 0 .
 - frequency is any number other than 1, 2, or 4.
 - basis < 0 or basis > 4 .
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

| DATA | DESCRIPTION |
|-----------|-----------------|
| 15-Feb-08 | Settlement date |
| 15-Nov-16 | Maturity date |

| DATA | DESCRIPTION |
|----------|-------------------------------------|
| 5.75% | Percent coupon |
| 95.04287 | Price |
| \\$100 | Redemption value |
| 2 | Frequency is semiannual (see above) |
| 0 | 30/360 basis (see above) |

The following DAX query:

```
EVALUATE
{
    YIELD(DATE(2008,2,15), DATE(2016,11,15), 0.0575, 95.04287, 100, 2,0)
}
```

Returns the yield on a bond with the terms specified above.

| [VALUE] |
|--------------------|
| 0.0650000068807314 |

YIELDDISC

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the annual yield for a discounted security.

Syntax

```
YIELDDISC(<settlement>, <maturity>, <pr>, <redemption>[, <basis>])
```

Parameters

| TERM | DEFINITION |
|------------|---|
| settlement | The security's settlement date. The security settlement date is the date after the issue date when the security is traded to the buyer. |
| maturity | The security's maturity date. The maturity date is the date when the security expires. |
| pr | The security's price per \ \$100 face value. |
| redemption | The security's redemption value per \ \$100 face value. |
| basis | (Optional) The type of day count basis to use. If basis is omitted, it is assumed to be 0. The accepted values are listed below this table. |

The **basis** parameter accepts the following values:

| BASIS | DAY COUNT BASIS |
|--------------|------------------|
| 0 or omitted | US (NASD) 30/360 |
| 1 | Actual/actual |
| 2 | Actual/360 |
| 3 | Actual/365 |
| 4 | European 30/360 |

Return Value

The annual yield.

Remarks

- Dates are stored as sequential serial numbers so they can be used in calculations. In DAX, December 30, 1899 is day 0, and January 1, 2008 is 39448 because it is 39,448 days after December 30, 1899.

- The settlement date is the date a buyer purchases a coupon, such as a bond. The maturity date is the date when a coupon expires. For example, suppose a 30-year bond is issued on January 1, 2008, and is purchased by a buyer six months later. The issue date would be January 1, 2008, the settlement date would be July 1, 2008, and the maturity date would be January 1, 2038, which is 30 years after the January 1, 2008, issue date.
- settlement and maturity are truncated to integers.
- basis is rounded to the nearest integer.
- An error is returned if:
 - settlement or maturity is not a valid date.
 - settlement \geq maturity.
 - pr \leq 0.
 - redemption \leq 0.
 - basis < 0 or basis > 4.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

| | |
|-----------|------------------|
| DATA | - |
| 16-Feb-08 | Settlement date |
| 1-Mar-08 | Maturity date |
| 99.795 | Price |
| \\$100 | Redemption value |
| 2 | Actual/360 basis |

The following DAX query:

```
EVALUATE
{
    YIELDDISC(DATE(2008,2,16), DATE(2008,3,1), 99.795, 100, 2)
}
```

Returns the security's annual yield, given the terms specified above.

| |
|--------------------|
| [VALUE] |
| 0.0528225719868583 |

YIELDMAT

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the annual yield of a security that pays interest at maturity.

Syntax

```
YIELDMAT(<settlement>, <maturity>, <issue>, <rate>, <pr>[, <basis>])
```

Parameters

| TERM | DEFINITION |
|------------|---|
| settlement | The security's settlement date. The security settlement date is the date after the issue date when the security is traded to the buyer. |
| maturity | The security's maturity date. The maturity date is the date when the security expires. |
| issue | The security's issue date. |
| rate | The security's interest rate at date of issue. |
| pr | The security's price per \$100 face value. |
| basis | (Optional) The type of day count basis to use. If basis is omitted, it is assumed to be 0. The accepted values are listed below this table. |

The **basis** parameter accepts the following values:

| BASIS | DAY COUNT BASIS |
|--------------|------------------|
| 0 or omitted | US (NASD) 30/360 |
| 1 | Actual/actual |
| 2 | Actual/360 |
| 3 | Actual/365 |
| 4 | European 30/360 |

Return Value

The annual yield.

Remarks

- Dates are stored as sequential serial numbers so they can be used in calculations. In DAX, December 30, 1899 is day 0, and January 1, 2008 is 39448 because it is 39,448 days after December 30, 1899.
- The settlement date is the date a buyer purchases a coupon, such as a bond. The maturity date is the date when a coupon expires. For example, suppose a 30-year bond is issued on January 1, 2008, and is purchased by a buyer six months later. The issue date would be January 1, 2008, the settlement date would be July 1, 2008, and the maturity date would be January 1, 2038, which is 30 years after the January 1, 2008, issue date.
- settlement, maturity, and issue are truncated to integers.
- basis is rounded to the nearest integer.
- An error is returned if:
 - settlement, maturity, or issue is not a valid date.
 - maturity > settlement > issue is not satisfied.
 - rate < 0.
 - pr ≤ 0.
 - basis < 0 or basis > 4.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

| DATA | DESCRIPTION |
|-----------|---------------------------|
| 15-Mar-08 | Settlement date |
| 3-Nov-08 | Maturity date |
| 8-Nov-07 | Issue date |
| 6.25% | Percent semiannual coupon |
| 100.0123 | Price |
| 0 | 30/360 basis (see above) |

The following DAX query:

```
EVALUATE
{
    YIELDMAT(DATE(2008,3,15), DATE(2008,11,3), DATE(2007,11,8), 0.0625, 100.0123, 0)
}
```

Returns the yield for a security using the terms specified above.

| [VALUE] |
|--------------------|
| 0.0609543336915387 |

Information functions

10/31/2022 • 2 minutes to read • [Edit Online](#)

DAX information functions look at the cell or row that is provided as an argument and tells you whether the value matches the expected type. For example, the ISERROR function returns TRUE if the value that you reference contains an error.

In this category

| FUNCTION | DESCRIPTION |
|-------------------------------------|---|
| COLUMNSTATISTICS | Returns a table of statistics regarding every column in every table in the model. |
| CONTAINS | Returns true if values for all referred columns exist, or are contained, in those columns; otherwise, the function returns false. |
| CONTAINSROW | Returns TRUE if a row of values exists or contained in a table, otherwise returns FALSE. |
| CONTAINSSTRING | Returns TRUE or FALSE indicating whether one string contains another string. |
| CONTAINSSTRINGEXACT | Returns TRUE or FALSE indicating whether one string contains another string. |
| CUSTOMDATA | Returns the content of the CustomData property in the connection string. |
| HASONEFILTER | Returns TRUE when the number of directly filtered values on <i>columnName</i> is one; otherwise returns FALSE. |
| HASONEVALUE | Returns TRUE when the context for <i>columnName</i> has been filtered down to one distinct value only. Otherwise is FALSE. |
| ISAFTER | A boolean function that emulates the behavior of a Start At clause and returns true for a row that meets all of the condition parameters. |
| ISBLANK | Checks whether a value is blank, and returns TRUE or FALSE. |
| ISCROSSFILTERED | Returns TRUE when <i>columnName</i> or another column in the same or related table is being filtered. |
| ISEMPTY | Checks if a table is empty. |
| ISERROR | Checks whether a value is an error, and returns TRUE or FALSE. |
| ISEVEN | Returns TRUE if number is even, or FALSE if number is odd. |

| FUNCTION | DESCRIPTION |
|-----------------------------|---|
| ISFILTERED | Returns TRUE when <i>columnName</i> is being filtered directly. |
| ISINSCOPE | Returns true when the specified column is the level in a hierarchy of levels. |
| ISLOGICAL | Checks whether a value is a logical value, (TRUE or FALSE), and returns TRUE or FALSE. |
| ISNONTEXT | Checks if a value is not text (blank cells are not text), and returns TRUE or FALSE. |
| ISNUMBER | Checks whether a value is a number, and returns TRUE or FALSE. |
| ISODD | Returns TRUE if number is odd, or FALSE if number is even. |
| ISONORAFTER | A boolean function that emulates the behavior of a Start At clause and returns true for a row that meets all of the condition parameters. |
| ISSELECTEDMEASURE | Used by expressions for calculation items to determine the measure that is in context is one of those specified in a list of measures. |
| ISSUBTOTAL | Creates another column in a SUMMARIZE expression that returns True if the row contains subtotal values for the column given as argument, otherwise returns False. |
| ISTEXT | Checks if a value is text, and returns TRUE or FALSE. |
| NONVISUAL | Marks a value filter in a SUMMARIZECOLUMNS expression as non-visual. |
| SELECTEDMEASURE | Used by expressions for calculation items to reference the measure that is in context. |
| SELECTEDMEASUREFORMATSTRING | Used by expressions for calculation items to retrieve the format string of the measure that is in context. |
| SELECTEDMEASURENAME | Used by expressions for calculation items to determine the measure that is in context by name. |
| USERCULTURE | Returns the locale for the current user. |
| USERNAME | Returns the domain name and username from the credentials given to the system at connection time. |
| USEROBJECTID | Returns the current user's Object ID or SID. |
| USERPRINCIPALNAME | Returns the user principal name. |

COLUMNSTATISTICS

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns a table of statistics regarding every column in every table in the model.

Syntax

```
COLUMNSTATISTICS ()
```

Parameters

This function does not take any parameters.

Return value

A table of statistics. Each row of this table represents a different column in the model. Table columns include:

- **Table Name:** The current column's table.
- **Column Name:** The current column's name.
- **Min:** The minimum value found within the current column.
- **Max:** The maximum value found within the current column.
- **Cardinality:** The number of distinct values found within the current column.
- **Max Length:** The length of the longest string found within the current column (only applicable for string columns).

Remarks

- Columns in an error state and columns from query-scope calculated tables do not appear in the result table.
- If a filter from the filter context is applied to COLUMNSTATISTICS(), an error is returned.

Example

Examples in this article can be used with the sample Adventure Works DW 2020 Power BI Desktop model. To get the model, see [DAX sample model](#).

The following DAX query:

```
DEFINE
    TABLE FilteredProduct =
        FILTER (
            Product,
            [Color] == "Blue"
        )
    COLUMN Customer[Location] = [State-Province] & " " & [Country-Region]

EVALUATE
    COLUMNSTATISTICS ()
```

Returns a table with statistics regarding all columns from all tables in the model. The table also includes statistics

for the query-scope calculated column, Customer[Location]. However, the table does not include the columns from the query-scope calculated table, FilteredProduct.

| Table Name | Column Name | Min | Max | Cardinality | Max Length |
|-----------------|--|-----------------------------------|-----------------------|-------------|------------|
| Customer | RowNumber-2662979B-1795-4F74-8F37-6A1BA8059B61 | | | 18485 | |
| Customer | CustomerKey | -1 | 29483 | 18485 | |
| Customer | Customer ID | [Not Applicable] | AW00029483 | 18485 | 16 |
| Customer | Customer | [Not Applicable] | Zoe Watson | 18401 | 26 |
| Customer | City | [Not Applicable] | York | 270 | 21 |
| Customer | State-Province | [Not Applicable] | Yveline | 54 | 19 |
| Customer | Country-Region | [Not Applicable] | United States | 7 | 16 |
| Customer | Postal Code | [Not Applicable] | YO15 | 324 | 16 |
| Customer | Location | [Not Applicable] [Not Applicable] | Yveline France | 54 | 33 |
| Date | RowNumber-2662979B-1795-4F74-8F37-6A1BA8059B61 | | | 1461 | |
| Date | DateKey | 20170701 | 20210630 | 1461 | |
| Date | Date | 7/1/2017 12:00:00 AM | 6/30/2021 12:00:00 AM | 1461 | |
| Date | Fiscal Year | FY2018 | FY2021 | 4 | 6 |
| Date | Fiscal Quarter | FY2018 Q1 | FY2021 Q4 | 16 | 9 |
| Date | Month | 2017 Aug | 2021 May | 48 | 8 |
| Date | MonthKey | 201707 | 202106 | 48 | |
| Date | Full Date | 2017 Aug, 01 | 2021 May, 31 | 1461 | 12 |
| Sales Territory | RowNumber-2662979B-1795-4F74-8F37-6A1BA8059B61 | | | 11 | |
| Sales Territory | SalesTerritoryKey | 1 | 11 | 11 | |
| Sales Territory | SalesTerritoryRegion | Australia | United Kingdom | 11 | |

See also

[Filter context](#)

[CALCULATETABLE function](#)

CONTAINS

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns true if values for all referred columns exist, or are contained, in those columns; otherwise, the function returns false.

Syntax

```
CONTAINS(<table>, <columnName>, <value>[, <columnName>, <value>]...)
```

Parameters

| TERM | DEFINITION |
|------------|---|
| table | Any DAX expression that returns a table of data. |
| columnName | The name of an existing column, using standard DAX syntax. It cannot be an expression. |
| value | Any DAX expression that returns a single scalar value, that is to be sought in <i>columnName</i> . The expression is to be evaluated exactly once and before it is passed to the argument list. |

Return value

A value of **TRUE** if each specified *value* can be found in the corresponding *columnName*, or are contained, in those columns; otherwise, the function returns **FALSE**.

Remarks

- The arguments *columnName* and *value* must come in pairs; otherwise an error is returned.
- *columnName* must belong to the specified *table*, or to a table that is related to *table*.
- If *columnName* refers to a column in a related table then it must be fully qualified; otherwise, an error is returned.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following example creates a measure that tells you whether there were any Internet sales of product 214 and to customer 11185 at the same time.

```
= CONTAINS(InternetSales, [ProductKey], 214, [CustomerKey], 11185)
```

CONTAINSROW function

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns TRUE if there exists at least one row where all columns have specified values.

Syntax

```
CONTAINSROW(<Table>, <Value> [, <Value> [, ...] ] )
```

Parameters

| TERM | DEFINITION |
|-------|---|
| Table | A table to test. |
| Value | Any valid DAX expression that returns a scalar value. |

Return value

TRUE or FALSE.

Remarks

- Except syntax, the IN operator and CONTAINSROW function are functionally equivalent.

```
<scalarExpr> IN <tableExpr>  
( <scalarExpr1>, <scalarExpr2>, ... ) IN <tableExpr>
```

- The number of scalarExprN must match the number of columns in tableExpr.
 - NOT IN is not an operator in DAX. To perform the logical negation of the IN operator, put NOT in front of the entire expression. For example, NOT [Color] IN { "Red", "Yellow", "Blue" }.
- Unlike the = operator, the IN operator and the CONTAINSROW function perform strict comparison. For example, the BLANK value does not match 0.

Examples

Examples in this article can be used with the sample Adventure Works DW 2020 Power BI Desktop model. To get the model, see [DAX sample model](#).

Example 1

The following DAX queries:


```

EVALUATE
FILTER (
    ALL ( Product[Color] ),
    ( [Color] )
    IN {
        "Red",
        "Yellow",
        "Blue"
    }
)
ORDER BY [Color]

```

and

```

EVALUATE
FILTER (
    ALL ( Product[Color] ),
    CONTAINSROW (
        {
            "Red",
            "Yellow",
            "Blue"
        },
        [Color]
    )
)
ORDER BY [Color]

```

Return the following table with a single column:

| [COLOR] |
|---------|
| Blue |
| Red |
| Yellow |

Example 2

The following equivalent DAX queries:

```

EVALUATE
FILTER (
    ALL ( Product[Color] ),
    NOT [Color]
    IN {
        "Red",
        "Yellow",
        "Blue"
    }
)
ORDER BY [Color]

```

and

```
EVALUATE
FILTER (
    ALL ( Product[Color] ),
    NOT CONTAINSROW (
        {
            "Red",
            "Yellow",
            "Blue"
        },
        [Color]
    )
)
ORDER BY [Color]
```

Return the following table with a single column:

| [COLOR] |
|--------------|
| Black |
| Grey |
| Multi |
| NA |
| Silver |
| Silver\Black |
| White |

See also

[IN operator](#)

[DAX queries](#)

CONTAINSSTRING

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns TRUE or FALSE indicating whether one string contains another string.

Syntax

```
CONTAINSSTRING(<within_text>, <find_text>)
```

Parameters

| TERM | DEFINITION |
|-------------|---|
| within_text | The text in which you want to search for find_text. |
| find_text | The text you want to find. |

Return value

TRUE if find_text is a substring of within_text; otherwise FALSE.

Remarks

- CONTAINSSTRING is not case-sensitive.
- You can use `?` and `*` wildcard characters. Use `~` to escape wildcard characters.

Example

DAX query

```
EVALUATE
    ROW(
        "Case 1", CONTAINSSTRING("abcd", "bc"),
        "Case 2", CONTAINSSTRING("abcd", "BC"),
        "Case 3", CONTAINSSTRING("abcd", "a*d"),
        "Case 4", CONTAINSSTRING("abcd", "ef")
    )
```

Returns

| [CASE 1] | [CASE 2] | [CASE 3] | [CASE 4] |
|----------|----------|----------|----------|
| TRUE | TRUE | TRUE | FALSE |

CONTAINSSTRINGEXACT

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns TRUE or FALSE indicating whether one string contains another string.

Syntax

```
CONTAINSSTRINGEXACT(<within_text>, <find_text>)
```

Parameters

| TERM | DEFINITION |
|-------------|---|
| within_text | The text in which you want to search for find_text. |
| find_text | The text you want to find. |

Return value

TRUE if find_text is a substring of within_text; otherwise FALSE.

Remarks

CONTAINSSTRINGEXACT is case-sensitive.

Example

DAX query

```
EVALUATE
    ROW(
        "Case 1", CONTAINSSTRINGEXACT("abcd", "bc"),
        "Case 2", CONTAINSSTRINGEXACT("abcd", "BC"),
        "Case 3", CONTAINSSTRINGEXACT("abcd", "a*d"),
        "Case 4", CONTAINSSTRINGEXACT("abcd", "ef")
    )
```

Returns

| [CASE 1] | [CASE 2] | [CASE 3] | [CASE 4] |
|----------|----------|----------|----------|
| TRUE | FALSE | FALSE | FALSE |

CUSTOMDATA

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the content of the **CustomData** property in the connection string.

Syntax

```
CUSTOMDATA()
```

Return value

The content of the **CustomData** property in the connection string.

Blank, if **CustomData** property was not defined at connection time.

Remarks

This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following DAX formula verifies if the CustomData property was set to "OK".

```
= IF(CUSTOMDATA()="OK", "Correct Custom data in connection string", "No custom data in connection string  
property or unexpected value")
```

HASONEFILTER

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns **TRUE** when the number of directly filtered values on *columnName* is one; otherwise returns **FALSE**.

Syntax

```
HASONEFILTER(<columnName>)
```

Parameters

| TERM | DEFINITION |
|------------|--|
| columnName | The name of an existing column, using standard DAX syntax. It cannot be an expression. |

Return value

TRUE when the number of directly filtered values on *columnName* is one; otherwise returns **FALSE**.

Remarks

- This function is similar to HASONEVALUE() with the difference that HASONEVALUE() works based on cross-filters while HASONEFILTER() works by a direct filter.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following example shows how to use HASONEFILTER() to return the filter for ResellerSales_USD[ProductKey] if there is one filter, or to return BLANK if there are no filters or more than one filter on ResellerSales_USD[ProductKey].

```
= IF(HASONEFILTER(ResellerSales_USD[ProductKey]),FILTERS(ResellerSales_USD[ProductKey]),BLANK())
```

HASONEVALUE

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns **TRUE** when the context for *columnName* has been filtered down to one distinct value only. Otherwise is **FALSE**.

Syntax

```
HASONEVALUE(<columnName>)
```

Parameters

| TERM | DEFINITION |
|------------|--|
| columnName | The name of an existing column, using standard DAX syntax. It cannot be an expression. |

Return value

TRUE when the context for *columnName* has been filtered down to one distinct value only. Otherwise is **FALSE**.

Remarks

- An equivalent expression for HASONEVALUE() is `COUNTROWS(VALUES(<columnName>)) = 1`.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following measure formula verifies if the context is being sliced by one value in order to estimate a percentage against a predefined scenario; in this case you want to compare Reseller Sales against sales in 2007, then you need to know if the context is filtered by single years. Also, if the comparison is meaningless you want to return BLANK.

```
=  
IF(HASONEVALUE(DateTime[CalendarYear]),SUM(ResellerSales_USD[SalesAmount_USD])/CALCULATE(SUM(ResellerSales_USD[SalesAmount_USD]),DateTime[CalendarYear]=2007),BLANK())
```

ISAFter

10/31/2022 • 2 minutes to read • [Edit Online](#)

A boolean function that emulates the behavior of a 'Start At' clause and returns true for a row that meets all of the condition parameters.

Based on the sort order, the first parameter is compared with the second parameter. If the sort order is ascending, the comparison to be done is first parameter greater than the second parameter. If the sort order is descending, the comparison to be done is second parameter less than the first parameter.

Syntax

```
ISAFter(<scalar_expression>, <scalar_expression>[, sort_order [, <scalar_expression>, <scalar_expression>[, sort_order]]...)
```

Parameters

| TERM | DEFINITION |
|-------------------|--|
| scalar expression | Any expression that returns a scalar value like a column reference or integer or string value. Typically the first parameter is a column reference and the second parameter is a scalar value. |
| sort order | (optional) The order in which the column is sorted. Can be ascending (ASC) or descending (DEC). By default the sort order is ascending. |

Return value

True or false.

Remarks

This function is similar to [ISONORAFTER](#). The difference is ISAFter returns true for values sorted strictly *after* the filter values, where ISONORAFTER returns true for values sorted *on or after* the filter values.

This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

Table name: 'Info'

| COUNTRY | STATE | COUNT | TOTAL |
|---------|-------|-------|-------|
| IND | JK | 20 | 800 |
| IND | MH | 25 | 1000 |

| COUNTRY | STATE | COUNT | TOTAL |
|---------|-------|-------|-------|
| IND | WB | 10 | 900 |
| USA | CA | 5 | 500 |
| USA | WA | 10 | 900 |

The following expression:

```

FILTER (
  Info,
  ISAFter (
    Info[Country], "IND", ASC,
    Info[State], "MH", ASC )
)
```

Returns:

| COUNTRY | STATE | COUNT | TOTAL |
|---------|-------|-------|-------|
| IND | WB | 10 | 900 |
| USA | CA | 5 | 500 |
| USA | WA | 10 | 900 |

See also

[ISONORAFter](#)

ISBLANK

10/31/2022 • 2 minutes to read • [Edit Online](#)

Checks whether a value is blank, and returns TRUE or FALSE.

Syntax

```
ISBLANK(<value>)
```

Parameters

| TERM | DEFINITION |
|-------|---|
| value | The value or expression you want to test. |

Return value

A Boolean value of TRUE if the value is blank; otherwise FALSE.

Remarks

To learn more about best practices when working with BLANKS, see [Avoid converting BLANKs to values in DAX](#).

Example

This formula computes the increase or decrease ratio in sales compared to the previous year. The example uses the IF function to check the value for the previous year's sales in order to avoid a divide by zero error.

```
//Sales to Previous Year Ratio  
  
= IF( ISBLANK('CalculatedMeasures'[PreviousYearTotalSales])  
    , BLANK()  
    , ( 'CalculatedMeasures'[Total Sales]-'CalculatedMeasures'[PreviousYearTotalSales] )  
      /'CalculatedMeasures'[PreviousYearTotalSales])
```

Result,

| ROW LABELS | TOTAL SALES | TOTAL SALES PREVIOUS YEAR | SALES TO PREVIOUS YEAR RATIO |
|-------------|------------------|---------------------------|------------------------------|
| 2005 | \$10,209,985.08 | | |
| 2006 | \$28,553,348.43 | \$10,209,985.08 | 179.66% |
| 2007 | \$39,248,847.52 | \$28,553,348.43 | 37.46% |
| 2008 | \$24,542,444.68 | \$39,248,847.52 | -37.47% |
| Grand Total | \$102,554,625.71 | | |

See also

[Information functions](#)

ISCROSSFILTERED

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns TRUE when the specified table or column is cross-filtered.

Syntax

```
ISCROSSFILTERED(<TableNameOrColumnName>)
```

Parameters

| TERM | DEFINITION |
|-----------------------|--|
| TableNameOrColumnName | The name of an existing table or column. It cannot be an expression. |

Return value

TRUE when *ColumnName* or a column of *TableName* is being cross-filtered. Otherwise returns FALSE.

Remarks

- A column or table is said to be cross-filtered when a filter is applied to *ColumnName*, any column of *TableName*, or to any column of a related table.
- A column or table is said to be filtered directly when a filter is applied to *ColumnName* or to any column of *TableName*. Therefore, the [ISFILTERED](#) function also returns TRUE when *ColumnName* or any column of *TableName* is filtered.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

See also

[ISFILTERED](#) function

[FILTERS](#) function

[HASONEFILTER](#) function

[HASONEVALUE](#) function

IEMPTY

10/31/2022 • 2 minutes to read • [Edit Online](#)

Checks if a table is empty.

Syntax

```
IEMPTY(<table_expression>)
```

Parameters

| TERM | DEFINITION |
|------------------|---|
| table_expression | A table reference or a DAX expression that returns a table. |

Return value

True if the table is empty (has no rows), if else, False.

Remarks

This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

For the below table named 'Info':

| COUNTRY | STATE | COUNTY | TOTAL |
|---------|-------|--------|-------|
| IND | JK | 20 | 800 |
| IND | MH | 25 | 1000 |
| IND | WB | 10 | 900 |
| USA | CA | 5 | 500 |
| USA | WA | 10 | 900 |

```
EVALUATE  
ROW("Any countries with count > 25?", NOT(IEMPTY(FILTER(Info, [County]>25))))
```

Return value: **FALSE**

ISERROR

10/31/2022 • 2 minutes to read • [Edit Online](#)

Checks whether a value is an error, and returns TRUE or FALSE.

Syntax

```
ISERROR(<value>)
```

Parameters

| TERM | DEFINITION |
|-------|-----------------------------|
| value | The value you want to test. |

Return value

A Boolean value of TRUE if the value is an error; otherwise FALSE.

Remarks

- For best practices when using ISERROR, see [Appropriate use of error functions](#).
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following example calculates the ratio of total Internet sales to total reseller sales. The ISERROR function is used to check for errors, such as division by zero. If there is an error a blank is returned, otherwise the ratio is returned.

```
= IF( ISERROR(  
    SUM('ResellerSales_USD'[SalesAmount_USD])  
    /SUM('InternetSales_USD'[SalesAmount_USD])  
    )  
    , BLANK()  
    , SUM('ResellerSales_USD'[SalesAmount_USD])  
    /SUM('InternetSales_USD'[SalesAmount_USD])  
    )
```

See also

[Information functions](#)

[IFERROR function](#)

[IF function](#)

ISEVEN

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns TRUE if number is even, or FALSE if number is odd.

Syntax

```
ISEVEN(number)
```

Parameters

| TERM | DEFINITION |
|--------|--|
| number | The value to test. If number is not an integer, it is truncated. |

Return value

Returns TRUE if number is even, or FALSE if number is odd.

Remarks

- If number is nonnumeric, ISEVEN returns the #VALUE! error value.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

ISFILTERED

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns TRUE when the specified table or column is being filtered directly.

Syntax

```
ISFILTERED(<TableNameOrColumnName>)
```

Parameters

| TERM | DEFINITION |
|-----------------------|--|
| TableNameOrColumnName | The name of an existing table or column. It cannot be an expression. |

Return value

TRUE when *ColumnName* or a column of *TableName* is being filtered directly. Otherwise returns FALSE.

Remarks

- A column or table is said to be filtered directly when a filter is applied to *ColumnName* or any column of *TableName*.
- A column or table is said to be cross-filtered when a filter is applied to *ColumnName*, any column of *TableName*, or to any column of a related table. Therefore, the [ISCROSSFILTERED](#) function also returns TRUE when *ColumnName*, any column of *TableName*, or a column of a related table is filtered.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

See also

[ISCROSSFILTERED](#) function

[FILTERS](#) function

[HASONEFILTER](#) function

[HASONEVALUE](#) function

ISINSCOPE

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns true when the specified column is the level in a hierarchy of levels.

Syntax

```
ISINSCOPE(<columnName>)
```

Parameters

| TERM | DEFINITION |
|------------|--|
| columnName | The name of an existing column, using standard DAX syntax. It cannot be an expression. |

Return value

TRUE when the specified column is the level in a hierarchy of levels.

Remarks

This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

```

DEFINE
MEASURE FactInternetSales[% of Parent] =
    SWITCH (TRUE(),
        ISINSCOPE(DimProduct[Subcategory]),
            DIVIDE(
                SUM(FactInternetSales[Sales Amount]),
                CALCULATE(
                    SUM(FactInternetSales[Sales Amount]),
                    ALLSELECTED(DimProduct[Subcategory]))
            ),
        ISINSCOPE(DimProduct[Category]),
            DIVIDE(
                SUM(FactInternetSales[Sales Amount]),
                CALCULATE(
                    SUM(FactInternetSales[Sales Amount]),
                    ALLSELECTED(DimProduct[Category]))
            ),
        1
    ) * 100
EVALUATE
    SUMMARIZECOLUMNS
    (
        ROLLUPADDISSUBTOTAL
        (
            DimProduct[Category], "Category Subtotal",
            DimProduct[Subcategory], "Subcategory Subtotal"
        ),
        TREATAS(
            {"Bike Racks", "Bike Stands", "Mountain Bikes", "Road Bikes", "Touring Bikes"},
            DimProduct[Subcategory]),
        "Sales", SUM(FactInternetSales[Sales Amount]),
        "% of Parent", [% of Parent]
    )
ORDER BY
    [Category Subtotal] DESC, [Category],
    [Subcategory Subtotal] DESC, [Subcategory]

```

Returns,

| DIMPRODUCT[C ATEGORY] | DIMPRODUCT[S UBCATEGORY] | [CATEGORY SUBTOTAL] | [SUBCATEGORY SUBTOTAL] | [SALES] | [% OF PARENT] |
|--------------------------|-----------------------------|------------------------|---------------------------|---------------|---------------|
| | | TRUE | TRUE | 28,397,095.65 | 100.00 |
| Accessories | | FALSE | TRUE | 78,951.00 | 0.28 |
| Accessories | Bike Racks | FALSE | FALSE | 39,360.00 | 49.85 |
| Accessories | Bike Stands | FALSE | FALSE | 39,591.00 | 50.15 |
| Bikes | | FALSE | TRUE | 28,318,144.65 | 99.72 |
| Bikes | Mountain Bikes | FALSE | FALSE | 9,952,759.56 | 35.15 |
| Bikes | Road Bikes | FALSE | FALSE | 14,520,584.04 | 51.28 |
| Bikes | Touring Bikes | FALSE | FALSE | 3,844,801.05 | 13.58 |

See also

SUMMARIZECOLUMNS function

CALCULATE function

ISLOGICAL

10/31/2022 • 2 minutes to read • [Edit Online](#)

Checks whether a value is a logical value, (TRUE or FALSE), and returns TRUE or FALSE.

Syntax

```
ISLOGICAL(<value>)
```

Parameters

| TERM | DEFINITION |
|-------|----------------------------------|
| value | The value that you want to test. |

Return value

TRUE if the value is a logical value; FALSE if any value other than TRUE OR FALSE.

Remarks

This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following three samples show the behavior of ISLOGICAL.

```
//RETURNS: Is Boolean type or Logical
= IF(ISLOGICAL(true), "Is Boolean type or Logical", "Is different type")

//RETURNS: Is Boolean type or Logical
= IF(ISLOGICAL(false), "Is Boolean type or Logical", "Is different type")

//RETURNS: Is different type
= IF(ISLOGICAL(25), "Is Boolean type or Logical", "Is different type")
```

See also

[Information functions](#)

ISNONTEXT

10/31/2022 • 2 minutes to read • [Edit Online](#)

Checks if a value is not text (blank cells are not text), and returns TRUE or FALSE.

Syntax

```
ISNONTEXT(<value>)
```

Parameters

| TERM | DEFINITION |
|-------|------------------------------|
| value | The value you want to check. |

Return value

TRUE if the value is not text or blank; FALSE if the value is text.

Remarks

- An empty string is considered text.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following examples show the behavior of the ISNONTEXT function.

```
//RETURNS: Is Non-Text
= IF(ISNONTEXT(1), "Is Non-Text", "Is Text")

//RETURNS: Is Non-Text
= IF(ISNONTEXT(BLANK()), "Is Non-Text", "Is Text")

//RETURNS: Is Text
= IF(ISNONTEXT(""), "Is Non-Text", "Is Text")
```

See also

[Information functions](#)

ISNUMBER

10/31/2022 • 2 minutes to read • [Edit Online](#)

Checks whether a value is a number, and returns TRUE or FALSE.

Syntax

```
ISNUMBER(<value>)
```

Parameters

| TERM | DEFINITION |
|-------|-----------------------------|
| value | The value you want to test. |

Return value

TRUE if the value is numeric; otherwise FALSE.

Remarks

This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following three samples show the behavior of ISNUMBER.

```
//RETURNS: Is number
= IF(ISNUMBER(0), "Is number", "Is Not number")

//RETURNS: Is number
= IF(ISNUMBER(3.1E-1), "Is number", "Is Not number")

//RETURNS: Is Not number
= IF(ISNUMBER("123"), "Is number", "Is Not number")
```

See also

[Information functions](#)

ISODD

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns TRUE if number is odd, or FALSE if number is even.

Syntax

```
ISODD(number)
```

Parameters

| TERM | DEFINITION |
|--------|--|
| number | The value to test. If number is not an integer, it is truncated. |

Return value

Returns TRUE if number is odd, or FALSE if number is even.

Remarks

- If number is nonnumeric, ISODD returns the #VALUE! error value.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

ISONORAFTER

10/31/2022 • 2 minutes to read • [Edit Online](#)

A boolean function that emulates the behavior of a Start At clause and returns true for a row that meets all of the condition parameters.

Based on the sort order, the first parameter is compared with the second parameter. If the sort order is ascending, the comparison to be done is first parameter greater than the second parameter. If the sort order is descending, the comparison to be done is second parameter less than the first parameter.

Syntax

```
ISONORAFTER(<scalar_expression>, <scalar_expression>[, sort_order [, <scalar_expression>,  
<scalar_expression>[, sort_order]]...)
```

Parameters

| TERM | DEFINITION |
|-------------------|--|
| scalar expression | Any expression that returns a scalar value like a column reference or integer or string value. Typically the first parameter is a column reference and the second parameter is a scalar value. |
| sort order | (optional) The order in which the column is sorted. Can be ascending (ASC) or descending (DESC). By default the sort order is ascending. |

Return value

True or false.

Remarks

This function is similar to [ISAFter](#). The difference is ISONORAFTER returns true for values sorted *on or after* the filter values, where ISAFter returns true for values sorted strictly *after* the filter values.

This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

For the following table named, Info:

| COUNTRY | STATE | COUNT | TOTAL |
|---------|-------|-------|-------|
| IND | JK | 20 | 800 |
| IND | MH | 25 | 1000 |

| COUNTRY | STATE | COUNT | TOTAL |
|---------|-------|-------|-------|
| IND | WB | 10 | 900 |
| USA | CA | 5 | 500 |
| USA | WA | 10 | 900 |

The following expression:

```

FILTER (
    Info,
    ISONORAFTER (
        Info[Country], "IND", ASC,
        Info[State], "MH", ASC )
)
```

Returns:

| COUNTRY | STATE | COUNT | TOTAL |
|---------|-------|-------|-------|
| IND | MH | 25 | 1000 |
| IND | WB | 10 | 900 |
| USA | CA | 5 | 500 |
| USA | WA | 10 | 900 |

See also

[ISAFATER](#)

ISSELECTEDMEASURE

10/31/2022 • 2 minutes to read • [Edit Online](#)

Used by expressions for calculation items to determine the measure that is in context is one of those specified in a list of measures.

Syntax

```
ISSELECTEDMEASURE( M1, M2, ... )
```

Parameters

| TERM | DEFINITION |
|-------------|---------------------|
| M1, M2, ... | A list of measures. |

Return value

A Boolean indicating whether the measure that is currently in context is one of those specified in the list of parameters.

Remarks

- Can only be referenced in the expression for a calculation item.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following calculation item expression checks if the current measure is one of those specified in the list of parameters. If the measures are renamed, formula fixup will reflect the name changes in the expression.

```
IF (
    ISSELECTEDMEASURE ( [Expense Ratio 1], [Expense Ratio 2] ),
    SELECTEDMEASURE (),
    DIVIDE ( SELECTEDMEASURE (), COUNTROWS ( DimDate ) )
)
```

See also

[SELECTEDMEASURE](#)

[SELECTEDMEASURENAME](#)

ISSUBTOTAL

10/31/2022 • 2 minutes to read • [Edit Online](#)

Creates another column in a [SUMMARIZE](#) expression that returns True if the row contains subtotal values for the column given as argument, otherwise returns False.

Syntax

```
ISSUBTOTAL(<columnName>)
```

With [SUMMARIZE](#),

```
SUMMARIZE(<table>, <groupBy_columnName>[, <groupBy_columnName>]...[, ROLLUP(<groupBy_columnName>[, <groupBy_columnName>...])][, <name>, {<expression>|ISSUBTOTAL(<columnName>)}]...)
```

Parameters

| TERM | DEFINITION |
|------------|--|
| columnName | The name of any column in table of the SUMMARIZE function or any column in a related table to table. |

Return value

A True value if the row contains a subtotal value for the column given as argument, otherwise returns False.

Remarks

- This function can only be used in the expression of a [SUMMARIZE](#) function.
- This function must be preceded by the name of the Boolean column.

Example

See [SUMMARIZE](#).

ISTEXT

10/31/2022 • 2 minutes to read • [Edit Online](#)

Checks if a value is text, and returns TRUE or FALSE.

Syntax

```
ISTEXT(<value>)
```

Parameters

| TERM | DEFINITION |
|-------|------------------------------|
| value | The value you want to check. |

Return value

TRUE if the value is text; otherwise FALSE.

Remarks

This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following examples show the behavior of the ISTEXT function.

```
//RETURNS: Is Text
= IF(ISTEXT("text"), "Is Text", "Is Non-Text")

//RETURNS: Is Text
= IF(ISTEXT(""), "Is Text", "Is Non-Text")

//RETURNS: Is Non-Text
= IF(ISTEXT(1), "Is Text", "Is Non-Text")

//RETURNS: Is Non-Text
= IF(ISTEXT(BLANK()), "Is Text", "Is Non-Text")
```

See also

[Information functions](#)

NONVISUAL

10/31/2022 • 2 minutes to read • [Edit Online](#)

Marks a value filter in a [SUMMARIZECOLUMNS](#) expression as non-visual. This function can only be used within a [SUMMARIZECOLUMNS](#) expression.

Syntax

```
NONVISUAL(<expression>)
```

Parameters

| TERM | DEFINITION |
|------------|---|
| expression | Any DAX expression that returns a single value (not a table). |

Return value

A table of values.

Remarks

- Marks a value filter in [SUMMARIZECOLUMNS](#) as not affecting measure values, but only applying to group-by columns.
- This function can only be used within a [SUMMARIZECOLUMNS](#) expression. It's used as either a filterTable argument of the [SUMMARIZECOLUMNS](#) function or a groupLevelFilter argument of the [ROLLUPADDISSUBTOTAL](#) or [ROLLUPISSUBTOTAL](#) function.

Example

See [SUMMARIZECOLUMNS](#).

SELECTEDMEASURE

10/31/2022 • 2 minutes to read • [Edit Online](#)

Used by expressions for calculation items to reference the measure that is in context.

Syntax

```
SELECTEDMEASURE()
```

Parameters

None

Return value

A reference to the measure that is currently in context when the calculation item is evaluated.

Remarks

- Can only be referenced in the expression for a calculation item.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following calculation item expression calculates the year-to-date for whatever the measure is in context.

```
CALCULATE(SELECTEDMEASURE(), DATESYTD(DimDate[Date]))
```

See also

[SELECTEDMEASURENAME](#)

[ISSELECTEDMEASURE](#)

SELECTEDMEASUREFORMATSTRING

10/31/2022 • 2 minutes to read • [Edit Online](#)

Used by expressions for calculation items to retrieve the format string of the measure that is in context.

Syntax

```
SELECTEDMEASUREFORMATSTRING()
```

Parameters

None

Return value

A string holding the format string of the measure that is currently in context when the calculation item is evaluated.

Remarks

- This function can only be referenced in expressions for calculation items in calculation groups. It is designed to be used by the **Format String Expression** property of calculation items.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following expression is evaluated by the Format String Expression property for a calculation item. If there is a single currency in filter context, the format string is retrieved from the DimCurrency[FormatString] column; otherwise the format string of the measure in context is used.

```
SELECTEDVALUE( DimCurrency[FormatString], SELECTEDMEASUREFORMATSTRING() )
```

See also

[SELECTEDMEASURE](#)
[ISSELECTEDMEASURE](#)

SELECTEDMEASURENAME

10/31/2022 • 2 minutes to read • [Edit Online](#)

Used by expressions for calculation items to determine the measure that is in context by name.

Syntax

```
SELECTEDMEASURENAME()
```

Parameters

None

Return value

A string value holding the name of the measure that is currently in context when the calculation item is evaluated.

Remarks

- Can only be referenced in the expression for a calculation item.
- This function is often used for debugging purposes when authoring calculation groups.

Example

The following calculation item expression checks if the current measure is Expense Ratio and conditionally applies calculation logic. Since the check is based on a string comparison, it is not subject to formula fixup and will not benefit from object renaming being automatically reflected. For a similar comparison that would benefit from formula fixup, please see the `ISSELECTEDMEASURE` function instead.

```
IF (
    SELECTEDMEASURENAME = "Expense Ratio",
    SELECTEDMEASURE (),
    DIVIDE ( SELECTEDMEASURE (), COUNTROWS ( DimDate ) )
)
```

See also

[SELECTEDMEASURE](#)
[ISSELECTEDMEASURE](#)

USERCULTURE

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the locale (language code-country code) for the current user, determined by the operating system, browser settings, or Power BI service.

Note: This function is currently supported in Power BI Premium per capacity, Power BI Premium per user, and Power BI Embedded only.

Syntax

```
USERCULTURE()
```

Parameters

This expression has no parameters.

Return value

Locale as a string.

Remarks

- In the Power BI service, locale is determined by **Settings > Language > Language Settings**. The default is determined by the user's browser language setting.
- When used in calculated table and calculated column expressions, the result may differ depending on whether the table is in DirectQuery or Import mode. When in DirectQuery mode, the result is determined by the language (locale) specified in Language Settings in the Power BI service. The default in Language Settings specifies locale is determined by the user's browser language setting, which means the same calculated table or column can return different results depending on the browser language settings for each user. When in Import mode, the result is statically determined during refresh and will not vary at query time. For managed refreshes, such as scheduled or interactive, locale is not based on the user's browser language setting but instead uses an invariant locale. The invariant locale, however, can be overridden by using the XMLA endpoint to specify a custom locale.
- When combined with the Field parameters feature in Power BI, USERCULTURE can be used to reliably translate dynamic visualization titles and captions when used in measure and row-level security (RLS) object expressions within the same model. However, expressions containing USERCULTURE called from outside the model, such as queries and live-connect report measures, should not be relied upon for correctly translated titles and captions.
- USERCULTURE returns the correct user locale when used in object expressions called from within the model such as measures, row-level security (RLS), and calculation items. However, it may not return the correct user locale when used in expressions from outside the model, such as queries and live-connect report measures.
- In Live-connect reports, USERCULTURE may not return the correct user locale when called from a report measure expression.

Example

For the following expression,

```
FORMAT(TODAY(), "dddd", USERCULTURE())
```

Depending on the language setting for the current user, USERCULTURE returns the current day, for example:

| LOCALE | FORMATTED WEEKDAY |
|--------------|-------------------|
| de-DE | Dienstag |
| en-US | Tuesday |
| es-ES_tradnl | martes |
| eu-ES | asteartea |
| it-IT | martedì |
| nl-NL | dinsdag |
| pl-PL | wtorek |
| ro-RO | marți |
| ru-RU | вторник |
| uk-UA | вівторок |

See also

[Expression-based titles in Power BI](#)

[USERNAME](#)

[USERPRINCIPALNAME](#)

[USEROBJECTID](#)

USERNAME

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the domain name and username from the credentials given to the system at connection time.

Syntax

```
USERNAME()
```

Parameters

This expression has no parameters.

Return value

The username from the credentials given to the system at connection time

Example

The following formula verifies if the user login is part of the UsersTable.

```
= IF(CONTAINS(UsersTable,UsersTable[login], USERNAME()), "Allowed", BLANK())
```

USEROBJECTID

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the current user's Object ID from Azure AD or security identifier (SID).

Syntax

```
USEROBJECTID()
```

Parameters

This expression has no parameters.

Return value

The current user's Object ID from Azure AD for Power BI or Azure Analysis Services models or SID for SQL Server Analysis Services models.

USERPRINCIPALNAME

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the user principal name.

Syntax

```
USERPRINCIPALNAME()
```

Parameters

This expression has no parameters.

Return value

The userprincipalname at connection time.

Logical functions

10/31/2022 • 2 minutes to read • [Edit Online](#)

Logical functions act upon an expression to return information about the values or sets in the expression. For example, you can use the IF function to check the result of an expression and create conditional results.

In this category

| FUNCTION | DESCRIPTION |
|---------------------------|--|
| AND | Checks whether both arguments are TRUE, and returns TRUE if both arguments are TRUE. |
| BITAND | Returns a bitwise 'AND' of two numbers. |
| BITLSHIFT | Returns a number shifted left by the specified number of bits. |
| BITOR | Returns a bitwise 'OR' of two numbers. |
| BITRSHIFT | Returns a number shifted right by the specified number of bits. |
| BITXOR | Returns a bitwise 'XOR' of two numbers. |
| COALESCE | Returns the first expression that does not evaluate to BLANK. |
| FALSE | Returns the logical value FALSE. |
| IF | Checks a condition, and returns one value when TRUE, otherwise it returns a second value. |
| IF.EAGER | Checks a condition, and returns one value when TRUE, otherwise it returns a second value. Uses an <i>eager</i> execution plan which always executes the branch expressions regardless of the condition expression. |
| IFERROR | Evaluates an expression and returns a specified value if the expression returns an error |
| NOT | Changes FALSE to TRUE, or TRUE to FALSE. |
| OR | Checks whether one of the arguments is TRUE to return TRUE. |
| SWITCH | Evaluates an expression against a list of values and returns one of multiple possible result expressions. |
| TRUE | Returns the logical value TRUE. |

AND

10/31/2022 • 2 minutes to read • [Edit Online](#)

Checks whether both arguments are TRUE, and returns TRUE if both arguments are TRUE. Otherwise returns false.

Syntax

```
AND(<logical1>,<logical2>)
```

Parameters

| TERM | DEFINITION |
|----------------------|--------------------------------------|
| logical_1, logical_2 | The logical values you want to test. |

Return value

Returns true or false depending on the combination of values that you test.

Remarks

The **AND** function in DAX accepts only two (2) arguments. If you need to perform an AND operation on multiple expressions, you can create a series of calculations or, better, use the AND operator (&&) to join all of them in a simpler expression.

Example 1

The following formula shows the syntax of the AND function.

```
= IF(AND(10 > 9, -10 < -1), "All true", "One or more false")
```

Because both conditions, passed as arguments, to the AND function are true, the formula returns "All True".

Example 2

The following sample uses the AND function with nested formulas to compare two sets of calculations at the same time. For each product category, the formula determines if the current year sales and previous year sales of the Internet channel are larger than the Reseller channel for the same periods. If both conditions are true, for each category the formula returns the value, "Internet hit".

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

Returns

| ROW LABELS | 2005 | 2006 | 2007 | 2008 | - | GRAND TOTAL |
|-------------------|------|------|------|--------------|---|-------------|
| Bib-Shorts | | | | | | |
| Bike Racks | | | | | | |
| Bike Stands | | | | Internet Hit | | |
| Bottles and Cages | | | | Internet Hit | | |
| Bottom Brackets | | | | | | |
| Brakes | | | | | | |
| Caps | | | | | | |
| Chains | | | | | | |
| Cleaners | | | | | | |
| Cranksets | | | | | | |
| Derailleurs | | | | | | |
| Fenders | | | | Internet Hit | | |
| Forks | | | | | | |
| Gloves | | | | | | |
| Handlebars | | | | | | |
| Headsets | | | | | | |
| Helmets | | | | | | |
| Hydration Packs | | | | | | |
| Jerseys | | | | | | |
| Lights | | | | | | |
| Locks | | | | | | |
| Mountain Bikes | | | | | | |

| ROW LABELS | 2005 | 2006 | 2007 | 2008 | - | GRAND TOTAL |
|-----------------|------|------|------|--------------|---|-------------|
| Mountain Frames | | | | | | |
| Panniers | | | | | | |
| Pedals | | | | | | |
| Pumps | | | | | | |
| Road Bikes | | | | | | |
| Road Frames | | | | | | |
| Saddles | | | | | | |
| Shorts | | | | | | |
| Socks | | | | | | |
| Tights | | | | | | |
| Tires and Tubes | | | | Internet Hit | | |
| Touring Bikes | | | | | | |
| Touring Frames | | | | | | |
| Vests | | | | | | |
| Wheels | | | | | | |
| | | | | | | |
| Grand Total | | | | | | |

See also

[Logical functions](#)

BITAND

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns a bitwise AND of two numbers.

Syntax

```
BITAND(<number>, <number>)
```

Parameters

| TERM | DEFINITION |
|--------|--|
| Number | Any scalar expression that returns number. If not an integer, it is truncated. |

Return value

A bitwise AND of two numbers.

Remarks

- This function supports both positive and negative numbers.

Example

The following DAX query:

```
EVALUATE { BITAND(13, 11) }
```

Returns 9.

See also

[BITLSHIFT](#)

[BITRSHIFT](#)

[BITOR](#)

[BITXOR](#)

BITLSHIFT

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns a number shifted left by the specified number of bits.

Syntax

```
BITLSHIFT(<Number>, <Shift_Amount>)
```

Parameters

| TERM | DEFINITION |
|--------------|--|
| Number | Any DAX expression that returns an integer expression. |
| Shift_Amount | Any DAX expression that returns an integer expression. |

Return value

An integer value.

Remarks

- Be sure to understand the nature of bitshift operations and overflow/underflow of integers before using DAX bitshift functions.
- If Shift_Amount is negative, it will shift in the opposite direction.
- If absolute value of Shift_Amount is larger than 64, there will be no error but will result in overflow/underflow.
- There's no limit on Number, but the result may overflow/underflow.

Examples

Example 1

The following DAX query:

```
EVALUATE
{ BITLSHIFT(2, 3) }
```

Returns 16.

Example 2

The following DAX query:

```
EVALUATE
{ BITLSHIFT(128, -1) }
```

Returns 64.

Example 3

The following DAX query:

```
Define
    Measure Sales[LeftShift] = BITLSHIFT(SELECTEDVALUE(Sales[Amount]), 3)

EVALUATE
SUMMARIZECOLUMNS(
    Sales[Amount],
    "LEFTSHIFT",
    [LeftShift]
)
```

Shifts left each sales amount with 3 bits and returns the bit-shifted sales amount.

See also

[BITRSHIFT](#)

[BITAND](#)

[BITOR](#)

[BITXOR](#)

BITOR

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns a bitwise OR of two numbers.

Syntax

```
BITOR(<number>, <number>)
```

Parameters

| TERM | DEFINITION |
|--------|--|
| Number | Any scalar expression that returns number. If not an integer, it is truncated. |

Return value

A bitwise OR of two numbers.

Remarks

- This function supports both positive and negative numbers.

Example

The following DAX query:

```
EVALUATE
{ BITOR(9, 10) }
```

Returns 11.

See also

[BITAND](#)

[BITXOR](#)

[BITLSHIFT](#)

[BITRSHIFT](#)

BITRSHIFT

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns a number shifted right by the specified number of bits.

Syntax

```
BITRSHIFT(<Number>, <Shift_Amount>)
```

Parameters

| TERM | DEFINITION |
|--------------|--|
| Number | Any DAX expression that returns an integer expression. |
| Shift_Amount | Any DAX expression that returns an integer expression. |

Return value

An integer value.

Remarks

- Be sure to understand the nature of bitshift operations and overflow/underflow of integers before using DAX bitshift functions.
- If Shift_Amount is negative, it will shift in the opposite direction.
- If absolute value of Shift_Amount is larger than 64, there will be no error but will result in overflow/underflow.
- There's no limit on Number, but the result may overflow/underflow.

Examples

Example 1

The following DAX query:

```
EVALUATE
{ BITRSHIFT(16, 3) }
```

Returns 2.

Example 2

The following DAX query:

```
EVALUATE
{ BITRSHIFT(1024, -3) }
```

Returns 8192.

Example 3

The following DAX query:

```
Define
    Measure Sales[RightShift] = BITRSHIFT(SELECTEDVALUE(Sales[Amount]), 3)

EVALUATE
SUMMARIZECOLUMNS(
    Sales[Amount],
    "RIGHTSHIFT",
    [RightShift]
)
```

Shifts right each sales amount with 3 bits and returns the bit-shifted sales amount.

See also

[BITLSHIFT](#)

[BITAND](#)

[BITOR](#)

[BITXOR](#)

BITXOR

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns a bitwise XOR of two numbers.

Syntax

```
BITXOR(<number>, <number>)
```

Parameters

| TERM | DEFINITION |
|--------|--|
| Number | Any scalar expression that returns number. If not an integer, it is truncated. |

Return value

A bitwise XOR of two numbers.

Remarks

- This function supports both positive and negative numbers.

Example

The following DAX query:

```
EVALUATE { BITXOR(9, 10) }
```

Returns 3.

See also

[BITOR](#)

[BITAND](#)

[BITLSHIFT](#)

[BITRSHIFT](#)

COALESCE

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the first expression that does not evaluate to BLANK. If all expressions evaluate to BLANK, BLANK is returned.

Syntax

```
COALESCE(<expression>, <expression>[, <expression>]...)
```

Parameters

| TERM | DEFINITION |
|------------|--|
| expression | Any DAX expression that returns a scalar expression. |

Return value

A scalar value coming from one of the expressions or BLANK if all expressions evaluate to BLANK.

Remarks

Input expressions may be of different data types.

Example 1

The following DAX query:

```
EVALUATE { COALESCE(BLANK(), 10, DATE(2008, 3, 3)) }
```

Returns `10`, which is the first expression that does not evaluate to BLANK.

Example 2

The following DAX expression:

```
= COALESCE(SUM(FactInternetSales[SalesAmount]), 0)
```

Returns the sum of all values in the SalesAmount column in the FactInternetSales table, or `0`. This can be used to convert BLANK values of total sales to `0`.

FALSE

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the logical value FALSE.

Syntax

```
FALSE()
```

Return value

Always FALSE.

Remarks

The word FALSE is also interpreted as the logical value FALSE.

Example

The formula returns the logical value FALSE when the value in the column, 'InternetSales_USD'[SalesAmount_USD], is less than or equal to 200000.

```
= IF(SUM('InternetSales_USD'[SalesAmount_USD]) >200000, TRUE(), false())
```

The following table shows the results when the example formula is used with 'ProductCategory'[ProductCategoryName] in Row Labels and 'DateTime'[CalendarYear] in Column Labels.

| ROW LABELS | 2005 | 2006 | 2007 | 2008 | - | GRAND TOTAL |
|-------------|-------|-------|-------|-------|-------|-------------|
| Accessories | FALSE | FALSE | TRUE | TRUE | FALSE | TRUE |
| Bikes | TRUE | TRUE | TRUE | TRUE | FALSE | TRUE |
| Clothing | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE |
| Components | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| Grand Total | TRUE | TRUE | TRUE | TRUE | FALSE | TRUE |

See also

[TRUE function](#)

[NOT function](#)

[IF function](#)

IF

10/31/2022 • 2 minutes to read • [Edit Online](#)

Checks a condition, and returns one value when it's TRUE, otherwise it returns a second value.

Syntax

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

Parameters

| TERM | DEFINITION |
|----------------|---|
| logical_test | Any value or expression that can be evaluated to TRUE or FALSE. |
| value_if_true | The value that's returned if the logical test is TRUE. |
| value_if_false | (Optional) The value that's returned if the logical test is FALSE. If omitted, BLANK is returned. |

Return value

Either **value_if_true**, **value_if_false**, or BLANK.

Remarks

- The IF function can return a variant data type if **value_if_true** and **value_if_false** are of different data types, but the function attempts to return a single data type if both **value_if_true** and **value_if_false** are of numeric data types. In the latter case, the IF function will implicitly convert data types to accommodate both values.

For example, the formula `IF(<condition>, TRUE(), 0)` returns TRUE or 0, but the formula

`IF(<condition>, 1.0, 0)` returns only decimal values even though **value_if_false** is of the whole number data type. To learn more about implicit data type conversion, see [Data types](#).

- To execute the branch expressions regardless of the condition expression, use [IFEAGER](#) instead.

Examples

The following **Product** table calculated column definitions use the IF function in different ways to classify each product based on its list price.

The first example tests whether the **List Price** column value is less than 500. When this condition is true, the value **Low** is returned. Because there's no **value_if_false** value, BLANK is returned.

Examples in this article can be used with the sample Adventure Works DW 2020 Power BI Desktop model. To get the model, see [DAX sample model](#).

```
Price Group =  
IF(  
    'Product'[List Price] < 500,  
    "Low"  
)
```

The second example uses the same test, but this time includes a **value_if_false** value. So, the formula classifies each product as either **Low** or **High**.

```
Price Group =  
IF(  
    'Product'[List Price] < 500,  
    "Low",  
    "High"  
)
```

The third example uses the same test, but this time nests an IF function to perform an additional test. So, the formula classifies each product as either **Low**, **Medium**, or **High**.

```
Price Group =  
IF(  
    'Product'[List Price] < 500,  
    "Low",  
    IF(  
        'Product'[List Price] < 1500,  
        "Medium",  
        "High"  
    )  
)
```

TIP

When you need to nest multiple IF functions, the [SWITCH](#) function might be a better option. This function provides a more elegant way to write an expression that returns more than two possible values.

See also

[IF.EAGER function](#)

[SWITCH function \(DAX\)](#)

[Logical functions](#)

IF.EAGER

10/31/2022 • 2 minutes to read • [Edit Online](#)

Checks a condition, and returns one value when TRUE, otherwise it returns a second value. It uses an *eager* execution plan which always executes the branch expressions regardless of the condition expression.

Syntax

```
IF.EAGER(<logical_test>, <value_if_true>[, <value_if_false>])
```

Parameters

| TERM | DEFINITION |
|----------------|---|
| logical_test | Any value or expression that can be evaluated to TRUE or FALSE. |
| value_if_true | The value that's returned if the logical test is TRUE. |
| value_if_false | (Optional) The value that's returned if the logical test is FALSE. If omitted, BLANK is returned. |

Return value

Either **value_if_true**, **value_if_false**, or BLANK.

Remarks

- The IF.EAGER function can return a variant data type if **value_if_true** and **value_if_false** are of different data types, but the function attempts to return a single data type if both **value_if_true** and **value_if_false** are of numeric data types. In the latter case, the IF.EAGER function will implicitly convert data types to accommodate both values.

For example, the formula `IF.EAGER(<condition>, TRUE(), 0)` returns TRUE or 0, but the formula `IF.EAGER(<condition>, 1.0, 0)` returns only decimal values even though **value_if_false** is of the whole number data type. To learn more about implicit data type conversion, see [Data types](#).

- IF.EAGER has the same functional behavior as the IF function, but performance may differ due to differences in execution plans. `IF.EAGER(<logical_test>, <value_if_true>, <value_if_false>)` has the same execution plan as the following DAX expression:

```
VAR _value_if_true = <value_if_true>
VAR _value_if_false = <value_if_false>
RETURN
IF (<logical_test>, _value_if_true, _value_if_false)
```

Note: The two branch expressions are evaluated regardless of the condition expression.

Examples

See [IF Examples](#).

See also

[IF function](#)

[Logical functions](#)

IFERROR

10/31/2022 • 2 minutes to read • [Edit Online](#)

Evaluates an expression and returns a specified value if the expression returns an error; otherwise returns the value of the expression itself.

Syntax

```
IFERROR(value, value_if_error)
```

Parameters

| TERM | DEFINITION |
|----------------|--------------------------|
| value | Any value or expression. |
| value_if_error | Any value or expression. |

Return value

A scalar of the same type as **value**

Remarks

- You can use the IFERROR function to trap and handle errors in an expression.
- If **value** or **value_if_error** is an empty cell, IFERROR treats it as an empty string value ("").
- The IFERROR function is based on the IF function, and uses the same error messages, but has fewer arguments. The relationship between the IFERROR function and the IF function as follows:

```
IFERROR(A,B) := IF(ISERROR(A), B, A)
```

Values that are returned for A and B must be of the same data type; therefore, the column or expression used for **value** and the value returned for **value_if_error** must be the same data type.

- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.
- For best practices when using IFERROR, see [Appropriate use of error functions](#).

Example

The following example returns 9999 if the expression 25/0 evaluates to an error. If the expression returns a value other than error, that value is passed to the invoking expression.

```
= IFERROR(25/0,9999)
```

See also

NOT

10/31/2022 • 2 minutes to read • [Edit Online](#)

Changes FALSE to TRUE, or TRUE to FALSE.

Syntax

```
NOT(<logical>)
```

Parameters

| TERM | DEFINITION |
|---------|---|
| logical | A value or expression that can be evaluated to TRUE or FALSE. |

Return value

TRUE OR FALSE.

Example

The following example retrieves values from the calculated column that was created to illustrate the IF function. For that example, the calculated column was named using the default name, **Calculated Column1**, and contains the following formula: `= IF([Orders]<300,"true","false")`

The formula checks the value in the column, [Orders], and returns "true" if the number of orders is under 300.

Now create a new calculated column, **Calculated Column2**, and type the following formula.

```
= NOT([CalculatedColumn1])
```

For each row in **Calculated Column1**, the values "true" and "false" are interpreted as the logical values TRUE or FALSE, and the NOT function returns the logical opposite of that value.

See also

[TRUE function](#)

[FALSE function](#)

[IF function](#)

OR

10/31/2022 • 2 minutes to read • [Edit Online](#)

Checks whether one of the arguments is TRUE to return TRUE. The function returns FALSE if both arguments are FALSE.

Syntax

```
OR(<logical1>,<logical2>)
```

Parameters

| TERM | DEFINITION |
|----------------------|--------------------------------------|
| logical_1, logical_2 | The logical values you want to test. |

Return value

A Boolean value. The value is TRUE if any of the two arguments is TRUE; the value is FALSE if both the arguments are FALSE.

Remarks

- The **OR** function in DAX accepts only two (2) arguments. If you need to perform an OR operation on multiple expressions, you can create a series of calculations or, better, use the OR operator (||) to join all of them in a simpler expression.
- The function evaluates the arguments until the first TRUE argument, then returns TRUE.

Example

The following example shows how to use the OR function to obtain the sales people that belong to the Circle of Excellence. The Circle of Excellence recognizes those who have achieved more than a million dollars in Touring Bikes sales or sales of over two and a half million dollars in 2007.

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

Returns

| ROW LABELS | 2005 | 2006 | 2007 | 2008 | - | GRAND TOTAL |
|---------------|------|------|------|------|---|-------------|
| Abbas, Syed E | | | | | | |

See also

[Logical functions](#)

SWITCH

10/31/2022 • 2 minutes to read • [Edit Online](#)

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

Syntax

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

Parameters

| TERM | DEFINITION |
|------------|---|
| expression | Any DAX expression that returns a single scalar value, where the expression is to be evaluated multiple times (for each row/context). |
| value | A constant value to be matched with the results of <i>expression</i> . |
| result | Any scalar expression to be evaluated if the results of <i>expression</i> match the corresponding <i>value</i> . |
| else | Any scalar expression to be evaluated if the result of <i>expression</i> doesn't match any of the <i>value</i> arguments. |

Return value

A scalar value coming from one of the *result* expressions, if there was a match with *value*, or from the *else* expression, if there was no match with any *value*.

Remarks

All result expressions and the else expression must be of the same data type.

Example

The following example creates a calculated column of month names.

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

TRUE

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the logical value TRUE.

Syntax

```
TRUE()
```

Return value

Always TRUE.

Remarks

The word TRUE is also interpreted as the logical value TRUE.

Example

The formula returns the logical value TRUE when the value in the column, 'InternetSales_USD'[SalesAmount_USD], is greater than 200000.

```
= IF(SUM('InternetSales_USD'[SalesAmount_USD]) >200000, TRUE(), false())
```

The following table shows the results when the example formula is used in a report with 'ProductCategory'[ProductCategoryName] in Row Labels and 'DateTime'[CalendarYear] in Column Labels.

| ROW LABELS | 2005 | 2006 | 2007 | 2008 | - | GRAND TOTAL |
|-------------|-------|-------|-------|-------|-------|-------------|
| Accessories | FALSE | FALSE | TRUE | TRUE | FALSE | TRUE |
| Bikes | TRUE | TRUE | TRUE | TRUE | FALSE | TRUE |
| Clothing | FALSE | FALSE | FALSE | FALSE | FALSE | TRUE |
| Components | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| | FALSE | FALSE | FALSE | FALSE | FALSE | FALSE |
| Grand Total | TRUE | TRUE | TRUE | TRUE | FALSE | TRUE |

See also

[FALSE](#)

[NOT](#)

[IF](#)

Math and Trig functions

10/31/2022 • 2 minutes to read • [Edit Online](#)

The mathematical functions in Data Analysis Expressions (DAX) are very similar to the Excel mathematical and trigonometric functions. This section lists the mathematical functions provided by DAX.

In this category

| FUNCTION | DESCRIPTION |
|--------------------------|--|
| ABS | Returns the absolute value of a number. |
| ACOS | Returns the arccosine, or inverse cosine, of a number. |
| ACOSH | Returns the inverse hyperbolic cosine of a number. |
| ACOT | Returns the arccotangent, or inverse cotangent, of a number. |
| ACOTH | Returns the inverse hyperbolic cotangent of a number. |
| ASIN | Returns the arcsine, or inverse sine, of a number. |
| ASINH | Returns the inverse hyperbolic sine of a number. |
| ATAN | Returns the arctangent, or inverse tangent, of a number. |
| ATANH | Returns the inverse hyperbolic tangent of a number. |
| CEILING | Rounds a number up, to the nearest integer or to the nearest multiple of significance. |
| CONVERT | Converts an expression of one data type to another. |
| COS | Returns the cosine of the given angle. |
| COSH | Returns the hyperbolic cosine of a number. |
| COT | Returns the cotangent of an angle specified in radians. |
| COTH | Returns the hyperbolic cotangent of a hyperbolic angle. |
| CURRENCY | Evaluates the argument and returns the result as currency data type. |
| DEGREES | Converts radians into degrees. |
| DIVIDE | Performs division and returns alternate result or BLANK() on division by 0. |

| FUNCTION | DESCRIPTION |
|-------------|---|
| EVEN | Returns number rounded up to the nearest even integer. |
| EXP | Returns e raised to the power of a given number. |
| FACT | Returns the factorial of a number, equal to the series $1*2*3*...*$, ending in the given number. |
| FLOOR | Rounds a number down, toward zero, to the nearest multiple of significance. |
| GCD | Returns the greatest common divisor of two or more integers. |
| INT | Rounds a number down to the nearest integer. |
| ISO.CEILING | Rounds a number up, to the nearest integer or to the nearest multiple of significance. |
| LCM | Returns the least common multiple of integers. |
| LN | Returns the natural logarithm of a number. |
| LOG | Returns the logarithm of a number to the base you specify. |
| LOG10 | Returns the base-10 logarithm of a number. |
| MOD | Returns the remainder after a number is divided by a divisor. The result always has the same sign as the divisor. |
| MROUND | Returns a number rounded to the desired multiple. |
| ODD | Returns number rounded up to the nearest odd integer. |
| PI | Returns the value of Pi, 3.14159265358979, accurate to 15 digits. |
| POWER | Returns the result of a number raised to a power. |
| QUOTIENT | Performs division and returns only the integer portion of the division result. |
| RADIANS | Converts degrees to radians. |
| RAND | Returns a random number greater than or equal to 0 and less than 1, evenly distributed. |
| RANDBETWEEN | Returns a random number in the range between two numbers you specify. |
| ROUND | Rounds a number to the specified number of digits. |
| ROUNDDOWN | Rounds a number down, toward zero. |

| FUNCTION | DESCRIPTION |
|----------|--|
| ROUNDUP | Rounds a number up, away from 0 (zero). |
| SIGN | Determines the sign of a number, the result of a calculation, or a value in a column. |
| SIN | Returns the sine of the given angle. |
| SINH | Returns the hyperbolic sine of a number. |
| SQRT | Returns the square root of a number. |
| SQRTPI | Returns the square root of (number * pi). |
| TAN | Returns the tangent of the given angle. |
| TANH | Returns the hyperbolic tangent of a number. |
| TRUNC | Truncates a number to an integer by removing the decimal, or fractional, part of the number. |

ABS

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the absolute value of a number.

Syntax

```
ABS(<number>)
```

Parameters

| TERM | DEFINITION |
|--------|---|
| number | The number for which you want the absolute value. |

Return value

A decimal number.

Remarks

The absolute value of a number is a decimal number, whole or decimal, without its sign. You can use the ABS function to ensure that only non-negative numbers are returned from expressions when nested in functions that require a positive number.

Example

The following example returns the absolute value of the difference between the list price and the dealer price, which you might use in a new calculated column, **DealerMarkup**.

```
= ABS([DealerPrice]-[ListPrice])
```

See also

[Math and Trig functions](#)

[SIGN function](#)

ACOS

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the arccosine, or inverse cosine, of a number. The arccosine is the angle whose cosine is *number*. The returned angle is given in radians in the range 0 (zero) to pi.

Syntax

ACOS(*number*)

Parameters

| TERM | DEFINITION |
|--------|--|
| Number | The cosine of the angle you want and must be from -1 to 1. |

Return value

Returns the arccosine, or inverse cosine, of a number.

Remarks

If you want to convert the result from radians to degrees, multiply it by 180/PI() or use the DEGREES function.

Example

| FORMULA | DESCRIPTION | RESULT |
|-----------------------|---------------------------------------|-------------|
| = ACOS(-0.5) | Arccosine of -0.5 in radians, 2*pi/3. | 2.094395102 |
| = ACOS(-0.5)*180/PI() | Arccosine of -0.5 in degrees. | 120 |

ACOSH

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the inverse hyperbolic cosine of a number. The number must be greater than or equal to 1. The inverse hyperbolic cosine is the value whose hyperbolic cosine is *number*, so ACOSH(COSH(number)) equals number.

Syntax

ACOSH(number)

Parameters

| TERM | DEFINITION |
|--------|---|
| number | Any real number equal to or greater than 1. |

Return value

Returns the inverse hyperbolic cosine of a number.

Remarks

This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

| FORMULA | DESCRIPTION | RESULT |
|-------------|----------------------------------|----------|
| = ACOSH(1) | Inverse hyperbolic cosine of 1. | 0 |
| = ACOSH(10) | Inverse hyperbolic cosine of 10. | 2.993228 |

ACOT

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the principal value of the arccotangent, or inverse cotangent of a number.

Syntax

```
ACOT(number)
```

Parameters

| TERM | DEFINITION |
|--------|--|
| Number | The cosine of the angle you want. Must be a real number. |

Return value

A single decimal value.

ACOTH

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the inverse hyperbolic cotangent of a number.

Syntax

```
ACOTH(number)
```

Parameters

| TERM | DEFINITION |
|--------|--|
| Number | The absolute value of Number must be greater than 1. |

Return value

A single decimal value.

Remarks

This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

ASIN

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the arcsine, or inverse sine, of a number. The arcsine is the angle whose sine is *number*. The returned angle is given in radians in the range $-\pi/2$ to $\pi/2$.

Syntax

ASIN(number)

Parameters

| TERM | DEFINITION |
|--------|--|
| number | The sine of the angle you want and must be from -1 to 1. |

Return value

Returns the arcsine, or inverse sine, of a number.

Remarks

To express the arcsine in degrees, multiply the result by $180/\pi$ () or use the DEGREES function.

Example

| FORMULA | DESCRIPTION | RESULT |
|-----------------------|--------------------------------------|--------------|
| = ASIN(-0.5) | Arcsine of -0.5 in radians, $-\pi/6$ | -0.523598776 |
| = ASIN(-0.5)*180/PI() | Arcsine of -0.5 in degrees | -30 |
| = DEGREES(ASIN(-0.5)) | Arcsine of -0.5 in degrees | -30 |

ASINH

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the inverse hyperbolic sine of a number. The inverse hyperbolic sine is the value whose hyperbolic sine is *number*, so ASINH(SINH(number)) equals *number*.

Syntax

ASINH(number)

Parameters

| TERM | DEFINITION |
|--------|------------------|
| number | Any real number. |

Return value

Returns the inverse hyperbolic sine of a number.

Remarks

This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

| FORMULA | DESCRIPTION | RESULT |
|---------------|---------------------------------|--------------|
| = ASINH(-2.5) | Inverse hyperbolic sine of -2.5 | -1.647231146 |
| = ASINH(10) | Inverse hyperbolic sine of 10 | 2.99822295 |

ATAN

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the arctangent, or inverse tangent, of a number. The arctangent is the angle whose tangent is *number*. The returned angle is given in radians in the range $-\pi/2$ to $\pi/2$.

Syntax

ATAN(*number*)

Parameters

| TERM | DEFINITION |
|--------|------------------------------------|
| number | The tangent of the angle you want. |

Return value

Returns the inverse hyperbolic tangent of a number.

Remarks

To express the arctangent in degrees, multiply the result by $180/\pi$ or use the DEGREES function.

Example

| FORMULA | DESCRIPTION | RESULT |
|--------------------|-------------------------------------|-------------|
| = ATAN(1) | Arctangent of 1 in radians, $\pi/4$ | 0.785398163 |
| = ATAN(1)*180/PI() | Arctangent of 1 in degrees | 45 |

ATANH

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the inverse hyperbolic tangent of a number. Number must be between -1 and 1 (excluding -1 and 1). The inverse hyperbolic tangent is the value whose hyperbolic tangent is *number*, so ATANH(TANH(number)) equals *number*.

Syntax

ATANH(number)

Parameters

| TERM | DEFINITION |
|--------|-----------------------------------|
| number | Any real number between 1 and -1. |

Return value

Returns the inverse hyperbolic tangent of a number.

Remarks

This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

| FORMULA | DESCRIPTION | RESULT |
|---------------------|--|--------------|
| = ATANH(0.76159416) | Inverse hyperbolic tangent of 0.76159416 | 1.00000001 |
| = ATANH(-0.1) | | -0.100335348 |
| | | |

See also

[ATAN function](#)

CEILING

10/31/2022 • 2 minutes to read • [Edit Online](#)

Rounds a number up, to the nearest integer or to the nearest multiple of significance.

Syntax

```
CEILING(<number>, <significance>)
```

Parameters

| TERM | DEFINITION |
|--------------|--|
| number | The number you want to round, or a reference to a column that contains numbers. |
| significance | The multiple of significance to which you want to round. For example, to round to the nearest integer, type 1. |

Return value

A number rounded as specified.

Remarks

- There are two CEILING functions in DAX, with the following differences:
 - The CEILING function emulates the behavior of the CEILING function in Excel.
 - The ISO.CEILING function follows the ISO-defined behavior for determining the ceiling value.
- The two functions return the same value for positive numbers, but different values for negative numbers. When using a positive multiple of significance, both CEILING and ISO.CEILING round negative numbers upward (toward positive infinity). When using a negative multiple of significance, CEILING rounds negative numbers downward (toward negative infinity), while ISO.CEILING rounds negative numbers upward (toward positive infinity).
- The return type is usually of the same type of the significant argument, with the following exceptions:
 - If the number argument type is currency, the return type is currency.
 - If the significance argument type is Boolean, the return type is integer.
 - If the significance argument type is non-numeric, the return type is real.

Example 1

The following formula returns 4.45. This might be useful if you want to avoid using smaller units in your pricing. If an existing product is priced at \$4.42, you can use CEILING to round prices up to the nearest unit of five cents.

```
= CEILING(4.42,0.05)
```

Example 2

The following formula returns similar results as the previous example, but uses numeric values stored in the column, **ProductPrice**.

```
= CEILING([ProductPrice],0.05)
```

See also

[Math and Trig functions](#)

[FLOOR function](#)

[ISO.CEILING function](#)

[ROUNDUP function](#)

CONVERT

10/31/2022 • 2 minutes to read • [Edit Online](#)

Converts an expression of one data type to another.

Syntax

```
CONVERT(<Expression>, <Datatype>)
```

Parameters

| TERM | DEFINITION |
|------------|--|
| Expression | Any valid expression. |
| Datatype | An enumeration that includes: INTEGER(Whole Number), DOUBLE(Decimal Number), STRING(Text), BOOLEAN(True/False), CURRENCY(Fixed Decimal Number), DATETIME(Date, Time, etc). |

Return value

Returns the value of <Expression>, translated to <Datatype>.

Remarks

- The function returns an error when a value cannot be converted to the specified data type.
- DAX calculated columns must be of a single data type. Since MEDIAN and MEDIANX functions over an integer column return mixed data types, either integer or double, the following calculated column expression will return an error as a result: `MedianNumberCarsOwned = MEDIAN(DimCustomer[NumberCarsOwned])`.
- To avoid mixed data types, change the expression to always return the double data type, for example: `MedianNumberCarsOwned = MEDIANX(DimCustomer, CONVERT([NumberCarsOwned], DOUBLE))`.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

DAX query

```
EVALUATE { CONVERT(DATE(1900, 1, 1), INTEGER) }
```

Returns

[VALUE]

2

COS

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the cosine of the given angle.

Syntax

`COS(number)`

Parameters

| TERM | DEFINITION |
|--------|---|
| number | Required. The angle in radians for which you want the cosine. |

Return value

Returns the cosine of the given angle.

Remarks

If the angle is in degrees, either multiply the angle by $\text{PI()}/180$ or use the `RADIANS` function to convert the angle to radians.

Example

| FORMULA | DESCRIPTION | RESULT |
|---------------------------------|-------------------------|-----------|
| <code>= COS(1.047)</code> | Cosine of 1.047 radians | 0.5001711 |
| <code>= COS(60*PI()/180)</code> | Cosine of 60 degrees | 0.5 |
| <code>= COS(RADIANS(60))</code> | Cosine of 60 degrees | 0.5 |

COSH

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the hyperbolic cosine of a number.

Syntax

COSH(number)

Parameters

| TERM | DEFINITION |
|--------|---|
| number | Required. Any real number for which you want to find the hyperbolic cosine. |

Return value

The hyperbolic cosine of a number.

Remarks

- The formula for the hyperbolic cosine is:

$$\text{COSH}(z) = \frac{e^z + e^{-z}}{2}$$

- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

| FORMULA | DESCRIPTION | RESULT |
|----------------|---|-----------|
| = COSH(4) | Hyperbolic cosine of 4 | 27.308233 |
| = COSH(EXP(1)) | Hyperbolic cosine of the base of the natural logarithm. | 7.6101251 |

COT

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the cotangent of an angle specified in radians.

Syntax

```
COT (<number>)
```

Parameters

| TERM | DEFINITION |
|--------|--|
| number | The angle in radians for which you want the cotangent. |

Return value

The cotangent of the given angle.

Remarks

- The absolute value of number must be less than 2^{27} and cannot be 0.
- If number is outside its constraints, an error is returned.
- If number is a non-numeric value, an error is returned.

Example

The following DAX query,

```
EVALUATE { COT(30) }
```

Returns

```
[VALUE]
```

```
-0.156119952161659
```


COTH

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the hyperbolic cotangent of a hyperbolic angle.

Syntax

COTH (<number>)

Parameters

| TERM | DEFINITION |
|--------|--|
| number | The hyperbolic angle in radians for which you want the hyperbolic cotangent. |

Return value

The hyperbolic cotangent of the given angle.

Remarks

- The hyperbolic cotangent is an analog of the ordinary (circular) cotangent.
- The absolute value of number must be less than 2^{27} and cannot be 0.
- If number is outside its constraints, an error is returned
- If number is a non-numeric value, an error is returned.
- The following equation is used:

$$\text{COTH}(N) = \frac{1}{\text{TANH}(N)} = \frac{\text{COSH}(N)}{\text{SINH}(N)} = \frac{e^N + e^{-N}}{e^N - e^{-N}}$$

- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following DAX query,

```
EVALUATE { COTH(2) }
```

Returns

[VALUE]

1.03731472072755

CURRENCY

10/31/2022 • 2 minutes to read • [Edit Online](#)

Evaluates the argument and returns the result as currency data type.

Syntax

```
CURRENCY(<value>)
```

Parameters

| TERM | DEFINITION |
|-------|---|
| value | Any DAX expression that returns a single scalar value where the expression is to be evaluated exactly once before all other operations. |

Return value

The value of the expression evaluated and returned as a currency type value.

Remarks

- The CURRENCY function rounds up the 5th significant decimal, in value, to return the 4th decimal digit. Rounding up occurs if the 5th significant decimal is equal or larger than 5. For example, if value is 3.66666666666666 then converting to currency returns \ \$3.6667. However, if value is 3.0123456789 then converting to currency returns \ \$3.0123.
- If the data type of the expression is TrueFalse then CURRENCY(<TrueFalse>) will return \ \$1.0000 for True values and \ \$0.0000 for False values.
- If the data type of the expression is Text then CURRENCY(<Text>) will try to convert text to a number. If conversion succeeds the number will be converted to currency, otherwise an error is returned.
- If the data type of the expression is DateTime then CURRENCY(<DateTime>) will convert the datetime value to a number and that number to currency. DateTime values have an integer part that represents the number of days between the given date and 1900-03-01 and a fraction that represents the fraction of a day (where 12 hours or noon is 0.5 day). If the value of the expression is not a proper DateTime value an error is returned.

Example

Convert number 1234.56 to currency data type.

```
= CURRENCY(1234.56)
```

Returns value 1234.56000.

DEGREES

10/31/2022 • 2 minutes to read • [Edit Online](#)

Converts radians into degrees.

Syntax

```
DEGREES(angle)
```

Parameters

| TERM | DEFINITION |
|--------------|--|
| <i>angle</i> | Required. The angle in radians that you want to convert. |

Example

| FORMULA | DESCRIPTION | RESULT |
|-----------------|-----------------------|--------|
| = DEGREES(PI()) | Degrees of pi radians | 180 |

DIVIDE

10/31/2022 • 2 minutes to read • [Edit Online](#)

Performs division and returns alternate result or BLANK() on division by 0.

Syntax

```
DIVIDE(<numerator>, <denominator> [,<alternateresult>])
```

Parameters

| TERM | DEFINITION |
|-----------------|---|
| numerator | The dividend or number to divide. |
| denominator | The divisor or number to divide by. |
| alternateresult | (Optional) The value returned when division by zero results in an error. When not provided, the default value is BLANK(). |

Return value

- A decimal number.

Remarks

- Alternate result on divide by 0 must be a constant.
- For best practices when using DIVIDE, see [DIVIDE function vs. divide operator \(/\) in DAX](#).

Example

The following example returns 2.5.

```
= DIVIDE(5,2)
```

Example 1

The following example returns BLANK.

```
= DIVIDE(5,0)
```

Example 2

The following example returns 1.

```
= DIVIDE(5,0,1)
```

See also

[QUOTIENT function](#)

[Math and Trig functions](#)

EVEN

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns number rounded up to the nearest even integer. You can use this function for processing items that come in twos. For example, a packing crate accepts rows of one or two items. The crate is full when the number of items, rounded up to the nearest two, matches the crate's capacity.

Syntax

EVEN(number)

Parameters

| TERM | DEFINITION |
|--------|---------------------|
| number | The value to round. |

Return value

Returns number rounded up to the nearest even integer.

Remarks

- If number is nonnumeric, EVEN returns the #VALUE! error value.
- Regardless of the sign of number, a value is rounded up when adjusted away from zero. If number is an even integer, no rounding occurs.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

| FORMULA | DESCRIPTION | RESULT |
|-------------|--|--------|
| = EVEN(1.5) | Rounds 1.5 to the nearest even integer | 2 |
| = EVEN(3) | Rounds 3 to the nearest even integer | 4 |
| = EVEN(2) | Rounds 2 to the nearest even integer | 2 |
| = EVEN(-1) | Rounds -1 to the nearest even integer | -2 |

EXP

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns e raised to the power of a given number. The constant e equals 2.71828182845904, the base of the natural logarithm.

Syntax

```
EXP(<number>)
```

Parameters

| TERM | DEFINITION |
|--------|--|
| number | The exponent applied to the base e. The constant e equals 2.71828182845904, the base of the natural logarithm. |

Return value

A decimal number.

Exceptions

Remarks

- EXP is the inverse of LN, which is the natural logarithm of the given number.
- To calculate powers of bases other than e, use the exponentiation operator (^). For more information, see [DAX Operator Reference](#).

Example

The following formula calculates e raised to the power of the number contained in the column, `[Power]`.

```
= EXP([Power])
```

See also

[Math and Trig functions](#)

[LN function](#)

[EXP function](#)

[LOG function](#)

[LOG function](#)

FACT

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the factorial of a number, equal to the series $1*2*3*...*$, ending in the given number.

Syntax

```
FACT(<number>)
```

Parameters

| TERM | DEFINITION |
|--------|--|
| number | The non-negative number for which you want to calculate the factorial. |

Return value

A decimal number.

Remarks

- If the number is not an integer, it is truncated and an error is returned. If the result is too large, an error is returned.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following formula returns the factorial for the series of integers in the column, `[Values]`.

```
= FACT([Values])
```

The following table shows the expected results:

| VALUES | RESULTS |
|--------|---------|
| 0 | 1 |
| 1 | 1 |
| 2 | 2 |
| 3 | 6 |
| 4 | 24 |

| VALUES | RESULTS |
|--------|---------------------|
| 5 | 120 |
| 170 | 7.257415615308E+306 |

See also

[Math and Trig functions](#)

[TRUNC function](#)

FLOOR

10/31/2022 • 2 minutes to read • [Edit Online](#)

Rounds a number down, toward zero, to the nearest multiple of significance.

Syntax

```
FLOOR(<number>, <significance>)
```

Parameters

| TERM | DEFINITION |
|--------------|---|
| number | The numeric value you want to round. |
| significance | The multiple to which you want to round. The arguments number and significance must either both be positive, or both be negative. |

Return value

A decimal number.

Remarks

- If either argument is nonnumeric, FLOOR returns ****#VALUE!****error value.
- If number and significance have different signs, FLOOR returns the ****#NUM!****error value.
- Regardless of the sign of the number, a value is rounded down when adjusted away from zero. If the number is an exact multiple of significance, no rounding occurs.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following formula takes the values in the [Total Product Cost] column from the table, InternetSales, and rounds down to the nearest multiple of .1.

```
= FLOOR(InternetSales[Total Product Cost],.1)
```

The following table shows the expected results for some sample values:

| VALUES | EXPECTED RESULT |
|---------|-----------------|
| 10.8423 | 10.8 |
| 8.0373 | 8 |

| VALUES | EXPECTED RESULT |
|--------|-----------------|
| 2.9733 | 2.9 |

See also

[Math and Trig functions](#)

GCD

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the greatest common divisor of two or more integers. The greatest common divisor is the largest integer that divides both number1 and number2 without a remainder.

Syntax

```
GCD(number1, [number2], ...)
```

Parameters

| TERM | DEFINITION |
|-----------------------|---|
| number1, number2, ... | Number1 is required, subsequent numbers are optional. 1 to 255 values. If any value is not an integer, it is truncated. |

Return value

The greatest common divisor of two or more integers.

Remarks

- If any argument is nonnumeric, GCD returns the #VALUE! error value.
- If any argument is less than zero, GCD returns the #NUM! error value.
- One divides any value evenly.
- A prime number has only itself and one as even divisors.
- If a parameter to GCD is $\geq 2^{53}$, GCD returns the #NUM! error value.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

| FORMULA | DESCRIPTION | RESULT |
|---------------|---------------------------------------|--------|
| = GCD(5, 2) | Greatest common divisor of 5 and 2. | 1 |
| = GCD(24, 36) | Greatest common divisor of 24 and 36. | 12 |
| = GCD(7, 1) | Greatest common divisor of 7 and 1. | 1 |

INT

10/31/2022 • 2 minutes to read • [Edit Online](#)

Rounds a number down to the nearest integer.

Syntax

```
INT(<number>)
```

Parameters

| TERM | DEFINITION |
|--------|---|
| number | The number you want to round down to an integer |

Return value

A whole number.

Remarks

TRUNC and INT are similar in that both return integers. TRUNC removes the fractional part of the number. INT rounds numbers down to the nearest integer based on the value of the fractional part of the number. INT and TRUNC are different only when using negative numbers: `TRUNC(-4.3)` returns -4, but `INT(-4.3)` returns -5 because -5 is the lower number.

Example

The following expression rounds the value to 1. If you use the ROUND function, the result would be 2.

```
= INT(1.5)
```

See also

[Math and Trig functions](#)

[ROUND function](#)

[ROUNDUP function](#)

[ROUNDDOWN function](#)

[MROUND function](#)

ISO.CEILING

10/31/2022 • 2 minutes to read • [Edit Online](#)

Rounds a number up, to the nearest integer or to the nearest multiple of significance.

Syntax

```
ISO.CEILING(<number>[, <significance>])
```

Parameters

| TERM | DEFINITION |
|--------------|--|
| number | The number you want to round, or a reference to a column that contains numbers. |
| significance | (optional) The multiple of significance to which you want to round. For example, to round to the nearest integer, type 1. If the unit of significance is not specified, the number is rounded up to the nearest integer. |

Return value

A number, of the same type as the *number* argument, rounded as specified.

Remarks

There are two CEILING functions in DAX, with the following differences:

- The CEILING function emulates the behavior of the CEILING function in Excel.
- The ISO.CEILING function follows the ISO-defined behavior for determining the ceiling value.

The two functions return the same value for positive numbers, but different values for negative numbers. When using a positive multiple of significance, both CEILING and ISO.CEILING round negative numbers upward (toward positive infinity). When using a negative multiple of significance, CEILING rounds negative numbers downward (toward negative infinity), while ISO.CEILING rounds negative numbers upward (toward positive infinity).

The result type is usually the same type of the significance used as argument with the following exceptions:

- If the first argument is of currency type then the result will be currency type.
- If the optional argument is not included the result is of integer type.
- If the significance argument is of Boolean type then the result is of integer type.
- If the significance argument is non-numeric type then the result is of real type.

Example: Positive Numbers

The following formula returns 4.45. This might be useful if you want to avoid using smaller units in your pricing.

If an existing product is priced at \$4.42, you can use ISO.CEILING to round prices up to the nearest unit of five cents.

```
= ISO.CEILING(4.42,0.05)
```

Example: Negative Numbers

The following formula returns the ISO ceiling value of -4.40.

```
= ISO.CEILING(-4.42,0.05)
```

See also

[Math and Trig functions](#)

[FLOOR function](#)

[CEILING function](#)

[ROUNDUP function](#)

LCM

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the least common multiple of integers. The least common multiple is the smallest positive integer that is a multiple of all integer arguments number1, number2, and so on. Use LCM to add fractions with different denominators.

Syntax

```
LCM(number1, [number2], ...)
```

Parameters

| TERM | DEFINITION |
|----------------------|--|
| number1, number2,... | Number1 is required, subsequent numbers are optional. 1 to 255 values for which you want the least common multiple. If value is not an integer, it is truncated. |

Return value

Returns the least common multiple of integers.

Remarks

- If any argument is nonnumeric, LCM returns the #VALUE! error value.
- If any argument is less than zero, LCM returns the #NUM! error value.
- If $\text{LCM}(a,b) \geq 2^{53}$, LCM returns the #NUM! error value.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

| FORMULA | DESCRIPTION | RESULT |
|---------------|-------------------------------------|--------|
| = LCM(5, 2) | Least common multiple of 5 and 2. | 10 |
| = LCM(24, 36) | Least common multiple of 24 and 36. | 72 |

LN

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the natural logarithm of a number. Natural logarithms are based on the constant e (2.71828182845904).

Syntax

```
LN(<number>)
```

Parameters

| TERM | DEFINITION |
|--------|---|
| number | The positive number for which you want the natural logarithm. |

Return value

A decimal number.

Remarks

LN is the inverse of the EXP function.

Example

The following example returns the natural logarithm of the number in the column, `[Values]`.

```
= LN([Values])
```

See also

[Math and Trig functions](#)

[EXP function](#)

LOG

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the logarithm of a number to the base you specify.

Syntax

```
LOG(<number>,<base>)
```

Parameters

| TERM | DEFINITION |
|--------|--|
| number | The positive number for which you want the logarithm. |
| base | The base of the logarithm. If omitted, the base is 10. |

Return value

A decimal number.

Remarks

You might receive an error if the value is too large to be displayed.

The function LOG10 is similar, but always returns the common logarithm, meaning the logarithm for the base 10.

Example

The following formulas return the same result, 2.

```
= LOG(100,10)  
= LOG(100)  
= LOG10(100)
```

See also

[Math and Trig functions](#)

[EXP function](#)

[LOG function](#)

[LOG function](#)

LOG10

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the base-10 logarithm of a number.

Syntax

```
LOG10(<number>)
```

Parameters

| TERM | DEFINITION |
|--------|---|
| number | A positive number for which you want the base-10 logarithm. |

Return value

A decimal number.

Remarks

The LOG function lets you change the base of the logarithm, instead of using the base 10.

Example

The following formulas return the same result, 2:

```
= LOG(100,10)  
= LOG(100)  
= LOG10(100)
```

See also

[Math and Trig functions](#)

[EXP function](#)

[LOG function](#)

[LOG function](#)

MOD

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the remainder after a number is divided by a divisor. The result always has the same sign as the divisor.

Syntax

```
MOD(<number>, <divisor>)
```

Parameters

| TERM | DEFINITION |
|---------|--|
| number | The number for which you want to find the remainder after the division is performed. |
| divisor | The number by which you want to divide. |

Return value

A whole number.

Remarks

- If the divisor is 0 (zero), MOD returns an error. You cannot divide by 0.
- The MOD function can be expressed in terms of the INT function: $\text{MOD}(n, d) = n - d * \text{INT}(n/d)$

Example 1

The following formula returns 1, the remainder of 3 divided by 2.

```
= MOD(3,2)
```

Example 2

The following formula returns -1, the remainder of 3 divided by 2. Note that the sign is always the same as the sign of the divisor.

```
= MOD(-3,-2)
```

See also

[Math and Trig functions](#)

[ROUND function](#)

[ROUNDUP function](#)

[ROUNDDOWN function](#)

MROUND function

INT function

MROUND

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns a number rounded to the desired multiple.

Syntax

```
MROUND(<number>, <multiple>)
```

Parameters

| TERM | DEFINITION |
|----------|---|
| number | The number to round. |
| multiple | The multiple of significance to which you want to round the number. |

Return value

A decimal number.

Remarks

MROUND rounds up, away from zero, if the remainder of dividing **number** by the specified **multiple** is greater than or equal to half the value of **multiple**.

Example: Decimal Places

The following expression rounds 1.3 to the nearest multiple of .2. The expected result is 1.4.

```
= MROUND(1.3,0.2)
```

Example: Negative Numbers

The following expression rounds -10 to the nearest multiple of -3. The expected result is -9.

```
= MROUND(-10,-3)
```

Example: Error

The following expression returns an error, because the numbers have different signs.

```
= MROUND(5,-2)
```

See also

Math and Trig functions

ROUND function

ROUNDUP function

ROUNDDOWN function

MROUND function

INT function

ODD

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns number rounded up to the nearest odd integer.

Syntax

ODD(number)

Parameters

| TERM | DEFINITION |
|--------|-------------------------------|
| number | Required. The value to round. |

Return value

Returns number rounded up to the nearest odd integer.

Remarks

- If number is nonnumeric, ODD returns the #VALUE! error value.
- Regardless of the sign of number, a value is rounded up when adjusted away from zero. If number is an odd integer, no rounding occurs.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

| FORMULA | DESCRIPTION | RESULT |
|------------|--|--------|
| = ODD(1.5) | Rounds 1.5 up to the nearest odd integer. | 3 |
| = ODD(3) | Rounds 3 up to the nearest odd integer. | 3 |
| = ODD(2) | Rounds 2 up to the nearest odd integer. | 3 |
| = ODD(-1) | Rounds -1 up to the nearest odd integer. | -1 |
| = ODD(-2) | Rounds -2 up (away from 0) to the nearest odd integer. | -3 |

PI

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the value of Pi, 3.14159265358979, accurate to 15 digits.

Syntax

```
PI()
```

Return value

A decimal number with the value of Pi, 3.14159265358979, accurate to 15 digits.

Remarks

Pi is a mathematical constant. In DAX, Pi is represented as a real number accurate to 15 digits, the same as Excel.

Example

The following formula calculates the area of a circle given the radius in the column, `[Radius]`.

```
= PI()*([Radius]*2)
```

See also

[Math and Trig functions](#)

POWER

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the result of a number raised to a power.

Syntax

```
POWER(<number>, <power>)
```

Parameters

| TERM | DEFINITION |
|--------|--|
| number | The base number, which can be any real number. |
| power | The exponent to which the base number is raised. |

Return value

A decimal number.

Example

The following example returns 25.

```
= POWER(5,2)
```

See also

[Math and Trig functions](#)

QUOTIENT

10/31/2022 • 2 minutes to read • [Edit Online](#)

Performs division and returns only the integer portion of the division result. Use this function when you want to discard the remainder of division.

Syntax

```
QUOTIENT(<numerator>, <denominator>)
```

Parameters

| TERM | DEFINITION |
|-------------|--------------------------------------|
| numerator | The dividend, or number to divide. |
| denominator | The divisor, or number to divide by. |

Return value

A whole number.

Remarks

- If either argument is non-numeric, QUOTIENT returns the **#VALUE!** error value.
- You can use a column reference instead of a literal value for either argument. However, if the column that you reference contains a 0 (zero), an error is returned for the entire column of values.

Example

The following formulas return the same result, 2.

```
= QUOTIENT(5,2)
```

```
= QUOTIENT(10/2,2)
```

See also

[Math and Trig functions](#)

RADIANS

10/31/2022 • 2 minutes to read • [Edit Online](#)

Converts degrees to radians.

Syntax

```
RADIANS(angle)
```

Parameters

| TERM | DEFINITION |
|-------|---|
| angle | Required. An angle in degrees that you want to convert. |

Example

| FORMULA | DESCRIPTION | RESULT |
|----------------|---|----------|
| = RADIANS(270) | 270 degrees as radians (4.712389 or $3\pi/2$ radians) | 4.712389 |

RAND

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns a random number greater than or equal to 0 and less than 1, evenly distributed. The number that is returned changes each time the cell containing this function is recalculated.

Syntax

```
RAND()
```

Return value

A decimal number.

Remarks

- Recalculation depends on various factors, including whether the model is set to **Manual** or **Automatic** recalculation mode, and whether data has been refreshed.
- RAND and other volatile functions that do not have fixed values are not always recalculated. For example, execution of a query or filtering will usually not cause such functions to be re-evaluated. However, the results for these functions will be recalculated when the entire column is recalculated. These situations include refresh from an external data source or manual editing of data that causes re-evaluation of formulas that contain these functions.
- RAND is always recalculated if the function is used in the definition of a measure.
- RAND function cannot return a result of zero, to prevent errors such as division by zero.

Examples

To generate a random real number between two other numbers, use:

```
= RAND()*(b-a)+a
```

To generate a random number greater than 0 and less than 1:

```
= RAND()
```

To generate a random number greater than 0 and less than 100

```
= RAND()*100
```

To generate a random whole number greater than 0 and less than 100

```
INT(RAND()*100)
```

See also

[Math and Trig functions](#)

[Statistical functions](#)

RANDBETWEEN

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns a random number in the range between two numbers you specify.

Syntax

```
RANDBETWEEN(<bottom>,<top>)
```

Parameters

| TERM | DEFINITION |
|--------|--|
| Bottom | The smallest integer the function will return. |
| Top | The largest integer the function will return. |

Return value

A whole number.

Remarks

This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following formula returns a random number between 1 and 10.

```
= RANDBETWEEN(1,10)
```

See also

[Math and Trig functions](#)

[Statistical functions](#)

ROUND

10/31/2022 • 2 minutes to read • [Edit Online](#)

Rounds a number to the specified number of digits.

Syntax

```
ROUND(<number>, <num_digits>)
```

Parameters

| TERM | DEFINITION |
|------------|--|
| number | The number you want to round. |
| num_digits | The number of digits to which you want to round. A negative value rounds digits to the left of the decimal point; a value of zero rounds to the nearest integer. |

Return value

A decimal number.

Remarks

- If **num_digits** is greater than 0 (zero), then number is rounded to the specified number of decimal places.
- If **num_digits** is 0, the number is rounded to the nearest integer.
- If **num_digits** is less than 0, the number is rounded to the left of the decimal point.
- Related functions
 - To always round up (away from zero), use the ROUNDUP function.
 - To always round down (toward zero), use the ROUNDDOWN function.
 - To round a number to a specific multiple (for example, to round to the nearest multiple of 0.5), use the MROUND function.
 - Use the functions TRUNC and INT to obtain the integer portion of the number.

Example 1

The following formula rounds 2.15 up, to one decimal place. The expected result is 2.2.

```
= ROUND(2.15,1)
```

Example 2

The following formula rounds 21.5 to one decimal place to the left of the decimal point. The expected result is

20.

```
= ROUND(21.5,-1)
```

See also

[Math and Trig functions](#)

[ROUND](#)

[ROUNDDOWN](#)

[MROUND](#)

[INT](#)

[TRUNC](#)

ROUNDDOWN

10/31/2022 • 2 minutes to read • [Edit Online](#)

Rounds a number down, toward zero.

Syntax

```
ROUNDDOWN(<number>, <num_digits>)
```

Parameters

| TERM | DEFINITION |
|------------|---|
| number | A real number that you want rounded down. |
| num_digits | The number of digits to which you want to round. Negative rounds to the left of the decimal point; zero to the nearest integer. |

Return value

A decimal number.

Remarks

- If **num_digits** is greater than 0 (zero), then the value in **number** is rounded down to the specified number of decimal places.
- If **num_digits** is 0, then the value in **number** is rounded down to the nearest integer.
- If **num_digits** is less than 0, then the value in **number** is rounded down to the left of the decimal point.
- ROUNDDOWN behaves like ROUND, except that it always rounds a number down. The INT function also rounds down, but with INT the result is always an integer, whereas with ROUNDDOWN you can control the precision of the result.

Example 1

The following example rounds 3.14159 down to three decimal places. The expected result is 3.141.

```
= ROUNDDOWN(3.14159,3)
```

Example 2

The following example rounds the value of 31415.92654 down to 2 decimal places to the left of the decimal. The expected result is 31400.

```
= ROUNDDOWN(31415.92654, -2)
```

See also

[Math and Trig functions](#)

[ROUND](#)

[ROUNDUP](#)

[ROUNDDOWN](#)

[MROUND](#)

[INT](#)

ROUNDUP

10/31/2022 • 2 minutes to read • [Edit Online](#)

Rounds a number up, away from 0 (zero).

Syntax

```
ROUNDUP(<number>, <num_digits>)
```

Parameters

| TERM | DEFINITION |
|------------|---|
| number | A real number that you want to round up. |
| num_digits | The number of digits to which you want to round. A negative value for num_digits rounds to the left of the decimal point; if num_digits is zero or is omitted, number is rounded to the nearest integer. |

Return value

A decimal number.

Remarks

- If **num_digits** is greater than 0 (zero), then the number is rounded up to the specified number of decimal places.
- If **num_digits** is 0, then number is rounded up to the nearest integer.
- If **num_digits** is less than 0, then number is rounded up to the left of the decimal point.
- ROUNDUP behaves like ROUND, except that it always rounds a number up.

Example

The following formula rounds Pi to four decimal places. The expected result is 3.1416.

```
= ROUNDUP(PI(),4)
```

Example: Decimals as Second Argument

The following formula rounds 1.3 to the nearest multiple of 0.2. The expected result is 2.

```
= ROUNDUP(1.3,0.2)
```

Example: Negative Number as Second Argument

The following formula rounds the value in the column, **FreightCost**, with the expected results shown in the following table:

```
= ROUNDUP([Values],-1)
```

When **num_digits** is less than zero, the number of places to the left of the decimal sign is increased by the value you specify.

| FREIGHTCOST | EXPECTED RESULT |
|-------------|-----------------|
| 13.25 | 20 |
| 2.45 | 10 |
| 25.56 | 30 |
| 1.34 | 10 |
| 345.01 | 350 |

See also

- Math and Trig functions
- ROUND
- ROUNDDOWN
- MROUND
- INT

SIGN

10/31/2022 • 2 minutes to read • [Edit Online](#)

Determines the sign of a number, the result of a calculation, or a value in a column. The function returns 1 if the number is positive, 0 (zero) if the number is zero, or -1 if the number is negative.

Syntax

```
SIGN(<number>)
```

Parameters

| TERM | DEFINITION |
|--------|---|
| number | Any real number, a column that contains numbers, or an expression that evaluates to a number. |

Return value

A whole number. The possible Return values are 1, 0, and -1.

| RETURN VALUE | DESCRIPTION |
|--------------|------------------------|
| 1 | The number is positive |
| 0 | The number is zero |
| -1 | The number is negative |

Example

The following formula returns the sign of the result of the expression that calculates sale price minus cost.

```
= SIGN( ([Sale Price] - [Cost]) )
```

See also

[Math and Trig functions](#)

SIN

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the sine of the given angle.

Syntax

```
SIN(number)
```

Parameters

| TERM | DEFINITION |
|--------|---|
| number | Required. The angle in radians for which you want the sine. |

Return value

Returns the sine of the given angle.

Remarks

If an argument is in degrees, multiply it by $\text{PI()}/180$ or use the **RADIANS** function to convert it to radians.

Example

| FORMULA | DESCRIPTION | RESULT |
|--------------------|--|--------|
| = SIN(PI()) | Sine of pi radians (0, approximately). | 0.0 |
| = SIN(PI()/2) | Sine of pi/2 radians. | 1.0 |
| = SIN(30*PI()/180) | Sine of 30 degrees. | 0.5 |
| = SIN(RADIANS(30)) | Sine of 30 degrees. | 0.5 |

SINH

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the hyperbolic sine of a number.

Syntax

`SINH(number)`

Parameters

| TERM | DEFINITION |
|--------|----------------------------|
| number | Required. Any real number. |

Return value

Returns the hyperbolic sine of a number.

Remarks

- The formula for the hyperbolic sine is:

$$\text{SINH}(z) = \frac{e^z - e^{-z}}{2}$$

- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

Probability of obtaining a result of less than 1.03 seconds.

```
= 2.868*SINH(0.0342*1.03)
```

Returns, 0.1010491

SQRT

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the square root of a number.

Syntax

```
SQRT(<number>)
```

Parameters

| TERM | DEFINITION |
|--------|---|
| number | The number for which you want the square root, a column that contains numbers, or an expression that evaluates to a number. |

Return value

A decimal number.

Remarks

If the number is negative, the SQRT function returns an error.

Example

The following formula,

```
= SQRT(25)
```

See also

[Math and Trig functions](#)

SQRTPI

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the square root of (number * pi).

Syntax

```
SQRTPI(number)
```

Parameters

| TERM | DEFINITION |
|--------|---|
| number | Required. The number by which pi is multiplied. |

Return value

Returns the square root of (number * pi).

Example

| FORMULA | DESCRIPTION | RESULT |
|-------------|------------------------|----------|
| = SQRTPI(1) | Square root of pi. | 1.772454 |
| = SQRTPI(2) | Square root of 2 * pi. | 2.506628 |

TAN

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the tangent of the given angle.

Syntax

TAN(number)

Parameters

| TERM | DEFINITION |
|--------|--|
| number | Required. The angle in radians for which you want the tangent. |

Return value

Returns the tangent of the given angle.

Remarks

If your argument is in degrees, multiply it by $\text{PI()}/180$ or use the RADIANS function to convert it to radians.

Example

| FORMULA | DESCRIPTION | RESULT |
|--------------------|------------------------------------|---------|
| = TAN(0.785) | Tangent of 0.785 radians (0.99920) | 0.99920 |
| = TAN(45*PI()/180) | Tangent of 45 degrees (1) | 1 |
| = TAN(RADIANS(45)) | Tangent of 45 degrees (1) | 1 |

TANH

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the hyperbolic tangent of a number.

Syntax

TANH(number)

Parameters

| TERM | DEFINITION |
|--------|----------------------------|
| number | Required. Any real number. |

Return value

Returns the hyperbolic tangent of a number.

Remarks

- The formula for the hyperbolic tangent is:

$$\text{TANH}(z) = \frac{\text{SINH}(z)}{\text{COSH}(z)}$$

- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

| FORMULA | DESCRIPTION | RESULT |
|-------------|--------------------------------------|-----------|
| = TANH(-2) | Hyperbolic tangent of -2 (-0.96403) | -0.964028 |
| = TANH(0) | Hyperbolic tangent of 0 (0) | 0 |
| = TANH(0.5) | Hyperbolic tangent of 0.5 (0.462117) | 0.462117 |

TRUNC

10/31/2022 • 2 minutes to read • [Edit Online](#)

Truncates a number to an integer by removing the decimal, or fractional, part of the number.

Syntax

```
TRUNC(<number>,<num_digits>)
```

Parameters

| TERM | DEFINITION |
|------------|---|
| number | The number you want to truncate. |
| num_digits | A number specifying the precision of the truncation; if omitted, 0 (zero) |

Return value

A whole number.

Remarks

TRUNC and INT are similar in that both return integers. TRUNC removes the fractional part of the number. INT rounds numbers down to the nearest integer based on the value of the fractional part of the number. INT and TRUNC are different only when using negative numbers: `TRUNC(-4.3)` returns -4, but `INT(-4.3)` returns -5 because -5 is the smaller number.

Example 1

The following formula returns 3, the integer part of pi.

```
= TRUNC(PI())
```

Example 2

The following formula returns -8, the integer part of -8.9.

```
= TRUNC(-8.9)
```

See also

[Math and Trig functions](#)

[ROUND](#)

[ROUNDUP](#)

[ROUNDDOWN](#)

MROUND
INT

Other functions

10/31/2022 • 2 minutes to read • [Edit Online](#)

These functions perform unique actions that cannot be defined by any of the categories.

In this category

| FUNCTION | DESCRIPTION |
|-----------------------|--|
| BLANK | Returns a blank. |
| ERROR | Raises an error with an error message. |

BLANK

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns a blank.

Syntax

```
BLANK()
```

Return value

A blank.

Remarks

- Blanks are not equivalent to nulls. DAX uses blanks for both database nulls and for blank cells in Excel.
- Some DAX functions treat blank cells somewhat differently from Microsoft Excel. Blanks and empty strings ("") are not always equivalent, but some operations may treat them as such.

Example

The following example illustrates how you can work with blanks in formulas. The formula calculates the ratio of sales between the Resellers and the Internet channels. However, before attempting to calculate the ratio the denominator should be checked for zero values. If the denominator is zero then a blank value should be returned; otherwise, the ratio is calculated.

```
= IF( SUM(InternetSales_USD[SalesAmount_USD])= 0 , BLANK() ,  
SUM(ResellerSales_USD[SalesAmount_USD])/SUM(InternetSales_USD[SalesAmount_USD]) )
```

The table shows the expected results when this formula is used to create a table visualization.

| ROW LABELS | ACCESSORIES | BIKES | CLOTHING | GRAND TOTAL |
|-------------|-------------|-------|----------|-------------|
| 2005 | | 2.65 | | 2.89 |
| 2006 | | 3.33 | | 4.03 |
| 2007 | 1.04 | 2.92 | 6.63 | 3.51 |
| 2008 | 0.41 | 1.53 | 2.00 | 1.71 |
| Grand Total | 0.83 | 2.51 | 5.45 | 2.94 |

In the original data source, the column evaluated by the BLANK function might have included text, empty strings, or nulls. If the original data source was a SQL Server database, nulls and empty strings are different kinds of data. However, for this operation an implicit type cast is performed and DAX treats them as the same.

See also

[Text functions](#)

[ISBLANK function](#)

ERROR

10/31/2022 • 2 minutes to read • [Edit Online](#)

Raises an error with an error message.

Syntax

```
ERROR(<text>)
```

Parameters

| TERM | DEFINITION |
|------|--|
| text | A text string containing an error message. |

Return value

None

Remarks

- The ERROR function can be placed in a DAX expression anywhere a scalar value is expected.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example 1

The following DAX query:

```
DEFINE
MEASURE DimProduct[Measure] =
    IF(
        SELECTEDVALUE(DimProduct[Color]) = "Red",
        ERROR("red color encountered"),
        SELECTEDVALUE(DimProduct[Color])
    )
EVALUATE SUMMARIZECOLUMNS(DimProduct[Color], "Measure", [Measure])
ORDER BY [Color]
```

Fails and raises an error message containing "red color encountered".

Example 2

The following DAX query:

```

DEFINE
MEASURE DimProduct[Measure] =
    IF(
        SELECTEDVALUE(DimProduct[Color]) = "Magenta",
        ERROR("magenta color encountered"),
        SELECTEDVALUE(DimProduct[Color])
    )
EVALUATE SUMMARIZECOLUMNS(DimProduct[Color], "Measure", [Measure])
ORDER BY [Color]

```

Returns the following table:

| DIMPRODUCT[COLOR] | [MEASURE] |
|-------------------|--------------|
| Black | Black |
| Blue | Blue |
| Grey | Grey |
| Multi | Multi |
| NA | NA |
| Red | Red |
| Silver | Silver |
| Silver\Black | Silver\Black |
| White | White |
| Yellow | Yellow |

Because Magenta is not one of the product colors, the ERROR function is not executed.

Parent and Child functions

10/31/2022 • 2 minutes to read • [Edit Online](#)

These functions manage data that is presented as parent/child hierarchies. To learn more, see [Understanding functions for Parent-Child Hierarchies in DAX](#).

In this category

| FUNCTION | DESCRIPTION |
|---------------------------------|---|
| PATH | Returns a delimited text string with the identifiers of all the parents of the current identifier. |
| PATHCONTAINS | Returns TRUE if the specified <i>item</i> exists within the specified <i>path</i> . |
| PATHITEM | Returns the item at the specified <i>position</i> from a string resulting from evaluation of a PATH function. |
| PATHITEMREVERSE | Returns the item at the specified <i>position</i> from a string resulting from evaluation of a PATH function. |
| PATHLENGTH | Returns the number of parents to the specified item in a given PATH result, including self. |

Understanding functions for parent-child hierarchies in DAX

10/31/2022 • 3 minutes to read • [Edit Online](#)

DAX provides five functions to help users manage data that is presented as a parent-child hierarchy in their models. With this functions a user can obtain the entire lineage of parents a row has, how many levels has the lineage to the top parent, who is the parent n-levels above the current row, who is the n-descendant from the top of the current row hierarchy and is certain parent a parent in the current row hierarchy?

Parent-child functions in DAX

The following table contains a Parent-Child hierarchy on the columns: **EmployeeKey** and **ParentEmployeeKey** that is used in all the functions examples.

| EMPLOYEEKEY | PARENTEMPLOYEEKEY |
|-------------|-------------------|
| 112 | |
| 14 | 112 |
| 3 | 14 |
| 11 | 3 |
| 13 | 3 |
| 162 | 3 |
| 117 | 162 |
| 221 | 162 |
| 81 | 162 |

In the above table you can see that employee 112 has no parent defined, employee 14 has employee 112 as manager (ParentEmployeeKey), employee 3 has employee 14 as manager and employees 11, 13, and 162 have employee 3 as manager. The above helps to understand that employee 112 has no manager above her/him and she/he is the top manager for all employees shown here; also, employee 3 reports to employee 14 and employees 11, 13, 162 report to 3.

The following table presents the available functions, a brief description of the function and an example of the function over the same data shown above.

PATH function - Returns a delimited text with the identifiers of all the parents to the current row, starting with the oldest or top most until current.

| EMPLOYEEKEY | PARENTEMPLOYEEKEY | PATH |
|-------------|-------------------|------|
| 112 | | 112 |

| EMPLOYEEKEY | PARENTEMPLOYEEKEY | PATH |
|-------------|-------------------|------------------|
| 14 | 112 | 112 14 |
| 3 | 14 | 112 14 3 |
| 11 | 3 | 112 14 3 11 |
| 13 | 3 | 112 14 3 13 |
| 162 | 3 | 112 14 3 162 |
| 117 | 162 | 112 14 3 162 117 |
| 221 | 162 | 112 14 3 162 221 |
| 81 | 162 | 112 14 3 162 81 |

PATHLENGTH function - Returns the number of levels in a given PATH(), starting at current level until the oldest or top most parent level. In the following example column PathLength is defined as '`= PATHLENGTH([Path])`'; the example includes all data from the Path() example to help understand how this function works.

| EMPLOYEEKEY | PARENTEMPLOYEEKEY | PATH | PATHLENGTH |
|-------------|-------------------|------------------|------------|
| 112 | | 112 | 1 |
| 14 | 112 | 112 14 | 2 |
| 3 | 14 | 112 14 3 | 3 |
| 11 | 3 | 112 14 3 11 | 4 |
| 13 | 3 | 112 14 3 13 | 4 |
| 162 | 3 | 112 14 3 162 | 4 |
| 117 | 162 | 112 14 3 162 117 | 5 |
| 221 | 162 | 112 14 3 162 221 | 5 |
| 81 | 162 | 112 14 3 162 81 | 5 |

PATHITEM function - Returns the item at the specified position from a PATH() like result, counting from left to right. In the following example column PathItem - 4th from left is defined as '`= PATHITEM([Path], 4)`'; this example returns the EmployeeKey at fourth position in the Path string from the left, using the same sample data from the Path() example.

| EMPLOYEEKEY | PARENTEMPLOYEEKEY | PATH | PATHITEM - 4TH FROM LEFT |
|-------------|-------------------|--------|--------------------------|
| 112 | | 112 | |
| 14 | 112 | 112 14 | |

| EMPLOYEEKEY | PARENTEMPLOYEEKEY | PATH | PATHITEM - 4TH FROM LEFT |
|-------------|-------------------|------|--------------------------|
|-------------|-------------------|------|--------------------------|

| | | | |
|-----|-----|------------------|-----|
| 3 | 14 | 112 14 3 | |
| 11 | 3 | 112 14 3 11 | 11 |
| 13 | 3 | 112 14 3 13 | 13 |
| 162 | 3 | 112 14 3 162 | 162 |
| 117 | 162 | 112 14 3 162 117 | 162 |
| 221 | 162 | 112 14 3 162 221 | 162 |
| 81 | 162 | 112 14 3 162 81 | 162 |

PATHITEMREVERSE function - Returns the item at *position* from a PATH() like function result, counting backwards from right to left.

In the following example column PathItemReverse - 3rd from right is defined as '`= PATHITEMREVERSE([Path], 3)`'; this example returns the EmployeeKey at third position in the Path string from the right, using the same sample data from the Path() example.

| EMPLOYEEKEY | PARENTEMPLOYEEKEY | PATH | PATHITEMREVERSE - 3RD FROM RIGHT |
|-------------|-------------------|------------------|----------------------------------|
| 112 | | 112 | |
| 14 | 112 | 112 14 | |
| 3 | 14 | 112 14 3 | 112 |
| 11 | 3 | 112 14 3 11 | 14 |
| 13 | 3 | 112 14 3 13 | 14 |
| 162 | 3 | 112 14 3 162 | 14 |
| 117 | 162 | 112 14 3 162 117 | 3 |
| 221 | 162 | 112 14 3 162 221 | 3 |
| 81 | 162 | 112 14 3 162 81 | 3 |

PATHCONTAINS function - Returns **TRUE** if the specified *item* exists within the specified *path*. In the following example column PathContains - employee 162 is defined as '`= PATHCONTAINS([Path], "162")`'; this example returns **TRUE** if the given path contains employee 162. This example uses the results from the Path() example above.

| EMPLOYEEKEY | PARENTEMPLOYEEKEY | PATH | PATHCONTAINS - EMPLOYEE 162 |
|-------------|-------------------|------------------|--------------------------------|
| 112 | | 112 | FALSE |
| 14 | 112 | 112 14 | FALSE |
| 3 | 14 | 112 14 3 | FALSE |
| 11 | 3 | 112 14 3 11 | FALSE |
| 13 | 3 | 112 14 3 13 | FALSE |
| 162 | 3 | 112 14 3 162 | TRUE |
| 117 | 162 | 112 14 3 162 117 | TRUE |

PATH

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns a delimited text string with the identifiers of all the parents of the current identifier, starting with the oldest and continuing until current.

Syntax

```
PATH(<ID_columnName>, <parent_columnName>)
```

Parameters

| TERM | DEFINITION |
|-------------------|--|
| ID_columnName | The name of an existing column containing the unique identifier for rows in the table. This cannot be an expression. The data type of the value in <i>ID_columnName</i> must be text or integer, and must also be the same data type as the column referenced in <i>parent_columnName</i> . |
| parent_columnName | The name of an existing column containing the unique identifier for the parent of the current row. This cannot be an expression. The data type of the value in <i>parent_columnName</i> data type must be text or integer, and must be the same data type as the value in <i>ID_columnName</i> . |

Return value

A delimited text string containing the identifiers of all the parents to the current identifier.

Remarks

- This function is used in tables that have some kind of internal hierarchy, to return the items that are related to the current row value. For example, in an Employees table that contains employees, the managers of employees, and the managers of the managers, you can return the path that connects an employee to his or her manager.
- The path is not constrained to a single level of parent-child relationships; it can return related rows that are several levels up from the specified starting row.
 - The delimiter used to separate the ascendants is the vertical bar, '|'.
 - The values in *ID_columnName* and *parent_columnName* must have the same data type, text or integer.
 - Values in *parent_columnName* must be present in *ID_columnName*. That is, you cannot look up a parent if there is no value at the child level.
 - If *parent_columnName* is BLANK then PATH() returns *ID_columnName* value. In other words, if you look for the manager of an employee but the *parent_columnName* column has no data, the PATH function returns just the employee ID.
 - If *ID_columnName* has duplicates and *parent_columnName* is the same for those duplicates then PATH() returns the common *parent_columnName* value; however, if *parent_columnName* value is different for those duplicates then PATH() returns an error. In other words, if you have two listings for

the same employee ID and they have the same manager ID, the PATH function returns the ID for that manager. However, if there are two identical employee IDs that have different manager IDs, the PATH function returns an error.

- If *ID_columnName* is BLANK then PATH() returns BLANK.
- If *ID_columnName* contains a vertical bar '|' then PATH() returns an error.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following example creates a calculated column that lists all the managers for each employee.

```
= PATH(Employee[EmployeeKey], Employee[ParentEmployeeKey])
```

PATHCONTAINS

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns **TRUE** if the specified *item* exists within the specified *path*.

Syntax

```
PATHCONTAINS(<path>, <item>)
```

Parameters

| TERM | DEFINITION |
|------|---|
| path | A string created as the result of evaluating a PATH function. |
| item | A text expression to look for in the path result. |

Return value

A value of **TRUE** if *item* exists in *path*; otherwise **FALSE**.

Remarks

- If *item* is an integer number it is converted to text and then the function is evaluated. If conversion fails then the function returns an error.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following example creates a calculated column that takes a manager ID and checks a set of employees. If the manager ID is among the list of managers returned by the PATH function, the PATHCONTAINS function returns true; otherwise it returns false.

```
= PATHCONTAINS(PATH(Employee[EmployeeKey], Employee[ParentEmployeeKey]), "23")
```

PATHITEM

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the item at the specified *position* from a string resulting from evaluation of a PATH function. Positions are counted from left to right.

Syntax

```
PATHITEM(<path>, <position>[, <type>])
```

Parameters

| TERM | DEFINITION |
|----------|---|
| path | A text string in the form of the results of a PATH function. |
| position | An integer expression with the position of the item to be returned. |
| type | (Optional)An enumeration that defines the data type of the result: |

type enumeration

| ENUMERATION | ALTERNATE ENUMERATION | DESCRIPTION |
|-------------|-----------------------|--|
| TEXT | 0 | Results are returned with the data type text. (default). |
| INTEGER | 1 | Results are returned as integers. |

Return value

The identifier returned by the PATH function at the specified position in the list of identifiers. Items returned by the PATH function are ordered by most distant to current.

Remarks

- This function can be used to return a specific level from a hierarchy returned by a PATH function. For example, you could return just the skip-level managers for all employees.
- If you specify a number for *position* that is less than one (1) or greater than the number of elements in *path*, the PATHITEM function returns BLANK
- If *type* is not a valid enumeration element an error is returned.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following example returns the third tier manager of the current employee; it takes the employee and manager IDs as the input to a PATH function that returns a string with the hierarchy of parents to current employee. From that string PATHITEM returns the third entry as an integer.

```
= PATHITEM(PATH(Employee[EmployeeKey], Employee[ParentEmployeeKey]), 3, 1)
```

PATHITEMREVERSE

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the item at the specified *position* from a string resulting from evaluation of a PATH function. Positions are counted backwards from right to left.

Syntax

```
PATHITEMREVERSE(<path>, <position>[, <type>])
```

Parameters

| TERM | DEFINITION |
|----------|---|
| path | A text string resulting from evaluation of a PATH function. |
| position | An integer expression with the position of the item to be returned. Position is counted backwards from right to left. |
| type | (Optional)An enumeration that defines the data type of the result: |

type enumeration

| ENUMERATION | ALTERNATE ENUMERATION | DESCRIPTION |
|-------------|-----------------------|--|
| TEXT | 0 | Results are returned with the data type text. (default). |
| INTEGER | 1 | Results are returned as integers. |

Return value

The n-position ascendant in the given path, counting from current to the oldest.

Remarks

- This function can be used to get an individual item from a hierarchy resulting from a PATH function.
- This function reverses the standard order of the hierarchy, so that closest items are listed first. For example, if the PATH function returns a list of managers above an employee in a hierarchy, the PATHITEMREVERSE function returns the employee's immediate manager in position 2 because position 1 contains the employee's id.
- If the number specified for *position* is less than one (1) or greater than the number of elements in *path*, the PATHITEM function returns BLANK.
- If *type* is not a valid enumeration element an error is returned.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following example takes an employee ID column as the input to a PATH function, and reverses the list of grandparent elements that are returned. The position specified is 3 and the return type is 1; therefore, the PATHITEMREVERSE function returns an integer representing the manager two levels up from the employee.

```
= PATHITEMREVERSE(PATH(Employee[EmployeeKey], Employee[ParentEmployeeKey]), 3, 1)
```

PATHLENGTH

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the number of parents to the specified item in a given PATH result, including self.

Syntax

```
PATHLENGTH(<path>)
```

Parameters

| TERM | DEFINITION |
|------|---|
| path | A text expression resulting from evaluation of a PATH function. |

Return value

The number of items that are parents to the specified item in a given PATH result, including the specified item.

Remarks

This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following example takes an employee ID as input to a PATH function and returns a list of the managers above that employee in the hierarchy. The PATHLENGTH function takes that result and counts the different levels of employees and managers, including the employee you started with.

```
= PATHLENGTH(PATH(Employee[EmployeeKey], Employee[ParentEmployeeKey]))
```


Relationship functions

10/31/2022 • 2 minutes to read • [Edit Online](#)

Functions in this category are for managing and utilizing relationships between tables.

In this category

| FUNCTION | DESCRIPTION |
|---------------------------------|---|
| CROSSFILTER | Specifies the cross-filtering direction to be used in a calculation for a relationship that exists between two columns. |
| RELATED | Returns a related value from another table. |
| RELATEDTABLE | Evaluates a table expression in a context modified by the given filters. |
| USERELATIONSHIP | Specifies the relationship to be used in a specific calculation as the one that exists between columnName1 and columnName2. |

CROSSFILTER

10/31/2022 • 2 minutes to read • [Edit Online](#)

Specifies the cross-filtering direction to be used in a calculation for a relationship that exists between two columns.

Syntax

```
CROSSFILTER(<columnName1>, <columnName2>, <direction>)
```

Parameters

| TERM | DEFINITION |
|-------------|--|
| columnName1 | The name of an existing column, using standard DAX syntax and fully qualified, that usually represents the many side of the relationship to be used; if the arguments are given in reverse order the function will swap them before using them. This argument cannot be an expression. |
| columnName2 | The name of an existing column, using standard DAX syntax and fully qualified, that usually represents the one side or lookup side of the relationship to be used; if the arguments are given in reverse order the function will swap them before using them. This argument cannot be an expression. |
| Direction | <p>The cross-filter direction to be used. Must be one of the following:</p> <p>None - No cross-filtering occurs along this relationship.</p> <p>Both - Filters on either side filters the other side.</p> <p>OneWay - Filters on the one side or the lookup side of a relationship filter the other side. This option cannot be used with a one-to-one relationship . Don't use this option on a many-to-many relationship because it is unclear which side is the lookup side; use OneWay_LeftFiltersRight or OneWay_RightFiltersLeft instead.</p> <p>OneWay_LeftFiltersRight - Filters on the side of <columnName1> filter the side of <columnName2>. This option cannot be used with a one-to-one or many-to-one relationship.</p> <p>OneWay_RightFiltersLeft - Filters on the side of <columnName2> filter the side of <columnName1>. This option cannot be used with a one-to-one or many-to-one relationship.</p> |

Return value

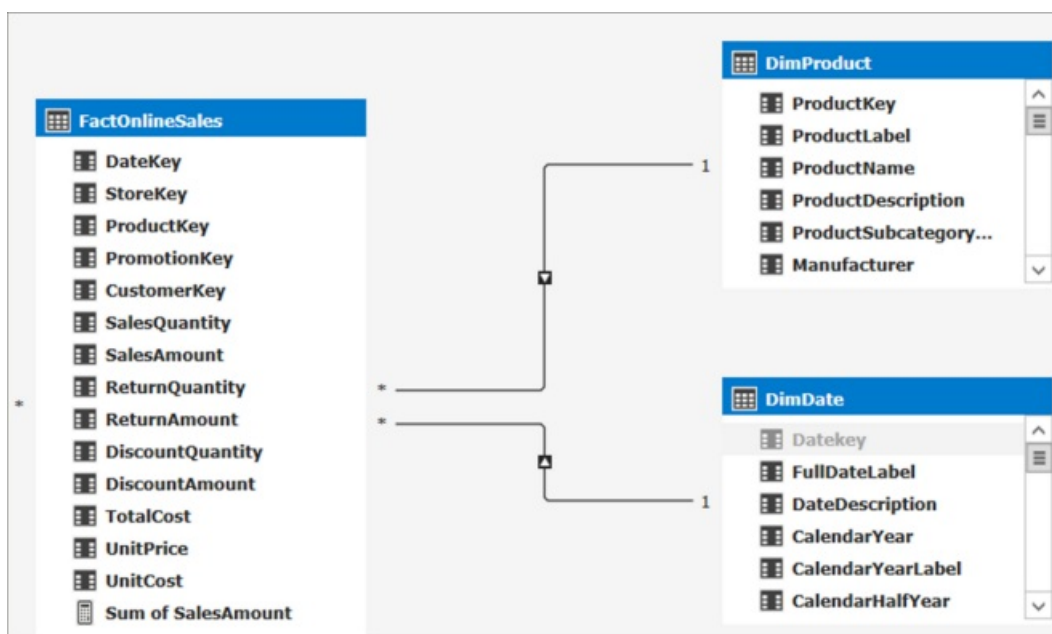
The function returns no value; the function only sets the cross-filtering direction for the indicated relationship, for the duration of the query.

Remarks

- In the case of a 1:1 relationship, there is no difference between the one and both direction.
- CROSSFILTER can only be used in functions that take a filter as an argument, for example: CALCULATE, CALCULATETABLE, CLOSINGBALANCEMONTH, CLOSINGBALANCEQUARTER, CLOSINGBALANCEYEAR, OPENINGBALANCEMONTH, OPENINGBALANCEQUARTER, OPENINGBALANCEYEAR, TOTALMTD, TOTALQTD and TOTALYTD functions.
- CROSSFILTER uses existing relationships in the model, identifying relationships by their ending point columns.
- In CROSSFILTER, the cross-filtering setting of a relationship is not important; that is, whether the relationship is set to filter one, or both directions in the model does not affect the usage of the function. CROSSFILTER will override any existing cross-filtering setting.
- An error is returned if any of the columns named as an argument is not part of a relationship or the arguments belong to different relationships.
- If CALCULATE expressions are nested, and more than one CALCULATE expression contains a CROSSFILTER function, then the innermost CROSSFILTER is the one that prevails in case of a conflict or ambiguity.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

In the following model diagram, both DimProduct and DimDate have a single direction relationship with FactOnlineSales.



By default, we cannot get the Count of Products sold by year:

| Row Labels | Sum of SalesAmount | Distinct Count of ProductKey |
|--------------------|------------------------|------------------------------|
| 2005 | \$3,266,373.66 | 606 |
| 2006 | \$6,530,343.53 | 606 |
| 2007 | \$9,791,060.30 | 606 |
| 2008 | \$9,770,899.74 | 606 |
| 2009 | | 606 |
| 2010 | | 606 |
| Grand Total | \$29,358,677.22 | 606 |

There are two ways to get the count of products by year:

- Turn on bi-directional cross-filtering on the relationship. This will change how filters work for all data between these two tables.
- Use the CROSSFILTER function to change how the relationships work for just this measure.

When using DAX, we can use the CROSSFILTER function to change how the cross-filter direction behaves between two columns defined by a relationship. In this case, the DAX expression looks like this:

```
BiDi:= CALCULATE([Distinct Count of ProductKey], CROSSFILTER(FactInternetSales[ProductKey],
DimProduct[ProductKey] , Both))
```

By using the CROSSFILTER function in our measure expression, we get the expected results:

| Row Labels | Sum of SalesAmount | Distinct Count of ProductKey | BiDi |
|--------------------|------------------------|------------------------------|------------|
| 2005 | \$3,266,373.66 | 606 | 25 |
| 2006 | \$6,530,343.53 | 606 | 56 |
| 2007 | \$9,791,060.30 | 606 | 133 |
| 2008 | \$9,770,899.74 | 606 | 102 |
| 2009 | | 606 | |
| 2010 | | 606 | |
| Grand Total | \$29,358,677.22 | 606 | 606 |

RELATED

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns a related value from another table.

Syntax

```
RELATED(<column>)
```

Parameters

| TERM | DEFINITION |
|--------|---|
| column | The column that contains the values you want to retrieve. |

Return value

A single value that is related to the current row.

Remarks

- The RELATED function requires that a relationship exists between the current table and the table with related information. You specify the column that contains the data that you want, and the function follows an existing many-to-one relationship to fetch the value from the specified column in the related table. If a relationship does not exist, you must create a relationship.
- When the RELATED function performs a lookup, it examines all values in the specified table regardless of any filters that may have been applied.
- The RELATED function needs a row context; therefore, it can only be used in calculated column expression, where the current row context is unambiguous, or as a nested function in an expression that uses a table scanning function. A table scanning function, such as SUMX, gets the value of the current row value and then scans another table for instances of that value.
- The RELATED function cannot be used to fetch a column across a [limited relationship](#).

Example

In the following example, the measure Non USA Internet Sales is created to produce a sales report that excludes sales in the United States. In order to create the measure, the InternetSales_USD table must be filtered to exclude all sales that belong to the United States in the SalesTerritory table. The United States, as a country, appears 5 times in the SalesTerritory table; once for each of the following regions: Northwest, Northeast, Central, Southwest, and Southeast.

The first approach to filter the Internet Sales, in order to create the measure, could be to add a filter expression like the following:

```

FILTER('InternetSales_USD'
, 'InternetSales_USD'[SalesTerritoryKey]<>1 && 'InternetSales_USD'[SalesTerritoryKey]<>2 &&
'InternetSales_USD'[SalesTerritoryKey]<>3 && 'InternetSales_USD'[SalesTerritoryKey]<>4 &&
'InternetSales_USD'[SalesTerritoryKey]<>5)

```

However, this approach is counterintuitive, prone to typing errors, and might not work if any of the existing regions is split in the future.

A better approach would be to use the existing relationship between InternetSales_USD and SalesTerritory and explicitly state that the country must be different from the United States. To do so, create a filter expression like the following:

```

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

```

This expression uses the RELATED function to lookup the country value in the SalesTerritory table, starting with the value of the key column, SalesTerritoryKey, in the InternetSales_USD table. The result of the lookup is used by the filter function to determine if the InternetSales_USD row is filtered or not.

NOTE

If the example does not work, you might need to create a relationship between the tables.

```

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

```

The following table shows only totals for each region, to prove that the filter expression in the measure, Non USA Internet Sales, works as intended.

| ROW LABELS | INTERNET SALES | NON USA INTERNET SALES |
|----------------|-----------------|------------------------|
| Australia | \$4,999,021.84 | \$4,999,021.84 |
| Canada | \$1,343,109.10 | \$1,343,109.10 |
| France | \$2,490,944.57 | \$2,490,944.57 |
| Germany | \$2,775,195.60 | \$2,775,195.60 |
| United Kingdom | \$5,057,076.55 | \$5,057,076.55 |
| United States | \$9,389,479.79 | |
| Grand Total | \$26,054,827.45 | \$16,665,347.67 |

The following shows what that you might get if you used this measure in a report table visual:

| ROW LABELS | ACCESSORIES | BIKES | CLOTHING | GRAND TOTAL |
|------------|-------------|----------------|----------|----------------|
| 2005 | | \$1,526,481.95 | | \$1,526,481.95 |

| ROW LABELS | ACCESSORIES | BIKES | CLOTHING | GRAND TOTAL |
|-------------|--------------|-----------------|--------------|-----------------|
| 2006 | | \$3,554,744.04 | | \$3,554,744.04 |
| 2007 | \$156,480.18 | \$5,640,106.05 | \$70,142.77 | \$5,866,729.00 |
| 2008 | \$228,159.45 | \$5,386,558.19 | \$102,675.04 | \$5,717,392.68 |
| Grand Total | \$384,639.63 | \$16,107,890.23 | \$172,817.81 | \$16,665,347.67 |

See also

[RELATEDTABLE](#)

[Filter functions](#)

RELATEDTABLE

10/31/2022 • 2 minutes to read • [Edit Online](#)

Evaluates a table expression in a context modified by the given filters.

Syntax

```
RELATEDTABLE(<tableName>)
```

Parameters

| TERM | DEFINITION |
|-----------|--|
| tableName | The name of an existing table using standard DAX syntax. It cannot be an expression. |

Return value

A table of values.

Remarks

- The RELATEDTABLE function changes the context in which the data is filtered, and evaluates the expression in the new context that you specify.
- This function is a shortcut for CALCULATETABLE function with no logical expression.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following example uses the RELATEDTABLE function to create a calculated column with the Internet Sales in the Product Category table:

```
= SUMX( RELATEDTABLE('InternetSales_USD')  
    , [SalesAmount_USD])
```

The following table shows the results:

| PRODUCT CATEGORY KEY | PRODUCT CATEGORY ALTERNATEKEY | PRODUCT CATEGORY NAME | INTERNET SALES |
|----------------------|-------------------------------|-----------------------|-----------------|
| 1 | 1 | Bikes | \$28,318,144.65 |
| 2 | 2 | Components | |
| 3 | 3 | Clothing | \$339,772.61 |

| PRODUCT CATEGORY KEY | PRODUCT CATEGORY ALTERNATEKEY | PRODUCT CATEGORY NAME | INTERNET SALES |
|----------------------|----------------------------------|--------------------------|----------------|
| 4 | 4 | Accessories | \$700,759.96 |

See also

[CALCULATETABLE](#)
[Filter functions](#)

USERELATIONSHIP

10/31/2022 • 2 minutes to read • [Edit Online](#)

Specifies the relationship to be used in a specific calculation as the one that exists between columnName1 and columnName2.

Syntax

```
USERELATIONSHIP(<columnName1>,<columnName2>)
```

Parameters

| TERM | DEFINITION |
|-------------|--|
| columnName1 | The name of an existing column, using standard DAX syntax and fully qualified, that usually represents the many side of the relationship to be used; if the arguments are given in reverse order the function will swap them before using them. This argument cannot be an expression. |
| columnName2 | The name of an existing column, using standard DAX syntax and fully qualified, that usually represents the one side or lookup side of the relationship to be used; if the arguments are given in reverse order the function will swap them before using them. This argument cannot be an expression. |

Return value

The function returns no value; the function only enables the indicated relationship for the duration of the calculation.

Remarks

- USERELATIONSHIP can only be used in functions that take a filter as an argument, for example: CALCULATE, CALCULATETABLE, CLOSINGBALANCEMONTH, CLOSINGBALANCEQUARTER, CLOSINGBALANCEYEAR, OPENINGBALANCEMONTH, OPENINGBALANCEQUARTER, OPENINGBALANCEYEAR, TOTALMTD, TOTALQTD and TOTALYTD functions.

- USERELATIONSHIP cannot be used when row level security is defined for the table in which the measure is included. For example,

```
CALCULATE(SUM([SalesAmount]), USERELATIONSHIP(FactInternetSales[CustomerKey],  
DimCustomer[CustomerKey]))
```

will return an error if row level security is defined for DimCustomer.

- USERELATIONSHIP uses existing relationships in the model, identifying relationships by their ending point columns.
- In USERELATIONSHIP, the status of a relationship is not important; that is, whether the relationship is active or not does not affect the usage of the function. Even if the relationship is inactive, it will be used and overrides any other active relationships that might be present in the model but not mentioned in the function arguments.

- An error is returned if any of the columns named as an argument is not part of a relationship or the arguments belong to different relationships.
- If multiple relationships are needed to join table A to table B in a calculation, each relationship must be indicated in a different USERELATIONSHIP function.
- If CALCULATE expressions are nested, and more than one CALCULATE expression contains a USERELATIONSHIP function, then the innermost USERELATIONSHIP is the one that prevails in case of a conflict or ambiguity.
- Up to 10 USERELATIONSHIP functions can be nested; however, your expression might have a deeper level of nesting, ie. the following sample expression is nested 3 levels deep but only 2 for USERELATIONSHIP:

```
=CALCULATE(CALCULATE( CALCULATE( <anyExpression>, USERELATIONSHIP( t1[colA], t2[colB])),
t99[colZ]=999), USERELATIONSHIP( t1[colA], t2[colA]))
```

- For 1-to-1 relationships, USERELATIONSHIP will only activate the relationship in one direction. In particular, filters will only be able to flow from *columnName2's* table to *columnName1's* table. If bi-directional cross-filtering is desired, two USERELATIONSHIPS with opposite directionality can be used in the same calculation. For example,

```
CALCULATE(..., USERELATIONSHIP(T1[K], T2[K]), USERELATIONSHIP(T2[K], T1[K])) .
```

Example

The following sample shows how to override the default, active, relationship between InternetSales and DateTime tables. The default relationship exists between the OrderDate column, in the InternetSales table, and the Date column, in the DateTime table.

To calculate the sum of internet sales and allow slicing by ShippingDate instead of the traditional OrderDate, create measure, [InternetSales by ShippingDate] using the following expression:

```
= CALCULATE(SUM(InternetSales[SalesAmount]), USERELATIONSHIP(InternetSales[ShippingDate], DateTime[Date]))
```

Relationships between InternetSales[ShipmentDate] and DateTime[Date] must exist and should not be the active relationship; also, the relationship between InternetSales[OrderDate] and DateTime[Date] should exist and should be the active relationship.

Statistical functions

10/31/2022 • 2 minutes to read • [Edit Online](#)

Statistical functions calculate values related to statistical distributions and probability, such as standard deviation and number of permutations.

In this category

| FUNCTION | DESCRIPTION |
|---------------------------------|--|
| BETA.DIST | Returns the beta distribution. |
| BETA.INV | Returns the inverse of the beta cumulative probability density function (BETA.DIST). |
| CHISQ.DIST | Returns the chi-squared distribution. |
| CHISQ.DIST. RT | Returns the right-tailed probability of the chi-squared distribution. |
| CHISQ.INV | Returns the inverse of the left-tailed probability of the chi-squared distribution. |
| CHISQ.INV. RT | Returns the inverse of the right-tailed probability of the chi-squared distribution. |
| COMBIN | Returns the number of combinations for a given number of items. |
| COMBINA | Returns the number of combinations (with repetitions) for a given number of items. |
| CONFIDENCE.NORM | The confidence interval is a range of values. |
| CONFIDENCE.T | Returns the confidence interval for a population mean, using a Student's t distribution. |
| EXPON.DIST | Returns the exponential distribution. |
| GEOMEAN | Returns the geometric mean of the numbers in a column. |
| GEOMEANX | Returns the geometric mean of an expression evaluated for each row in a table. |
| MEDIAN | Returns the median of numbers in a column. |
| MEDIANX | Returns the median number of an expression evaluated for each row in a table. |

| FUNCTION | DESCRIPTION |
|-----------------|--|
| NORM.DIST | Returns the normal distribution for the specified mean and standard deviation. |
| NORM.INV | The inverse of the normal cumulative distribution for the specified mean and standard deviation. |
| NORM.S.DIST | Returns the standard normal distribution (has a mean of zero and a standard deviation of one). |
| NORM.S.INV | Returns the inverse of the standard normal cumulative distribution. |
| PERCENTILE.EXC | Returns the k-th percentile of values in a range, where k is in the range 0..1, exclusive. |
| PERCENTILE.INC | Returns the k-th percentile of values in a range, where k is in the range 0..1, inclusive. |
| PERCENTILEX.EXC | Returns the percentile number of an expression evaluated for each row in a table. |
| PERCENTILEX.INC | Returns the percentile number of an expression evaluated for each row in a table. |
| PERMUT | Returns the number of permutations for a given number of objects that can be selected from number objects. |
| POISSON.DIST | Returns the Poisson distribution. |
| RANK.EQ | Returns the ranking of a number in a list of numbers. |
| RANKX | Returns the ranking of a number in a list of numbers for each row in the <i>table</i> argument. |
| SAMPLE | Returns a sample of N rows from the specified table. |
| STDEV.P | Returns the standard deviation of the entire population. |
| STDEV.S | Returns the standard deviation of a sample population. |
| STDEVX.P | Returns the standard deviation of the entire population. |
| STDEVX.S | Returns the standard deviation of a sample population. |
| T.DIST | Returns the Student's left-tailed t-distribution. |
| T.DIST.2T | Returns the two-tailed Student's t-distribution. |
| T.DISTRT | Returns the right-tailed Student's t-distribution. |
| T.INV | Returns the left-tailed inverse of the Student's t-distribution. |

| FUNCTION | DESCRIPTION |
|----------|---|
| T.INV.2t | Returns the two-tailed inverse of the Student's t-distribution. |
| VAR.P | Returns the variance of the entire population. |
| VAR.S | Returns the variance of a sample population. |
| VARX.P | Returns the variance of the entire population. |
| VARX.S | Returns the variance of a sample population. |

BETA.DIST

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the beta distribution. The beta distribution is commonly used to study variation in the percentage of something across samples, such as the fraction of the day people spend watching television.

Syntax

```
BETA.DIST(x,alpha,beta,cumulative,[A],[B])
```

Parameters

| TERM | DEFINITION |
|-------|---|
| x | The value between A and B at which to evaluate the function |
| Alpha | A parameter of the distribution. |
| Beta | A parameter of the distribution. |
| A | Optional. A lower bound to the interval of x. |
| B | Optional. An upper bound to the interval of x. |

Return value

Returns the beta distribution.

Remarks

- If any argument is nonnumeric, BETA.DIST returns the #VALUE! error value.
- If any argument is not an integer, it is rounded.
- If $\alpha \leq 0$ or $\beta \leq 0$, BETA.DIST returns the #NUM! error value.
- If $x < A$, $x > B$, or $A = B$, BETA.DIST returns the #NUM! error value.
- If you omit values for A and B, BETA.DIST uses the standard cumulative beta distribution, so that $A = 0$ and $B = 1$.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

BETA.INV

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the inverse of the beta cumulative probability density function (BETA.DIST).

If $\text{probability} = \text{BETA.DIST}(x, \dots, \text{TRUE})$, then $\text{BETA.INV}(\text{probability}, \dots) = x$. The beta distribution can be used in project planning to model probable completion times given an expected completion time and variability.

Syntax

```
BETA.INV(probability, alpha, beta, [A], [B])
```

Parameters

| TERM | DEFINITION |
|-------------|--|
| Probability | A probability associated with the beta distribution. |
| Alpha | A parameter of the distribution. |
| Beta | A parameter the distribution. |
| A | Optional. A lower bound to the interval of x. |
| B | Optional. An upper bound to the interval of x. |

Return value

Returns the inverse of the beta cumulative probability density function (BETA.DIST).

Remarks

- If any argument is nonnumeric, BETA.INV returns the #VALUE! error value.
- If any argument is not an integer, it is rounded.
- If $\alpha \leq 0$ or $\beta \leq 0$, BETA.INV returns the #NUM! error value.
- If $\text{probability} \leq 0$ or $\text{probability} > 1$, BETA.INV returns the #NUM! error value.
- If you omit values for A and B, BETA.INV uses the standard cumulative beta distribution, so that $A = 0$ and $B = 1$.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

CHISQ.DIST

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the chi-squared distribution.

The chi-squared distribution is commonly used to study variation in the percentage of something across samples, such as the fraction of the day people spend watching television.

Syntax

```
CHISQ.DIST(<x>, <deg_freedom>, <cumulative>)
```

Parameters

| TERM | DEFINITION |
|-------------|--|
| x | The value at which you want to evaluate the distribution. |
| Deg_freedom | The number of degrees of freedom. |
| cumulative | A logical value that determines the form of the function. If cumulative is TRUE, CHISQ.DIST returns the cumulative distribution function; if FALSE, it returns the probability density function. |

Return value

The chi-squared distribution.

Remarks

- If x or deg_freedom is nonnumeric, an error is returned.
- If deg_freedom is not an integer, it is rounded.
- If $x < 0$, an error is returned.
- If $\text{deg_freedom} < 1$ or $\text{deg_freedom} > 10^{10}$, an error is returned.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following DAX query,

```
EVALUATE { CHISQ.DIST(2, 2, TRUE) }
```

Returns

| |
|-------------------|
| [VALUE] |
| 0.632120558828558 |

CHISQ.DIST.RT

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the right-tailed probability of the chi-squared distribution.

The chi-squared distribution is associated with a chi-squared test. Use the chi-squared test to compare observed and expected values. For example, a genetic experiment might hypothesize that the next generation of plants will exhibit a certain set of colors. By comparing the observed results with the expected ones, you can decide whether your original hypothesis is valid.

Syntax

```
CHISQ.DIST.RT(<x>, <deg_freedom>)
```

Parameters

| TERM | DEFINITION |
|-------------|---|
| x | The value at which you want to evaluate the distribution. |
| Deg_freedom | The number of degrees of freedom. |

Return value

The right-tailed probability of the chi-squared distribution.

Remarks

- If x or deg_freedom is nonnumeric, an error is returned.
- If deg_freedom is not an integer, it is rounded.
- If $x < 0$, an error is returned.
- If $\text{deg_freedom} < 1$ or $\text{deg_freedom} > 10^{10}$, an error is returned.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following DAX query,

```
EVALUATE { CHISQ.DIST.RT(2, 5) }
```

Returns

[VALUE]

0.84914503608461

CHISQ.INV

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the inverse of the left-tailed probability of the chi-squared distribution.

The chi-squared distribution is commonly used to study variation in the percentage of something across samples, such as the fraction of the day people spend watching television.

Syntax

```
CHISQ.INV(probability,deg_freedom)
```

Parameters

| TERM | DEFINITION |
|-------------|---|
| Probability | A probability associated with the chi-squared distribution. |
| Deg_freedom | The number of degrees of freedom. |

Return value

Returns the inverse of the left-tailed probability of the chi-squared distribution.

Remarks

- If argument is nonnumeric, CHISQ.INV returns the #VALUE! error value.
- If probability < 0 or probability > 1, CHISQ.INV returns the #NUM! error value.
- If deg_freedom is not an integer, it is rounded.
- If deg_freedom < 0 or deg_freedom > 10^10, CHISQ.INV returns the #NUM! error value.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

| FORMULA | DESCRIPTION | RESULT |
|---------------------|---|-------------|
| = CHISQ.INV(0.93,1) | Inverse of the left-tailed probability of the chi-squared distribution for 0.93, using 1 degree of freedom. | 5.318520074 |
| = CHISQ.INV(0.6,2) | Inverse of the left-tailed probability of the chi-squared distribution for 0.6, using 2 degrees of freedom. | 1.832581464 |

CHISQ.INV.RT

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the inverse of the right-tailed probability of the chi-squared distribution.

If $\text{probability} = \text{CHISQ.DIST.RT}(x, \dots)$, then $\text{CHISQ.INV.RT}(\text{probability}, \dots) = x$. Use this function to compare observed results with expected ones in order to decide whether your original hypothesis is valid.

Syntax

```
CHISQ.INV.RT(probability,deg_freedom)
```

Parameters

| TERM | DEFINITION |
|-------------|---|
| Probability | A probability associated with the chi-squared distribution. |
| Deg_freedom | The number of degrees of freedom. |

Return value

Returns the inverse of the right-tailed probability of the chi-squared distribution.

Remarks

- If either argument is nonnumeric, CHISQ.INV.RT returns the #VALUE! error value.
- If $\text{probability} < 0$ or $\text{probability} > 1$, CHISQ.INV.RT returns the #NUM! error value.
- If deg_freedom is not an integer, it is rounded.
- If $\text{deg_freedom} < 1$, CHISQ.INV.RT returns the #NUM! error value.
- Given a value for probability, CHISQ.INV.RT seeks that value x such that $\text{CHISQ.DIST.RT}(x, \text{deg_freedom}) = \text{probability}$. Thus, precision of CHISQ.INV.RT depends on precision of CHISQ.DIST.RT. CHISQ.INV.RT uses an iterative search technique. If the search has not converged after 64 iterations, the function returns the #N/A error value.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

COMBIN

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the number of combinations for a given number of items. Use COMBIN to determine the total possible number of groups for a given number of items.

Syntax

```
COMBIN(number, number_chosen)
```

Parameters

| TERM | DEFINITION |
|---------------|--|
| number | The number of items. |
| number_chosen | The number of items in each combination. |

Return value

Returns the number of combinations for a given number of items.

Remarks

- Numeric arguments are truncated to integers.
- If either argument is nonnumeric, COMBIN returns the #VALUE! error value.
- If number < 0, number_chosen < 0, or number < number_chosen, COMBIN returns the #NUM! error value.
- A combination is any set or subset of items, regardless of their internal order. Combinations are distinct from permutations, for which the internal order is significant.

- The number of combinations is as follows, where number = n and number_chosen = k :

$${}^nP_k = \frac{P_{n,k}}{k!} = \frac{n!}{k!(n-k)!}$$

Where

$$P_{n,k} = \frac{n!}{(n-k)!}$$

- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

| FORMULA | DESCRIPTION | RESULT |
|---------------|---|--------|
| = COMBIN(8,2) | Possible two-person teams that can be formed from 8 candidates. | 28 |

COMBINA

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the number of combinations (with repetitions) for a given number of items.

Syntax

```
COMBINA(number, number_chosen)
```

Parameters

| TERM | DEFINITION |
|---------------|---|
| number | Must be greater than or equal to 0, and greater than or equal to Number_chosen. Non-integer values are truncated. |
| number_chosen | Must be greater than or equal to 0. Non-integer values are truncated. |

Return value

Returns the number of combinations (with repetitions) for a given number of items.

Remarks

- If the value of either argument is outside of its constraints, COMBINA returns the #NUM! error value.
- If either argument is a non-numeric value, COMBINA returns the #VALUE! error value.
- The following equation is used, where \$N\$ is Number and \$M\$ is Number_chosen:
$${}^{N+M-1}C_{N-1}$$
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

| FORMULA | DESCRIPTION | RESULT |
|-----------------|---|--------|
| = COMBINA(4,3) | Returns the number of combinations (with repetitions) for 4 and 3. | 20 |
| = COMBINA(10,3) | Returns the number of combinations (with repetitions) for 10 and 3. | 220 |

CONFIDENCE.NORM

10/31/2022 • 2 minutes to read • [Edit Online](#)

The confidence interval is a range of values. Your sample mean, \bar{x} , is at the center of this range and the range is $\bar{x} \pm \text{CONFIDENCE.NORM}$. For example, if \bar{x} is the sample mean of delivery times for products ordered through the mail, $\bar{x} \pm \text{CONFIDENCE.NORM}$ is a range of population means. For any population mean, μ_0 , in this range, the probability of obtaining a sample mean further from μ_0 than \bar{x} is greater than alpha; for any population mean, μ_0 , not in this range, the probability of obtaining a sample mean further from μ_0 than \bar{x} is less than alpha. In other words, assume that we use \bar{x} , standard_dev , and size to construct a two-tailed test at significance level alpha of the hypothesis that the population mean is μ_0 . Then we will not reject that hypothesis if μ_0 is in the confidence interval and will reject that hypothesis if μ_0 is not in the confidence interval. The confidence interval does not allow us to infer that there is probability $1 - \alpha$ that our next package will take a delivery time that is in the confidence interval.

Syntax

```
CONFIDENCE.NORM(alpha, standard_dev, size)
```

Parameters

| TERM | DEFINITION |
|-------------------|---|
| alpha | The significance level used to compute the confidence level. The confidence level equals $100 \times (1 - \alpha)\%$, or in other words, an alpha of 0.05 indicates a 95 percent confidence level. |
| standard_dev | The population standard deviation for the data range and is assumed to be known. |
| standard_dev,size | The sample size. |

Return value

A range of values

Remarks

- If any argument is nonnumeric, CONFIDENCE.NORM returns the #VALUE! error value.
- If $\alpha \leq 0$ or $\alpha \geq 1$, CONFIDENCE.NORM returns the #NUM! error value.
- If $\text{standard_dev} \leq 0$, CONFIDENCE.NORM returns the #NUM! error value.
- If size is not an integer, it is rounded.
- If $\text{size} < 1$, CONFIDENCE.NORM returns the #NUM! error value.
- If we assume alpha equals 0.05, we need to calculate the area under the standard normal curve that equals $(1 - \alpha)$, or 95 percent. This value is ± 1.96 . The confidence interval is therefore:

$$\bar{x} \pm 1.96 \times \left(\frac{\sigma}{\sqrt{n}} \right)$$

- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

CONFIDENCE.T

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the confidence interval for a population mean, using a Student's t distribution.

Syntax

```
CONFIDENCE.T(alpha,standard_dev,size)
```

Parameters

| TERM | DEFINITION |
|--------------|---|
| alpha | The significance level used to compute the confidence level. The confidence level equals 100*(1 - alpha)%, or in other words, an alpha of 0.05 indicates a 95 percent confidence level. |
| standard_dev | The population standard deviation for the data range and is assumed to be known. |
| size | The sample size. |

Return value

Returns the confidence interval for a population mean, using a Student's t distribution.

Remarks

- If any argument is nonnumeric, CONFIDENCE.T returns the #VALUE! error value.
- If $\alpha \leq 0$ or $\alpha \geq 1$, CONFIDENCE.T returns the #NUM! error value.
- If $\text{standard_dev} \leq 0$, CONFIDENCE.T returns the #NUM! error value.
- If size is not an integer, it is rounded.
- If size equals 1, CONFIDENCE.T returns #DIV/0! error value.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

| FORMULA | DESCRIPTION | RESULT |
|---------------------------|---|-------------|
| = CONFIDENCE.T(0.05,1,50) | Confidence interval for the mean of a population based on a sample size of 50, with a 5% significance level and a standard deviation of 1. This is based on a Student's t-distribution. | 0.284196855 |

EXPON.DIST

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the exponential distribution. Use EXPON.DIST to model the time between events, such as how long an automated bank teller takes to deliver cash. For example, you can use EXPON.DIST to determine the probability that the process takes at most 1 minute.

Syntax

EXPON.DIST(x,lambda,cumulative)

Parameters

| TERM | DEFINITION |
|------------|--|
| x | Required. The value of the function. |
| lambda | Required. The parameter value. |
| cumulative | Required. A logical value that indicates which form of the exponential function to provide. If cumulative is TRUE, EXPON.DIST returns the cumulative distribution function; if FALSE, it returns the probability density function. |

Return value

Returns the exponential distribution.

Remarks

- If x or lambda is nonnumeric, EXPON.DIST returns the #VALUE! error value.
- If x or lambda is not an integer, it is rounded.
- If $x < 0$, EXPON.DIST returns the #NUM! error value.
- If $\lambda \leq 0$, EXPON.DIST returns the #NUM! error value.

- The equation for the probability density function is:

$$f(x; \lambda) = \lambda e^{-\lambda x}$$

- The equation for the cumulative distribution function is:

$$F(x; \lambda) = 1 - e^{-\lambda x}$$

- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

GEOMEAN

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the geometric mean of the numbers in a column.

To return the geometric mean of an expression evaluated for each row in a table, use [GEOMEANX function](#).

Syntax

```
GEOMEAN(<column>)
```

Parameters

| TERM | DEFINITION |
|--------|--|
| column | The column that contains the numbers for which the geometric mean is to be computed. |

Return value

A decimal number.

Remarks

- Only the numbers in the column are counted. Blanks, logical values, and text are ignored.
- GEOMEAN(Table[Column]) is equivalent to GEOMEANX(Table, Table[Column])
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following computes the geometric mean of the Return column in the Investment table:

```
= GEOMEAN( Investment[Return] )
```

See also

[GEOMEANX function](#)

GEOMEANX

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the geometric mean of an expression evaluated for each row in a table.

To return the geometric mean of the numbers in a column, use [GEOMEAN function](#).

Syntax

```
GEOMEANX(<table>, <expression>)
```

Parameters

| TERM | DEFINITION |
|------------|---|
| table | The table containing the rows for which the expression will be evaluated. |
| expression | The expression to be evaluated for each row of the table. |

Return value

A decimal number.

Remarks

- The GEOMEANX function takes as its first argument a table, or an expression that returns a table. The second argument is a column that contains the numbers for which you want to compute the geometric mean, or an expression that evaluates to a column.
- Only the numbers in the column are counted. Blanks, logical values, and text are ignored.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following computes the geometric mean of the ReturnPct column in the Investments table:

```
= GEOMEANX( Investments, Investments[ReturnPct] + 1 )
```

See also

[GEOMEAN function](#)

MEDIAN

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the median of numbers in a column.

To return the median of an expression evaluated for each row in a table, use [MEDIANX function](#).

Syntax

```
MEDIAN(<column>)
```

Parameters

| TERM | DEFINITION |
|--------|--|
| column | The column that contains the numbers for which the median is to be computed. |

Return value

A decimal number.

Remarks

- Only the numbers in the column are counted. Blanks are ignored. Logical values, dates, and text are not supported.
- MEDIAN(Table[Column]) is equivalent to MEDIANX(Table, Table[Column]).
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following computes the median of a column named Age in a table named Customers:

```
= MEDIAN( Customers[Age] )
```

See also

[MEDIANX function](#)

MEDIANX)

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the median number of an expression evaluated for each row in a table.

To return the median of numbers in a column, use [MEDIAN function](#).

Syntax

```
MEDIANX(<table>, <expression>)
```

Parameters

| TERM | DEFINITION |
|------------|---|
| table | The table containing the rows for which the expression will be evaluated. |
| expression | The expression to be evaluated for each row of the table. |

Return value

A decimal number.

Remarks

- The MEDIANX function takes as its first argument a table, or an expression that returns a table. The second argument is a column that contains the numbers for which you want to compute the median, or an expression that evaluates to a column.
- Only the numbers in the column are counted.
- Logical values and text are ignored.
- MEDIANX does not ignore blanks; however, MEDIAN does ignore blanks
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following computes the median age of customers who live in the USA.

```
= MEDIANX( FILTER(Customers, RELATED( Geography[Country]="USA" ) ), Customers[Age] )
```

See also

[MEDIAN function](#)

NORM.DIST

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the normal distribution for the specified mean and standard deviation.

Syntax

```
NORM.DIST(X, Mean, Standard_dev, Cumulative)
```

Parameters

| TERM | DEFINITION |
|--------------|---|
| X | The value for which you want the distribution. |
| Mean | The arithmetic mean of the distribution. |
| Standard_dev | The standard deviation of the distribution. |
| Cumulative* | A logical value that determines the form of the function. If cumulative is TRUE, NORM.DIST returns the cumulative distribution function; if FALSE, it returns the probability density function. |

Return value

The normal distribution for the specified mean and standard deviation.

Remarks

This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

```
EVALUATE { NORM.DIST(42, 40, 1.5, TRUE) }
```

Returns

[VALUE]

0.908788780274132

See also

[NORM.S.DIST function](#)

[NORM.INV function](#)

[NORM.S.INV](#)

NORM.INV

10/31/2022 • 2 minutes to read • [Edit Online](#)

The inverse of the normal cumulative distribution for the specified mean and standard deviation.

Syntax

```
NORM.INV(Probability, Mean, Standard_dev)
```

Parameters

| TERM | DEFINITION |
|--------------|---|
| Probability | A probability corresponding to the normal distribution. |
| Mean | The arithmetic mean of the distribution. |
| Standard_dev | The standard deviation of the distribution. |

Return value

Returns the inverse of the normal cumulative distribution for the specified mean and standard deviation.

Remarks

This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

```
EVALUATE { NORM.INV(0.908789, 40, 1.5) }
```

Returns

[VALUE]

42.00000200956628780274132

See also

[NORM.S.INV](#)

[NORM.S.DIST function](#)

[NORM.DIST function](#)

NORM.S.DIST

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the standard normal distribution (has a mean of zero and a standard deviation of one).

Syntax

```
NORM.S.DIST(Z, Cumulative)
```

Parameters

| TERM | DEFINITION |
|------------|---|
| Z | The value for which you want the distribution. |
| Cumulative | Cumulative is a logical value that determines the form of the function. If cumulative is TRUE, NORM.S.DIST returns the cumulative distribution function; if FALSE, it returns the probability density function. |

Return value

The standard normal distribution (has a mean of zero and a standard deviation of one).

Remarks

This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

```
EVALUATE { NORM.S.DIST(1.333333, TRUE) }
```

Returns

[VALUE]

0.908788725604095

See also

[NORM.INV function](#)

[NORM.DIST function](#)

[NORM.S.INV](#)

NORM.S.INV

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the inverse of the standard normal cumulative distribution. The distribution has a mean of zero and a standard deviation of one.

Syntax

```
NORM.S.INV(Probability)
```

Parameters

| TERM | DEFINITION |
|-------------|---|
| Probability | A probability corresponding to the normal distribution. |

Return value

The inverse of the standard normal cumulative distribution. The distribution has a mean of zero and a standard deviation of one.

Remarks

This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

```
EVALUATE { NORM.S.INV(0.908789) }
```

Returns

[VALUE]

1.33333467304411

See also

[NORM.INV](#)

[NORM.S.DIST function](#)

[NORM.DIST function](#)

PERCENTILE.EXC

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the k-th percentile of values in a range, where k is in the range 0..1, exclusive.

To return the percentile number of an expression evaluated for each row in a table, use [PERCENTILEX.EXC function](#).

Syntax

```
PERCENTILE.EXC(<column>, <k>)
```

Parameters

| TERM | DEFINITION |
|--------|---|
| column | A column containing the values that define relative standing. |
| k | The percentile value in the range 0..1, exclusive. |

Return value

The k-th percentile of values in a range, where k is in the range 0..1, exclusive.

Remarks

- If column is empty, BLANK() is returned.
- If k is zero or blank, percentile rank of $1/(n+1)$ returns the smallest value. If zero, it is out of range and an error is returned.
- If k is nonnumeric or outside the range 0 to 1, an error is returned.
- If k is not a multiple of $1/(n + 1)$, PERCENTILE.EXC will interpolate to determine the value at the k-th percentile.
- PERCENTILE.EXC will interpolate when the value for the specified percentile is between two values in the array. If it cannot interpolate for the k percentile specified, an error is returned.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

See also

[PERCENTILEX.EXC](#)

PERCENTILE.INC

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the k-th percentile of values in a range, where k is in the range 0..1, inclusive.

To return the percentile number of an expression evaluated for each row in a table, use [PERCENTILEX.INC](#).

Syntax

```
PERCENTILE.INC(<column>, <k>)
```

Parameters

| TERM | DEFINITION |
|--------|---|
| column | A column containing the values that define relative standing. |
| k | The percentile value in the range 0..1, inclusive. |

Return value

The k-th percentile of values in a range, where k is in the range 0..1, inclusive.

Remarks

- If column is empty, BLANK() is returned.
- If k is zero or blank, percentile rank of $1/(n+1)$ returns the smallest value. If zero, it is out of range and an error is returned.
- If k is nonnumeric or outside the range 0 to 1, an error is returned.
- If k is not a multiple of $1/(n + 1)$, PERCENTILE.INC will interpolate to determine the value at the k-th percentile.
- PERCENTILE.INC will interpolate when the value for the specified percentile is between two values in the array. If it cannot interpolate for the k percentile specified, an error is returned.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

See also

[PERCENTILEX.INC](#)

PERCENTILEX.EXC

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the percentile number of an expression evaluated for each row in a table.

To return the percentile of numbers in a column, use [PERCENTILE.EXC function](#).

Syntax

```
PERCENTILEX.EXC(<table>, <expression>, k)
```

Parameters

| TERM | DEFINITION |
|------------|---|
| table | The table containing the rows for which the expression will be evaluated. |
| expression | The expression to be evaluated for each row of the table. |
| k | The desired percentile value in the range 0 to 1 exclusive. |

Return value

The percentile number of an expression evaluated for each row in a table.

Remarks

- If k is zero or blank, percentile rank of $1/(n+1)$ returns the smallest value. If zero, it is out of range and an error is returned.
- If k is nonnumeric or outside the range 0 to 1, an error is returned.
- If k is not a multiple of $1/(n + 1)$, PERCENTILEX.EXC will interpolate to determine the value at the k-th percentile.
- PERCENTILEX.EXC will interpolate when the value for the specified percentile is between two values in the array. If it cannot interpolate for the k percentile specified, an error is returned.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

See also

[PERCENTILE.EXC](#)

PERCENTILEX.INC

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the percentile number of an expression evaluated for each row in a table.

To return the percentile of numbers in a column, use [PERCENTILE.INC](#).

Syntax

```
PERCENTILEX.INC(<table>, <expression>;, k)
```

Parameters

| TERM | DEFINITION |
|------------|---|
| table | The table containing the rows for which the expression will be evaluated. |
| expression | The expression to be evaluated for each row of the table. |
| k | The desired percentile value in the range 0 to 1 inclusive. |

Return value

The percentile number of an expression evaluated for each row in a table.

Remarks

- If k is zero or blank, percentile rank of $1/(n - 1)$ returns the smallest value. If zero, it is out of range and an error is returned.
- If k is nonnumeric or outside the range 0 to 1, an error is returned.
- If k is not a multiple of $1/(n - 1)$, PERCENTILEX.EXC will interpolate to determine the value at the k-th percentile.
- PERCENTILEX.INC will interpolate when the value for the specified percentile is between two values in the array. If it cannot interpolate for the k percentile specified, an error is returned.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

See also

[PERCENTILE.INC](#)

PERMUT

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the number of permutations for a given number of objects that can be selected from number objects. A permutation is any set or subset of objects or events where internal order is significant. Permutations are different from combinations, for which the internal order is not significant. Use this function for lottery-style probability calculations.

Syntax

```
PERMUT(number, number_chosen)
```

Parameters

| TERM | DEFINITION |
|---------------|--|
| number | Required. An integer that describes the number of objects. |
| number_chosen | Required. An integer that describes the number of objects in each permutation. |

Return value

Returns the number of permutations for a given number of objects that can be selected from number objects

Remarks

- Both arguments are truncated to integers.
- If number or number_chosen is nonnumeric, PERMUT returns the #VALUE! error value.
- If number ≤ 0 or if number_chosen < 0 , PERMUT returns the #NUM! error value.
- If number $<$ number_chosen, PERMUT returns the #NUM! error value.
- The equation for the number of permutations is:

$${}_nP_k = \frac{n!}{(n-k)!}$$

- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

In the following formula, permutations possible for a group of 3 objects where 2 are chosen:

```
= PERMUT(3,2)
```

Result,

POISSON.DIST

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the Poisson distribution. A common application of the Poisson distribution is predicting the number of events over a specific time, such as the number of cars arriving at a toll plaza in 1 minute.

Syntax

```
POISSON.DIST(x,mean,cumulative)
```

Parameters

| TERM | DEFINITION |
|------------|--|
| x | Required. The number of events. |
| mean | Required. The expected numeric value. |
| cumulative | Required. A logical value that determines the form of the probability distribution returned. If cumulative is TRUE, POISSON.DIST returns the cumulative Poisson probability that the number of random events occurring will be between zero and x inclusive; if FALSE, it returns the Poisson probability mass function that the number of events occurring will be exactly x. |

Return value

Returns the Poisson distribution.

Remarks

- If x is not an integer, it is rounded.
- If x or mean is nonnumeric, POISSON.DIST returns the #VALUE! error value.
- If $x < 0$, POISSON.DIST returns the #NUM! error value.
- If $\text{mean} < 0$, POISSON.DIST returns the #NUM! error value.
- POISSON.DIST is calculated as follows.

- For cumulative = FALSE:

$$\text{POISSON} = \frac{e^{-\lambda} \lambda^x}{x!}$$

- For cumulative = TRUE:

$$\text{CUMPOISSON} = \sum_{k=0}^x \frac{e^{-\lambda} \lambda^k}{k!}$$

- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

RANK.EQ

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the ranking of a number in a list of numbers.

Syntax

```
RANK.EQ(<value>, <columnName>[, <order>])
```

Parameters

| TERM | DEFINITION |
|------------|---|
| value | Any DAX expression that returns a single scalar value whose rank is to be found. The expression is to be evaluated exactly once, before the function is evaluated, and its value passed to the argument list. |
| columnName | The name of an existing column against which ranks will be determined. It cannot be an expression or a column created using these functions: ADDCOLUMNS, ROW or SUMMARIZE. |
| order | (Optional) A value that specifies how to rank <i>number</i> , low to high or high to low: |

order values

| VALUE | ALTERNATE VALUE | DESCRIPTION |
|----------|-----------------|--|
| 0 (zero) | FALSE | Ranks in descending order of <i>columnName</i> . If <i>value</i> is equal to the highest number in <i>columnName</i> then RANK.EQ is 1. |
| 1 | TRUE | Ranks in ascending order of <i>columnName</i> . If <i>value</i> is equal to the lowest number in <i>columnName</i> then RANK.EQ is 1. |

Return value

A number indicating the rank of *value* among the numbers in *columnName*.

Remarks

- columnName* cannot refer to any column created using these functions: ADDCOLUMNS, ROW or SUMMARIZE.I
- If *value* is not in *columnName* or value is a blank, then **RANK.EQ** returns a blank value.
- Duplicate values of *value* receive the same rank value; the next rank value assigned will be the rank value

plus the number of duplicate values. For example if five (5) values are tied with a rank of 11 then the next value will receive a rank of 16 (11 + 5).

- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example 1

The following example creates a calculated column that ranks the values in SalesAmount_USD, from the *InternetSales_USD* table, against all numbers in the same column.

```
= RANK.EQ(InternetSales_USD[SalesAmount_USD], InternetSales_USD[SalesAmount_USD])
```

Example 2

The following example ranks a subset of values against a given sample. Assume that you have a table of local students with their performance in a specific national test and, also, you have the entire set of scores in that national test. The following calculated column will give you the national ranking for each of the local students.

```
= RANK.EQ(Students[Test_Score], NationalScores[Test_Score])
```

RANKX

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the ranking of a number in a list of numbers for each row in the *table* argument.

Syntax

```
RANKX(<table>, <expression>[, <value>[, <order>[, <ties>]]])
```

Parameters

table

Any DAX expression that returns a table of data over which the expression is evaluated.

expression

Any DAX expression that returns a single scalar value. The expression is evaluated for each row of *table*, to generate all possible values for ranking. See the remarks section to understand the function behavior when *expression* evaluates to BLANK.

value

(Optional) Any DAX expression that returns a single scalar value whose rank is to be found. See the remarks section to understand the function's behavior when *value* is not found in the expression.

When the *value* parameter is omitted, the value of expression at the current row is used instead.

order

(Optional) A value that specifies how to rank *value*, low to high or high to low:

| VALUE | ALTERNATE VALUE | DESCRIPTION |
|----------|-----------------|---|
| 0 (zero) | FALSE | <p>Ranks in descending order of values of expression. If value is equal to the highest number in expression then RANKX returns 1.</p> <p>This is the default value when order parameter is omitted.</p> |
| 1 | TRUE | <p>Ranks in ascending order of expression. If value is equal to the lowest number in expression then RANKX returns 1.</p> |

ties

(Optional) An enumeration that defines how to determine ranking when there are ties.

| ENUMERATION | DESCRIPTION |
|-------------|-------------|
|-------------|-------------|

| ENUMERATION | DESCRIPTION |
|-------------|--|
| Skip | <p>The next rank value, after a tie, is the rank value of the tie plus the count of tied values. For example if five (5) values are tied with a rank of 11 then the next value will receive a rank of 16 (11 + 5).</p> <p>This is the default value when <i>ties</i> parameter is omitted.</p> |
| Dense | <p>The next rank value, after a tie, is the next rank value. For example if five (5) values are tied with a rank of 11 then the next value will receive a rank of 12.</p> |

Return value

The rank number of *value* among all possible values of *expression* evaluated for all rows of *table* numbers.

Remarks

- If *expression* or *value* evaluates to BLANK it is treated as a 0 (zero) for all expressions that result in a number, or as an empty text for all text expressions.
- If *value* is not among all possible values of *expression* then RANKX temporarily adds *value* to the values from *expression* and re-evaluates RANKX to determine the proper rank of *value*.
- Optional arguments might be skipped by placing an empty comma (,) in the argument list, i.e.
RANKX(Inventory, [InventoryCost],,, "Dense")
- Equality related comparisons (=, <, >, >= and <=) between values with the *Decimal Number* data type can potentially return unexpected results when using RANKX function. Incorrect results can occur because values with *Decimal Number* data type are stored as IEEE Standard 754 floating point numbers and have inherent limitations in their precision. To avoid unexpected results, change the data type to *Fixed Decimal Number* or do a forced rounding using [ROUND](#).
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following calculated column in the Products table calculates the sales ranking for each product in the Internet channel.

```
= RANKX(ALL(Products), SUMX(RELATEDTABLE(InternetSales), [SalesAmount]))
```

SAMPLE

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns a sample of N rows from the specified table.

Syntax

```
SAMPLE(<n_value>, <table>, <orderBy_expression>, [<order>[, <orderBy_expression>, [<order>]]...])
```

Parameters

| TERM | DEFINITION |
|--------------------|--|
| n_value | The number of rows to return. It is any DAX expression that returns a single scalar value, where the expression is to be evaluated multiple times (for each row/context). If a non-integer value (or expression) is entered, the result is cast as an integer. |
| table | Any DAX expression that returns a table of data from where to extract the 'n' sample rows. |
| orderBy_expression | (Optional) Any scalar DAX expression where the result value is evaluated for each row of <i>table</i> . |
| order | (Optional) A value that specifies how to sort <i>orderBy_expression</i> values, ascending or descending: 0 (zero), sorts in descending order of values of <i>order_by</i> . 1, ranks in ascending order of <i>order_by</i> . |

Return value

A table consisting of a sample of N rows of *table* or an empty table if *n_value* is 0 (zero) or less. If OrderBy arguments are provided, the sample will be stable and deterministic, returning the first row, the last row, and evenly distributed rows between them. If no ordering is specified, the sample will be random, not stable, and not deterministic.

Remarks

- If *n_value* is 0 (zero) or less then SAMPLE returns an empty table.
- In order to avoid duplicate values in the sample, the table provided as the second argument should be grouped by the column used for sorting.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

STDEV.S

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the standard deviation of a sample population.

Syntax

```
STDEV.S(<ColumnName>)
```

Parameters

| TERM | DEFINITION |
|------------|--|
| columnName | The name of an existing column using standard DAX syntax, usually fully qualified. It cannot be an expression. |

Return value

A number that represents the standard deviation of a sample population.

Exceptions

Remarks

- STDEV.S assumes that the column refers to a sample of the population. If your data represents the entire population, then compute the standard deviation by using STDEV.P.

- STDEV.S uses the following formula:

$$\sqrt{\sum (x - \tilde{x})^2 / (n-1)}$$

where \tilde{x} is the average value of x for the sample population and n is the population size.

- Blank rows are filtered out from *columnName* and not considered in the calculations.
- An error is returned if *columnName* contains less than 2 non-blank rows.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following example shows the formula for a measure that calculates the standard deviation of the column, SalesAmount_USD, when the table InternetSales_USD is the sample population.

```
= STDEV.S(InternetSales_USD[SalesAmount_USD])
```

STDEV.P

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the standard deviation of the entire population.

Syntax

```
STDEV.P(<ColumnName>)
```

Parameters

| TERM | DEFINITION |
|------------|--|
| columnName | The name of an existing column using standard DAX syntax, usually fully qualified. It cannot be an expression. |

Return value

A number representing the standard deviation of the entire population.

Remarks

- STDEV.P assumes that the column refers to the entire population. If your data represents a sample of the population, then compute the standard deviation by using STDEV.S.
- STDEV.P uses the following formula:

$$\sqrt{\sum (x - \bar{x})^2 / n}$$

where \bar{x} is the average value of x for the entire population and n is the population size.

- Blank rows are filtered out from *columnName* and not considered in the calculations.
- An error is returned if *columnName* contains less than 2 non-blank rows
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following example shows the formula for a measure that calculates the standard deviation of the column, SalesAmount_USD, when the table InternetSales_USD is the entire population.

```
= STDEV.P(InternetSales_USD[SalesAmount_USD])
```


STDEVX.S

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the standard deviation of a sample population.

Syntax

```
STDEVX.S(<table>, <expression>)
```

Parameters

| TERM | DEFINITION |
|------------|---|
| table | Any DAX expression that returns a single scalar value, where the expression is to be evaluated multiple times (for each row/context). |
| expression | Any DAX expression that returns a single scalar value, where the expression is to be evaluated multiple times (for each row/context). |

Return value

A number with the standard deviation of a sample population.

Exceptions

Remarks

- STDEVX.S evaluates *expression* for each row of *table* and returns the standard deviation of *expression* assuming that *table* refers to a sample of the population. If *table* represents the entire population, then compute the standard deviation by using STDEVX.P.

- STDEVX.S uses the following formula:

$$\sqrt{\sum (x - \tilde{x})^2 / (n-1)}$$

where \tilde{x} is the average value of x for the entire population and n is the population size.

- Blank rows are filtered out from *columnName* and not considered in the calculations.
- An error is returned if *columnName* contains less than 2 non-blank rows.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following example shows the formula for a calculated column that estimates the standard deviation of the unit price per product for a sample population, when the formula is used in the Product table.

```
= STDEVX.S(RELATEDTABLE(InternetSales_USD), InternetSales_USD[UnitPrice_USD] -  
(InternetSales_USD[DiscountAmount_USD]/InternetSales_USD[OrderQuantity]))
```

STDEVX.P

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the standard deviation of the entire population.

Syntax

```
STDEVX.P(<table>, <expression>)
```

Parameters

| TERM | DEFINITION |
|------------|---|
| table | Any DAX expression that returns a single scalar value, where the expression is to be evaluated multiple times (for each row/context). |
| expression | Any DAX expression that returns a single scalar value, where the expression is to be evaluated multiple times (for each row/context). |

Return value

A number that represents the standard deviation of the entire population.

Remarks

- STDEVX.P evaluates *expression* for each row of *table* and returns the standard deviation of expression assuming that *table* refers to the entire population. If the data in *table* represents a sample of the population, you should compute the standard deviation by using STDEVX.S instead.

- STDEVX.P uses the following formula:

$$\sqrt{\sum (x - \tilde{x})^2 / n}$$

where \tilde{x} is the average value of *x* for the entire population and *n* is the population size.

- Blank rows are filtered out from *columnName* and not considered in the calculations.
- An error is returned if *columnName* contains less than 2 non-blank rows
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following example shows the formula for a calculated column that calculates the standard deviation of the unit price per product, when the formula is used in the *Product* table.

```
= STDEVX.P(RELATERTABLE(InternetSales_USD), InternetSales_USD[UnitPrice_USD] -  
(InternetSales_USD[DiscountAmount_USD]/InternetSales_USD[OrderQuantity]))
```

T.DIST

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the Student's left-tailed t-distribution.

Syntax

```
T.DIST(X,Deg_freedom,Cumulative)
```

Parameters

| TERM | DEFINITION |
|-------------|--|
| X | The numeric value at which to evaluate the distribution. |
| Deg_freedom | An integer indicating the number of degrees of freedom. |
| Cumulative | A logical value that determines the form of the function. If cumulative is TRUE, T.DIST returns the cumulative distribution function; if FALSE, it returns the probability density function. |

Return value

The Student's left-tailed t-distribution.

Remarks

This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

```
EVALUATE { T.DIST(60, 1, TRUE) }
```

Returns,

[VALUE]

0.994695326367377

See also

[T.DIST.2T](#)

[T.DIST.RT](#)

[T.INV](#)

[T.INV.2t](#)

T.DIST.2T

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the two-tailed Student's t-distribution.

Syntax

```
T.DIST.2T(X,Deg_freedom)
```

Parameters

| TERM | DEFINITION |
|-------------|--|
| X | The numeric value at which to evaluate the distribution. |
| Deg_freedom | An integer indicating the number of degrees of freedom. |

Return value

The two-tailed Student's t-distribution.

Remarks

This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

```
EVALUATE { T.DIST.2T(1.959999998, 60) }
```

Returns

[VALUE]

0.054644929975921

See also

[T.DIST](#)

[T.DIST.RT](#)

[T.INV](#)

[T.INV.2t](#)

T.DIST.RT

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the right-tailed Student's t-distribution.

Syntax

```
T.DIST.RT(X,Deg_freedom)
```

Parameters

| TERM | DEFINITION |
|-------------|--|
| X | The numeric value at which to evaluate the distribution. |
| Deg_freedom | An integer indicating the number of degrees of freedom. |

Return value

The right-tailed Student's t-distribution.

Remarks

This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

```
EVALUATE { T.DIST.RT(1.959999998, 60) }
```

Returns

[VALUE]

0.0273224649879605

See also

[T.DIST](#)

[T.DIST.2T](#)

[T.INV](#)

[T.INV.2t](#)

T.INV

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the left-tailed inverse of the Student's t-distribution.

Syntax

```
T.INV(Probability,Deg_freedom)
```

Parameters

| TERM | DEFINITION |
|-------------|---|
| Probability | The probability associated with the Student's t-distribution. |
| Deg_freedom | The number of degrees of freedom with which to characterize the distribution. |

Return value

The left-tailed inverse of the Student's t-distribution.

Remarks

This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

```
EVALUATE { T.INV(0.75, 2) }
```

Returns

[VALUE]

0.816496580927726

See also

[T.INV.2T](#)

[T.DIST](#)

[T.DIST.2T](#)

[T.DIST.RT](#)

T.INV.2T

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the two-tailed inverse of the Student's t-distribution.

Syntax

```
T.INV.2T(Probability,Deg_freedom)
```

Parameters

| TERM | DEFINITION |
|-------------|---|
| Probability | The probability associated with the Student's t-distribution. |
| Deg_freedom | The number of degrees of freedom with which to characterize the distribution. |

Return value

The two-tailed inverse of the Student's t-distribution.

Remarks

This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

```
EVALUATE { T.INV.2T(0.546449, 60) }
```

Returns

[VALUE]

0.606533075825759

See also

[T.INV](#)

[T.DIST](#)

[T.DIST.2T](#)

[T.DIST.RT](#)

VAR.S

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the variance of a sample population.

Syntax

```
VAR.S(<columnName>)
```

Parameters

| TERM | DEFINITION |
|------------|--|
| columnName | The name of an existing column using standard DAX syntax, usually fully qualified. It cannot be an expression. |

Return value

A number with the variance of a sample population.

Remarks

- VAR.S assumes that the column refers to a sample of the population. If your data represents the entire population, then compute the variance by using VAR.P.
- VAR.S uses the following formula:

$$\sum (x - \tilde{x})^2 / (n - 1)$$

where \tilde{x} is the average value of x for the sample population

and n is the population size

- Blank rows are filtered out from *columnName* and not considered in the calculations.
- An error is returned if *columnName* contains less than 2 non-blank rows.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following example shows the formula for a measure that calculates the variance of the SalesAmount_USD column from the InternetSales_USD for a sample population.

```
= VAR.S(InternetSales_USD[SalesAmount_USD])
```

VAR.P

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the variance of the entire population.

Syntax

```
VAR.P(<columnName>)
```

Parameters

| TERM | DEFINITION |
|------------|--|
| columnName | The name of an existing column using standard DAX syntax, usually fully qualified. It cannot be an expression. |

Return value

A number with the variance of the entire population.

Remarks

- VAR.P assumes that the column refers the entire population. If your data represents a sample of the population, then compute the variance by using VAR.S.

- VAR.P uses the following formula:

$$\sum (x - \tilde{x})^2 / n$$

where \tilde{x} is the average value of x for the entire population

and n is the population size

- Blank rows are filtered out from *columnName* and not considered in the calculations.
- An error is returned if *columnName* contains less than 2 non-blank rows
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following example shows the formula for a measure that estimates the variance of the SalesAmount_USD column from the InternetSales_USD table, for the entire population.

```
= VAR.P(InternetSales_USD[SalesAmount_USD])
```

VARX.S

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the variance of a sample population.

Syntax

```
VARX.S(<table>, <expression>)
```

Parameters

| TERM | DEFINITION |
|------------|---|
| table | Any DAX expression that returns a table of data. |
| expression | Any DAX expression that returns a single scalar value, where the expression is to be evaluated multiple times (for each row/context). |

Return value

A number that represents the variance of a sample population.

Remarks

- VARX.S evaluates *expression* for each row of *table* and returns the variance of *expression*; on the assumption that *table* refers to a sample of the population. If *table* represents the entire population, then you should compute the variance by using VARX.P.

- VAR.S uses the following formula:

$$\sum (x - \tilde{x})^2 / (n-1)$$

where \tilde{x} is the average value of x for the sample population

and n is the population size

- Blank rows are filtered out from *columnName* and not considered in the calculations.
- An error is returned if *columnName* contains less than 2 non-blank rows.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following example shows the formula for a calculated column that estimates the variance of the unit price per product for a sample population, when the formula is used in the Product table.

```
= VARX.S(InternetSales_USD, InternetSales_USD[UnitPrice_USD] -  
(InternetSales_USD[DiscountAmount_USD]/InternetSales_USD[OrderQuantity]))
```

VARX.P

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the variance of the entire population.

Syntax

```
VARX.P(<table>, <expression>)
```

Parameters

| TERM | DEFINITION |
|------------|---|
| table | Any DAX expression that returns a table of data. |
| expression | Any DAX expression that returns a single scalar value, where the expression is to be evaluated multiple times (for each row/context). |

Return value

A number with the variance of the entire population.

Remarks

- VARX.P evaluates <expression> for each row of <table> and returns the variance of <expression> assuming that <table> refers to the entire population.. If <table> represents a sample of the population, then compute the variance by using VARX.S.

- VARX.P uses the following formula:

$$\sum (x - \tilde{x})^2 / n$$

where \tilde{x} is the average value of x for the entire population

and n is the population size

- Blank rows are filtered out from *columnName* and not considered in the calculations.
- An error is returned if *columnName* contains less than 2 non-blank rows
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following example shows the formula for a calculated column that calculates the variance of the unit price per product, when the formula is used in the Product table

```
= VARX.P(InternetSales_USD, InternetSales_USD[UnitPrice_USD] -  
(InternetSales_USD[DiscountAmount_USD]/InternetSales_USD[OrderQuantity]))
```

Table manipulation functions

10/31/2022 • 2 minutes to read • [Edit Online](#)

These functions return a table or manipulate existing tables.

In this category

| FUNCTION | DESCRIPTION |
|---------------------------------|--|
| ADDCOLUMNS | Adds calculated columns to the given table or table expression. |
| ADDMISSINGITEMS | Adds combinations of items from multiple columns to a table if they do not already exist. |
| CROSSJOIN | Returns a table that contains the Cartesian product of all rows from all tables in the arguments. |
| CURRENTGROUP | Returns a set of rows from the table argument of a <code>GROUPBY</code> expression. |
| DATATABLE | Provides a mechanism for declaring an inline set of data values. |
| DETAILROWS | Evaluates a Detail Rows Expression defined for a measure and returns the data. |
| DISTINCT column | Returns a one-column table that contains the distinct values from the specified column. |
| DISTINCT table | Returns a table by removing duplicate rows from another table or expression. |
| EXCEPT | Returns the rows of one table which do not appear in another table. |
| FILTERS | Returns a table of values directly applied as filters to <i>columnName</i> . |
| GENERATE | Returns a table with the Cartesian product between each row in <i>table1</i> and the table that results from evaluating <i>table2</i> in the context of the current row from <i>table1</i> . |
| GENERATEALL | Returns a table with the Cartesian product between each row in <i>table1</i> and the table that results from evaluating <i>table2</i> in the context of the current row from <i>table1</i> . |
| GENERATESERIES | Returns a single column table containing the values of an arithmetic series. |

| FUNCTION | DESCRIPTION |
|----------------------|---|
| GROUPBY | Similar to the SUMMARIZE function, GROUPBY does not do an implicit CALCULATE for any extension columns that it adds. |
| IGNORE | Modifies SUMMARIZECOLUMNS by omitting specific expressions from the BLANK/NULL evaluation. |
| INTERSECT | Returns the row intersection of two tables, retaining duplicates. |
| NATURALINNERJOIN | Performs an inner join of a table with another table. |
| NATURALLEFTOUTERJOIN | Performs a join of the LeftTable with the RightTable. |
| ROLLUP | Modifies the behavior of SUMMARIZE by adding rollup rows to the result on columns defined by the groupBy_columnName parameter. |
| ROLLUPADDISSUBTOTAL | Modifies the behavior of SUMMARIZECOLUMNS by adding rollup/subtotal rows to the result based on the groupBy_columnName columns. |
| ROLLUPISSUBTOTAL | Pairs rollup groups with the column added by ROLLUPADDISSUBTOTAL within an ADDMISSINGITEMS expression. |
| ROLLUPGROUP | Modifies the behavior of SUMMARIZE and SUMMARIZECOLUMNS by adding rollup rows to the result on columns defined by the the groupBy_columnName parameter. |
| ROW | Returns a table with a single row containing values that result from the expressions given to each column. |
| SELECTCOLUMNS | Adds calculated columns to the given table or table expression. |
| SUBSTITUTEWITHINDEX | Returns a table which represents a left semijoin of the two tables supplied as arguments. |
| SUMMARIZE | Returns a summary table for the requested totals over a set of groups. |
| SUMMARIZECOLUMNS | Returns a summary table over a set of groups. |
| Table Constructor | Returns a table of one or more columns. |
| TOPN | Returns the top N rows of the specified table. |
| TREATAS | Applies the result of a table expression as filters to columns from an unrelated table. |
| UNION | Creates a union (join) table from a pair of tables. |

| FUNCTION | DESCRIPTION |
|----------|--|
| VALUES | Returns a one-column table that contains the distinct values from the specified table or column. |

ADDCOLUMNS

10/31/2022 • 2 minutes to read • [Edit Online](#)

Adds calculated columns to the given table or table expression.

Syntax

```
ADDCOLUMNS(<table>, <name>, <expression>[, <name>, <expression>]...)
```

Parameters

| TERM | DEFINITION |
|------------|---|
| table | Any DAX expression that returns a table of data. |
| name | The name given to the column, enclosed in double quotes. |
| expression | Any DAX expression that returns a scalar expression, evaluated for each row of <i>table</i> . |

Return value

A table with all its original columns and the added ones.

Remarks

This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following example returns an extended version of the Product Category table that includes total sales values from the reseller channel and the internet sales.

```
ADDCOLUMNS(ProductCategory,
    , "Internet Sales", SUMX(RELATEDTABLE(InternetSales_USD), InternetSales_USD[SalesAmount_USD])
    , "Reseller Sales", SUMX(RELATEDTABLE(ResellerSales_USD),
    ResellerSales_USD[SalesAmount_USD]))
```

The following table shows a preview of the data as it would be received by any function expecting to receive a table:

| PRODUCTCATEGORY[PRODUCTCATEGORY NAME] | PRODUCTCATEGORY[PRODUCTCATEGORY ALTERNATEKEY] | PRODUCTCATEGORY[PRODUCTCATEGORY KEY] | [INTERNET SALES] | [RESELLER SALES] |
|--|--|---|------------------|------------------|
| Bikes | 1 | 1 | 25107749.77 | 63084675.04 |
| Components | 2 | 2 | | 11205837.96 |

| PRODUCTCATEGORY[PRODUCTCATEGORY NAME] | PRODUCTCATEGORY[PRODUCTCATEGORY ALTERNATEKEY] | PRODUCTCATEGORY[PRODUCTCATEGORY KEY] | [INTERNET SALES] | [RESELLER SALES] |
|--|--|---|------------------|------------------|
| Clothing | 3 | 3 | 306157.5829 | 1669943.267 |
| Accessories | 4 | 4 | 640920.1338 | 534301.9888 |

ADDMISSINGITEMS

10/31/2022 • 2 minutes to read • [Edit Online](#)

Adds rows with empty values to a table returned by [SUMMARIZECOLUMNS](#).

Syntax

```
ADDMISSINGITEMS ( [ <showAll_columnName> [ , <showAll_columnName> [ , ... ] ] ], <table> [ , <groupBy_columnName> [ , [<filterTable>] [ , <groupBy_columnName> [ , [<filterTable>] [ , ... ] ] ] ] ] )
```

Parameters

| TERM | DEFINITION |
|--------------------|---|
| showAll_columnName | (Optional) A column for which to return items with no data for the measures used. If not specified, all columns are returned. |
| table | A SUMMARIZECOLUMNS table. |
| groupBy_columnName | (Optional) A column to group by in the supplied table argument. |
| filterTable | (Optional) A table expression that defines which rows are returned. |

Return value

A table with one or more columns.

Remarks

This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

With SUMMARIZECOLUMNS

A table returned by [SUMMARIZECOLUMNS](#) will include only rows with values. By wrapping a [SUMMARIZECOLUMNS](#) expression within an ADDMISSINGITEMS expression, rows containing no values are also returned.

Example

Without ADDMISSINGITEMS, the following query:

```
SUMMARIZECOLUMNS(  
    'Sales'[CustomerId],  
    "Total Qty", SUM ( Sales[TotalQty] )  
)
```

Returns,

| CUSTOMERID | TOTALQTY |
|------------|----------|
| A | 5 |
| B | 3 |
| C | 3 |
| E | 2 |

With ADDMISSINGITEMS, the following query:

```
EVALUATE
ADDMISSINGITEMS (
  'Sales'[CustomerId],
  SUMMARIZECOLUMNS(
    'Sales'[CustomerId],
    "Total Qty", SUM ( Sales[TotalQty] )
  ),
  'Sales'[CustomerId]
)
```

Returns,

| CUSTOMERID | TOTALQTY |
|------------|----------|
| A | 5 |
| B | 3 |
| C | 3 |
| D | |
| E | 2 |
| F | |

CROSSJOIN

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns a table that contains the Cartesian product of all rows from all tables in the arguments. The columns in the new table are all the columns in all the argument tables.

Syntax

```
CROSSJOIN(<table>, <table>[, <table>]...)
```

Parameters

| TERM | DEFINITION |
|-------|---|
| table | Any DAX expression that returns a table of data |

Return value

A table that contains the Cartesian product of all rows from all tables in the arguments.

Remarks

- Column names from *table* arguments must all be different in all tables or an error is returned.
- The total number of rows returned by CROSSJOIN() is equal to the product of the number of rows from all tables in the arguments; also, the total number of columns in the result table is the sum of the number of columns in all tables. For example, if **TableA** has **rA** rows and **cA** columns, and **TableB** has **rB** rows and **cB** columns, and **TableC** has **rC** rows and **cC** columns; then, the resulting table has **rA × rB × rC** rows and **cA + cB + cC** columns.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following example shows the results of applying CROSSJOIN() to two tables: **Colors** and **Stationery**.

The table **Colors** contains colors and patterns:

| COLOR | PATTERN |
|-------|-------------------|
| Red | Horizontal Stripe |
| Green | Vertical Stripe |
| Blue | Crosshatch |

The table **Stationery** contains fonts and presentation:

| FONT | PRESENTATION |
|------------|--------------|
| serif | embossed |
| sans-serif | engraved |

The expression to generate the cross join is presented below:

```
CROSSJOIN( Colors, Stationery)
```

When the above expression is used wherever a table expression is expected, the results of the expression would be as follows:

| COLOR | PATTERN | FONT | PRESENTATION |
|-------|-------------------|------------|--------------|
| Red | Horizontal Stripe | serif | embossed |
| Green | Vertical Stripe | serif | embossed |
| Blue | Crosshatch | serif | embossed |
| Red | Horizontal Stripe | sans-serif | engraved |
| Green | Vertical Stripe | sans-serif | engraved |
| Blue | Crosshatch | sans-serif | engraved |

CURRENTGROUP

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns a set of rows from the table argument of a [GROUPBY](#) expression that belong to the current row of the [GROUPBY](#) result.

Syntax

```
CURRENTGROUP ( )
```

Parameters

None

Return value

The rows in the table argument of the [GROUPBY](#) function corresponding to one group of values of the groupBy_columnName arguments.

Remarks

- This function can only be used within a [GROUPBY](#) expression.
- This function takes no arguments and is only supported as the first argument to one of the following aggregation functions: [AVERAGEX](#), [COUNTAX](#), [COUNTX](#), [GEOMEANX](#), [MAXX](#), [MINX](#), [PRODUCTX](#), [STDEVX.S](#), [STDEVX.P](#), [SUMX](#), [VARX.S](#), [VARX.P](#).

Example

See [GROUPBY](#).

DATATABLE

10/31/2022 • 2 minutes to read • [Edit Online](#)

Provides a mechanism for declaring an inline set of data values.

Syntax

```
DATATABLE (ColumnName1, DataType1, ColumnName2, DataType2..., {{Value1, Value2...}, {ValueN, ValueN+1...}}...)
```

Parameters

| TERM | DEFINITION |
|------------|--|
| ColumnName | Any DAX expression that returns a table. |
| DataType | An enumeration that includes: INTEGER, DOUBLE, STRING, BOOLEAN, CURRENCY, DATETIME |
| Value | <p>A single argument using Excel syntax for a one dimensional array constant, nested to provide an array of arrays. This argument represents the set of data values that will be in the table</p> <p>For example, { {values in row1}, {values in row2}, {values in row3}, etc. } Where {values in row1} is a comma delimited set of constant expressions, namely a combination of constants, combined with a handful of basic functions including DATE, TIME, and BLANK, as well as a plus operator between DATE and TIME and a unary minus operator so that negative values can be expressed.</p> <p>The following are all valid values: 3, -5, BLANK(), "2009-04-15 02:45:21". Values may not refer to anything outside the immediate expression, and cannot refer to columns, tables, relationships, or anything else.</p> <p>A missing value will be treated identically to BLANK(). For example, the following are the same: {1,2,BLANK(),4} {1,2,,4}</p> |

Return value

A table declaring an inline set of values.

Remarks

- Unlike DATATABLE, [Table Constructor](#) allows any scalar expressions as input values.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

```
= DataTable("Name", STRING,  
            "Region", STRING  
            ,{  
                {" User1", "East"},  
                {" User2", "East"},  
                {" User3", "West"},  
                {" User4", "West"},  
                {" User4", "East"}  
            }  
            )
```


DETAILROWS

10/31/2022 • 2 minutes to read • [Edit Online](#)

Evaluates a Detail Rows Expression defined for a measure and returns the data.

Syntax

```
DETAILROWS([Measure])
```

Parameters

| TERM | DEFINITION |
|---------|--------------------|
| Measure | Name of a measure. |

Return value

A table with the data returned by the Detail Rows Expression. If no Detail Rows Expression is defined, the data for the table containing the measure is returned.

DISTINCT (column)

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns a one-column table that contains the distinct values from the specified column. In other words, duplicate values are removed and only unique values are returned.

NOTE

This function cannot be used to Return values into a cell or column on a worksheet; rather, you nest the DISTINCT function within a formula, to get a list of distinct values that can be passed to another function and then counted, summed, or used for other operations.

Syntax

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

Parameters

| TERM | DEFINITION |
|--------|--|
| column | The column from which unique values are to be returned. Or, an expression that returns a column. |

Return value

A column of unique values.

Remarks

- The results of DISTINCT are affected by the current filter context. For example, if you use the formula in the following example to create a measure, the results would change whenever the table was filtered to show only a particular region or a time period.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Related functions

There is another version of the DISTINCT function, [DISTINCT \(table\)](#), that returns a table by removing duplicate rows from another table or expression..

The VALUES function is similar to DISTINCT; it can also be used to return a list of unique values, and generally will return exactly the same results as DISTINCT. However, in some context VALUES will return one additional special value. For more information, see [VALUES function](#).

Example

The following formula counts the number of unique customers who have generated orders over the internet channel. The table that follows illustrates the possible results when the formula is added to a report.

```
= COUNTROWS(DISTINCT(InternetSales_USD[CustomerKey]))
```

You cannot paste the list of values that **DISTINCT** returns directly into a column. Instead, you pass the results of the **DISTINCT** function to another function that counts, filters, or aggregates values by using the list. To make the example as simple as possible, here the table of distinct values has been passed to the **COUNTROWS** function.

| ROW LABELS | ACCESSORIES | BIKES | CLOTHING | GRAND TOTAL |
|-------------|-------------|-------|----------|-------------|
| 2005 | | 1013 | | 1013 |
| 2006 | | 2677 | | 2677 |
| 2007 | 6792 | 4875 | 2867 | 9309 |
| 2008 | 9435 | 5451 | 4196 | 11377 |
| Grand Total | 15114 | 9132 | 6852 | 18484 |

Also, note that the results are not additive. That is to say, the total number of unique customers in *2007* is not the sum of unique customers of *Accessories*, *Bikes* and *Clothing* for that year. The reason is that a customer can be counted in multiple groups.

See also

[Filter functions](#)

[FILTER function](#)

[RELATED function](#)

[VALUES function](#)

DISTINCT (table)

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns a table by removing duplicate rows from another table or expression.

Syntax

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

Parameters

| TERM | DEFINITION |
|-------|---|
| table | The table from which unique rows are to be returned. The table can also be an expression that results in a table. |

Return value

A table containing only distinct rows.

Related functions

There is another version of the DISTINCT function, [DISTINCT \(column\)](#), that takes a column name as input parameter.

Example

The following query:

```
EVALUATE DISTINCT( { (1, "A"), (2, "B"), (1, "A") } )
```

Returns table:

| [VALUE1] | [VALUE2] |
|----------|----------|
| 1 | A |
| 2 | B |

See also

[Filter functions](#)

[DISTINCT \(column\)](#)

[FILTER function](#)

[RELATED function](#)

[VALUES function](#)

EXCEPT

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the rows of one table which do not appear in another table.

Syntax

```
EXCEPT(<table_expression1>, <table_expression2>)
```

Parameters

| TERM | DEFINITION |
|------------------|--|
| Table_expression | Any DAX expression that returns a table. |

Return value

A table that contains the rows of one table minus all the rows of another table.

Remarks

- If a row appears at all in both tables, it and its duplicates are not present in the result set. If a row appears in only table_expression1, it and its duplicates will appear in the result set.
- The column names will match the column names in table_expression1.
- The returned table has lineage based on the columns in table_expression1 , regardless of the lineage of the columns in the second table. For example, if the first column of first table_expression has lineage to the base column C1 in the model, the Except will reduce the rows based on the availability of values in the first column of second table_expression and keep the lineage on base column C1 intact.
- The two tables must have the same number of columns.
- Columns are compared based on positioning, and data comparison with no type coercion.
- The set of rows returned depends on the order of the two expressions.
- The returned table does not include columns from tables related to table_expression1.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

States1

| STATE |
|-------|
| A |
| B |

| STATE |
|-------|
| B |
| B |
| C |
| D |
| D |

States2

| STATE |
|-------|
| B |
| C |
| D |
| D |
| D |
| E |
| E |
| E |

Except(States1, States2)

| STATE |
|-------|
| A |

Except(States2, States1)

| STATE |
|-------|
| E |
| E |
| E |

FILTERS

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the values that are directly applied as filters to *columnName*.

Syntax

```
FILTERS(<columnName>)
```

Parameters

| TERM | DESCRIPTION |
|------------|--|
| columnName | The name of an existing column, using standard DAX syntax. It cannot be an expression. |

Return value

The values that are directly applied as filters to *columnName*.

Remarks

This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following example shows how to determine the number of direct filters a column has.

```
= COUNTROWS(FILTERS(ResellerSales_USD[ProductKey]))
```

This example lets you know how many direct filters on ResellerSales_USD[ProductKey] have been applied to the context where the expression is being evaluated.

GENERATE

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns a table with the Cartesian product between each row in *table1* and the table that results from evaluating *table2* in the context of the current row from *table1*.

Syntax

```
GENERATE(<table1>, <table2>)
```

Parameters

| TERM | DEFINITION |
|--------|--|
| table1 | Any DAX expression that returns a table. |
| table2 | Any DAX expression that returns a table. |

Return value

A table with the Cartesian product between each row in *table1* and the table that results from evaluating *table2* in the context of the current row from *table1*

Remarks

- If the evaluation of *table2* for the current row in *table1* returns an empty table, then the result table will not contain the current row from *table1*. This is different than `GENERATEALL()` where the current row from *table1* will be included in the results and columns corresponding to *table2* will have null values for that row.
- All column names from *table1* and *table2* must be different or an error is returned.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

In the following example the user wants a summary table of the sales by Region and Product Category for the Resellers channel, like the following table:

| SALESTERRITORY[SALESTERRITORYGROUP] | PRODUCTCATEGORY[PRODUCTCATEGORYNAME] | [RESELLER SALES] |
|-------------------------------------|--------------------------------------|------------------|
| Europe | Accessories | \$ 142,227.27 |
| Europe | Bikes | \$ 9,970,200.44 |
| Europe | Clothing | \$ 365,847.63 |

| SALESTERRITORY[SALESTERRITORYGROUP] | PRODUCTCATEGORY[PRODUCTCATEGORYNAME] | [RESELLER SALES] |
|-------------------------------------|--------------------------------------|------------------|
| Europe | Components | \$ 2,214,440.19 |
| North America | Accessories | \$ 379,305.15 |
| North America | Bikes | \$ 52,403,796.85 |
| North America | Clothing | \$ 1,281,193.26 |
| North America | Components | \$ 8,882,848.05 |
| Pacific | Accessories | \$ 12,769.57 |
| Pacific | Bikes | \$ 710,677.75 |
| Pacific | Clothing | \$ 22,902.38 |
| Pacific | Components | \$ 108,549.71 |

The following formula produces the above table:

```
GENERATE(
SUMMARIZE(SalesTerritory, SalesTerritory[SalesTerritoryGroup])
,SUMMARIZE(ProductCategory
, [ProductCategoryName]
, "Reseller Sales", SUMX(RELATEDTABLE(ResellerSales_USD), ResellerSales_USD[SalesAmount_USD])
)
)
```

1. The first SUMMARIZE statement, `SUMMARIZE(SalesTerritory, SalesTerritory[SalesTerritoryGroup])`, produces a table of territory groups, where each row is a territory group, as shown below:

| SALESTERRITORY[SALESTERRITORYGROUP] |
|-------------------------------------|
| North America |
| Europe |
| Pacific |
| NA |

2. The second SUMMARIZE statement,

```
SUMMARIZE(ProductCategory, [ProductCategoryName], "Reseller Sales",
SUMX(RELATEDTABLE(ResellerSales_USD), ResellerSales_USD[SalesAmount_USD]))
```

, produces a table of Product Category groups with the Reseller sales for each group, as shown below:

| PRODUCTCATEGORY[PRODUCTCATEGORYNAME] | [RESELLER SALES] |
|--------------------------------------|------------------|
| Bikes | \$ 63,084,675.04 |
| Components | \$ 11,205,837.96 |

| PRODUCTCATEGORY[PRODUCTCATEGORYNAME] | [RESELLER SALES] |
|--------------------------------------|------------------|
| Clothing | \$ 1,669,943.27 |
| Accessories | \$ 534,301.99 |

3. However, when you take the above table and evaluate it under the context of each row from the territory groups table, you obtain different results for each territory.

GENERATEALL

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns a table with the Cartesian product between each row in *table1* and the table that results from evaluating *table2* in the context of the current row from *table1*.

Syntax

```
GENERATEALL(<table1>, <table2>)
```

Parameters

| TERM | DEFINITION |
|--------|--|
| table1 | Any DAX expression that returns a table. |
| table2 | Any DAX expression that returns a table. |

Return value

A table with the Cartesian product between each row in *table1* and the table that results from evaluating *table2* in the context of the current row from *table1*

Remarks

- If the evaluation of *table2* for the current row in *table1* returns an empty table, then the current row from *table1* will be included in the results and columns corresponding to *table2* will have null values for that row. This is different than GENERATE() where the current row from *table1* will **not** be included in the results.
- All column names from *table1* and *table2* must be different or an error is returned.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

In the following example, the user wants a summary table of the sales by Region and Product Category for the Resellers channel, like the following table:

| SALESTERRITORY[SALESTERRITORYGROUP] | PRODUCTCATEGORY[PRODUCTCATEGORYNAME] | [RESELLER SALES] |
|-------------------------------------|--------------------------------------|------------------|
| Europe | Accessories | \$ 142,227.27 |
| Europe | Bikes | \$ 9,970,200.44 |
| Europe | Clothing | \$ 365,847.63 |

| SALESTERRITORY[SALESTERRITORYGROUP] | PRODUCTCATEGORY[PRODUCTCATEGORYNAME] | [RESELLER SALES] |
|-------------------------------------|--------------------------------------|------------------|
| Europe | Components | \$ 2,214,440.19 |
| NA | Accessories | |
| NA | Bikes | |
| NA | Clothing | |
| NA | Components | |
| North America | Accessories | \$ 379,305.15 |
| North America | Bikes | \$ 52,403,796.85 |
| North America | Clothing | \$ 1,281,193.26 |
| North America | Components | \$ 8,882,848.05 |
| Pacific | Accessories | \$ 12,769.57 |
| Pacific | Bikes | \$ 710,677.75 |
| Pacific | Clothing | \$ 22,902.38 |
| Pacific | Components | \$ 108,549.71 |

The following formula produces the above table:

```

GENERATEALL(
  SUMMARIZE(SalesTerritory, SalesTerritory[SalesTerritoryGroup])
, SUMMARIZE(ProductCategory
, [ProductCategoryName]
, "Reseller Sales", SUMX(RELATEDTABLE(ResellerSales_USD), ResellerSales_USD[SalesAmount_USD])
)
)

```

1. The first SUMMARIZE produces a table of territory groups, where each row is a territory group, like those listed below:

| SALESTERRITORY[SALESTERRITORYGROUP] |
|-------------------------------------|
| North America |
| Europe |
| Pacific |
| NA |

2. The second SUMMARIZE produces a table of Product Category groups with the Reseller sales for each group, as shown below:

| PRODUCTCATEGORY[PRODUCTCATEGORYNAME] | [RESELLER SALES] |
|--------------------------------------|------------------|
| Bikes | \$ 63,084,675.04 |
| Components | \$ 11,205,837.96 |
| Clothing | \$ 1,669,943.27 |
| Accessories | \$ 534,301.99 |

- However, when you take the above table and evaluate the table under the context of each row from the territory groups table, you obtain different results for each territory.

GENERATESERIES

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns a single column table containing the values of an arithmetic series, that is, a sequence of values in which each differs from the preceding by a constant quantity. The name of the column returned is Value.

Syntax

```
GENERATESERIES(<startValue>, <endValue>[, <incrementValue>])
```

Parameters

| TERM | DEFINITION |
|----------------|--|
| startValue | The initial value used to generate the sequence. |
| endValue | The end value used to generate the sequence. |
| incrementValue | (Optional) The increment value of the sequence. When not provided, the default value is 1. |

Return value

A single column table containing the values of an arithmetic series. The name of the column is Value.

Remarks

- When startValue is less than endValue, an empty table is returned.
- incrementValue must be a positive value.
- The sequence stops at the last value that is less than or equal to endValue.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example 1

The following DAX query:

```
EVALUATE GENERATESERIES(1, 5)
```

Returns the following table with a single column:

| [VALUE] |
|---------|
| 1 |
| 2 |

| [VALUE] |
|---------|
| 3 |
| 4 |
| 5 |

Example 2

The following DAX query:

```
EVALUATE GENERATESERIES(1.2, 2.4, 0.4)
```

Returns the following table with a single column:

| [VALUE] |
|---------|
| 1.2 |
| 1.6 |
| 2 |
| 2.4 |

Example 3

The following DAX query:

```
EVALUATE GENERATESERIES(CURRENCY(10), CURRENCY(12.4), CURRENCY(0.5))
```

Returns the following table with a single column:

| [VALUE] |
|---------|
| 10 |
| 10.5 |
| 11 |
| 11.5 |
| 12 |

GROUPBY

10/31/2022 • 2 minutes to read • [Edit Online](#)

The GROUPBY function is similar to the [SUMMARIZE](#) function. However, GROUPBY does not do an implicit [CALCULATE](#) for any extension columns that it adds. GROUPBY permits a new function, [CURRENTGROUP](#), to be used inside aggregation functions in the extension columns that it adds. GROUPBY is used to perform multiple aggregations in a single table scan.

Syntax

```
GROUPBY (<table> [, <groupBy_columnName> [, <groupBy_columnName> [, ...]] [, <name>, <expression> [, <name>, <expression> [, ...]])
```

Parameters

| TERM | DEFINITION |
|--------------------|--|
| table | Any DAX expression that returns a table of data. |
| groupBy_columnName | The name of an existing column in the table (or in a related table,) by which the data is to be grouped. This parameter cannot be an expression. |
| name | The name given to a new column that is being added to the list of GroupBy columns, enclosed in double quotes. |
| expression | One of the X aggregation functions with the first argument being CURRENTGROUP(). See With CURRENTGROUP section below for the full list of supported X aggregation functions. |

Return value

A table with the selected columns for the groupBy_columnName arguments and the extension columns designated by the name arguments.

Remarks

- The GROUPBY function does the following:
 - Start with the specified table (and all related tables in the "to-one" direction).
 - Create a grouping using all of the GroupBy columns (which are required to exist in the table from step #1.).
 - Each group is one row in the result, but represents a set of rows in the original table.
 - For each group, evaluate the extension columns being added. Unlike the SUMMARIZE function, an implied CALCULATE is not performed, and the group isn't placed into the filter context.
- Each column for which you define a name must have a corresponding expression; otherwise, an error is returned. The first argument, name, defines the name of the column in the results. The second argument, expression, defines the calculation performed to obtain the value for each row in that column.

- `groupBy_columnName` must be either in table or in a related table.
- Each name must be enclosed in double quotation marks.
- The function groups a selected set of rows into a set of summary rows by the values of one or more `groupBy_columnName` columns. One row is returned for each group.
- `GROUPBY` is primarily used to perform aggregations over intermediate results from DAX table expressions. For efficient aggregations over physical tables in the model, consider using [SUMMARIZECOLUMNS](#) or [SUMMARIZE](#) function.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

With CURRENTGROUP

[CURRENTGROUP](#) can only be used in an expression that defines an extension column within the `GROUPBY` function. In-effect, [CURRENTGROUP](#) returns a set of rows from the table argument of `GROUPBY` that belong to the current row of the `GROUPBY` result. The [CURRENTGROUP](#) function takes no arguments and is only supported as the first argument to one of the following aggregation functions: [AVERAGEX](#), [COUNTAX](#), [COUNTX](#), [GEOMEANX](#), [MAXX](#), [MINX](#), [PRODUCTX](#), [STDEVX.S](#), [STDEVX.P](#), [SUMX](#), [VARX.S](#), [VARX.P](#).

Example

The following example first calculates the total sales grouped by country and product category over physical tables by using the [SUMMARIZECOLUMNS](#) function. It then uses the `GROUPBY` function to scan the intermediate result from the first step to find the maximum sales in each country across the product categories.

```
DEFINE
VAR SalesByCountryAndCategory =
SUMMARIZECOLUMNS(
    Geography[Country],
    Product[Category],
    "Total Sales", SUMX(Sales, Sales[Price] * Sales[Qty])
)

EVALUATE
GROUPBY(
    SalesByCountryAndCategory,
    Geography[Country],
    "Max Sales", MAXX(CURRENTGROUP(), [Total Sales])
)
```

See also

[SUMMARIZE function](#)

[SUMMARIZECOLUMNS function](#)

IGNORE

10/31/2022 • 2 minutes to read • [Edit Online](#)

Modifies the behavior of the [SUMMARIZECOLUMNS](#) function by omitting specific expressions from the BLANK/NULL evaluation. Rows for which all expressions not using IGNORE return BLANK/NULL will be excluded independent of whether the expressions which do use IGNORE evaluate to BLANK/NULL or not. This function can only be used within a [SUMMARIZECOLUMNS](#) expression.

Syntax

```
IGNORE(<expression>)
```

With SUMMARIZECOLUMNS,

```
SUMMARIZECOLUMNS(<groupBy_columnName>[, <groupBy_columnName >]..., [<filterTable>]...[, <name>, IGNORE(...)]...)
```

Parameters

| TERM | DEFINITION |
|------------|---|
| expression | Any DAX expression that returns a single value (not a table). |

Return value

The function does not return a value.

Remarks

IGNORE can only be used as an expression argument to [SUMMARIZECOLUMNS](#).

Example

See [SUMMARIZECOLUMNS](#).

INTERSECT

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the row intersection of two tables, retaining duplicates.

Syntax

```
INTERSECT(<table_expression1>, <table_expression2>)
```

Parameters

| TERM | DEFINITION |
|------------------|--|
| Table_expression | Any DAX expression that returns a table. |

Return value

A table that contains all the rows in table_expression1 that are also in table_expression2

Exceptions

Remarks

- Intersect is not commutative. In general, Intersect(T1, T2) will have a different result set than Intersect(T2, T1).
- Duplicate rows are retained. If a row appears in table_expression1 and table_expression2, it and all duplicates in table_expression_1 are included in the result set.
- The column names will match the column names in table_expression1.
- The returned table has lineage based on the columns in table_expression1 , regardless of the lineage of the columns in the second table. For example, if the first column of first table_expression has lineage to the base column C1 in the model, the intersect will reduce the rows based on the intersect on first column of second table_expression and keep the lineage on base column C1 intact.
- Columns are compared based on positioning, and data comparison with no type coercion.
- The returned table does not include columns from tables related to table_expression1.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

States1

| STATE |
|-------|
| A |

| STATE |
|-------|
| A |
| B |
| B |
| B |
| C |
| D |
| D |

States2

| STATE |
|-------|
| B |
| C |
| D |
| D |
| D |
| E |

Intersect(States1, States2)

| STATE |
|-------|
| B |
| B |
| B |
| C |
| D |
| D |

Intersect(States2, States1)

| STATE |
|-------|
|-------|

| |
|-------|
| STATE |
| B |
| C |
| D |
| D |
| D |

NATURALINNERJOIN

10/31/2022 • 2 minutes to read • [Edit Online](#)

Performs an inner join of a table with another table.

Syntax

```
NATURALINNERJOIN(<LeftTable>, <RightTable>)
```

Parameters

| TERM | DEFINITION |
|------------|--|
| LeftTable | A table expression defining the table on the left side of the join. |
| RightTable | A table expression defining the table on the right side of the join. |

Return value

A table which includes only rows for which the values in the common columns specified are present in both tables. The table returned will have the common columns from the left table and other columns from both the tables.

Remarks

- Tables are joined on common columns (by name) in the two tables. If the two tables have no common column names, an error is returned.
- There is no sort order guarantee for the results.
- Columns being joined on must have the same data type in both tables.
- Only columns from the same source table (have the same lineage) are joined on. For example, Products[ProductID], WebSales[ProductID], StoreSales[ProductID] with many-to-one relationships between WebSales and StoreSales and the Products table based on the ProductID column, WebSales and StoreSales tables are joined on [ProductID].
- Strict comparison semantics are used during join. There is no type coercion; for example, 1 does not equal 1.0.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

See also

[NATURALLEFTOUTERJOIN](#)

NATURALLEFTOUTERJOIN

10/31/2022 • 2 minutes to read • [Edit Online](#)

Performs a join of the LeftTable with the RightTable by using the Left Outer Join semantics.

Syntax

```
NATURALLEFTOUTERJOIN(<LeftTable>, <RightTable>)
```

Parameters

| TERM | DEFINITION |
|------------|--|
| LeftTable | A table expression defining the table on the left side of the join. |
| RightTable | A table expression defining the table on the right side of the join. |

Return value

A table which includes only rows from RightTable for which the values in the common columns specified are also present in LeftTable. The table returned will have the common columns from the left table and the other columns from both the tables.

Remarks

- Tables are joined on common columns (by name) in the two tables. If the two tables have no common column names, an error is returned.
- There is no sort order guarantee for the results.
- Columns being joined on must have the same data type in both tables.
- Only columns from the same source table (have the same lineage) are joined on. For example, Products[ProductID], WebSales[ProductID], StoreSales[ProductID] with many-to-one relationships between WebSales and StoreSales and the Products table based on the ProductID column, WebSales and StoreSales tables are joined on [ProductID].
- Strict comparison semantics are used during join. There is no type coercion; for example, 1 does not equal 1.0.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

See also

[NATURALINNERJOIN](#)

ROLLUP

10/31/2022 • 2 minutes to read • [Edit Online](#)

Modifies the behavior of the [SUMMARIZE](#) function by adding rollup rows to the result on columns defined by the `groupBy_columnName` parameter. This function can only be used within a [SUMMARIZE](#) expression.

Syntax

```
ROLLUP ( <groupBy_columnName> [, <groupBy_columnName> [, ... ] ] )
```

With [SUMMARIZE](#),

```
SUMMARIZE(<table>, <groupBy_columnName>[, <groupBy_columnName>]...[, ROLLUP(<groupBy_columnName>[, <groupBy_columnName>...])][, <name>, <expression>]...)
```

Parameters

| TERM | DEFINITION |
|---------------------------------|--|
| <code>groupBy_columnName</code> | The qualified name of an existing column or <code>ROLLUPGROUP</code> function to be used to create summary groups based on the values found in it. This parameter cannot be an expression. |

Return value

This function does not return a value. It only specifies the set of columns to be subtotaled.

Remarks

This function can only be used within a [SUMMARIZE](#) expression.

Example

See [SUMMARIZE](#).

ROLLUPADDISSUBTOTAL

10/31/2022 • 2 minutes to read • [Edit Online](#)

Modifies the behavior of the [SUMMARIZECOLUMNS](#) function by adding rollup/subtotal rows to the result based on the `groupBy_columnName` columns. This function can only be used within a [SUMMARIZECOLUMNS](#) expression.

Syntax

```
ROLLUPADDISSUBTOTAL ( [<grandtotalFilter>], <groupBy_columnName>, <name> [, [<groupLevelFilter>] [, <groupBy_columnName>, <name> [, [<groupLevelFilter>] [, ... ] ] ] ] )
```

Parameters

| TERM | DEFINITION |
|---------------------------------|--|
| <code>grandtotalFilter</code> | (Optional) Filter to be applied to the grandtotal level. |
| <code>groupBy_columnName</code> | Name of an existing column used to create summary groups based on the values found in it. Cannot be an expression. |
| <code>name</code> | Name of an ISSUBTOTAL column. The values of the column are calculated using the ISSUBTOTAL function. |
| <code>groupLevelFilter</code> | (Optional) Filter to be applied to the current level. |

Return value

The function does not return a value.

Remarks

None

Example

See [SUMMARIZECOLUMNS](#).

ROLLUPGROUP

10/31/2022 • 2 minutes to read • [Edit Online](#)

Modifies the behavior of the [SUMMARIZE](#) and [SUMMARIZECOLUMNS](#) functions by adding rollup rows to the result on columns defined by the the `groupBy_columnName` parameter. This function can only be used within a [SUMMARIZE](#) or [SUMMARIZECOLUMNS](#) expression.

Syntax

```
ROLLUPGROUP ( <groupBy_columnName> [ , <groupBy_columnName> [ , ... ] ] )
```

Parameters

| TERM | DEFINITION |
|---------------------------------|---|
| <code>groupBy_columnName</code> | The qualified name of an existing column or ROLLUPGROUP function to be used to create summary groups based on the values found in it. This parameter cannot be an expression. |

Return value

This function does not return a value. It marks a set of columns to be treated as a single group during subtotaling by [ROLLUP](#) or [ROLLUPADDISSUBTOTAL](#).

Remarks

ROLLUPGROUP can only be used as a `groupBy_columnName` argument to [ROLLUP](#), [ROLLUPADDISSUBTOTAL](#), or [ROLLUPISSUBTOTAL](#).

Example

See [SUMMARIZE](#) and [SUMMARIZECOLUMNS](#).

ROLLUPISSUBTOTAL

10/31/2022 • 2 minutes to read • [Edit Online](#)

Pairs rollup groups with the column added by [ROLLUPADDISSUBTOTAL](#). This function can only be used within an [ADDMISSINGITEMS](#) expression.

Syntax

```
ROLLUPISSUBTOTAL ( [<grandTotalFilter>], <groupBy_columnName>, <isSubtotal_columnName> [,  
[<groupLevelFilter>] [, <groupBy_columnName>, <isSubtotal_columnName> [, [<groupLevelFilter>] [, ... ] ] ] ] )
```

Parameters

| TERM | DEFINITION |
|-----------------------|--|
| grandTotalFilter | (Optional) Filter to be applied to the grandtotal level. |
| groupBy_columnName | Name of an existing column used to create summary groups based on the values found in it. Cannot be an expression. |
| isSubtotal_columnName | Name of an ISSUBTOTAL column. The values of the column are calculated using the ISSUBTOTAL function. |
| groupLevelFilter | (Optional) Filter to be applied to the current level. |

Return value

None

Remarks

This function can only be used within an [ADDMISSINGITEMS](#) expression.

ROW function

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns a table with a single row containing values that result from the expressions given to each column.

Syntax

```
ROW(<name>, <expression>[[,<name>, <expression>]...])
```

Parameters

| TERM | DEFINITION |
|------------|--|
| name | The name given to the column, enclosed in double quotes. |
| expression | Any DAX expression that returns a single scalar value to populate. <i>name</i> . |

Return value

A single row table

Remarks

- Arguments must always come in pairs of *name* and *expression*.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following example returns a single row table with the total sales for internet and resellers channels.

```
ROW("Internet Total Sales (USD)", SUM(InternetSales_USD[SalesAmount_USD]),  
    "Resellers Total Sales (USD)", SUM(ResellerSales_USD[SalesAmount_USD]))
```

SELECTCOLUMNS

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns a table with selected columns from the table and new columns specified by the DAX expressions.

Syntax

```
SELECTCOLUMNS(<Table>, [<Name>], <Expression>, <Name>], ...)
```

Parameters

| TERM | DEFINITION |
|------------|---|
| Table | Any DAX expression that returns a table. |
| Name | The name given to the column, enclosed in double quotes. |
| Expression | Any expression that returns a scalar value like a column reference, integer, or string value. |

Return value

A table with the same number of rows as the table specified as the first argument. The returned table has one column for each pair of <Name>, <Expression> arguments, and each expression is evaluated in the context of a row from the specified <Table> argument.

Remarks

SELECTCOLUMNS has the same signature as ADDCOLUMNS, and has the same behavior except that instead of starting with the <Table> specified, SELECTCOLUMNS starts with an empty table before adding columns.

This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

For the following table named **Customer**:

| COUNTRY | STATE | COUNT | TOTAL |
|---------|-------|-------|-------|
| IND | JK | 20 | 800 |
| IND | MH | 25 | 1000 |
| IND | WB | 10 | 900 |
| USA | CA | 5 | 500 |
| USA | WA | 10 | 900 |

```
SELECT COLUMNS(Customer, "Country, State", [Country] & ", " & [State])
```

Returns,

| COUNTRY, STATE |
|----------------|
| IND, JK |
| IND, MH |
| IND, WB |
| USA, CA |
| USA, WA |

SUBSTITUTEWITHINDEX

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns a table which represents a left semijoin of the two tables supplied as arguments. The semijoin is performed by using common columns, determined by common column names and common data type . The columns being joined on are replaced with a single column in the returned table which is of type integer and contains an index. The index is a reference into the right join table given a specified sort order.

Columns in the right/second table supplied which do not exist in the left/first table supplied are not included in the returned table and are not used to join on.

The index starts at 0 (0-based) and is incremented by one for each additional row in the right/second join table supplied. The index is based on the sort order specified for the right/second join table.

Syntax

```
SUBSTITUTEWITHINDEX(<table>, <indexColumnName>, <indexColumnsTable>, [<orderBy_expression>, [<order>]][, <orderBy_expression>, [<order>]]...)
```

Parameters

| TERM | DEFINITION |
|--------------------|--|
| table | A table to be filtered by performing a left semijoin with the table specified as the third argument (indexColumnsTable). This is the table on the left side of the left semijoin so the table returned includes the same columns as this table except that all common columns of the two tables will be replaced with a single index column in the table returned. |
| indexColumnName | A string which specifies the name of the index column which is replacing all the common columns in the two tables supplied as arguments to this function. |
| indexColumnsTable | The second table for the left semijoin. This is the table on the right side of the left semijoin. Only values present in this table will be returned by the function. Also, the columns of this table (based on column names) will be replaced with a single index column in the table returned by this function. |
| orderBy_expression | Any DAX expression where the result value is used to specify the desired sort order of the indexColumnsTable table for generating correct index values. The sort order specified for the indexColumnsTable table defines the index of each row in the table and that index is used in the table returned to represent combinations of values in the indexColumnsTable as they appear in the table supplied as the first argument to this function. |

| TERM | DEFINITION |
|-------|--|
| order | <p>(Optional) A value that specifies how to sort orderBy_expression values, ascending or descending:</p> <p>Value: Desc. Alternative value: 0(zero)/FALSE. Sorts in descending order of values of orderBy_expression. This is the default value when order parameter is omitted.</p> <p>Value: ASC. Alternative value: 1/TRUE. Ranks in ascending order of orderBy_expression.</p> |

Return value

A table which includes only those values present in the indexColumnsTable table and which has an index column instead of all columns present (by name) in the indexColumnsTable table.

Remarks

- This function does not guarantee any result sort order.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

SUMMARIZE

10/31/2022 • 4 minutes to read • [Edit Online](#)

Returns a summary table for the requested totals over a set of groups.

Syntax

```
SUMMARIZE (<table>, <groupBy_columnName>[, <groupBy_columnName>]...[, <name>, <expression>]...)
```

Parameters

| TERM | DEFINITION |
|--------------------|--|
| table | Any DAX expression that returns a table of data. |
| groupBy_ColumnName | (Optional) The qualified name of an existing column used to create summary groups based on the values found in it. This parameter cannot be an expression. |
| name | The name given to a total or summarize column, enclosed in double quotes. |
| expression | Any DAX expression that returns a single scalar value, where the expression is to be evaluated multiple times (for each row/context). |

Return value

A table with the selected columns for the *groupBy_columnName* arguments and the summarized columns designed by the name arguments.

Remarks

- Each column for which you define a name must have a corresponding expression; otherwise, an error is returned. The first argument, name, defines the name of the column in the results. The second argument, expression, defines the calculation performed to obtain the value for each row in that column.
- groupBy_columnName must be either in *table* or in a related table to *table*.
- Each name must be enclosed in double quotation marks.
- The function groups a selected set of rows into a set of summary rows by the values of one or more groupBy_columnName columns. One row is returned for each group.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following example returns a summary of the reseller sales grouped around the calendar year and the product category name, this result table allows you to do analysis over the reseller sales by year and product

category.

```
SUMMARIZE(ResellerSales_USD
    , DateTime[CalendarYear]
    , ProductCategory[ProductCategoryName]
    , "Sales Amount (USD)", SUM(ResellerSales_USD[SalesAmount_USD])
    , "Discount Amount (USD)", SUM(ResellerSales_USD[DiscountAmount])
)
```

The following table shows a preview of the data as it would be received by any function expecting to receive a table:

| DATETIME[CALENDARYEAR] | PRODUCTCATEGORY[PRODUCTCATEGORYNAME] | [SALES AMOUNT (USD)] | [DISCOUNT AMOUNT (USD)] |
|------------------------|--------------------------------------|----------------------|-------------------------|
| 2008 | Bikes | 12968255.42 | 36167.6592 |
| 2005 | Bikes | 6958251.043 | 4231.1621 |
| 2006 | Bikes | 18901351.08 | 178175.8399 |
| 2007 | Bikes | 24256817.5 | 276065.992 |
| 2008 | Components | 2008052.706 | 39.9266 |
| 2005 | Components | 574256.9865 | 0 |
| 2006 | Components | 3428213.05 | 948.7674 |
| 2007 | Components | 5195315.216 | 4226.0444 |
| 2008 | Clothing | 366507.844 | 4151.1235 |
| 2005 | Clothing | 31851.1628 | 90.9593 |
| 2006 | Clothing | 455730.9729 | 4233.039 |
| 2007 | Clothing | 815853.2868 | 12489.3835 |
| 2008 | Accessories | 153299.924 | 865.5945 |
| 2005 | Accessories | 18594.4782 | 4.293 |
| 2006 | Accessories | 86612.7463 | 1061.4872 |
| 2007 | Accessories | 275794.8403 | 4756.6546 |

With ROLLUP

The addition of the [ROLLUP](#) syntax modifies the behavior of the SUMMARIZE function by adding rollup rows to the result on the groupBy_columnName columns. [ROLLUP](#) can only be used within a SUMMARIZE expression.

Example

The following example adds rollup rows to the Group-By columns of the SUMMARIZE function call:

```

SUMMARIZE(ResellerSales_USD
, ROLLUP( DateTime[CalendarYear], ProductCategory[ProductCategoryName])
, "Sales Amount (USD)", SUM(ResellerSales_USD[SalesAmount_USD])
, "Discount Amount (USD)", SUM(ResellerSales_USD[DiscountAmount])
)

```

Returns the following table,

| DATETIME[CALENDARYEAR] | PRODUCTCATEGORY[PRODUCTCATEGORYNAME] | [SALES AMOUNT (USD)] | [DISCOUNT AMOUNT (USD)] |
|------------------------|--------------------------------------|----------------------|-------------------------|
| 2008 | Bikes | 12968255.42 | 36167.6592 |
| 2005 | Bikes | 6958251.043 | 4231.1621 |
| 2006 | Bikes | 18901351.08 | 178175.8399 |
| 2007 | Bikes | 24256817.5 | 276065.992 |
| 2008 | Components | 2008052.706 | 39.9266 |
| 2005 | Components | 574256.9865 | 0 |
| 2006 | Components | 3428213.05 | 948.7674 |
| 2007 | Components | 5195315.216 | 4226.0444 |
| 2008 | Clothing | 366507.844 | 4151.1235 |
| 2005 | Clothing | 31851.1628 | 90.9593 |
| 2006 | Clothing | 455730.9729 | 4233.039 |
| 2007 | Clothing | 815853.2868 | 12489.3835 |
| 2008 | Accessories | 153299.924 | 865.5945 |
| 2005 | Accessories | 18594.4782 | 4.293 |
| 2006 | Accessories | 86612.7463 | 1061.4872 |
| 2007 | Accessories | 275794.8403 | 4756.6546 |
| 2008 | | 15496115.89 | 41224.3038 |
| 2005 | | 7582953.67 | 4326.4144 |
| 2006 | | 22871907.85 | 184419.1335 |
| 2007 | | 30543780.84 | 297538.0745 |
| | | 76494758.25 | 527507.9262 |

With ROLLUPGROUP

The addition of [ROLLUPGROUP](#) inside a [ROLLUP](#) syntax can be used to prevent partial subtotals in rollup rows. [ROLLUPGROUP](#) can only be used within a [ROLLUP](#), [ROLLUPADDISSUBTOTAL](#), or [ROLLUPISSUBTOTAL](#) expression.

Example

The following example shows only the grand total of all years and categories without the subtotal of each year with all categories:

```
SUMMARIZE(ResellerSales_USD
    , ROLLUP(ROLLUPGROUP( DateTime[CalendarYear], ProductCategory[ProductCategoryName]))
    , "Sales Amount (USD)", SUM(ResellerSales_USD[SalesAmount_USD])
    , "Discount Amount (USD)", SUM(ResellerSales_USD[DiscountAmount])
)
```

Returns the following table,

| DATETIME[CALENDARYEAR] | PRODUCTCATEGORY[PRODUCTCATEGORYNAME] | [SALES AMOUNT (USD)] | [DISCOUNT AMOUNT (USD)] |
|------------------------|--------------------------------------|----------------------|-------------------------|
| 2008 | Bikes | 12968255.42 | 36167.6592 |
| 2005 | Bikes | 6958251.043 | 4231.1621 |
| 2006 | Bikes | 18901351.08 | 178175.8399 |
| 2007 | Bikes | 24256817.5 | 276065.992 |
| 2008 | Components | 2008052.706 | 39.9266 |
| 2005 | Components | 574256.9865 | 0 |
| 2006 | Components | 3428213.05 | 948.7674 |
| 2007 | Components | 5195315.216 | 4226.0444 |
| 2008 | Clothing | 366507.844 | 4151.1235 |
| 2005 | Clothing | 31851.1628 | 90.9593 |
| 2006 | Clothing | 455730.9729 | 4233.039 |
| 2007 | Clothing | 815853.2868 | 12489.3835 |
| 2008 | Accessories | 153299.924 | 865.5945 |
| 2005 | Accessories | 18594.4782 | 4.293 |
| 2006 | Accessories | 86612.7463 | 1061.4872 |
| 2007 | Accessories | 275794.8403 | 4756.6546 |
| | | 76494758.25 | 527507.9262 |

| | | | |
|----------------------------|--|----------------------|----------------------------|
| DATETIME[CALENDAR YEAR] | PRODUCTCATEGORY[PROD UCTCATEGORYNAME] | [SALES AMOUNT (USD)] | [DISCOUNT AMOUNT (USD)] |
|----------------------------|--|----------------------|----------------------------|

With ISSUBTOTAL

With [ISSUBTOTAL](#), you can create another column in the SUMMARIZE expression that returns True if the row contains subtotal values for the column given as argument to [ISSUBTOTAL](#), otherwise returns False. [ISSUBTOTAL](#) can only be used within a SUMMARIZE expression.

Example

The following sample generates an [ISSUBTOTAL](#) column for each of the [ROLLUP](#) columns in the given SUMMARIZE function call:

```
SUMMARIZE(ResellerSales_USD
, ROLLUP( DateTime[CalendarYear], ProductCategory[ProductCategoryName])
, "Sales Amount (USD)", SUM(ResellerSales_USD[SalesAmount_USD])
, "Discount Amount (USD)", SUM(ResellerSales_USD[DiscountAmount])
, "Is Sub Total for DateTimeCalendarYear", ISSUBTOTAL(DateTime[CalendarYear])
, "Is Sub Total for ProductCategoryName", ISSUBTOTAL(ProductCategory[ProductCategoryName])
)
```

Returns the following table,

| [IS SUB TOTAL FOR DATETIMECALEN DARYEAR] | [IS SUB TOTAL FOR PRODUCTCATEG ORYNAME] | DATETIME[CALE NDARYEAR] | PRODUCTCATEG ORY[PRODUCTC ATEGORYNAME] | [SALES AMOUNT (USD)] | [DISCOUNT AMOUNT (USD)] |
|---|--|----------------------------|--|-------------------------|----------------------------|
| FALSE | FALSE | | | | |
| FALSE | FALSE | 2008 | Bikes | 12968255.42 | 36167.6592 |
| FALSE | FALSE | 2005 | Bikes | 6958251.043 | 4231.1621 |
| FALSE | FALSE | 2006 | Bikes | 18901351.08 | 178175.8399 |
| FALSE | FALSE | 2007 | Bikes | 24256817.5 | 276065.992 |
| FALSE | FALSE | 2008 | Components | 2008052.706 | 39.9266 |
| FALSE | FALSE | 2005 | Components | 574256.9865 | 0 |
| FALSE | FALSE | 2006 | Components | 3428213.05 | 948.7674 |
| FALSE | FALSE | 2007 | Components | 5195315.216 | 4226.0444 |
| FALSE | FALSE | 2008 | Clothing | 366507.844 | 4151.1235 |
| FALSE | FALSE | 2005 | Clothing | 31851.1628 | 90.9593 |
| FALSE | FALSE | 2006 | Clothing | 455730.9729 | 4233.039 |

| [IS SUB TOTAL FOR DATETIMECALEN DARYEAR] | [IS SUB TOTAL FOR PRODUCTCATEG ORYNAME] | DATETIME[CALE NDARYEAR] | PRODUCTCATEG ORY[PRODUCTC ATEGORYNAME] | [SALES AMOUNT (USD)] | [DISCOUNT AMOUNT (USD)] |
|---|--|----------------------------|--|-------------------------|----------------------------|
| FALSE | FALSE | 2007 | Clothing | 815853.2868 | 12489.3835 |
| FALSE | FALSE | 2008 | Accessories | 153299.924 | 865.5945 |
| FALSE | FALSE | 2005 | Accessories | 18594.4782 | 4.293 |
| FALSE | FALSE | 2006 | Accessories | 86612.7463 | 1061.4872 |
| FALSE | FALSE | 2007 | Accessories | 275794.8403 | 4756.6546 |
| FALSE | TRUE | | | | |
| FALSE | TRUE | 2008 | | 15496115.89 | 41224.3038 |
| FALSE | TRUE | 2005 | | 7582953.67 | 4326.4144 |
| FALSE | TRUE | 2006 | | 22871907.85 | 184419.1335 |
| FALSE | TRUE | 2007 | | 30543780.84 | 297538.0745 |
| TRUE | TRUE | | | 76494758.25 | 527507.9262 |

See also

[SUMMARIZECOLUMNS](#)

SUMMARIZECOLUMNS

10/31/2022 • 5 minutes to read • [Edit Online](#)

Returns a summary table over a set of groups.

Syntax

```
SUMMARIZECOLUMNS( <groupBy_columnName> [, <groupBy_columnName >]..., [<filterTable>]...[, <name>, <expression>] ...)
```

Parameters

| TERM | DEFINITION |
|--------------------|--|
| groupBy_columnName | A fully qualified column reference (Table[Column]) to a base table for which the distinct values are included in the returned table. Each groupBy_columnName column is cross-joined (different tables) or auto-existed (same table) with the subsequent specified columns. |
| filterTable | A table expression which is added to the filter context of all columns specified as groupBy_columnName arguments. The values present in the filter table are used to filter before cross-join/auto-exist is performed. |
| name | A string representing the column name to use for the subsequent expression specified. |
| expression | Any DAX expression that returns a single value (not a table). |

Return value

A table which includes combinations of values from the supplied columns based on the grouping specified. Only rows for which at least one of the supplied expressions return a non-blank value are included in the table returned. If all expressions evaluate to BLANK/NULL for a row, that row is not included in the table returned.

Remarks

- This function does not guarantee any sort order for the results.
- A column cannot be specified more than once in the groupBy_columnName parameter. For example, the following formula is invalid.

```
SUMMARIZECOLUMNS( Sales[StoreId], Sales[StoreId] )
```

- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Filter context

Consider the following query:

```
SUMMARIZECOLUMNS (
    'Sales Territory'[Category],
    FILTER('Customer', 'Customer' [First Name] = "Alicia")
)
```

In this query, without a measure the groupBy columns do not contain any columns from the FILTER expression (for example, from Customer table). The filter is not applied to the groupBy columns. The Sales Territory and Customer tables may be indirectly related through the Reseller sales fact table. Since they're not directly related, the filter expression is a no-op and the groupBy columns are not impacted.

However, with this query:

```
SUMMARIZECOLUMNS (
    'Sales Territory'[Category], 'Customer' [Education],
    FILTER('Customer', 'Customer' [First Name] = "Alicia")
)
```

The groupBy columns contain a column which is impacted by the filter and that filter is applied to the groupBy results.

With IGNORE

The **IGNORE** syntax can be used to modify the behavior of the SUMMARIZECOLUMNS function by omitting specific expressions from the BLANK/NULL evaluation. Rows for which all expressions not using **IGNORE** return BLANK/NULL will be excluded independent of whether the expressions which do use **IGNORE** evaluate to BLANK/NULL or not. **IGNORE** can only be used within a SUMMARIZECOLUMNS expression.

Example

```
SUMMARIZECOLUMNS(
    Sales[CustomerId], "Total Qty",
    IGNORE( SUM( Sales[Qty] ) ),
    "BlankIfTotalQtyIsNot3", IF( SUM( Sales[Qty] )=3, 3 )
)
```

This rolls up the Sales[CustomerId] column, creating a subtotal for all customers in the given grouping. Without **IGNORE**, the result is:

| CUSTOMERID | TOTALQTY | BLANKIFTOTALQTYISNOT3 |
|------------|----------|-----------------------|
| A | 5 | |
| B | 3 | 3 |
| C | 3 | 3 |

With **IGNORE**,

| CUSTOMERID | TOTALQTY | BLANKIFTOTALQTYISNOT3 |
|------------|----------|-----------------------|
| B | 3 | 3 |
| C | 3 | 3 |

All expression ignored,


```
SUMMARIZECOLUMNS(
    Sales[CustomerId], "Blank",
    IGNORE( Blank() ), "BlankIfTotalQtyIsNot5",
    IGNORE( IF( SUM( Sales[Qty] )=5, 5 ) )
)
```

Even though both expressions return blank for some rows, they're included since there are no unignored expressions which return blank.

| CUSTOMERID | TOTALQTY | BLANKIFTOTALQTYISNOT3 |
|------------|----------|-----------------------|
| A | | 5 |
| B | | |
| C | | |

With NONVISUAL

The **NONVISUAL** function marks a value filter in SUMMARIZECOLUMNS function as not affecting measure values, but only applying to groupBy columns. **NONVISUAL** can only be used within a SUMMARIZECOLUMNS expression.

Example

```
DEFINE
MEASURE FactInternetSales[Sales] = SUM(FactInternetSales[Sales Amount])
EVALUATE
SUMMARIZECOLUMNS
(
    DimDate[CalendarYear],
    NONVISUAL(TREATAS({2007, 2008}, DimDate[CalendarYear])),
    "Sales", [Sales],
    "Visual Total Sales", CALCULATE([Sales], ALLSELECTED(DimDate[CalendarYear]))
)
ORDER BY [CalendarYear]
```

Returns the result where [Visual Total Sales] is the total across all years:

| DIMDATE[CALENDARYEAR] | [SALES] | [VISUAL TOTAL SALES] |
|-----------------------|--------------|----------------------|
| 2007 | 9,791,060.30 | 29,358,677.22 |
| 2008 | 9,770,899.74 | 29,358,677.22 |

In contrast, the same query *without* the NONVISUAL function:

```

DEFINE
MEASURE FactInternetSales[Sales] = SUM(FactInternetSales[Sales Amount])
EVALUATE
SUMMARIZECOLUMNS
(
    DimDate[CalendarYear],
    TREATAS({2007, 2008}, DimDate[CalendarYear]),
    "Sales", [Sales],
    "Visual Total Sales", CALCULATE([Sales], ALLSELECTED(DimDate[CalendarYear]))
)
ORDER BY [CalendarYear]

```

Returns the result where [Visual Total Sales] is the total across the two selected years:

| DIMDATE[CALENDARYEAR] | [SALES] | [VISUAL TOTAL SALES] |
|-----------------------|--------------|----------------------|
| 2007 | 9,791,060.30 | 19,561,960.04 |
| 2008 | 9,770,899.74 | 19,561,960.04 |

With ROLLUPADDISSUBTOTAL

The addition of the [ROLLUPADDISSUBTOTAL](#) syntax modifies the behavior of the SUMMARIZECOLUMNS function by adding rollup/subtotal rows to the result based on the `groupBy_columnName` columns. [ROLLUPADDISSUBTOTAL](#) can only be used within a SUMMARIZECOLUMNS expression.

Example with single subtotal

```

DEFINE
VAR vCategoryFilter =
    TREATAS({"Accessories", "Clothing"}, Product[Category])
VAR vSubcategoryFilter =
    TREATAS({"Bike Racks", "Mountain Bikes"}, Product[Subcategory])
EVALUATE
SUMMARIZECOLUMNS
(
    ROLLUPADDISSUBTOTAL
    (
        Product[Category], "IsCategorySubtotal", vCategoryFilter,
        Product[Subcategory], "IsSubcategorySubtotal", vSubcategoryFilter
    ),
    "Total Qty", SUM(Sales[Qty])
)
ORDER BY
[IsCategorySubtotal] DESC, [Category],
[IsSubcategorySubtotal] DESC, [Subcategory]

```

Returns the following table,

| CATEGORY | SUBCATEGORY | ISCATEGORYSUBTOTAL | ISSUBCATEGORYSUBTOTAL | TOTAL QTY |
|-------------|-------------|--------------------|-----------------------|-----------|
| | | True | True | 60398 |
| Accessories | | False | True | 36092 |
| Accessories | Bike Racks | False | False | 328 |

| CATEGORY | SUBCATEGORY | ISCATEGORYSUBTOTAL | ISSUBCATEGORYSUBTOTAL | TOTAL QTY |
|----------|----------------|--------------------|-----------------------|-----------|
| Bikes | Mountain Bikes | False | False | 4970 |
| Clothing | | False | True | 9101 |

Example with multiple subtotals

```
SUMMARIZECOLUMNS (
    Regions[State], ROLLUPADDISSUBTOTAL ( Sales[CustomerId], "IsCustomerSubtotal" ),
    ROLLUPADDISSUBTOTAL ( Sales[Date], "IsDateSubtotal"), "Total Qty", SUM( Sales[Qty] )
)
```

Sales is grouped by state, by customer, by date, with subtotals for 1. Sales by state, by date 2. Sales by State, by Customer 3. Rolled up on both customer and date leading to sales by state.

Returns the following table,

| CUSTOMERID | ISCUSTOMERSUBTOTAL | STATE | TOTAL QTY | DATE | ISDATESUBTOTAL |
|------------|--------------------|-------|-----------|-----------|----------------|
| A | FALSE | WA | 5 | 7/10/2014 | |
| B | FALSE | WA | 1 | 7/10/2014 | |
| B | FALSE | WA | 2 | 7/11/2014 | |
| C | FALSE | OR | 2 | 7/10/2014 | |
| C | FALSE | OR | 1 | 7/11/2014 | |
| | TRUE | WA | 6 | 7/10/2014 | |
| | TRUE | WA | 2 | 7/11/2014 | |
| | TRUE | OR | 2 | 7/10/2014 | |
| | TRUE | OR | 1 | 7/11/2014 | |
| A | FALSE | WA | 5 | | TRUE |
| B | FALSE | WA | 3 | | TRUE |
| C | FALSE | OR | 3 | | TRUE |
| | TRUE | WA | 8 | | TRUE |
| | TRUE | OR | 3 | | TRUE |

With ROLLUPGROUP

Like with the [SUMMARIZE](#) function, [ROLLUPGROUP](#) can be used together with [ROLLUPADDISSUBTOTAL](#) to specify which summary groups/granularities (subtotals) to include, reducing the number of subtotal rows

returned. [ROLLUPGROUP](#) can only be used within a [SUMMARIZECOLUMNS](#) or [SUMMARIZE](#) expression.

Example with multiple subtotals

```
SUMMARIZECOLUMNS(  
    ROLLUPADDISSUBTOTAL( Sales[CustomerId], "IsCustomerSubtotal" ),  
    ROLLUPADDISSUBTOTAL(ROLLUPGROUP(Regions[City], Regions[State]), "IsCityStateSubtotal"), "Total Qty", SUM(  
    Sales[Qty] )  
)
```

Still grouped by City and State, but rolled together when reporting a subtotal returns the following table,

| STATE | CUSTOMERID | ISCUSTOMERSUB TOTAL | TOTAL QTY | CITY | ISCITYSTATESUB TOTAL |
|-------|------------|------------------------|-----------|----------|-------------------------|
| WA | A | FALSE | 2 | Bellevue | FALSE |
| WA | B | FALSE | 2 | Bellevue | FALSE |
| WA | A | FALSE | 3 | Redmond | FALSE |
| WA | B | FALSE | 1 | Redmond | FALSE |
| OR | C | FALSE | 3 | Portland | FALSE |
| WA | | TRUE | 4 | Bellevue | FALSE |
| WA | | TRUE | 4 | Redmond | FALSE |
| OR | | TRUE | 3 | Portland | FALSE |
| | A | FALSE | 5 | | FALSE |
| | B | FALSE | 3 | | TRUE |
| | C | FALSE | 3 | | TRUE |
| | | TRUE | 11 | | TRUE |

See also

[SUMMARIZE](#)

Table constructor

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns a table of one or more columns.

Syntax

```
{ <scalarExpr1>, <scalarExpr2>, ... }  
{ ( <scalarExpr1>, <scalarExpr2>, ... ), ( <scalarExpr1>, <scalarExpr2>, ... ), ... }
```

Parameters

| TERM | DEFINITION |
|-------------|---|
| scalarExprN | Any DAX expression that returns a scalar value. |

Return value

A table of one or more columns. When there is only one column, the name of the column is Value. When there are N columns where $N > 1$, the names of the columns from left to right are Value1, Value2, ..., ValueN.

Remarks

- The first syntax returns a table of a single column. The second syntax returns a table of one or more columns.
- The number of scalar expressions must be the same for all rows.
- When the data types of the values for a column are different in different rows, all values are converted to a common data type.

Example 1

The following DAX queries:

```
EVALUATE { 1, 2, 3 }
```

and

```
EVALUATE { (1), (2), (3) }
```

Return the following table of a single column:

| [VALUE] |
|---------|
| 1 |
| 2 |

| |
|---------|
| [VALUE] |
| 3 |

Example 2

The following DAX query:

```
EVALUATE
{
    (1.5, DATE(2017, 1, 1), CURRENCY(199.99), "A"),
    (2.5, DATE(2017, 1, 2), CURRENCY(249.99), "B"),
    (3.5, DATE(2017, 1, 3), CURRENCY(299.99), "C")
}
```

Returns,

| [VALUE1] | [VALUE2] | [VALUE3] | [VALUE4] |
|----------|----------|----------|----------|
| 1.5 | 1/1/2017 | 199.99 | A |
| 2.5 | 1/2/2017 | 249.99 | B |
| 3.5 | 1/3/2017 | 299.99 | C |

Example 3

The following DAX query:

```
EVALUATE { 1, DATE(2017, 1, 1), TRUE, "A" }
```

Returns the following table of a single column of String data type:

| [VALUE] |
|----------|
| 1 |
| 1/1/2017 |
| TRUE |
| A |

TOPN

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the top N rows of the specified table.

Syntax

```
TOPN(<N_Value>, <Table>, <OrderBy_Expression>, [<Order>[, <OrderBy_Expression>, [<Order>]]...])
```

Parameters

| PARAMETER | DEFINITION |
|--------------------|--|
| N_Value | The number of rows to return. Any DAX expression that returns a scalar value, where the expression is to be evaluated multiple times (for each row/context). See Remarks to better understand when the number of rows returned could be larger than <i>n_value</i> . |
| Table | Any DAX expression that returns a table of data from where to extract the top 'n' rows. See Remarks to better understand when an empty table is returned. |
| OrderBy_Expression | Any DAX expression where the result value is used to sort the table and evaluated for each row of <i>table</i> . |
| Order | (Optional) A value that specifies how to sort <i>OrderBy_Expression</i> values: - 0 (zero) or FALSE. Sorts in descending order of values of <i>Order</i> . Default when <i>Order</i> parameter is omitted. - 1 or TRUE. Ranks in ascending order of <i>OrderBy</i> . |

Return value

A table with the top N rows of *Table* or an empty table if *N_Value* is 0 (zero) or less. Rows are not sorted in any particular order.

Remarks

- If there is a tie, in *Order_By* values, at the N-th row of the table, then all tied rows are returned. Then, when there are ties at the N-th row the function might return more than n rows.
- If N_Value is 0 (zero) or less, TOPN returns an empty table.
- TOPN does not guarantee any sort order for the results.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following measure formula returns the top 10 sold products by sales amount.

```
= SUMX(
    TOPN(10,
        SUMMARIZE(Product, [ProductKey], "TotalSales",
            SUMX(RELATED(InternetSales_USD[SalesAmount_USD]),
                InternetSales_USD[SalesAmount_USD]) + SUMX(RELATED(ResellerSales_USD[SalesAmount_USD]),
                    ResellerSales_USD[SalesAmount_USD])
            )
        )
    )
)
```


TREATAS

10/31/2022 • 2 minutes to read • [Edit Online](#)

Applies the result of a table expression as filters to columns from an unrelated table.

Syntax

```
TREATAS(table_expression, <column>[, <column>[, <column>[,...]]] )
```

Parameters

| TERM | DEFINITION |
|------------------|---|
| table_expression | An expression that results in a table. |
| column | One or more existing columns. It cannot be an expression. |

Return value

A table that contains all the rows in column(s) that are also in table_expression.

Remarks

- The number of columns specified must match the number of columns in the table expression and be in the same order.
- If a value returned in the table expression does not exist in the column, it is ignored. For example, `TREATAS({"Red", "Green", "Yellow"}, DimProduct[Color])` sets a filter on column `DimProduct[Color]` with three values "Red", "Green", and "Yellow". If "Yellow" does not exist in `DimProduct[Color]`, the effective filter values would be "Red" and "Green".
- Best for use when a relationship does not exist between the tables. If you have multiple relationships between the tables involved, consider using [USERELATIONSHIP](#) instead.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

In the following example, the model contains two unrelated product tables. If a user applies a filter to `DimProduct1[ProductCategory]` selecting Bikes, Seats, Tires, the same filter, Bikes, Seats, Tires is applied to `DimProduct2[ProductCategory]`.

```
CALCULATE(  
    SUM(Sales[Amount]),  
    TREATAS(VALUES(DimProduct1[ProductCategory]), DimProduct2[ProductCategory])  
)
```

See also

INTERSECT
FILTER
USERRELATIONSHIP

UNION

10/31/2022 • 2 minutes to read • [Edit Online](#)

Creates a union (join) table from a pair of tables.

Syntax

```
UNION(<table_expression1>, <table_expression2> [,<table_expression>]...)
```

Parameters

| TERM | DEFINITION |
|------------------|--|
| table_expression | Any DAX expression that returns a table. |

Return value

A table that contains all the rows from each of the two table expressions.

Remarks

- The two tables must have the same number of columns.
- Columns are combined by position in their respective tables.
- The column names in the return table will match the column names in table_expression1.
- Duplicate rows are retained.
- The returned table has lineage where possible. For example, if the first column of each table_expression has lineage to the same base column C1 in the model, the first column in the UNION result will have lineage to C1. However, if combined columns have lineage to different base columns, or if there is an extension column, the resulting column in UNION will have no lineage.
- When data types differ, the resulting data type is determined based on the rules for data type coercion.
- The returned table will not contain columns from related tables.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following expression creates a union by combining the USAInventory table and the INDInventory table into a single table:

```
UNION(UsaInventory, IndInventory)
```

USAInventory

| COUNTRY | STATE | COUNT | TOTAL |
|---------|-------|-------|-------|
| USA | CA | 5 | 500 |
| USA | WA | 10 | 900 |

INDInventory

| COUNTRY | STATE | COUNT | TOTAL |
|---------|-------|-------|-------|
| IND | JK | 20 | 800 |
| IND | MH | 25 | 1000 |
| IND | WB | 10 | 900 |

Return table,

| COUNTRY | STATE | COUNT | TOTAL |
|---------|-------|-------|-------|
| USA | CA | 5 | 500 |
| USA | WA | 10 | 900 |
| IND | JK | 20 | 800 |
| IND | MH | 25 | 1000 |
| IND | WB | 10 | 900 |

VALUES

10/31/2022 • 3 minutes to read • [Edit Online](#)

When the input parameter is a column name, returns a one-column table that contains the distinct values from the specified column. Duplicate values are removed and only unique values are returned. A BLANK value can be added. When the input parameter is a table name, returns the rows from the specified table. Duplicate rows are preserved. A BLANK row can be added.

NOTE

This function cannot be used to Return values into a cell or column on a worksheet; rather, you use it as an intermediate function, nested in a formula, to get a list of distinct values that can be counted or used to filter or sum other values.

Syntax

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

Parameters

| TERM | DEFINITION |
|-------------------------|--|
| TableName or ColumnName | A column from which unique values are to be returned, or a table from which rows are to be returned. |

Return value

When the input parameter is a column name, a single column table. When the input parameter is a table name, a table of the same columns is returned.

Remarks

- When you use the VALUES function in a context that has been filtered, the unique values returned by VALUES are affected by the filter. For example, if you filter by Region, and return a list of the values for City, the list will include only those cities in the regions permitted by the filter. To return all of the cities, regardless of existing filters, you must use the ALL function to remove filters from the table. The second example demonstrates use of ALL with VALUES.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.
- For best practices when using VALUES, see [Use SELECTEDVALUE instead of VALUES](#).

Related functions

In most scenarios, when the argument is a column name, the results of the VALUES function are identical to those of the DISTINCT function. Both functions remove duplicates and return a list of the possible values in the specified column. However, the VALUES function can also return a blank value. This blank value is useful in cases where you are looking up distinct values from a related table, but a value used in the relationship is missing from one table. In database terminology, this is termed a violation of referential integrity. Such mismatches in

data can occur when one table is being updated and the related table is not.

When the argument is a table name, the result of the VALUES function returns all rows in the specified table plus a blank row, if there is a violation of referential integrity. The DISTINCT function removes duplicate rows and returns unique rows in the specified table.

NOTE

The DISTINCT function allows a column name or any valid table expression to be its argument but the VALUES function only accepts a column name or a table name as the argument.

The following table summarizes the mismatch between data that can occur in two related tables when referential integrity is not preserved.

| MYORDERS TABLE | MYSALES TABLE |
|------------------------------------|---------------|
| June 1 | June 1 sales |
| June 2 | June 2 sales |
| (no order dates have been entered) | June 3 sales |

If you use the DISTINCT function to return a list of dates, only two dates would be returned. However, if you use the VALUES function, the function returns the two dates plus an additional blank member. Also, any row from the MySales table that does not have a matching date in the MyOrders table will be "matched" to this unknown member.

Example

The following formula counts the number of unique invoices (sales orders), and produces the following results when used in a report that includes the Product Category Names:

```
= COUNTROWS(VALUES('InternetSales_USD'[SalesOrderNumber]))
```

Returns

| ROW LABELS | COUNT INVOICES |
|-------------|----------------|
| Accessories | 18,208 |
| Bikes | 15,205 |
| Clothing | 7,461 |
| Grand Total | 27,659 |

See also

- [FILTER function](#)
- [COUNTROWS function](#)
- [Filter functions](#)

Text functions

10/31/2022 • 2 minutes to read • [Edit Online](#)

Data Analysis Expressions (DAX) includes a set of text functions based on the library of string functions in Excel, but which have been modified to work with tables and columns in tabular models. This section describes text functions available in the DAX language.

In this category

| FUNCTION | DESCRIPTION |
|-------------------------------|---|
| COMBINEVALUES | Joins two or more text strings into one text string. |
| CONCATENATE | Joins two text strings into one text string. |
| CONCATENATEX | Concatenates the result of an expression evaluated for each row in a table. |
| EXACT | Compares two text strings and returns TRUE if they are exactly the same, FALSE otherwise. |
| FIND | Returns the starting position of one text string within another text string. |
| FIXED | Rounds a number to the specified number of decimals and returns the result as text. |
| FORMAT | Converts a value to text according to the specified format. |
| LEFT | Returns the specified number of characters from the start of a text string. |
| LEN | Returns the number of characters in a text string. |
| LOWER | Converts all letters in a text string to lowercase. |
| MID | Returns a string of characters from the middle of a text string, given a starting position and length. |
| REPLACE | REPLACE replaces part of a text string, based on the number of characters you specify, with a different text string. |
| REPT | Repeats text a given number of times. |
| RIGHT | RIGHT returns the last character or characters in a text string, based on the number of characters you specify. |
| SEARCH | Returns the number of the character at which a specific character or text string is first found, reading left to right. |

| FUNCTION | DESCRIPTION |
|------------|---|
| SUBSTITUTE | Replaces existing text with new text in a text string. |
| TRIM | Removes all spaces from text except for single spaces between words. |
| UNICHAR | Returns the Unicode character referenced by the numeric value. |
| UNICODE | Returns the numeric code corresponding to the first character of the text string. |
| UPPER | Converts a text string to all uppercase letters. |
| VALUE | Converts a text string that represents a number to a number. |

COMBINEVALUES

10/31/2022 • 2 minutes to read • [Edit Online](#)

Joins two or more text strings into one text string. The primary purpose of this function is to support multi-column relationships in DirectQuery models. See **Remarks** for details.

Syntax

```
COMBINEVALUES(<delimiter>, <expression>, <expression>[, <expression>]...)
```

Parameters

| TERM | DEFINITION |
|------------|---|
| delimiter | A separator to use during concatenation. Must be a constant value. |
| expression | A DAX expression whose value will be be joined into a single text string. |

Return value

A concatenated string.

Remarks

- The COMBINEVALUES function assumes, but does not validate, that when the input values are different, the output strings are also different. Based on this assumption, when COMBINEVALUES is used to create calculated columns in order to build a relationship that joins multiple columns from two DirectQuery tables, an optimized join condition is generated at query time. For example, if users want to create a relationship between Table1(Column1, Column2) and Table2(Column1, Column2), they can create two calculated columns, one on each table, as:

```
Table1[CalcColumn] = COMBINEVALUES(",", Table1[Column1], Table1[Column2])
```

and

```
Table2[CalcColumn] = COMBINEVALUES(",", Table2[Column1], Table2[Column2])
```

And then create a relationship between `Table1[CalcColumn]` and `Table2[CalcColumn]`. Unlike other DAX functions and operators, which are translated literally to the corresponding SQL operators and functions, the above relationship generates a SQL join predicate as:

```
(Table1.Column1 = Table2.Column1 OR Table1.Column1 IS NULL AND Table2.Column1 IS NULL)
```

and

```
(Table1.Column2 = Table2.Column2 OR Table1.Column2 IS NULL AND Table2.Column2 IS NULL)
```

- The join predicate can potentially deliver much better query performance than one that involves complex SQL operators and functions.
- The COMBINEVALUES function relies on users to choose the appropriate delimiter to ensure that unique combinations of input values produce distinct output strings but it does not validate that the assumption is true. For example, if users choose "|" as the delimiter, but one row in Table1 has Table1[Column1] = " | " and Table2 [Column2] = " " , while one row in Table2 has Table2[Column1] = " " and Table2[Column2] = " | " , the two concatenated outputs will be the same " | | " , which seem to indicate that the two rows are a match in the join operation. The two rows are not joined together if both tables are from the same DirectQuery source although they are joined together if both tables are imported.

Example

The following DAX query:

```
EVALUATE  
DISTINCT (  
    SELECTCOLUMNS ( Date, "Month", COMBINEVALUES ( " , ", [MonthName], [CalendarYear] ) )  
)
```

Returns the following single column table:

| [MONTH] |
|-----------------|
| January, 2020 |
| February, 2020 |
| March, 2020 |
| April, 2020 |
| May, 2020 |
| June, 2020 |
| July, 2020 |
| August, 2020 |
| September, 2020 |
| October, 2020 |
| November, 2020 |
| December, 2020 |
| January, 2021 |

| [MONTH] |
|-----------------|
| January, 2021 |
| February, 2021 |
| March, 2021 |
| April, 2021 |
| May, 2021 |
| June, 2021 |
| July, 2021 |
| August, 2021 |
| September, 2021 |
| October, 2021 |
| November, 2021 |
| December, 2021 |

CONCATENATE

10/31/2022 • 2 minutes to read • [Edit Online](#)

Joins two text strings into one text string.

Syntax

```
CONCATENATE(<text1>, <text2>)
```

Parameters

| TERM | DEFINITION |
|-------|--|
| text1 | The first text string to be joined into a single text string. The string can include text or numbers. You can also use column references. |
| text2 | The second text string to be joined into a single text string. The string can include text or numbers. You can also use column references. |

Return value

A concatenated string.

Remarks

- The CONCATENATE function joins two text strings into one text string. The joined items can be text, numbers, Boolean values represented as text, or a combination of those items. You can also use a column reference if the column contains appropriate values.
- The CONCATENATE function in DAX accepts only two arguments, whereas the Excel CONCATENATE function accepts up to 255 arguments. If you need to concatenate multiple columns, you can create a series of calculations or use the concatenation operator (&) to join all of them in a simpler expression.
- If you want to use text strings directly, rather than using a column reference, you must enclose each string in double quotation marks.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example: Concatenation of Literals

The sample formula creates a new string value by combining two string values that you provide as arguments.

```
= CONCATENATE("Hello ", "World")
```

Example: Concatenation of strings in columns

The sample formula returns the customer's full name as listed in a phone book. Note how a nested function is

used as the second argument. This is one way to concatenate multiple strings when you have more than two values that you want to use as arguments.

```
= CONCATENATE(Customer[LastName], CONCATENATE(", ", Customer[FirstName]))
```

Example: Conditional concatenation of strings in columns

The sample formula creates a new calculated column in the Customer table with the full customer name as a combination of first name, middle initial, and last name. If there is no middle name, the last name comes directly after the first name. If there is a middle name, only the first letter of the middle name is used and the initial letter is followed by a period.

```
= CONCATENATE( [FirstName]&" ", CONCATENATE( IF( LEN([MiddleName])>1, LEFT([MiddleName],1)&" ", ""), [LastName]))
```

This formula uses nested CONCATENATE and IF functions, together with the ampersand (&) operator, to conditionally concatenate three string values and add spaces as separators.

Example: Concatenation of columns with different data types

The following example demonstrates how to concatenate values in columns that have different data types. If the value that you are concatenating is numeric, the value will be implicitly converted to text. If both values are numeric, both values will be cast to text and concatenated as if they were strings.

| PRODUCT DESCRIPTION | PRODUCT ABBREVIATION (COLUMN 1 OF COMPOSITE KEY) | PRODUCT NUMBER (COLUMN 2 OF COMPOSITE KEY) | NEW GENERATED KEY COLUMN |
|---------------------|--|--|-----------------------------|
| Mountain bike | MTN | 40 | MTN40 |
| Mountain bike | MTN | 42 | MTN42 |

```
= CONCATENATE('Products'[Product abbreviation], 'Products'[Product number])
```

The CONCATENATE function in DAX accepts only two arguments, whereas the Excel CONCATENATE function accepts up to 255 arguments. If you need to add more arguments, you can use the ampersand (&) operator. For example, the following formula produces the results, MTN-40 and MTN-42.

```
= [Product abbreviation] & "-" & [Product number]
```

See also

[CONCATENATEX](#)

CONCATENATEX

10/31/2022 • 2 minutes to read • [Edit Online](#)

Concatenates the result of an expression evaluated for each row in a table.

Syntax

```
CONCATENATEX(<table>, <expression>[, <delimiter> [, <orderBy_expression> [, <order>]]...])
```

Parameters

| TERM | DEFINITION |
|--------------------|---|
| table | The table containing the rows for which the expression will be evaluated. |
| expression | The expression to be evaluated for each row of <i>table</i> . |
| delimiter | (Optional) A separator to use during concatenation. |
| orderBy_expression | (Optional) Any DAX expression where the result value is used to sort the concatenated values in the output string. It is evaluated for each row of <i>table</i> . |
| order | (Optional) A value that specifies how to sort <i>orderBy_expression</i> values, ascending or descending. |

The optional **order** parameter accepts the following values:

| VALUE | ALTERNATE VALUES | DESCRIPTION |
|----------|------------------|--|
| 0 (zero) | FALSE, DESC | Sorts in descending order of values of <i>orderBy_expression</i> . This is the default value when the <i>order</i> parameter is omitted. |
| 1 | TRUE, ASC | Sorts in ascending order of values of <i>orderBy_expression</i> . |

Return value

A concatenated string.

Remarks

- This function takes as its first argument a table or an expression that returns a table. The second argument is a column that contains the values you want to concatenate, or an expression that returns a value.
- Concatenated values are not necessarily sorted in any particular order, unless *orderBy_expression* is

specified.

- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

Employees table

| FIRSTNAME | LASTNAME |
|-----------|----------|
| Alan | Brewer |
| Michael | Blythe |

The following formula:

```
= CONCATENATEX(Employees, [FirstName] & " " & [LastName], ",")
```

Returns:

"Alan Brewer, Michael Blythe"

EXACT

10/31/2022 • 2 minutes to read • [Edit Online](#)

Compares two text strings and returns TRUE if they are exactly the same, otherwise returns FALSE. EXACT is case-sensitive but ignores formatting differences. EXACT is case-sensitive

Syntax

```
EXACT(<text1>,<text2>)
```

Parameters

| TERM | DEFINITION |
|-------|--|
| text1 | The first text string or column that contains text. |
| text2 | The second text string or column that contains text. |

Return value

True or False. (Boolean)

Example

The following formula used in a calculated column in the Product table checks the value of Product for the current row against the value of Model for the current row, and returns True if they are the same, and returns False if they are different.

Examples in this article can be used with the sample Adventure Works DW 2020 Power BI Desktop model. To get the model, see [DAX sample model](#).

```
= EXACT([Product], [Model])
```

See also

[Text functions](#)

FIND

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the starting position of one text string within another text string. FIND is case-sensitive.

Syntax

```
FIND(<find_text>, <within_text>[, [<start_num>][, <NotFoundValue>]])
```

Parameters

| TERM | DEFINITION |
|---------------|--|
| find_text | The text you want to find. Use double quotes (empty text) to match the first character in within_text . |
| within_text | The text containing the text you want to find. |
| start_num | (optional) The character at which to start the search; if omitted, start_num = 1. The first character in within_text is character number 1. |
| NotFoundValue | (optional, but strongly recommended) The value that should be returned when the operation does not find a matching substring, typically 0, -1, or BLANK(). If not specified, an error is returned. |

Return value

Number that shows the starting point of the text string you want to find.

Remarks

- Whereas Microsoft Excel has multiple versions of the FIND function to accommodate single-byte character set (SBCS) and double-byte character set (DBCS) languages, DAX uses Unicode and counts each character the same way; therefore, you do not need to use a different version depending on the character type.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.
- FIND does not support wildcards. To use wildcards, use [SEARCH](#).

Example

The following DAX query finds the position of the first letter of "Bike", in the string that contains the reseller name. If not found, Blank is returned.

Keep in mind, FIND is case-sensitive. In this example, if "bike" were used in the <find_text> argument, no results would be returned. Use [SEARCH](#) for case-insensitive.

Examples in this article can be used with the sample Adventure Works DW 2020 Power BI Desktop model. To get the model, see [DAX sample model](#).

```
EVALUATE
CALCULATETABLE (
    ADDCOLUMNS (
        TOPN ( 10, SUMMARIZE('Reseller', [Reseller], [Business Type])),
        "Position of Bike", FIND ( "Bike", 'Reseller'[Reseller], 1, BLANK () )
    ),
    'Reseller'[Business Type] IN { "Specialty Bike Shop", "Value Added Reseller", "Warehouse"}
)
```

Returns,

| RESELLER | BUSINESS TYPE | POSITION OF BIKE |
|-----------------------|----------------------|------------------|
| Volume Bike Sellers | Warehouse | 8 |
| Mass Market Bikes | Value Added Reseller | 13 |
| Twin Cycles | Value Added Reseller | |
| Rich Department Store | Warehouse | |
| Rental Gallery | Specialty Bike Shop | |
| Budget Toy Store | Warehouse | |
| Global Sports Outlet | Warehouse | |
| Online Bike Catalog | Warehouse | 8 |
| Helmets and Cycles | Value Added Reseller | |
| Jumbo Bikes | Specialty Bike Shop | 7 |

See also

[SEARCH](#)

[Text functions](#)

FIXED

10/31/2022 • 2 minutes to read • [Edit Online](#)

Rounds a number to the specified number of decimals and returns the result as text. You can specify that the result be returned with or without commas.

Syntax

```
FIXED(<number>, <decimals>, <no_commas>)
```

Parameters

| TERM | DEFINITION |
|-----------|---|
| number | The number you want to round and convert to text, or a column containing a number. |
| decimals | (optional) The number of digits to the right of the decimal point; if omitted, 2. |
| no_commas | (optional) A logical value: if 1, do not display commas in the returned text; if 0 or omitted, display commas in the returned text. |

Return value

A number represented as text.

Remarks

- If the value used for the **decimals** parameter is negative, **number** is rounded to the left of the decimal point.
- If you omit **decimals**, it is assumed to be 2.
- If **no_commas** is 0 or is omitted, then the returned text includes commas as usual.
- The major difference between formatting a cell containing a number by using a command and formatting a number directly with the FIXED function is that FIXED converts its result to text. A number formatted with a command from the formatting menu is still a number.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following formula used in a calculated column gets the numeric value for the current row in Product[List Price] and returns it as text with 2 decimal places and no commas.

Examples in this article can be used with the sample Adventure Works DW 2020 Power BI Desktop model. To get the model, see [DAX sample model](#).

```
= FIXED([List Price],2,1)
```

See also

[CEILING](#)

[FLOOR](#)

[ISO.CEILING](#)

[MROUND](#)

[ROUND](#)

[ROUNDDOWN](#)

[ROUNDUP](#)

FORMAT

10/31/2022 • 13 minutes to read • [Edit Online](#)

Converts a value to text according to the specified format.

Syntax

```
FORMAT(<value>, <format_string>[, <locale_name>])
```

Parameters

| TERM | DEFINITION |
|---------------|---|
| value | A value or expression that evaluates to a single value. |
| format_string | A string with the formatting template. |
| locale_name | (Optional) Name of the locale to be used by the function. Possible values are strings accepted by the Windows API function LocaleNameToLCID() . |

Return value

A string containing **value** formatted as defined by **format_string**.

NOTE

If **value** is BLANK, the function returns an empty string.

If **format_string** is BLANK, the value is formatted with a "General Number" or "General Date" format (according to **value** data type).

Remarks

- Predefined format strings use the model culture property when formatting the result. By default, the model culture property is set according to the user locale of the computer. For new Power BI Desktop models, the model culture property can be changed in Options > Regional Settings > Model language. For Analysis Services, model culture is set according to the Language property initially defined by the instance.
- The format strings supported as an argument to the DAX FORMAT function are based on the format strings used by Visual Basic (OLE Automation), not on the format strings used by the .NET Framework. Therefore, you might get unexpected results or an error if the argument doesn't match any defined format strings. For example, "p" as an abbreviation for "Percent" isn't supported. Strings that you provide as an argument to the FORMAT function that aren't included in the list of predefined format strings are handled as part of a custom format string, or as a string literal.
- To learn more specifying a locale with FORMAT, check out this [video](#).
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-

level security (RLS) rules.

Examples

Format strings

```
= FORMAT( 12345.67, "General Number")
= FORMAT( 12345.67, "Currency")
= FORMAT( 12345.67, "Fixed")
= FORMAT( 12345.67, "Standard")
= FORMAT( 12345.67, "Percent")
= FORMAT( 12345.67, "Scientific")
```

Returns:

12345.67 "General Number" displays the number with no formatting.

\$12,345.67 "Currency" displays the number with your currency locale formatting. The sample here shows the default United States currency formatting.

12345.67 "Fixed" displays at least one digit to the left of the decimal separator and two digits to the right of the decimal separator.

12,345.67 "Standard" displays at least one digit to the left of the decimal separator and two digits to the right of the decimal separator, and includes thousand separators. The sample here shows the default United States number formatting.

1,234,567.00 % "Percent" displays the number as a percentage (multiplied by 100) with formatting and the percent sign at the right of the number separated by a single space.

1.23E+04 "Scientific" displays the number in scientific notation with two decimal digits.

Datetime with optional locale_name

```
= FORMAT( dt"2020-12-15T12:30:59", BLANK(), "en-US" )
= FORMAT( dt"2020-12-15T12:30:59", BLANK(), "en-GB" )
= FORMAT( dt"2020-12-15T12:30:59", "mm/dd/yyyy", "en-GB" )
```

Returns:

12/15/2020 12:30:59 PM Where month precedes day and time is 12-hour format.

15/12/2020 12:30:59 Where day precedes month and time is 24-hour format.

12/15/2020 12:30:59 Where month precedes day and time is 24-hour format. Because a non-locale dependent format string is specified, the locale is not applied and the non-locale format is returned.

Predefined numeric formats

The following predefined numeric formats can be specified in the **format_string** argument:

| FORMAT | DESCRIPTION |
|------------------|---|
| "General Number" | Displays number with no thousand separators. |
| "Currency" | Displays number with thousand separators, if appropriate; displays two digits to the right of the decimal separator. Output is based on system locale settings. |

| FORMAT | DESCRIPTION |
|--------------|--|
| "Fixed" | Displays at least one digit to the left and two digits to the right of the decimal separator. |
| "Standard" | Displays number with thousand separators, at least one digit to the left and two digits to the right of the decimal separator. |
| "Percent" | Displays number multiplied by 100 with a percent sign (%) appended immediately to the right; always displays two digits to the right of the decimal separator. |
| "Scientific" | Uses standard scientific notation, providing two significant digits. |
| "Yes/No" | Displays No if number is 0; otherwise, displays Yes. |
| "True/False" | Displays False if number is 0; otherwise, displays True. |
| "On/Off" | Displays Off if number is 0; otherwise, displays On. |

Custom numeric formats

A custom format expression for numbers can have from one to three sections separated by semicolons. If the format string argument contains one of the named numeric formats, only one section is allowed.

| IF YOU USE | THE RESULT IS |
|------------------|--|
| One section only | The format expression applies to all values. |
| Two sections | The first section applies to positive values and zeros, the second to negative values. |
| Three sections | The first section applies to positive values, the second to negative values, and the third to zeros. |

"\$#,##0;(\$#,##0)"

If you include semicolons with nothing between them, the missing section is defined using the format of the positive value. For example, the following format displays positive and negative values using the format in the first section and displays "Zero" if the value is zero.

"\$#,##0"

If you include semicolons with nothing between them, the missing section is shown using the format of the positive value.

Custom numeric format characters

The following custom numeric format characters can be specified in the **format_string** argument:

| CHARACTER | DESCRIPTION |
|-----------|--|
| None | Display the number with no formatting. |
| (0) | Digit placeholder. Display a digit or a zero. If the expression has a digit in the position where the 0 appears in the format string, display it; otherwise, display a zero in that position. If the number has fewer digits than there are zeros (on either side of the decimal) in the format expression, display leading or trailing zeros. If the number has more digits to the right of the decimal separator than there are zeros to the right of the decimal separator in the format expression, round the number to as many decimal places as there are zeros. If the number has more digits to the left of the decimal separator than there are zeros to the left of the decimal separator in the format expression, display the extra digits without modification. |
| (#) | Digit placeholder. Display a digit or nothing. If the expression has a digit in the position where the # appears in the format string, display it; otherwise, display nothing in that position. This symbol works like the 0 digit placeholder, except that leading and trailing zeros aren't displayed if the number has the same or fewer digits than there are # characters on either side of the decimal separator in the format expression. |
| (.) | Decimal placeholder. In some locales, a comma is used as the decimal separator. The decimal placeholder determines how many digits are displayed to the left and right of the decimal separator. If the format expression contains only number signs to the left of this symbol, numbers smaller than 1 begin with a decimal separator. To display a leading zero displayed with fractional numbers, use 0 as the first digit placeholder to the left of the decimal separator. The actual character used as a decimal placeholder in the formatted output depends on the Number Format recognized by your system. |
| (%) | Percentage placeholder. The expression is multiplied by 100. The percent character (%) is inserted in the position where it appears in the format string. |
| (,) | Thousand separator. In some locales, a period is used as a thousand separator. The thousand separator separates thousands from hundreds within a number that has four or more places to the left of the decimal separator. Standard use of the thousand separator is specified if the format contains a thousand separator surrounded by digit placeholders (0 or #). Two adjacent thousand separators or a thousand separator immediately to the left of the decimal separator (whether or not a decimal is specified) means "scale the number by dividing it by 1000, rounding as needed." For example, you can use the format string "##0,," to represent 100 million as 100. Numbers smaller than 1 million are displayed as 0. Two adjacent thousand separators in any position other than immediately to the left of the decimal separator are treated simply as specifying the use of a thousand separator. The actual character used as the thousand separator in the formatted output depends on the Number Format recognized by your system. |

| CHARACTER | DESCRIPTION |
|---------------|---|
| (:) | Time separator. In some locales, other characters may be used to represent the time separator. The time separator separates hours, minutes, and seconds when time values are formatted. The actual character used as the time separator in formatted output is determined by your system settings. |
| (/) | Date separator. In some locales, other characters may be used to represent the date separator. The date separator separates the day, month, and year when date values are formatted. The actual character used as the date separator in formatted output is determined by your system settings. |
| (E- E+ e- e+) | Scientific format. If the format expression contains at least one digit placeholder (0 or #) to the right of E-, E+, e-, or e+, the number is displayed in scientific format and E or e is inserted between the number and its exponent. The number of digit placeholders to the right determines the number of digits in the exponent. Use E- or e- to place a minus sign next to negative exponents. Use E+ or e+ to place a minus sign next to negative exponents and a plus sign next to positive exponents. |
| - + \$ () | Display a literal character. To display a character other than one of those listed, precede it with a backslash (\) or enclose it in double quotation marks (" "). |
| (\) | Display the next character in the format string. To display a character that has special meaning as a literal character, precede it with a backslash (\). The backslash itself isn't displayed. Using a backslash is the same as enclosing the next character in double quotation marks. To display a backslash, use two backslashes (\). Examples of characters that can't be displayed as literal characters are the date-formatting and time-formatting characters (a, c, d, h, m, n, p, q, s, t, w, y, /, and :), the numeric-formatting characters (#, 0, %, E, e, comma, and period), and the string-formatting characters (@, &, <, >, and !). |
| ("ABC") | Display the string inside the double quotation marks (" "). |

Predefined date/time formats

The following predefined date/time formats can be specified in the **format_string** argument. When using formats other than these, they are interpreted as a custom date/time format:

| FORMAT | DESCRIPTION |
|------------------------------|--|
| "General Date" | Displays a date and/or time. For example, 3/12/2008 11:07:31 AM. Date display is determined by your application's current culture value. |
| "Long Date" or "Medium Date" | Displays a date according to your current culture's long date format. For example, Wednesday, March 12, 2008. |

| FORMAT | DESCRIPTION |
|----------------|--|
| "Short Date" | Displays a date using your current culture's short date format. For example, 3/12/2008. |
| "Long Time" or | Displays a time using your current culture's long time format; typically includes hours, minutes, seconds. For example, 11:07:31 AM. |
| "Medium Time" | Displays a time in 12 hour format. For example, 11:07 AM. |
| "Short Time" | Displays a time in 24 hour format. For example, 11:07. |

Custom date/time formats

The following format characters can be specified in the **format_string** to create custom date/time formats:

| CHARACTER | DESCRIPTION |
|-----------|--|
| (:) | Time separator. In some locales, other characters may be used to represent the time separator. The time separator separates hours, minutes, and seconds when time values are formatted. The actual character used as the time separator in formatted output is determined by your system settings. |
| (/) | Date separator. In some locales, other characters may be used to represent the date separator. The date separator separates the day, month, and year when date values are formatted. The actual character used as the date separator in formatted output is determined by your system settings. |
| (\) | Backslash. Displays the next character as a literal character. So, it's not interpreted as a formatting character. |
| (") | Double quote. Text enclosed within double quotes is displayed. So, it's not interpreted as formatting characters. |
| c | Display the date as <code>dddd</code> and display the time as <code>ttttt</code> , in that order. Display only date information if there is no fractional part to the date serial number; display only time information if there is no integer portion. |
| d | Display the day as a number without a leading zero (1-31). |
| dd | Display the day as a number with a leading zero (01-31). |
| ddd | Display the day as an abbreviation (Sun-Sat). Localized. |
| dddd | Display the day as a full name (Sunday-Saturday). Localized. |
| dddddd | Display the date as a complete date (including day, month, and year), formatted according to your system's short date format setting. The default short date format is <code>mm/dd/yyyy</code> . |

| CHARACTER | DESCRIPTION |
|-----------|---|
| dddddd | Display a date serial number as a complete date (including day, month, and year) formatted according to the long date setting recognized by your system. The default long date format is <code>dddd, mmmm d, yyyy</code> . |
| w | Display the day of the week as a number (1 for Sunday through 7 for Saturday). |
| ww | Display the week of the year as a number (1-54). |
| m | Display the month as a number without a leading zero (1-12). If <code>m</code> immediately follows <code>h</code> or <code>hh</code> , minute rather than the month is displayed. |
| mm | Display the month as a number with a leading zero (01-12). If <code>mm</code> immediately follows <code>h</code> or <code>hh</code> , minute rather than the month is displayed. |
| mmm | Display the month as an abbreviation (Jan-Dec). Localized. |
| mmm | Display the month as a full month name (January-December). Localized. |
| q | Display the quarter of the year as a number (1-4). |
| y | Display the day of the year as a number (1-366). |
| yy | Display the year as a 2-digit number (00-99). |
| yyyy | Display the year as a 4-digit number (100-9999). |
| h | Display the hour as a number without a leading zero (0-23). |
| hh | Display the hour as a number with a leading zero (00-23). |
| n | Display the minute as a number without a leading zero (0-59). |
| nn | Display the minute as a number with a leading zero (00-59). |
| s | Display the second as a number without a leading zero (0-59). |
| ss | Display the second as a number with a leading zero (00-59). |
| tttt | Display a time as a complete time (including hour, minute, and second), formatted using the time separator defined by the time format recognized by your system. A leading zero is displayed if the leading zero option is selected and the time is before 10:00 A.M. or P.M. The default time format is <code>h:mm:ss</code> . |

| CHARACTER | DESCRIPTION |
|-----------|--|
| AM/PM | Use the 12-hour clock and display an uppercase AM with any hour before noon; display an uppercase PM with any hour between noon and 11:59 P.M. |
| am/pm | Use the 12-hour clock and display a lowercase AM with any hour before noon; display a lowercase PM with any hour between noon and 11:59 P.M. |
| A/P | Use the 12-hour clock and display an uppercase A with any hour before noon; display an uppercase P with any hour between noon and 11:59 P.M. |
| a/p | Use the 12-hour clock and display a lowercase A with any hour before noon; display a lowercase P with any hour between noon and 11:59 P.M. |
| AMPM | Use the 12-hour clock and display the AM string literal as defined by your system with any hour before noon; display the PM string literal as defined by your system with any hour between noon and 11:59 P.M. AMPM can be either uppercase or lowercase, but the case of the string displayed matches the string as defined by your system settings. The default format is AM/PM. If your system is set to 24-hour clock, the string is typically set to an empty string. |

Date/time formatting uses the current user locale to format the string. For example, consider the date June 25, 2020. When it's formatted using format string "m/d/yyyy" it will be:

- User locale is United States of America (en-US): "6/25/2020"
- User locale is Germany (de-DE): "6.25.2020"

Custom date/time format examples

The following examples use the date/time Thursday, June 25, 2020, at 1:23:45 PM. Germany (de-DE) uses a 24-hour system. There's no equivalent of AM/PM.

| FORMAT | RESULT (EN-US) | RESULT (DE-DE) |
|----------|-------------------------|---------------------------|
| "c" | 06/25/2020 13:23:45 | 25.06.2020 13:23:45 |
| "d" | 25 | 25 |
| "dd" | 25 | 25 |
| "ddd" | Thu | Do |
| "dddd" | Thursday | Donnerstag |
| "ddddd" | 06/25/2020 | 25.06.2020 |
| "dddddd" | Thursday, June 25, 2020 | Donnerstag, 25. Juni 2020 |
| "w" | 5 | 5 |

| FORMAT | RESULT (EN-US) | RESULT (DE-DE) |
|----------------------------|------------------------|--------------------|
| "ww" | 26 | 26 |
| "m" | 6 | 6 |
| "mm" | 06 | 06 |
| "mmm" | Jun | Jun |
| "mmm" | June | Juni |
| "q" | 2 | 2 |
| "y" | 177 | 177 |
| "yy" | 20 | 20 |
| "yyyy" | 2020 | 2020 |
| ""Year"" yyyy" | Year 2020 | Year 2020 |
| "yyyy \Qq" | 2020 Q2 | 2020 Q2 |
| "dd/mm/yyyy" | 25/06/2020 | 25.06.2020 |
| "mm/dd/yyyy" | 06/25/2020 | 06.25.2020 |
| "h:nn:ss" | 13:23:45 | 13:23:45 |
| "h:nn:ss AMPM" | 1:23:45 PM | 1:23:45 |
| "hh:nn:ss" | 13:23:45 | 13:23:45 |
| "hh:nn:ss AMPM" | 01:23:45 PM | 01:23:45 |
| "ttttt" | 13:23:45 | 13:23:45 |
| "ttttt AMPM" | 13:23:45 PM | 13:23:45 |
| "mm/dd/yyyy hh:nn:ss AMPM" | 06/25/2020 01:23:45 PM | 6.25.2020 01:23:45 |

LEFT

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the specified number of characters from the start of a text string.

Syntax

```
LEFT(<text>, <num_chars>)
```

Parameters

| TERM | DEFINITION |
|-----------|---|
| text | The text string containing the characters you want to extract, or a reference to a column that contains text. |
| num_chars | (optional) The number of characters you want LEFT to extract; if omitted, 1. |

Return value

A text string.

Remarks

- Whereas Microsoft Excel contains different functions for working with text in single-byte and double-byte character languages, DAX works with Unicode and stores all characters as the same length; therefore, a single function is enough.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following example returns the first five characters of the company name in the column [ResellerName] and the first five letters of the geographical code in the column [GeographyKey] and concatenates them, to create an identifier.

```
= CONCATENATE(LEFT('Reseller'[ResellerName],LEFT(GeographyKey,3))
```

If the **num_chars** argument is a number that is larger than the number of characters available, the function returns the maximum characters available and does not raise an error. For example, the column [GeographyKey] contains numbers such as 1, 12 and 311; therefore the result also has variable length.

See also

[Text functions](#)

LEN

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the number of characters in a text string.

Syntax

```
LEN(<text>)
```

Parameters

| TERM | DEFINITION |
|------|---|
| text | The text whose length you want to find, or a column that contains text. Spaces count as characters. |

Return value

A whole number indicating the number of characters in the text string.

Remarks

- Whereas Microsoft Excel has different functions for working with single-byte and double-byte character languages, DAX uses Unicode and stores all characters with the same length.
- LEN always counts each character as 1, no matter what the default language setting is.
- If you use LEN with a column that contains non-text values, such as dates or Booleans, the function implicitly casts the value to text, using the current column format.

Example

The following formula sums the lengths of addresses in the columns, [AddressLine1] and [AddressLine2].

```
= LEN([AddressLine1])+LEN([AddressLine2])
```

LOWER

10/31/2022 • 2 minutes to read • [Edit Online](#)

Converts all letters in a text string to lowercase.

Syntax

```
LOWER(<text>)
```

Parameters

| TERM | DEFINITION |
|------|---|
| text | The text you want to convert to lowercase, or a reference to a column that contains text. |

Return value

Text in lowercase.

Remarks

Characters that are not letters are not changed. For example, the formula `= LOWER("123ABC")` returns **123abc**.

Example

The following formula gets each row in the column, [ProductCode], and converts the value to all lowercase. Numbers in the column are not affected.

```
= LOWER('New Products'[ProductCode])
```

See also

[Text functions](#)

MID

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns a string of characters from the middle of a text string, given a starting position and length.

Syntax

```
MID(<text>, <start_num>, <num_chars>)
```

Parameters

| TERM | DEFINITION |
|-----------|--|
| text | The text string from which you want to extract the characters, or a column that contains text. |
| start_num | The position of the first character you want to extract. Positions start at 1. |
| num_chars | The number of characters to return. |

Return value

A string of text of the specified length.

Remarks

Whereas Microsoft Excel has different functions for working with single-byte and double-byte characters languages, DAX uses Unicode and stores all characters with the same length.

Examples

The following expression,

```
MID("abcde",2,3))
```

Returns **"bcd"**.

The following expression,

```
MID('Reseller'[ResellerName],1,5))
```

Returns the same result as `LEFT([ResellerName],5)`. Both expressions return the first 5 letters of column, `[ResellerName]`.

See also

[Text functions](#)

REPLACE

10/31/2022 • 2 minutes to read • [Edit Online](#)

REPLACE replaces part of a text string, based on the number of characters you specify, with a different text string.

Syntax

```
REPLACE(<old_text>, <start_num>, <num_chars>, <new_text>)
```

Parameters

| TERM | DEFINITION |
|-----------|--|
| old_text | The string of text that contains the characters you want to replace, or a reference to a column that contains text. |
| start_num | The position of the character in old_text that you want to replace with new_text . |
| num_chars | The number of characters that you want to replace. Warning: If the argument, <i>num_chars</i> , is a blank or references a column that evaluates to a blank, the string for <i>new_text</i> is inserted at the position, <i>start_num</i> , without replacing any characters. This is the same behavior as in Excel. |
| new_text | The replacement text for the specified characters in old_text . |

Return value

A text string.

Remarks

- Whereas Microsoft Excel has different functions for use with single-byte and double-byte character languages, DAX uses Unicode and therefore stores all characters as the same length.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following formula creates a new calculated column that replaces the first two characters of the product code in column, [ProductCode], with a new two-letter code, OB.

```
= REPLACE('New Products'[Product Code],1,2,"OB")
```

See also

[Text functions](#)

[SUBSTITUTE function](#)

REPT

10/31/2022 • 2 minutes to read • [Edit Online](#)

Repeats text a given number of times. Use REPT to fill a cell with a number of instances of a text string.

Syntax

```
REPT(<text>, <num_times>)
```

Parameters

| TERM | DEFINITION |
|-----------|--|
| text | The text you want to repeat. |
| num_times | A positive number specifying the number of times to repeat text. |

Return value

A string containing the changes.

Remarks

- If **number_times** is 0 (zero), REPT returns a blank.
- If **number_times** is not an integer, it is truncated.
- The result of the REPT function cannot be longer than 32,767 characters, or REPT returns an error.

Example: Repeating Literal Strings

The following example returns the string, 85, repeated three times.

```
= REPT("85",3)
```

Example: Repeating Column Values

The following example returns the string in the column, [MyText], repeated for the number of times in the column, [MyNumber]. Because the formula extends for the entire column, the resulting string depends on the text and number value in each row.

```
= REPT([MyText],[MyNumber])
```

| MYTEXT | MYNUMBER | CALCULATEDCOLUMN1 |
|--------|----------|-------------------|
| Text | 2 | TextText |

| MYTEXT | MYNUMBER | CALCULATEDCOLUMN1 |
|--------|----------|-------------------|
| Number | 0 | |
| 85 | 3 | 858585 |

See also

[Text functions](#)

RIGHT

10/31/2022 • 2 minutes to read • [Edit Online](#)

RIGHT returns the last character or characters in a text string, based on the number of characters you specify.

Syntax

```
RIGHT(<text>, <num_chars>)
```

Parameters

| TERM | DEFINITION |
|-----------|---|
| text | The text string that contains the characters you want to extract, or a reference to a column that contains text. |
| num_chars | (optional) The number of characters you want RIGHT to extract; is omitted, 1. You can also use a reference to a column that contains numbers. |

If the column reference does not contain text, it is implicitly cast as text.

Return value

A text string containing the specified right-most characters.

Remarks

- RIGHT always counts each character, whether single-byte or double-byte, as 1, no matter what the default language setting is.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example: Returning a Fixed Number of Characters

The following formula returns the last two digits of the product code in the New Products table.

```
= RIGHT('New Products'[ProductCode],2)
```

Example: Using a Column Reference to Specify Character Count

The following formula returns a variable number of digits from the product code in the New Products table, depending on the number in the column, MyCount. If there is no value in the column, MyCount, or the value is a blank, RIGHT also returns a blank.

```
= RIGHT('New Products'[ProductCode],[MyCount])
```

See also

[Text functions](#)

[LEFT](#)

[MID](#)

SEARCH

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the number of the character at which a specific character or text string is first found, reading left to right. Search is case-insensitive and accent sensitive.

Syntax

```
SEARCH(<find_text>, <within_text>[, [<start_num>][, <NotFoundValue>]])
```

Parameters

| TERM | DEFINITION |
|---------------|---|
| find_text | The text that you want to find. You can use wildcard characters — the question mark (?) and asterisk (*) — in find_text . A question mark matches any single character; an asterisk matches any sequence of characters. If you want to find an actual question mark or asterisk, type a tilde (~) before the character. |
| within_text | The text in which you want to search for find_text , or a column containing text. |
| start_num | (optional) The character position in within_text at which you want to start searching. If omitted, 1. |
| NotFoundValue | (optional, but strongly recommended) The value that should be returned when the operation does not find a matching substring, typically 0, -1, or BLANK(). If not specified, an error is returned. |

Return value

The number of the starting position of the first text string from the first character of the second text string.

Remarks

- The search function is case insensitive. Searching for "N" will find the first occurrence of 'N' or 'n'.
- The search function is accent sensitive. Searching for "á" will find the first occurrence of 'á' but no occurrences of 'a', 'à', or the capitalized versions 'A', 'Á'.
- You can use the SEARCH function to determine the location of a character or text string within another text string, and then use the MID function to return the text, or use the REPLACE function to change the text.
- If the **find_text** cannot be found in **within_text**, the formula returns an error. This behavior is like Excel, which returns #VALUE if the substring is not found. Nulls in **within_text** will be interpreted as an empty string in this context.

- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following DAX query finds the position of the first letter of "cycle", in the string that contains the reseller name. If not found, Blank is returned.

SEARCH is case-insensitive. In this example, if "cycle" or "Cycle" is used in the <find_text> argument, results are returned for either case. Use [FIND](#) for case-sensitive.

Examples in this article can be used with the sample Adventure Works DW 2020 Power BI Desktop model. To get the model, see [DAX sample model](#).

```
EVALUATE
CALCULATETABLE (
    ADDCOLUMNS (
        TOPN ( 10, SUMMARIZE('Reseller', [Reseller], [Business Type])),
        "Position of cycle", SEARCH ( "cycle", 'Reseller'[Reseller], 1, BLANK () )
    ),
    'Reseller'[Business Type] IN { "Specialty Bike Shop", "Value Added Reseller", "Warehouse"}
)
```

Returns,

| RESELLER | BUSINESS TYPE | POSITION OF CYCLE |
|-----------------------|----------------------|-------------------|
| Volume Bike Sellers | Warehouse | |
| Mass Market Bikes | Value Added Reseller | |
| Twin Cycles | Value Added Reseller | 6 |
| Rich Department Store | Warehouse | |
| Rental Gallery | Specialty Bike Shop | |
| Budget Toy Store | Warehouse | |
| Global Sports Outlet | Warehouse | |
| Online Bike Catalog | Warehouse | |
| Helmets and Cycles | Value Added Reseller | 13 |
| Jumbo Bikes | Specialty Bike Shop | |

See also

[FIND](#)

[REPLACE](#)

[Text functions](#)

SUBSTITUTE

10/31/2022 • 2 minutes to read • [Edit Online](#)

Replaces existing text with new text in a text string.

Syntax

```
SUBSTITUTE(<text>, <old_text>, <new_text>, <instance_num>)
```

Parameters

| TERM | DEFINITION |
|--------------|---|
| text | The text in which you want to substitute characters, or a reference to a column containing text. |
| old_text | The existing text that you want to replace. |
| new_text | The text you want to replace old_text with. |
| instance_num | (optional) The occurrence of old_text you want to replace. If omitted, every instance of old_text is replaced |

Return value

A string of text.

Remarks

- Use the SUBSTITUTE function when you want to replace specific text in a text string; use the REPLACE function when you want to replace any text of variable length that occurs in a specific location in a text string.
- The SUBSTITUTE function is case-sensitive. If case does not match between **text** and **old_text**, SUBSTITUTE will not replace the text.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example: Substitution within a String

The following formula creates a copy of the column [Product Code] that substitutes the new product code **NW** for the old product code **PA** wherever it occurs in the column.

```
= SUBSTITUTE([Product Code], "NW", "PA")
```

See also

[Text functions](#)

REPLACE

TRIM

10/31/2022 • 2 minutes to read • [Edit Online](#)

Removes all spaces from text except for single spaces between words.

Syntax

```
TRIM(<text>)
```

Parameters

| TERM | DEFINITION |
|------|--|
| text | The text from which you want spaces removed, or a column that contains text. |

Return value

The string with spaces removed.

Remarks

- Use TRIM on text that you have received from another application that may have irregular spacing.
- The TRIM function was originally designed to trim the 7-bit ASCII space character (value 32) from text. In the Unicode character set, there is an additional space character called the nonbreaking space character that has a decimal value of 160. This character is commonly used in Web pages as the HTML entity, . By itself, the TRIM function does not remove this nonbreaking space character. For an example of how to trim both space characters from text, see [Remove spaces and nonprinting characters from text](#).

Example

The following formula creates a new string that does not have trailing white space.

```
= TRIM("A column with trailing spaces.  ")
```

When you create the formula, the formula is propagated through the row just as you typed it, so that you see the original string in each formula and the results are not apparent. However, when the formula is evaluated the string is trimmed.

You can verify that the formula produces the correct result by checking the length of the calculated column created by the previous formula, as follows:

```
= LEN([Calculated Column 1])
```

See also

[Text functions](#)

UNICHAR

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the Unicode character referenced by the numeric value.

Syntax

```
UNICHAR(number)
```

Parameters

| TERM | DEFINITION |
|--------|---|
| number | The Unicode number that represents the character. |

Return value

A character represented by the Unicode number.

Remarks

- If XML characters are not invalid, UNICHAR returns an error.
- If Unicode numbers are partial surrogates and data types are not valid, UNICHAR returns an error.
- If numbers are numeric values that fall outside the allowable range, UNICHAR returns an error.
- If number is zero (0), UNICHAR returns an error.
- The Unicode character returned can be a string of characters, for example in UTF-8 or UTF-16 codes.

Example

The following example returns the character represented by the Unicode number 66 (uppercase A).

```
= UNICHAR(65)
```

The following example returns the character represented by the Unicode number 32 (space character).

```
= UNICHAR(32)
```

The following example returns the character represented by the Unicode number 9733 (★ character).

```
= UNICHAR(9733)
```

UNICODE

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the number (code point) corresponding to the first character of the text.

Syntax

```
UNICODE( <Text> )
```

Parameters

| TERM | DEFINITION |
|------|---|
| Text | Text is the character for which you want the Unicode value. |

Return value

A numeric code for the first character in a text string.

UPPER

10/31/2022 • 2 minutes to read • [Edit Online](#)

Converts a text string to all uppercase letters.

Syntax

```
UPPER (<text>)
```

Parameters

| TERM | DEFINITION |
|------|--|
| text | The text you want converted to uppercase, or a reference to a column that contains text. |

Return value

Same text, in uppercase.

Example

The following formula converts the string in the column, [ProductCode], to all uppercase. Non-alphabetic characters are not affected.

```
= UPPER(['New Products'[Product Code])
```

See also

[Text functions](#)

[LOWER function](#)

VALUE

10/31/2022 • 2 minutes to read • [Edit Online](#)

Converts a text string that represents a number to a number.

Syntax

```
VALUE(<text>)
```

Parameters

| TERM | DEFINITION |
|------|---------------------------|
| text | The text to be converted. |

Return value

The converted number in decimal data type.

Remarks

- The value passed as the **text** parameter can be in any of the constant, number, date, or time formats recognized by the application or services you are using. If **text** is not in one of these formats, an error is returned.
- You do not generally need to use the VALUE function in a formula because the engine implicitly converts text to numbers as necessary.
- You can also use column references. For example, if you have a column that contains mixed number types, VALUE can be used to convert all values to a single numeric data type. However, if you use the VALUE function with a column that contains mixed numbers and text, the entire column is flagged with an error, because not all values in all rows can be converted to numbers.

Example

The following formula converts the typed string, "3", into the numeric value 3.

```
= VALUE("3")
```

See also

[Text functions](#)

Time intelligence functions

10/31/2022 • 3 minutes to read • [Edit Online](#)

Data Analysis Expressions (DAX) includes time-intelligence functions that enable you to manipulate data using time periods, including days, months, quarters, and years, and then build and compare calculations over those periods.

In this category

| FUNCTION | DESCRIPTION |
|---------------------------------------|---|
| CLOSINGBALANCEMONTH | Evaluates the expression at the last date of the month in the current context. |
| CLOSINGBALANCEQUARTER | Evaluates the expression at the last date of the quarter in the current context. |
| CLOSINGBALANCEYEAR | Evaluates the expression at the last date of the year in the current context. |
| DATEADD | Returns a table that contains a column of dates, shifted either forward or backward in time by the specified number of intervals from the dates in the current context. |
| DATESBETWEEN | Returns a table that contains a column of dates that begins with a specified start date and continues until a specified end date. |
| DATESINPERIOD | Returns a table that contains a column of dates that begins with a specified start date and continues for the specified number and type of date intervals. |
| DATESMTD | Returns a table that contains a column of the dates for the month to date, in the current context. |
| DATESQTD | Returns a table that contains a column of the dates for the quarter to date, in the current context. |
| DATESYTD | Returns a table that contains a column of the dates for the year to date, in the current context. |
| ENDOFMONTH | Returns the last date of the month in the current context for the specified column of dates. |
| ENDOFQUARTER | Returns the last date of the quarter in the current context for the specified column of dates. |
| ENDOFYEAR | Returns the last date of the year in the current context for the specified column of dates. |

| FUNCTION | DESCRIPTION |
|-----------------------|--|
| FIRSTDATE | Returns the first date in the current context for the specified column of dates. |
| FIRSTNONBLANK | Returns the first value in the column, column, filtered by the current context, where the expression is not blank |
| LASTDATE | Returns the last date in the current context for the specified column of dates. |
| LASTNONBLANK | Returns the last value in the column, column, filtered by the current context, where the expression is not blank. |
| NEXTDAY | Returns a table that contains a column of all dates from the next day, based on the first date specified in the dates column in the current context. |
| NEXTMONTH | Returns a table that contains a column of all dates from the next month, based on the first date in the dates column in the current context. |
| NEXTQUARTER | Returns a table that contains a column of all dates in the next quarter, based on the first date specified in the dates column, in the current context. |
| NEXTYEAR | Returns a table that contains a column of all dates in the next year, based on the first date in the dates column, in the current context. |
| OPENINGBALANCEMONTH | Evaluates the expression at the first date of the month in the current context. |
| OPENINGBALANCEQUARTER | Evaluates the expression at the first date of the quarter, in the current context. |
| OPENINGBALANCEYEAR | Evaluates the expression at the first date of the year in the current context. |
| PARALLELPERIOD | Returns a table that contains a column of dates that represents a period parallel to the dates in the specified dates column, in the current context, with the dates shifted a number of intervals either forward in time or back in time. |
| PREVIOUSDAY | Returns a table that contains a column of all dates representing the day that is previous to the first date in the dates column, in the current context. |
| PREVIOUSMONTH | Returns a table that contains a column of all dates from the previous month, based on the first date in the dates column, in the current context. |
| PREVIOUSQUARTER | Returns a table that contains a column of all dates from the previous quarter, based on the first date in the dates column, in the current context. |

| FUNCTION | DESCRIPTION |
|--------------------|---|
| PREVIOUSYEAR | Returns a table that contains a column of all dates from the previous year, given the last date in the dates column, in the current context. |
| SAMEPERIODLASTYEAR | Returns a table that contains a column of dates shifted one year back in time from the dates in the specified dates column, in the current context. |
| STARTOFMONTH | Returns the first date of the month in the current context for the specified column of dates. |
| STARTOFQUARTER | Returns the first date of the quarter in the current context for the specified column of dates. |
| STARTOFYEAR | Returns the first date of the year in the current context for the specified column of dates. |
| TOTALMTD | Evaluates the value of the expression for the month to date, in the current context. |
| TOTALQTD | Evaluates the value of the expression for the dates in the quarter to date, in the current context. |
| TOTALYTD | Evaluates the year-to-date value of the expression in the current context. |

CLOSINGBALANCEMONTH

10/31/2022 • 2 minutes to read • [Edit Online](#)

Evaluates the **expression** at the last date of the month in the current context.

Syntax

```
CLOSINGBALANCEMONTH(<expression>,<dates>[,<filter>])
```

Parameters

| PARAMETER | DEFINITION |
|------------|---|
| expression | An expression that returns a scalar value. |
| dates | A column that contains dates. |
| filter | (optional) An expression that specifies a filter to apply to the current context. |

Return value

A scalar value that represents the **expression** evaluated at the last date of the month in the current context.

Remarks

- The **dates** argument can be any of the following:
 - A reference to a date/time column.
 - A table expression that returns a single column of date/time values.
 - A Boolean expression that defines a single-column table of date/time values.

NOTE

Constraints on Boolean expressions are described in [CALCULATE function](#).

NOTE

The **filter** expression has restrictions described in [CALCULATE function](#).

- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following sample formula creates a measure that calculates the 'Month End Inventory Value' of the product inventory.

```
=  
CLOSINGBALANCEMONTH(SUMX(ProductInventory,ProductInventory[UnitCost]*ProductInventory[UnitsBalance]),DateTim  
e[DateKey])
```

See also

[Time intelligence functions](#)

[CLOSINGBALANCEYEAR function](#)

[CLOSINGBALANCEQUARTER function](#)

CLOSINGBALANCEQUARTER

10/31/2022 • 2 minutes to read • [Edit Online](#)

Evaluates the **expression** at the last date of the quarter in the current context.

Syntax

```
CLOSINGBALANCEQUARTER(<expression>,<dates>[,<filter>])
```

Parameters

| TERM | DEFINITION |
|------------|---|
| expression | An expression that returns a scalar value. |
| dates | A column that contains dates. |
| filter | (optional) An expression that specifies a filter to apply to the current context. |

Return value

A scalar value that represents the **expression** evaluated at the last date of the quarter in the current context.

Remarks

- The **dates** argument can be any of the following:
 - A reference to a date/time column.
 - A table expression that returns a single column of date/time values.
 - A Boolean expression that defines a single-column table of date/time values.

NOTE

Constraints on Boolean expressions are described in [CALCULATE function](#).

NOTE

The **filter** expression has restrictions described in [CALCULATE function](#).

- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following sample formula creates a measure that calculates the 'Quarter End Inventory Value' of the product inventory.

=

```
CLOSINGBALANCEQUARTER(SUMX(ProductInventory,ProductInventory[UnitCost]*ProductInventory[UnitsBalance]),DateT  
ime[DateKey])
```

See also

[Time intelligence functions](#)

[CLOSINGBALANCEYEAR function](#)

[CLOSINGBALANCEMONTH function](#)

CLOSINGBALANCEYEAR

10/31/2022 • 2 minutes to read • [Edit Online](#)

Evaluates the **expression** at the last date of the year in the current context.

Syntax

```
CLOSINGBALANCEYEAR(<expression>,<dates>[,<filter>][,<year_end_date>])
```

Parameters

| TERM | DEFINITION |
|---------------|---|
| expression | An expression that returns a scalar value. |
| dates | A column that contains dates. |
| filter | (optional) An expression that specifies a filter to apply to the current context. |
| year_end_date | (optional) A literal string with a date that defines the year-end date. The default is December 31. |

Return value

A scalar value that represents the **expression** evaluated at the last date of the year in the current context.

Remarks

- The **year_end_date** parameter is a string literal of a date, in the same locale as the locale of the client where the workbook was created. The year portion of the date is ignored.
- The **dates** argument can be any of the following:
 - A reference to a date/time column.
 - A table expression that returns a single column of date/time values.
 - A Boolean expression that defines a single-column table of date/time values.

NOTE

Constraints on Boolean expressions are described in [CALCULATE function](#).

NOTE

The **filter** expression has restrictions described in [CALCULATE function](#).

- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following sample formula creates a measure that calculates the 'Year End Inventory Value' of the product inventory.

```
=  
CLOSINGBALANCEYEAR(SUMX(ProductInventory,ProductInventory[UnitCost]*ProductInventory[UnitsBalance]),DateTime  
[DateKey])
```

See also

[Time intelligence functions](#)

[CLOSINGBALANCEYEAR function](#)

[CLOSINGBALANCEQUARTER function](#)

[CLOSINGBALANCEMONTH function](#)

DATEADD

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns a table that contains a column of dates, shifted either forward or backward in time by the specified number of intervals from the dates in the current context.

Syntax

```
DATEADD(<dates>,<number_of_intervals>,<interval>)
```

Parameters

| TERM | DEFINITION |
|---------------------|--|
| dates | A column that contains dates. |
| number_of_intervals | An integer that specifies the number of intervals to add to or subtract from the dates. |
| interval | The interval by which to shift the dates. The value for interval can be one of the following: <code>year</code> , <code>quarter</code> , <code>month</code> , <code>day</code> |

Return value

A table containing a single column of date values.

Remarks

The **dates** argument can be any of the following:

- A reference to a date/time column,
- A table expression that returns a single column of date/time values,
- A Boolean expression that defines a single-column table of date/time values.

NOTE

Constraints on Boolean expressions are described in the topic, [CALCULATE function](#).

- If the number specified for **number_of_intervals** is positive, the dates in **dates** are moved forward in time; if the number is negative, the dates in **dates** are shifted back in time.
- The **interval** parameter is an enumeration, not a set of strings; therefore values should not be enclosed in quotation marks. Also, the values: `year`, `quarter`, `month`, `day` should be spelled in full when using them.
- The result table includes only dates that exist in the **dates** column.
- If the dates in the current context do not form a contiguous interval, the function returns an error.

- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example - Shifting a set of dates

The following formula calculates dates that are one year before the dates in the current context.

```
= DATEADD(DateTime[DateKey],-1,year)
```

See also

[Time intelligence functions](#)

[Date and time functions](#)

DATESBETWEEN

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns a table that contains a column of dates that begins with a specified start date and continues until a specified end date.

This function is suited to pass as a filter to the [CALCULATE](#) function. Use it to filter an expression by a custom date range.

NOTE

If you're working with standard date intervals such as days, months, quarters, or years, it's recommended you use the better suited [DATESINPERIOD](#) function.

Syntax

```
DATESBETWEEN(<Dates>, <StartDate>, <EndDate>)
```

Parameters

| TERM | DEFINITION |
|-----------|--------------------|
| Dates | A date column. |
| StartDate | A date expression. |
| EndDate | A date expression. |

Return value

A table containing a single column of date values.

Remarks

- In the most common use case, **Dates** is a reference to the date column of a marked date table.
- If **StartDate** is BLANK, then **StartDate** will be the earliest value in the **Dates** column.
- If **EndDate** is BLANK, then **EndDate** will be the latest value in the **Dates** column.
- Dates used as the **StartDate** and **EndDate** are inclusive. So, for example, if the **StartDate** value is July 1, 2019, then that date will be included in the returned table (providing the date exists in the **Dates** column).
- The returned table can only contain dates stored in the **Dates** column. So, for example, if the **Dates** column starts from July 1, 2017, and the **StartDate** value is July 1, 2016, the returned table will start from July 1, 2017.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following **Sales** table measure definition uses the DATESBETWEEN function to produce a *life-to-date* (LTD) calculation. Life-to-date represents the accumulation of a measure over time since the very beginning of time.

Notice that the formula uses the [MAX](#) function. This function returns the latest date that's in the filter context. So, the DATESBETWEEN function returns a table of dates beginning from the earliest date until the latest date being reported.

Examples in this article can be used with the sample Adventure Works DW 2020 Power BI Desktop model. To get the model, see [DAX sample model](#).

```
Customers LTD =  
CALCULATE(  
    DISTINCTCOUNT(Sales[CustomerKey]),  
    DATESBETWEEN(  
        'Date'[Date],  
        BLANK(),  
        MAX('Date'[Date])  
    )  
)
```

Consider that the earliest date stored in the **Date** table is July 1, 2017. So, when a report filters the measure by the month of June 2020, the DATESBETWEEN function returns a date range from July 1, 2017 until June 30, 2020.

See also

- [Time intelligence functions \(DAX\)](#)
- [Date and time functions \(DAX\)](#)
- [DATESINPERIOD function \(DAX\)](#)

DATESINPERIOD

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns a table that contains a column of dates that begins with a specified start date and continues for the specified number and type of date intervals.

This function is suited to pass as a filter to the [CALCULATE](#) function. Use it to filter an expression by standard date intervals such as days, months, quarters, or years.

Syntax

```
DATESINPERIOD(<dates>, <start_date>, <number_of_intervals>, <interval>)
```

Parameters

| TERM | DEFINITION |
|---------------------|--|
| dates | A date column. |
| start_date | A date expression. |
| number_of_intervals | An integer that specifies the number of intervals to add to, or subtract from, the dates. |
| interval | The interval by which to shift the dates. The value for interval can be one of the following: <code>DAY</code> , <code>MONTH</code> , <code>QUARTER</code> , and <code>YEAR</code> . |

Return value

A table containing a single column of date values.

Remarks

- In the most common use case, **dates** is a reference to the date column of a marked date table.
- If the number specified for **number_of_intervals** is positive, dates are moved forward in time; if the number is negative, dates are shifted backward in time.
- The **interval** parameter is an enumeration. Valid values are `DAY`, `MONTH`, `QUARTER`, and `YEAR`. Because it's an enumeration, values aren't passed in as strings. So don't enclose them within quotation marks.
- The returned table can only contain dates stored in the **dates** column. So, for example, if the **dates** column starts from July 1, 2017, and the **start_date** value is July 1, 2016, the returned table will start from July 1, 2017.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following **Sales** table measure definition uses the DATESINPERIOD function to calculate revenue for the prior year (PY).

Notice the formula uses the [MAX](#) function. This function returns the latest date that's in the filter context. So, the DATESINPERIOD function returns a table of dates beginning from the latest date for the last year.

Examples in this article can be used with the sample Adventure Works DW 2020 Power BI Desktop model. To get the model, see [DAX sample model](#).

```
Revenue PY =  
CALCULATE(  
    SUM(Sales[Sales Amount]),  
    DATESINPERIOD(  
        'Date'[Date],  
        MAX('Date'[Date]),  
        -1,  
        YEAR  
    )  
)
```

Consider that the report is filtered by the month of June 2020. The MAX function returns June 30, 2020. The DATESINPERIOD function then returns a date range from July 1, 2019 until June 30, 2020. It's a year of date values starting from June 30, 2020 for the last year.

See also

[Time intelligence functions \(DAX\)](#)

[Date and time functions \(DAX\)](#)

[DATESBETWEEN function \(DAX\)](#)

DATESMTD

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns a table that contains a column of the dates for the month to date, in the current context.

Syntax

```
DATESMTD(<dates>)
```

Parameters

| TERM | DEFINITION |
|-------|-------------------------------|
| dates | A column that contains dates. |

Return value

A table containing a single column of date values.

Remarks

The **dates** argument can be any of the following:

- A reference to a date/time column.
- A table expression that returns a single column of date/time values.
- A Boolean expression that defines a single-column table of date/time values.

NOTE

Constraints on Boolean expressions are described in the topic, [CALCULATE function](#).

- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following sample formula creates a measure that calculates the 'Month To Date Total' for Internet Sales.

```
= CALCULATE(SUM(InternetSales_USD[SalesAmount_USD]), DATESMTD(DateTime[DateKey]))
```

See also

[Time intelligence functions](#)

[Date and time functions](#)

[DATESYTD function](#)

[DATESQTD function](#)

DATESQTD

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns a table that contains a column of the dates for the quarter to date, in the current context.

Syntax

```
DATESQTD(<dates>)
```

Parameters

| TERM | DEFINITION |
|-------|-------------------------------|
| dates | A column that contains dates. |

Return value

A table containing a single column of date values.

Remarks

The **dates** argument can be any of the following:

- A reference to a date/time column.
- A table expression that returns a single column of date/time values.
- A Boolean expression that defines a single-column table of date/time values.

NOTE

Constraints on Boolean expressions are described in the topic, [CALCULATE function](#).

- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following sample formula creates a measure that calculates the 'Quarterly Running Total' of Internet Sales.

```
= CALCULATE(SUM(InternetSales_USD[SalesAmount_USD]), DATESQTD(DateTime[DateKey]))
```

See also

[Time intelligence functions](#)

[Date and time functions](#)

[DATESYTD function](#)

[DATESMTD function](#)

DATESYTD

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns a table that contains a column of the dates for the year to date, in the current context.

Syntax

```
DATESYTD(<dates> [, <year_end_date>])
```

Parameters

| TERM | DEFINITION |
|---------------|---|
| dates | A column that contains dates. |
| year_end_date | (optional) A literal string with a date that defines the year-end date. The default is December 31. |

Return value

A table containing a single column of date values.

Remarks

The **dates** argument can be any of the following:

- A reference to a date/time column,
- A table expression that returns a single column of date/time values,
- A Boolean expression that defines a single-column table of date/time values.

NOTE

Constraints on Boolean expressions are described in the topic, [CALCULATE function](#).

- The **year_end_date** parameter is a string literal of a date, in the same locale as the locale of the client where the workbook was created. The year portion of the date is ignored.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following sample formula creates a measure that calculates the 'Running Total' for Internet sales.

```
= CALCULATE(SUM(InternetSales_USD[SalesAmount_USD]), DATESYTD(DateTime[DateKey]))
```

See also

Time intelligence functions

Date and time functions

DATESMTD function

DATESQTD function

ENDOFMONTH

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the last date of the month in the current context for the specified column of dates.

Syntax

```
ENDOFMONTH(<dates>)
```

Parameters

| TERM | DEFINITION |
|-------|-------------------------------|
| dates | A column that contains dates. |

Return value

A table containing a single column and single row with a date value.

Remarks

- The **dates** argument can be any of the following:
 - A reference to a date/time column.
 - A table expression that returns a single column of date/time values.
 - A Boolean expression that defines a single-column table of date/time values.
- Constraints on Boolean expressions are described in the topic, [CALCULATE function](#).
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following sample formula creates a measure that returns the end of the month, for the current context.

```
= ENDOFMONTH(DateTime[DateKey])
```

See also

[Date and time functions](#)

[Time intelligence functions](#)

[ENDOFYEAR function](#)

[ENDOFQUARTER function](#)

ENDOFQUARTER

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the last date of the quarter in the current context for the specified column of dates.

Syntax

```
ENDOFQUARTER(<dates>)
```

Parameters

| TERM | DEFINITION |
|-------|-------------------------------|
| dates | A column that contains dates. |

Return value

A table containing a single column and single row with a date value.

Remarks

- The **dates** argument can be any of the following:
 - A reference to a date/time column,
 - A table expression that returns a single column of date/time values,
 - A Boolean expression that defines a single-column table of date/time values.
- Constraints on Boolean expressions are described in the topic, [CALCULATE function](#).
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following sample formula creates a measure that returns the end of the quarter, for the current context.

```
= ENDOFQUARTER(DateTime[DateKey])
```

See also

[Date and time functions](#)

[Time intelligence functions](#)

[ENDOFYEAR function](#)

[ENDOFMONTH function](#)

ENDOFYEAR

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the last date of the year in the current context for the specified column of dates.

Syntax

```
ENDOFYEAR(<dates> [, <year_end_date>])
```

Parameters

| TERM | DEFINITION |
|---------------|---|
| dates | A column that contains dates. |
| year_end_date | (optional) A literal string with a date that defines the year-end date. The default is December 31. |

Return value

A table containing a single column and single row with a date value.

Remarks

- The **dates** argument can be any of the following:
 - A reference to a date/time column,
 - A table expression that returns a single column of date/time values,
 - A Boolean expression that defines a single-column table of date/time values.
- Constraints on Boolean expressions are described in the topic, [CALCULATE function](#).
- The **year_end_date** parameter is a string literal of a date, in the same locale as the locale of the client where the workbook was created. The year portion of the date is ignored.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following sample formula creates a measure that returns the end of the fiscal year that ends on June 30, for the current context.

```
= ENDOFYEAR(DateTime[DateKey], "06/30/2004")
```

See also

[Date and time functions](#)

[Time intelligence functions](#)

[ENDOFMONTH function](#)

FIRSTDATE

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the first date in the current context for the specified column of dates.

Syntax

```
FIRSTDATE(<dates>)
```

Parameters

| TERM | DEFINITION |
|-------|-------------------------------|
| dates | A column that contains dates. |

Return value

A table containing a single column and single row with a date value.

Remarks

- The **dates** argument can be any of the following:
 - A reference to a date/time column.
 - A table expression that returns a single column of date/time values.
 - A Boolean expression that defines a single-column table of date/time values.
- Constraints on Boolean expressions are described in the topic, [CALCULATE function](#).
- When the current context is a single date, the date returned by the FIRSTDATE and LASTDATE functions will be equal.
- The Return value is a table that contains a single column and single value. Therefore, this function can be used as an argument to any function that requires a table in its arguments. Also, the returned value can be used whenever a date value is required.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following sample formula creates a measure that obtains the first date when a sale was made in the Internet sales channel for the current context.

```
= FIRSTDATE('InternetSales_USD'[SaleDateKey])
```

See also

[Date and time functions](#)

[Time intelligence functions](#)

LASTDATE function

FIRSTNONBLANK function

FIRSTNONBLANK

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the first value in the column, **column**, filtered by the current context, where the expression is not blank.

Syntax

```
FIRSTNONBLANK(<column>, <expression>)
```

Parameters

| TERM | DEFINITION |
|------------|--|
| column | A column expression. |
| expression | An expression evaluated for blanks for each value of column . |

Return value

A table containing a single column and single row with the computed first value.

Remarks

- The **column** argument can be any of the following:
 - A reference to any column.
 - A table with a single column.
- A Boolean expression that defines a single-column table .
- Constraints on Boolean expressions are described in the topic, [CALCULATE function](#).
- This function is typically used to return the first value of a column for which the expression is not blank. For example, you could get the last value for which there were sales of a product.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

See also

[LASTNONBLANK function](#)

[Statistical functions](#)

FIRSTNONBLANKVALUE

10/31/2022 • 2 minutes to read • [Edit Online](#)

Evaluates an expression filtered by the sorted values of a column and returns the first value of the expression that is not blank.

Syntax

```
FIRSTNONBLANKVALUE(<column>, <expression>)
```

Parameters

| TERM | DEFINITION |
|------------|---|
| column | A column or an expression that returns a single-column table. |
| expression | An expression evaluated for each value of <column>. |

Return value

The first non-blank value of <expression> corresponding to the sorted values of <column>.

Remarks

- The column argument can be any of the following:
 - A reference to any column.
 - A table with a single column.
- This function is different from FIRSTNONBLANK in that the <column> is added to the filter context for the evaluation of <expression>.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following DAX query,

```
EVALUATE
SUMMARIZECOLUMNS(
    DimProduct[Class],
    "FNBV",
    FIRSTNONBLANKVALUE(
        DimDate[Date],
        SUM(FactInternetSales[SalesAmount])
    )
)
```

Returns,

| DIMPRODUCT[CLASS] | [FNBV] |
|-------------------|-----------|
| L | 699.0982 |
| H | 13778.24 |
| M | 1000.4375 |
| | 533.83 |

LASTDATE

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the last date in the current context for the specified column of dates.

Syntax

```
LASTDATE(<dates>)
```

Parameters

| TERM | DEFINITION |
|-------|-------------------------------|
| dates | A column that contains dates. |

Return value

A table containing a single column and single row with a date value.

Remarks

- The **dates** argument can be any of the following:
 - A reference to a date/time column,
 - A table expression that returns a single column of date/time values,
 - A Boolean expression that defines a single-column table of date/time values.
- Constraints on Boolean expressions are described in the topic, [CALCULATE function](#).
- When the current context is a single date, the date returned by the FIRSTDATE and LASTDATE functions will be equal.
- Technically, the Return value is a table that contains a single column and single value. Therefore, this function can be used as an argument to any function that requires a table in its arguments. Also, the returned value can be used whenever a date value is required.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following sample formula creates a measure that obtains the last date, for the current context, when a sale was made in the Internet sales channel.

```
= LASTDATE('InternetSales_USD'[SaleDateKey])
```

See also

[Date and time functions](#)

[Time intelligence functions](#)

FIRSTDATE function

LASTNONBLANK function

LASTNONBLANK

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the last value in the column, **column**, filtered by the current context, where the expression is not blank.

Syntax

```
LASTNONBLANK(<column>,<expression>)
```

Parameters

| TERM | DEFINITION |
|------------|--|
| column | A column expression. |
| expression | An expression evaluated for blanks for each value of column . |

Return value

A table containing a single column and single row with the computed last value.

Remarks

- The **column** argument can be any of the following:
 - A reference to any column.
 - A table with a single column.
 - A Boolean expression that defines a single-column table
- Constraints on Boolean expressions are described in the topic, [CALCULATE function](#).
- This function is typically used to return the last value of a column for which the expression is not blank. For example, you could get the last value for which there were sales of a product.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

See also

[FIRSTNONBLANK function](#)

[Statistical functions](#)

LASTNONBLANKVALUE

10/31/2022 • 2 minutes to read • [Edit Online](#)

Evaluates an expression filtered by the sorted values of a column and returns the last value of the expression that is not blank.

Syntax

```
LASTNONBLANKVALUE(<column>, <expression>)
```

Parameters

| TERM | DEFINITION |
|------------|---|
| column | A column or an expression that returns a single-column table. |
| expression | An expression evaluated for each value of <column>. |

Return value

The last non-blank value of <expression> corresponding to the sorted values of <column>.

Remarks

- The column argument can be any of the following:
 - A reference to any column.
 - A table with a single column.
- This function is different from LASTNONBLANK in that the <column> is added to the filter context for the evaluation of <expression>.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following DAX query,

```
EVALUATE
SUMMARIZECOLUMNS(
    DimProduct[Class],
    "LNBV",
    LASTNONBLANKVALUE(
        DimDate[Date],
        SUM(FactInternetSales[SalesAmount])
    )
)
```

Returns,

| DIMPRODUCT[CLASS] | [LNBV] |
|-------------------|--------|
| L | 132.44 |
| H | 137.6 |
| M | 84.97 |
| | 2288.6 |

NEXTDAY

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns a table that contains a column of all dates from the next day, based on the first date specified in the **dates** column in the current context.

Syntax

```
NEXTDAY(<dates>)
```

Parameters

| TERM | DEFINITION |
|-------|----------------------------|
| dates | A column containing dates. |

Return value

A table containing a single column of date values.

Remarks

- This function returns all dates from the next day to the first date in the input parameter. For example, if the first date in the **dates** argument refers to June 10, 2009; then this function returns all dates equal to June 11, 2009.
- The **dates** argument can be any of the following:
 - A reference to a date/time column.
 - A table expression that returns a single column of date/time values.
 - A Boolean expression that defines a single-column table of date/time values.
- Constraints on Boolean expressions are described in the topic, [CALCULATE function](#).
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following sample formula creates a measure that calculates the 'next day sales' of Internet sales.

```
= CALCULATE(SUM(InternetSales_USD[SalesAmount_USD]), NEXTDAY('DateTime'[DateKey]))
```

See also

[Time intelligence functions](#)

[Date and time functions](#)

[NEXTQUARTER function](#)

[NEXTMONTH function](#)

NEXTMONTH

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns a table that contains a column of all dates from the next month, based on the first date in the **dates** column in the current context.

Syntax

```
NEXTMONTH(<dates>)
```

Parameters

| TERM | DEFINITION |
|-------|----------------------------|
| dates | A column containing dates. |

Return value

A table containing a single column of date values.

Remarks

- This function returns all dates from the next day to the first date in the input parameter. For example, if the first date in the **dates** argument refers to June 10, 2009; then this function returns all dates for the month of July, 2009.
- The **dates** argument can be any of the following:
 - A reference to a date/time column.
 - A table expression that returns a single column of date/time values.
 - A Boolean expression that defines a single-column table of date/time values.
- Constraints on Boolean expressions are described in the topic, [CALCULATE function](#).
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following sample formula creates a measure that calculates the 'next month sales' for Internet sales.

```
= CALCULATE(SUM(InternetSales_USD[SalesAmount_USD]), NEXTMONTH('DateTime'[DateKey]))
```

See also

[Time intelligence functions](#)

[Date and time functions](#)

[NEXTDAY function](#)

[NEXTQUARTER function](#)

NEXTQUARTER

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns a table that contains a column of all dates in the next quarter, based on the first date specified in the **dates** column, in the current context.

Syntax

```
NEXTQUARTER(<dates>)
```

Parameters

| TERM | DEFINITION |
|-------|----------------------------|
| dates | A column containing dates. |

Return value

A table containing a single column of date values.

Remarks

- This function returns all dates in the next quarter, based on the first date in the input parameter. For example, if the first date in the **dates** column refers to June 10, 2009, this function returns all dates for the quarter July to September, 2009.
- The **dates** argument can be any of the following:
 - A reference to a date/time column.
 - A table expression that returns a single column of date/time values.
 - A Boolean expression that defines a single-column table of date/time values.
- Constraints on Boolean expressions are described in the topic, [CALCULATE function](#).
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following sample formula creates a measure that calculates the 'next quarter sales' for Internet sales.

```
= CALCULATE(SUM(InternetSales_USD[SalesAmount_USD]), NEXTQUARTER('DateTime'[DateKey]))
```

See also

[Time intelligence functions](#)

[Date and time functions](#)

[NEXTDAY function](#)

[NEXTMONTH function](#)

NEXTYEAR

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns a table that contains a column of all dates in the next year, based on the first date in the **dates** column, in the current context.

Syntax

```
NEXTYEAR(<dates>[, <year_end_date>])
```

Parameters

| TERM | DEFINITION |
|---------------|---|
| dates | A column containing dates. |
| year_end_date | (optional) A literal string with a date that defines the year-end date. The default is December 31. |

Return value

A table containing a single column of date values.

Remarks

- This function returns all dates in the next year, based on the first date in the input column. For example, if the first date in the **dates** column refers to the year 2007, this function returns all dates for the year 2008.
- The **dates** argument can be any of the following:
 - A reference to a date/time column.
 - A table expression that returns a single column of date/time values.
 - A Boolean expression that defines a single-column table of date/time values.
- Constraints on Boolean expressions are described in the topic, [CALCULATE function](#).
- The **year_end_date** parameter is a string literal of a date, in the same locale as the locale of the client where the workbook was created. The year portion of the date is ignored.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following sample formula creates a measure that calculates the 'next year sales' for Internet sales.

```
= CALCULATE(SUM(InternetSales_USD[SalesAmount_USD]), NEXTYEAR('DateTime'[DateKey]))
```


See also

[Time intelligence functions](#)

[Date and time functions](#)

[NEXTDAY function](#)

[NEXTQUARTER function](#)

[NEXTMONTH function](#)

OPENINGBALANCEMONTH

10/31/2022 • 2 minutes to read • [Edit Online](#)

Evaluates the **expression** at the first date of the month in the current context.

Syntax

```
OPENINGBALANCEMONTH(<expression>,<dates>[,<filter>])
```

Parameters

| TERM | DEFINITION |
|------------|---|
| expression | An expression that returns a scalar value. |
| dates | A column that contains dates. |
| filter | (optional) An expression that specifies a filter to apply to the current context. |

Return value

A scalar value that represents the **expression** evaluated at the first date of the month in the current context.

Remarks

- The **dates** argument can be any of the following:
 - A reference to a date/time column.
 - A table expression that returns a single column of date/time values.
 - A Boolean expression that defines a single-column table of date/time values.
- Constraints on Boolean expressions are described in the topic, [CALCULATE function](#).
- The **filter** expression has restrictions described in the topic, [CALCULATE function](#).
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following sample formula creates a measure that calculates the 'Month Start Inventory Value' of the product inventory.

```
=  
OPENINGBALANCEMONTH(SUMX(ProductInventory,ProductInventory[UnitCost]*ProductInventory[UnitsBalance]),DateTim  
e[DateKey])
```

See also

OPENINGBALANCEYEAR function
OPENINGBALANCEQUARTER function
Time intelligence functions
CLOSINGBALANCEMONTH function

OPENINGBALANCEQUARTER

10/31/2022 • 2 minutes to read • [Edit Online](#)

Evaluates the **expression** at the first date of the quarter, in the current context.

Syntax

```
OPENINGBALANCEQUARTER(<expression>,<dates>[,<filter>])
```

Parameters

| TERM | DEFINITION |
|------------|---|
| expression | An expression that returns a scalar value. |
| dates | A column that contains dates. |
| filter* | (optional) An expression that specifies a filter to apply to the current context. |

Return value

A scalar value that represents the **expression** evaluated at the first date of the quarter in the current context.

Remarks

- The **dates** argument can be any of the following:
 - A reference to a date/time column.
 - A table expression that returns a single column of date/time values.
 - A Boolean expression that defines a single-column table of date/time values.
- Constraints on Boolean expressions are described in the topic, [CALCULATE function](#).
- The **filter** expression has restrictions described in the topic, [CALCULATE function](#).
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following sample formula creates a measure that calculates the 'Quarter Start Inventory Value' of the product inventory.

```
=  
OPENINGBALANCEQUARTER(SUMX(ProductInventory,ProductInventory[UnitCost]*ProductInventory[UnitsBalance]),DateT  
ime[DateKey])
```

See also

OPENINGBALANCEYEAR function
OPENINGBALANCEMONTH function
Time intelligence functions
CLOSINGBALANCEQUARTER function

OPENINGBALANCEYEAR

10/31/2022 • 2 minutes to read • [Edit Online](#)

Evaluates the **expression** at the first date of the year in the current context.

Syntax

```
OPENINGBALANCEYEAR(<expression>,<dates>[,<filter>][,<year_end_date>])
```

Parameters

| TERM | DEFINITION |
|---------------|---|
| expression | An expression that returns a scalar value. |
| dates | A column that contains dates. |
| filter | (optional) An expression that specifies a filter to apply to the current context. |
| year_end_date | (optional) A literal string with a date that defines the year-end date. The default is December 31. |

Return value

A scalar value that represents the **expression** evaluated at the first date of the year in the current context.

Remarks

- The **dates** argument can be any of the following:
 - A reference to a date/time column.
 - A table expression that returns a single column of date/time values.
 - A Boolean expression that defines a single-column table of date/time values.
- Constraints on Boolean expressions are described in the topic, [CALCULATE function](#).
- The **filter** expression has restrictions described in the topic, [CALCULATE function](#).
- The **year_end_date** parameter is a string literal of a date, in the same locale as the locale of the client where the workbook was created. The year portion of the date is ignored.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following sample formula creates a measure that calculates the 'Year Start Inventory Value' of the product inventory.

=

```
OPENINGBALANCEYEAR(SUMX(ProductInventory,ProductInventory[UnitCost]*ProductInventory[UnitsBalance]),DateTime  
[DateKey])
```

See also

[OPENINGBALANCEQUARTER function](#)

[OPENINGBALANCEMONTH function](#)

[Time intelligence functions](#)

[CLOSINGBALANCEYEAR function](#)

PARALLELPERIOD

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns a table that contains a column of dates that represents a period parallel to the dates in the specified **dates** column, in the current context, with the dates shifted a number of intervals either forward in time or back in time.

Syntax

```
PARALLELPERIOD(<dates>,<number_of_intervals>,<interval>)
```

Parameters

| TERM | DEFINITION |
|---------------------|---|
| dates | A column that contains dates. |
| number_of_intervals | An integer that specifies the number of intervals to add to or subtract from the dates. |
| interval | The interval by which to shift the dates. The value for interval can be one of the following: <code>year</code> , <code>quarter</code> , <code>month</code> . |

Return value

A table containing a single column of date values.

Remarks

- This function takes the current set of dates in the column specified by **dates**, shifts the first date and the last date the specified number of intervals, and then returns all contiguous dates between the two shifted dates. If the interval is a partial range of month, quarter, or year then any partial months in the result are also filled out to complete the entire interval.
- The **dates** argument can be any of the following:
 - A reference to a date/time column,
 - A table expression that returns a single column of date/time values,
 - A Boolean expression that defines a single-column table of date/time values.
- Constraints on Boolean expressions are described in the topic, [CALCULATE function](#).
- If the number specified for **number_of_intervals** is positive, the dates in **dates** are moved forward in time; if the number is negative, the dates in **dates** are shifted back in time.
- The **interval** parameter is an enumeration, not a set of strings; therefore values should not be enclosed in quotation marks. Also, the values: `year`, `quarter`, `month` should be spelled in full when using them.
- The result table includes only dates that appear in the values of the underlying table column.
- The PARALLELPERIOD function is similar to the DATEADD function except that PARALLELPERIOD always

returns full periods at the given granularity level instead of the partial periods that DATEADD returns. For example, if you have a selection of dates that starts at June 10 and finishes at June 21 of the same year, and you want to shift that selection forward by one month then the PARALLELPERIOD function will return all dates from the next month (July 1 to July 31); however, if DATEADD is used instead, then the result will include only dates from July 10 to July 21.

- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following sample formula creates a measure that calculates the previous year sales for Internet sales.

```
= CALCULATE(SUM(InternetSales_USD[SalesAmount_USD]), PARALLELPERIOD(DateTime[DateKey],-1,year))
```

See also

[Time intelligence functions](#)

[Date and time functions](#)

[DATEADD function](#)

PREVIOUSDAY

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns a table that contains a column of all dates representing the day that is previous to the first date in the **dates** column, in the current context.

Syntax

```
PREVIOUSDAY(<dates>)
```

Parameters

| TERM | DEFINITION |
|-------|----------------------------|
| dates | A column containing dates. |

Return value

A table containing a single column of date values.

Remarks

- This function determines the first date in the input parameter, and then returns all dates corresponding to the day previous to that first date. For example, if the first date in the **dates** argument refers to June 10, 2009; this function returns all dates equal to June 9, 2009.
- The **dates** argument can be any of the following:
 - A reference to a date/time column.
 - A table expression that returns a single column of date/time values.
 - A Boolean expression that defines a single-column table of date/time values.
- Constraints on Boolean expressions are described in the topic, [CALCULATE function](#).
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following sample formula creates a measure that calculates the 'previous day sales' for Internet sales.

```
= CALCULATE(SUM(InternetSales_USD[SalesAmount_USD]), PREVIOUSDAY('DateTime'[DateKey]))
```

See also

[Time intelligence functions](#)

[Date and time functions](#)

[PREVIOUSMONTH function](#)

[PREVIOUSQUARTER function](#)

PREVIOUSMONTH

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns a table that contains a column of all dates from the previous month, based on the first date in the **Dates** column, in the current context.

Syntax

```
PREVIOUSMONTH(<Dates>)
```

Parameters

| TERM | DEFINITION |
|-------|----------------------------|
| Dates | A column containing dates. |

Return value

A table containing a single column of date values.

Remarks

- This function returns all dates from the previous month, using the first date in the column used as input. For example, if the first date in the **Dates** argument refers to June 10, 2009, this function returns all dates for the month of May, 2009.
- The **Dates** argument can be any of the following:
 - A reference to a date/time column.
 - A table expression that returns a single column of date/time values.
 - A Boolean expression that defines a single-column table of date/time values.
- Constraints on Boolean expressions are described in the topic, [CALCULATE](#).
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following sample formula creates a measure that calculates the 'previous month sales' for Internet sales.

```
= CALCULATE(SUM(InternetSales_USD[SalesAmount_USD]), PREVIOUSMONTH('DateTime'[DateKey]))
```

See also

[Time intelligence functions](#)

[Date and time functions](#)

[PREVIOUSDAY](#)

[PREVIOUSQUARTER](#)

PREVIOUSQUARTER

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns a table that contains a column of all dates from the previous quarter, based on the first date in the **dates** column, in the current context.

Syntax

```
PREVIOUSQUARTER(<dates>)
```

Parameters

| TERM | DEFINITION |
|-------|----------------------------|
| dates | A column containing dates. |

Return value

A table containing a single column of date values.

Remarks

- This function returns all dates from the previous quarter, using the first date in the input column. For example, if the first date in the **dates** argument refers to June 10, 2009, this function returns all dates for the quarter January to March, 2009.
- The **dates** argument can be any of the following:
 - A reference to a date/time column.
 - A table expression that returns a single column of date/time values.
 - A Boolean expression that defines a single-column table of date/time values.
- Constraints on Boolean expressions are described in the topic, [CALCULATE](#).
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following sample formula creates a measure that calculates the 'previous quarter sales' for Internet sales.

```
= CALCULATE(SUM(InternetSales_USD[SalesAmount_USD]), PREVIOUSQUARTER('DateTime'[DateKey]))
```

See also

[Time intelligence functions](#)

[Date and time functions](#)

[PREVIOUSMONTH](#)

[PREVIOUSDAY](#)

PREVIOUSYEAR

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns a table that contains a column of all dates from the previous year, given the last date in the **dates** column, in the current context.

Syntax

```
PREVIOUSYEAR(<dates>[, <year_end_date>])
```

Parameters

| TERM | DEFINITION |
|---------------|---|
| dates | A column containing dates. |
| year_end_date | (optional) A literal string with a date that defines the year-end date. The default is December 31. |

Return value

A table containing a single column of date values.

Remarks

- This function returns all dates from the previous year given the latest date in the input parameter. For example, if the latest date in the **dates** argument refers to the year 2009, then this function returns all dates for the year of 2008, up to the specified **year_end_date**.
- The **dates** argument can be any of the following:
 - A reference to a date/time column.
 - A table expression that returns a single column of date/time values.
 - A Boolean expression that defines a single-column table of date/time values.
- Constraints on Boolean expressions are described in the topic, [CALCULATE](#).
- The **year_end_date** parameter is a string literal of a date, in the same locale as the locale of the client where the workbook was created. The year portion of the date is ignored.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following sample formula creates a measure that calculates the previous year sales for Internet sales.

```
= CALCULATE(SUM(InternetSales_USD[SalesAmount_USD]), PREVIOUSYEAR('DateTime'[DateKey]))
```


See also

[Time intelligence functions](#)

[Date and time functions](#)

[PREVIOUSMONTH](#)

[PREVIOUSDAY](#)

[PREVIOUSQUARTER](#)

SAMEPERIODLASTYEAR

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns a table that contains a column of dates shifted one year back in time from the dates in the specified **dates** column, in the current context.

Syntax

```
SAMEPERIODLASTYEAR(<dates>)
```

Parameters

| TERM | DEFINITION |
|--------------|----------------------------|
| dates | A column containing dates. |

Return value

A single-column table of date values.

Remarks

- The **dates** argument can be any of the following:
 - A reference to a date/time column,
 - A table expression that returns a single column of date/time values,
 - A Boolean expression that defines a single-column table of date/time values.
- Constraints on Boolean expressions are described in the topic, [CALCULATE](#).
- The dates returned are the same as the dates returned by this equivalent formula:

DATEADD(dates, -1, year)
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following sample formula creates a measure that calculates the previous year sales of Reseller sales.

```
= CALCULATE(SUM(ResellerSales_USD[SalesAmount_USD]), SAMEPERIODLASTYEAR(DateTime[DateKey]))
```

See also

[Time intelligence functions](#)

[Date and time functions](#)

[PREVIOUSYEAR](#)

[PARALLELPERIOD](#)

STARTOFMONTH

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the first date of the month in the current context for the specified column of dates.

Syntax

```
STARTOFMONTH(<dates>)
```

Parameters

| TERM | DEFINITION |
|-------|-------------------------------|
| dates | A column that contains dates. |

Return value

A table containing a single column and single row with a date value.

Remarks

- The **dates** argument can be any of the following:
 - A reference to a date/time column.
 - A table expression that returns a single column of date/time values.
 - A Boolean expression that defines a single-column table of date/time values.
- Constraints on Boolean expressions are described in the topic, [CALCULATE](#).
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following sample formula creates a measure that returns the start of the month, for the current context.

```
= STARTOFMONTH(DateTime[DateKey])
```

See also

[Date and time functions](#)

[Time intelligence functions](#)

[STARTOFYEAR](#)

[STARTOFQUARTER](#)

STARTOFQUARTER

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the first date of the quarter in the current context for the specified column of dates.

Syntax

```
STARTOFQUARTER(<dates>)
```

Parameters

| TERM | DEFINITION |
|-------|-------------------------------|
| dates | A column that contains dates. |

Return value

A table containing a single column and single row with a date value.

Remarks

- The **dates** argument can be any of the following:
 - A reference to a date/time column.
 - A table expression that returns a single column of date/time values.
 - A Boolean expression that defines a single-column table of date/time values.
- Constraints on Boolean expressions are described in the topic, [CALCULATE](#).
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following sample formula creates a measure that returns the start of the quarter, for the current context.

```
= STARTOFQUARTER(DateTime[DateKey])
```

See also

[Date and time functions](#)

[Time intelligence functions](#)

[STARTOFYEAR](#)

[STARTOFMONTH](#)

STARTOFYEAR

10/31/2022 • 2 minutes to read • [Edit Online](#)

Returns the first date of the year in the current context for the specified column of dates.

Syntax

```
STARTOFYEAR(<dates>)
```

Parameters

| TERM | DEFINITION |
|-------------|-----------------------------------|
| dates | A column that contains dates. |
| YearEndDate | (Optional) A year end date value. |

Return value

A table containing a single column and single row with a date value.

Remarks

- The **dates** argument can be any of the following:
 - A reference to a date/time column.
 - A table expression that returns a single column of date/time values.
 - A Boolean expression that defines a single-column table of date/time values.
- Constraints on Boolean expressions are described in the topic, [CALCULATE](#).
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following sample formula creates a measure that returns the start of the year, for the current context.

```
= STARTOFYEAR(DateTime[DateKey])
```

See also

[Date and time functions](#)

[Time intelligence functions](#)

[STARTOFQUARTER](#)

[STARTOFMONTH](#)

TOTALMTD

10/31/2022 • 2 minutes to read • [Edit Online](#)

Evaluates the value of the **expression** for the month to date, in the current context.

Syntax

```
TOTALMTD(<expression>,<dates>[,<filter>])
```

Parameters

| PARAMETER | DEFINITION |
|------------|---|
| expression | An expression that returns a scalar value. |
| dates | A column that contains dates. |
| filter | (optional) An expression that specifies a filter to apply to the current context. |

Return value

A scalar value that represents the **expression** evaluated for the dates in the current month-to-date, given the dates in **dates**.

Remarks

- The **dates** argument can be any of the following:
 - A reference to a date/time column.
 - A table expression that returns a single column of date/time values.
 - A Boolean expression that defines a single-column table of date/time values.
- Constraints on Boolean expressions are described in the topic, [CALCULATE](#).
- The **filter** expression has restrictions described in the topic, [CALCULATE](#).
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following sample formula creates a measure that calculates the 'month running total' or 'month running sum' for Internet sales.

```
= TOTALMTD(SUM(InternetSales_USD[SalesAmount_USD]),DateTime[DateKey])
```

See also

[ALL](#)

CALCULATE
TOTALYTD
TOTALQTD

TOTALQTD

10/31/2022 • 2 minutes to read • [Edit Online](#)

Evaluates the value of the **expression** for the dates in the quarter to date, in the current context.

Syntax

```
TOTALQTD(<expression>,<dates>[,<filter>])
```

Parameters

| PARAMETER | DEFINITION |
|------------|---|
| expression | An expression that returns a scalar value. |
| dates | A column that contains dates. |
| filter | (optional) An expression that specifies a filter to apply to the current context. |

Return value

A scalar value that represents the **expression** evaluated for all dates in the current quarter to date, given the dates in **dates**.

Remarks

- The **dates** argument can be any of the following:
 - A reference to a date/time column.
 - A table expression that returns a single column of date/time values.
 - A Boolean expression that defines a single-column table of date/time values.
- Constraints on Boolean expressions are described in the topic, [CALCULATE](#).
- The **filter** expression has restrictions described in the topic, [CALCULATE](#).
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example

The following sample formula creates a measure that calculates the 'quarter running total' or 'quarter running sum' for Internet sales.

```
= TOTALQTD(SUM(InternetSales_USD[SalesAmount_USD]),DateTime[DateKey])
```

See also

[ALL](#)

CALCULATE
TOTALYTD
TOTALMTD

TOTALYTD

10/31/2022 • 2 minutes to read • [Edit Online](#)

Evaluates the year-to-date value of the **expression** in the current context.

Syntax

```
TOTALYTD(<expression>,<dates>[,<filter>][,<year_end_date>])
```

Parameters

| PARAMETER | DEFINITION |
|---------------|---|
| expression | An expression that returns a scalar value. |
| dates | A column that contains dates. |
| filter | (optional) An expression that specifies a filter to apply to the current context. |
| year_end_date | (optional) A literal string with a date that defines the year-end date. The default is December 31. |

Return value

A scalar value that represents the **expression** evaluated for the current year-to-date **dates**.

Remarks

- The **dates** argument can be any of the following:
 - A reference to a date/time column.
 - A table expression that returns a single column of date/time values.
 - A Boolean expression that defines a single-column table of date/time values.
- Constraints on Boolean expressions are described in the topic, [CALCULATE](#).
- The **filter** expression has restrictions described in the topic, [CALCULATE](#).
- The **year_end_date** parameter is a string literal of a date, in the same locale as the locale of the client where the workbook was created. The year portion of the date is not required and is ignored. For example, the following formula specifies a (fiscal) year_end_date of 6/30 in an EN-US locale workbook.

```
= TOTALYTD(SUM(InternetSales_USD[SalesAmount_USD]),DateTime[DateKey], ALL('DateTime'), "6/30")
```

In this example, year_end_date can be specified as "6/30", "Jun 30", "30 June", or any string that resolves to a month/day. However, it is recommended you specify year_end_date using "month/day" (as shown) to ensure the string resolves to a date.

- This function is not supported for use in DirectQuery mode when used in calculated columns or row-

level security (RLS) rules.

Example

The following sample formula creates a measure that calculates the 'year running total' or 'year running sum' for Internet sales.

```
= TOTALYTD(SUM(InternetSales_USD[SalesAmount_USD]),DateTime[DateKey])
```

See also

[ALL](#)

[CALCULATE](#)

[DATESYTD](#)

[TOTALMTD](#)

[TOTALQTD](#)

Statements

10/31/2022 • 2 minutes to read • [Edit Online](#)

In this category

| STATEMENT | DESCRIPTION |
|--------------------------|---|
| DEFINE | (Keyword) Introduces one or more one or more entity definitions that can be applied to one or more EVALUATE statements. |
| EVALUATE | (Keyword) Introduces a statement containing a table expression required to execute a DAX query. |
| MEASURE | (Keyword) Introduces a measure definition that can be used in one or more EVALUATE statements in a query. |
| ORDER BY | (Keyword) Introduces a statement that defines the sort order of query results returned by an EVALUATE statement. |
| START AT | (Keyword) Introduces a statement that defines the starting value at which the query results of an ORDER BY statement are returned. |
| VAR | (Keyword) Stores the result of an expression as a named variable, which can then be passed as an argument to other measure expressions. |

DEFINE

10/31/2022 • 2 minutes to read • [Edit Online](#)

Introduces a statement with one or more entity definitions that can be applied to one or more EVALUATE statements of a [DAX query](#).

Syntax

```
[DEFINE
(
  (MEASURE <table name>[<measure name>] = <scalar expression>) |
  (VAR <var name> = <table or scalar expression>) |
  (TABLE <table name> = <table expression>) |
  (COLUMN <table name>[<column name>] = <scalar expression>) |
) +
]

(EVALUATE <table expression>) +
```

Parameters

| TERM | DEFINITION |
|------------|--|
| Entity | MEASURE, VAR, TABLE ^[1] , or COLUMN ^[1] . |
| name | The name of a measure, var, table, or column definition. It cannot be an expression. The name does not have to be unique. The name exists only for the duration of the query. |
| expression | Any DAX expression that returns a table or scalar value. The expression can use any of the defined entities. If there is a need to convert a scalar expression into a table expression, wrap the expression inside a table constructor with curly braces <code>{ }</code> , or use the <code>ROW()</code> function to return a single row table. |

[1] **Caution:** Query scoped TABLE and COLUMN definitions are meant for internal use only. While you can define TABLE and COLUMN expressions for a query without syntax error, they may produce runtime errors and are not recommended.

Remarks

- A DAX query can have multiple EVALUATE statements, but can have only one DEFINE statement. Definitions in the DEFINE statement can apply to any EVALUATE statements in the query.
- At least one definition is required in a DEFINE statement.
- Measure definitions for a query override model measures of the same name.
- VAR names have unique restrictions. To learn more, see [VAR - Parameters](#).
- To learn more about how a DEFINE statement is used, see [DAX queries](#).

See also

[EVALUATE](#)

[VAR](#)

[MEASURE](#)

[DAX queries](#)

EVALUATE

10/31/2022 • 2 minutes to read • [Edit Online](#)

Introduces a statement containing a table expression required in a [DAX query](#).

Syntax

```
EVALUATE <table>
```

Parameters

| TERM | DEFINITION |
|-------|--------------------|
| table | A table expression |

Return value

The result of a table expression.

Remarks

- A DAX query can contain multiple EVALUATE statements.
- To learn more about how EVALUATE statements are used, see [DAX queries](#).

Example

```
EVALUATE  
    'Internet Sales'
```

Returns all rows and columns from the Internet Sales table, as a table.

See also

[ORDER BY](#)

[START AT](#)

[DEFINE](#)

[VAR](#)

[DAX queries](#)

MEASURE

10/31/2022 • 2 minutes to read • [Edit Online](#)

Introduces a measure definition in a DEFINE statement of a [DAX query](#).

Syntax

```
[DEFINE
    (
        MEASURE <table name>[<measure name>] = <scalar expression>
    ) +
]

(EVALUATE <table expression>) +
```

Parameters

| TERM | DEFINITION |
|-------------------|---|
| table name | The name of a table containing the measure. |
| measure name | The name of the measure. It cannot be an expression. The name does not have to be unique. The name exists only for the duration of the query. |
| scalar expression | A DAX expression that returns a scalar value. |

Return value

The calculated result of the measure expression.

Remarks

- Measure definitions for a query override model measures of the same name for the duration of the query. They will not affect the model measure.
- The measure expression can be used with any other expression in the same query.
- To learn more about how MEASURE statements are used, see [DAX queries](#).

See also

[DEFINE](#)

[EVALUATE](#)

[VAR](#)

[DAX queries](#)

ORDER BY

10/31/2022 • 2 minutes to read • [Edit Online](#)

Introduces a statement that defines sort order of query results returned by an EVALUATE statement in a [DAX query](#).

Syntax

```
[ORDER BY {<expression> [{ASC | DESC}]}[, ...]]
```

Parameters

| TERM | DEFINITION |
|------------|--|
| expression | Any DAX expression that returns a single scalar value. |
| ASC | (default) Ascending sort order. |
| DESC | Descending sort order. |

Return value

The result of an EVALUATE statement in ascending (ASC) or descending (DESC) order.

Remarks

To learn more about how ORDER BY statements are used, see [DAX queries](#).

See also

[START AT](#)

[EVALUATE](#)

[VAR](#)

[DEFINE](#)

[DAX queries](#)

START AT

10/31/2022 • 2 minutes to read • [Edit Online](#)

Introduces a statement that defines the starting value at which the query results of an ORDER BY clause in an EVALUATE statement in a [DAX query](#) are returned.

Syntax

```
[START AT {<value>|<parameter>} [, ...]]
```

Parameters

| TERM | DEFINITION |
|-----------|--|
| value | A constant value. Cannot be an expression. |
| parameter | The name of a parameter in an XMLA statement prefixed with an @ character. |

Remarks

- START AT arguments have a one-to-one correspondence with the columns in the ORDER BY statement. There can be as many arguments in the START AT statement as there are in the ORDER BY statement, but not more. The first argument in the START AT statement defines the starting value in column 1 of the ORDER BY columns. The second argument in the START AT statement defines the starting value in column 2 of the ORDER BY columns within the rows that meet the first value for column 1.
- To learn more about how START AT statements are used, see [DAX queries](#).

See also

[ORDER BY](#)

[EVALUATE](#)

[VAR](#)

[DEFINE](#)

[DAX queries](#)

VAR

10/31/2022 • 2 minutes to read • [Edit Online](#)

Stores the result of an expression as a named variable, which can then be passed as an argument to other measure expressions. Once resultant values have been calculated for a variable expression, those values do not change, even if the variable is referenced in another expression.

Syntax

```
VAR <name> = <expression>
```

Parameters

| TERM | DEFINITION |
|------------|---|
| name | The name of the variable (identifier). Delimiters are not supported. For example, 'varName' or [varName] will result in an error. Supported character set: a-z, A-Z, 0-9. 0-9 are not valid as first character. __ (double underscore) is allowed as a prefix to the identifier name. No other special characters are supported. Reserved keywords not allowed. Names of existing tables are not allowed. Empty spaces are not allowed. |
| expression | A DAX expression which returns a scalar or table value. |

Return value

A named variable containing the result of the expression argument.

Remarks

- An expression passed as an argument to VAR can contain another VAR declaration.
- When referencing a variable:
 - Measures cannot refer to variables defined outside the measure expression, but can refer to functional scope variables defined within the expression.
 - Variables can refer to measures.
 - Variables can refer to previously defined variables.
 - Columns in table variables cannot be referenced via TableName[ColumnName] syntax.
- For best practices when using VAR, see [Use variables to improve your DAX formulas](#).
- To learn more about how VAR is used within a DAX Query, see [DAX queries](#).

Example

To calculate a percentage of year-over-year growth without using a variable, you could create three separate

measures. This first measure calculates Sum of Sales Amount:

```
Sum of SalesAmount = SUM(SalesTable[SalesAmount])
```

A second measure calculates the sales amount for the previous year:

```
SalesAmount PreviousYear =  
    CALCULATE([Sum of SalesAmount],  
        SAMEPERIODLASTYEAR(Calendar[Date]))  
    )
```

You can then create a third measure that combines the other two measures to calculate a growth percentage. Notice the Sum of SalesAmount measure is used in two places; first to determine if there is a sale, then again to calculate a percentage.

```
Sum of SalesAmount YoY%: =  
    IF([Sum of SalesAmount] ,  
        DIVIDE(([Sum of SalesAmount] - [SalesAmount PreviousYear]), [Sum of SalesAmount])  
    )
```

By using a variable, you can create a single measure that calculates the same result:

```
YoY% = VAR Sales = SUM(SalesTable[SalesAmount])  
  
VAR SalesLastYear =  
    CALCULATE ( SUM ( SalesTable[SalesAmount] ), SAMEPERIODLASTYEAR ( 'Calendar'[Date] ) )  
  
return if(Sales, DIVIDE(Sales - SalesLastYear, Sales))
```

By using a variable, you can get the same outcome, but in a more readable way. And because the result of the expression is stored in the variable, the measure's performance can be significantly improved because it doesn't have to be recalculated each time it's used.

See also

[Use variables to improve your DAX formulas](#)
[DAX queries](#)

DAX glossary

10/31/2022 • 6 minutes to read • [Edit Online](#)

Analytic query

Power BI visuals query a data model by using an *analytic query*. An analytic query strives to reduce potentially large data volumes and model complexities using three distinct phases: Filter, group and summarize. An analytic query is created automatically when fields are assigned to the wells of report visuals. Report authors can control the behavior of field assignments by renaming fields, modifying the summarization technique, or disabling summarization to achieve grouping. At report design time, filters can be added to the report, a report page, or a visual. In reading view, filters can be modified in the **Filters** pane, or by interactions with slicers and other visuals (cross-filtering).

BLANK

DAX defines the absence of a value as BLANK. It's the equivalent of SQL NULL, but it doesn't behave exactly the same. It's more closely aligned to Excel and how it defines an empty cell. BLANK is evaluated as zero or an empty string when combined with other operations. For example, $\text{BLANK} + 20 = 20$. Always use capital letters; the plural is BLANKs, with a lowercase "s".

Calculated column

A model calculation used to add a column to a tabular model by writing a DAX formula. The formula must return a scalar value, and it's evaluated for each row in the table. A calculated column can be added to an Import or DirectQuery storage mode table.

Calculated measure

In tabular modeling, there's no such concept as a *calculated measure*. Use *measure* instead. The word *calculated* is used to describe calculated tables and calculated columns. It distinguishes them from tables and columns that originate from Power Query. Power Query doesn't have the concept of a measure.

Calculated table

A model calculation used to add a table to a tabular model by writing a DAX formula. The formula must return a table object. It results in a table that uses Import storage mode.

Calculation

A deliberate process that transforms one or more inputs into one or more results. In a tabular data model, a calculation can be a model object; either a calculated table, calculated column, or measure.

Context

Describes the environment in which a DAX formula is evaluated. There are two types of context: *Row context* and *filter context*. Row context represents the "current row", and is used to evaluate calculated column formulas and expressions used by table iterators. Filter context is used to evaluate measures, and it represents filters applied directly to model columns and filters propagated by model relationships.

DAX

Data Analysis Expressions (DAX) language is a formula language for Power Pivot in Excel, Power BI, Azure Analysis Services, and tabular modeling in SQL Server Analysis Services. You can also use DAX to add data model calculations and define row-level security (RLS) rules.

Dynamic security

When row-level security (RLS) rules are enforced by using the identity of the report user. Rules filter model tables by using the user's account name, which can be done with the `USERNAME` or `USERPRINCIPALNAME` functions. See [Row-level security](#).

Expression

A unit of DAX logic that's evaluated and returns a result. Expressions can declare variables in which case they're assigned a sub-expression and must include a `RETURN` statement that outputs a final expression. Expressions are constructed by using model objects (tables, columns, or measures), functions, operators, or constants.

Field

Data model resource presented in the **Fields** pane. Fields are used to configure report filters and visuals. Fields consist of model columns, hierarchy levels, and measures.

Formula

One or more DAX expressions used to define a model calculation. Inner expressions are called sub-expressions. Plural is *formulas*.

Function

DAX functions have arguments that allow passing in parameters. Formulas can use many function calls, possibly nesting functions within other functions. In a formula, function names must be followed by parentheses. Within the parentheses, parameters are passed in.

Implicit measure

An automatically generated calculation achieved by configuring a Power BI visual to summarize column values. **Numeric** columns support the greatest range of summarization, including: Sum, Average, Minimum, Maximum, Count (Distinct), Count, Standard deviation, Variance, or Median. Columns of other data types can be summarized, too. **Text** columns can be summarized by using: First (alphabetically), Last (alphabetically), Count (Distinct), or Count. **Date** columns can be summarized by using: Earliest, Latest, Count (Distinct), or Count. **Boolean** columns can be summarized by using: Count (Distinct), or Count.

Iterator function

A DAX function that enumerates all rows of a given table and evaluate a given expression for each row. It provides flexibility and control over how model calculations summarize data.

MDX

Multidimensional Expressions (MDX) language is a formula language for SQL Server Analysis Services multidimensional models (also known as *cubes*). MDX can be used to query tabular models, however it can't define implicit measures. It can only query measures that are already defined in the model.

Measure

A calculation that achieves summarization. Measures are either *implicit* or *explicit*. An explicit measure is a calculation added to a tabular data model by writing a DAX formula. A measure formula must return a scalar value. In the **Fields** pane, explicit measures are adorned with a calculator icon. Explicit measures are required when the model is queried by using Multidimensional Expressions (MDX), as is the case when using Analyze in Excel. An explicit measure is commonly just called a measure.

Measure group

A model table that contains at least one measure, and has no hierarchies or visible columns. In the **Fields** pane, each measure group is adorned with a multi-calculator icon. Measure groups are listed together at the top of the **Fields** pane, and sorted alphabetically by name.

Model calculation

A named formula that's used to add a calculated table, calculated column, or measure to a tabular data model. Its structure is <NAME> = <FORMULA>. Most calculations are added by data modelers in Power BI Desktop, but measures can also be added to a live connection report. See [Report measures](#).

Quick measures

A feature in Power BI Desktop that eliminates the need to write DAX formulas for commonly defined measures. Quick measures include average per category, rank, and difference from baseline.

Report measures

Also called *report-level measures*. They're added to a live connection report in Power BI Desktop by writing a DAX formula, but only for connections to Power BI models or Analysis Services tabular models.

Row-level security

Also called *RLS*. Design technique to restrict access to subsets of data for specific users. In a tabular model, it's achieved by creating model roles. Roles have rules, which are DAX expressions to filter table rows.

Scalar

In DAX, a scalar is a single value. A scalar can be of any data type: Decimal, Integer, DateTime, String, Currency, Boolean. A scalar value can be the result of an expression calculated from multiple values. For example, an aggregation function such as MAX() returns a single maximum value from a set of values from which to evaluate.

Summarization

An operation applied to the values of a column. See [measure](#).

Time intelligence

Time intelligence relates to calculations over time, like year-to-date (YTD).

Time intelligence function

DAX includes many time intelligence functions. Each time intelligence function achieves its result by modifying the filter context for date filters. Example functions: TOTALYTD and SAMEPERIODLASTYEAR.

Value, values

Data to be visualized.

What-if parameter

A Power BI Desktop feature that provides the ability to accept user input through slicers. Each parameter creates a single-column calculated table and a measure that returns a single-selected value. The measure can be used in model calculations to respond to the user's input.

DAX operators

10/31/2022 • 6 minutes to read • [Edit Online](#)

The Data Analysis Expression (DAX) language uses operators to create expressions that compare values, perform arithmetic calculations, or work with strings.

Types of operators

There are four different types of calculation operators: arithmetic, comparison, text concatenation, and logical.

Arithmetic operators

To perform basic mathematical operations such as addition, subtraction, or multiplication; combine numbers; and produce numeric results, use the following arithmetic operators.

| ARITHMETIC OPERATOR | MEANING | EXAMPLE |
|---------------------|---------------------|---------|
| + (plus sign) | Addition | 3+3 |
| – (minus sign) | Subtraction or sign | 3–1–1 |
| * (asterisk) | Multiplication | 3*3 |
| / (forward slash) | Division | 3/3 |
| ^ (caret) | Exponentiation | 16^4 |

NOTE

The plus sign can function both as a *binary operator* and as a *unary operator*. A binary operator requires numbers on both sides of the operator and performs addition. When you use values in a DAX formula on both sides of the binary operator, DAX tries to cast the values to numeric data types if they are not already numbers. In contrast, the unary operator can be applied to any type of argument. The plus symbol does not affect the type or value and is simply ignored, whereas the minus operator creates a negative value, if applied to a numeric value.

Comparison operators

You can compare two values with the following operators. When two values are compared by using these operators, the result is a logical value, either TRUE or FALSE.

| COMPARISON OPERATOR | MEANING | EXAMPLE |
|---------------------|--------------------------|-----------------------------|
| = | Equal to | [Region] = "USA" |
| == | Strict equal to | [Region] == "USA" |
| > | Greater than | [Sales Date] > "Jan 2009" |
| < | Less than | [Sales Date] < "Jan 1 2009" |
| >= | Greater than or equal to | [Amount] >= 20000 |

| COMPARISON OPERATOR | MEANING | EXAMPLE |
|---------------------|-----------------------|-------------------|
| <= | Less than or equal to | [Amount] <= 100 |
| <> | Not equal to | [Region] <> "USA" |

All comparison operators except == treat BLANK as equal to number 0, empty string "", DATE(1899, 12, 30), or FALSE. As a result, [Column] = 0 will be true when the value of [Column] is either 0 or BLANK. In contrast, [Column] == 0 is true only when the value of [Column] is 0.

Text concatenation operator

Use the ampersand (&) to join, or concatenate, two or more text strings to produce a single piece of text.

| TEXT OPERATOR | MEANING | EXAMPLE |
|---------------|--|--------------------------|
| & (ampersand) | Connects, or concatenates, two values to produce one continuous text value | [Region] & ", " & [City] |

Logical operators

Use logical operators (&&) and (||) to combine expressions to produce a single result.

| TEXT OPERATOR | MEANING | EXAMPLES |
|-----------------------|---|---|
| && (double ampersand) | Creates an AND condition between two expressions that each have a Boolean result. If both expressions return TRUE, the combination of the expressions also returns TRUE; otherwise the combination returns FALSE. | (([Region] = "France") && ([BikeBuyer] = "yes")) |
| (double pipe symbol) | Creates an OR condition between two logical expressions. If either expression returns TRUE, the result is TRUE; only when both expressions are FALSE is the result FALSE. | ((([Region] = "France") ([BikeBuyer] = "yes")) |
| IN | Creates a logical OR condition between each row being compared to a table. Note: the table constructor syntax uses curly braces. | 'Product'[Color] IN { "Red", "Blue", "Black" } |

Operators and precedence order

In some cases, the order in which calculation is performed can affect the Return value; therefore, it is important to understand how the order is determined and how you can change the order to obtain the desired results.

Calculation order

An expression evaluates the operators and values in a specific order. All expressions always begin with an equal sign (=). The equal sign indicates that the succeeding characters constitute an expression.

Following the equal sign are the elements to be calculated (the operands), which are separated by calculation operators. Expressions are always read from left to right, but the order in which the elements are grouped can be controlled to some degree by using parentheses.

Operator precedence

If you combine several operators in a single formula, the operations are ordered according to the following table. If the operators have equal precedence value, they are ordered from left to right. For example, if an expression contains both a multiplication and division operator, they are evaluated in the order that they appear in the expression, from left to right.

| OPERATOR | DESCRIPTION |
|----------------------------------|--|
| \wedge | Exponentiation |
| $-$ | Sign (as in -1) |
| $*$ and $/$ | Multiplication and division |
| $+$ and $-$ | Addition and subtraction |
| $\&$ | Connects two strings of text (concatenation) |
| $=, <, >, <=, >=, <>, \text{IN}$ | Comparison |
| NOT | NOT (unary operator) |

Using parentheses to control calculation order

To change the order of evaluation, you should enclose in parentheses that part of the formula that must be calculated first. For example, the following formula produces 11 because multiplication is calculated before addition. The formula multiplies 2 by 3, and then adds 5 to the result.

`=5+2*3`

In contrast, if you use parentheses to change the syntax, the order is changed so that 5 and 2 are added together, and the result multiplied by 3 to produce 21.

`=(5+2)*3`

In the following example, the parentheses around the first part of the formula force the calculation to evaluate the expression `(3 + 0.25)` first and then divide the result by the result of the expression, `(3 - 0.25)`.

`=(3 + 0.25)/(3 - 0.25)`

In the following example, the exponentiation operator is applied first, according to the rules of precedence for operators, and then the sign operator is applied. The result for this expression is -4.

`=-2^2`

To ensure that the sign operator is applied to the numeric value first, you can use parentheses to control operators, as shown in the following example. The result for this expression is 4.

`= (-2)^2`

Compatibility

DAX easily handles and compares various data types, much like Microsoft Excel. However, the underlying computation engine is based on SQL Server Analysis Services and provides additional advanced features of a relational data store, including richer support for date and time types. Therefore, in some cases the results of calculations or the behavior of functions may not be the same as in Excel. Moreover, DAX supports more data types than does Excel. This section describes the key differences.

Coercing data types of operands

In general, the two operands on the left and right sides of any operator should be the same data type. However, if the data types are different, DAX will convert them to a common data type to apply the operator in some cases:

1. Both operands are converted to the largest possible common data type.
2. The operator is applied, if possible.

For example, suppose you have two numbers that you want to combine. One number results from a formula, such as `= [Price] * .20`, and the result may contain many decimal places. The other number is an integer that has been provided as a string value.

In this case, DAX will convert both numbers to real numbers in a numeric format, using the largest numeric format that can store both kinds of numbers. Then DAX will apply the multiplication.

Depending on the data-type combination, type coercion may not be applied for comparison operations. For a complete list of data types supported by DAX, see [Data types supported in tabular models](#) and [Data types in Power BI Desktop](#).

Integer, Real Number, Currency, Date/time and Blank are considered numeric for comparison purposes. Blank evaluates to zero when performing a comparison. The following data-type combinations are supported for comparison operations.

| LEFT SIDE DATA TYPE | RIGHT SIDE DATA TYPE |
|---------------------|----------------------|
| Numeric | Numeric |
| Boolean | Boolean |
| String | String |

Other mixed data-type comparisons will return an error. For example, a formula such as `"1" > 0` returns an error stating that *DAX comparison operations do not support comparing values of type Text with values of type Integer*.

| DATA TYPES USED IN DAX | DATA TYPES USED IN EXCEL |
|------------------------|--------------------------|
| Numbers (I8, R8) | Numbers (R8) |
| String | String |
| Boolean | Boolean |
| DateTime | Variant |
| Currency | Currency |

Differences in precedence order

The precedence order of operations in DAX formulas is basically the same as that used by Microsoft Excel, but

some Excel operators are not supported, such as percent. Also, ranges are not supported.

Therefore, whenever you copy and paste formulas from Excel, be sure to review the formula carefully, as some operators or elements in the formulas may not be valid. When there is any doubt about the order in which operations are performed, it's recommended you use parentheses to control the order of operations and remove any ambiguity about the result.

See also

[DAX syntax](#)

[DAX parameter-naming](#)

DAX queries

10/31/2022 • 5 minutes to read • [Edit Online](#)

Reporting clients like Power BI and Excel execute DAX queries whenever a field is placed on a report or when a filter is applied. By using [SQL Server Management Studio](#) (SSMS), [Power BI Report Builder](#), and open-source tools like [DAX Studio](#), you can create and run your own DAX queries. DAX queries return results as a table right within the tool, allowing you to quickly create and test the performance of your DAX formulas.

Before learning about queries, it's important you have a solid understanding of DAX basics. If you haven't already, be sure to checkout [DAX overview](#).

Keywords

DAX queries have a simple syntax comprised of just one required keyword, **EVALUATE**, and several optional keywords: **ORDER BY**, **START AT**, **DEFINE**, **MEASURE**, **VAR**, **TABLE**, and **COLUMN**. Each keyword defines a statement used for the duration of the query.

EVALUATE (Required)

At the most basic level, a DAX query is an **EVALUATE** statement containing a table expression. At least one **EVALUATE** statement is required, however, a query can contain any number of **EVALUATE** statements.

EVALUATE Syntax

```
EVALUATE <table>
```

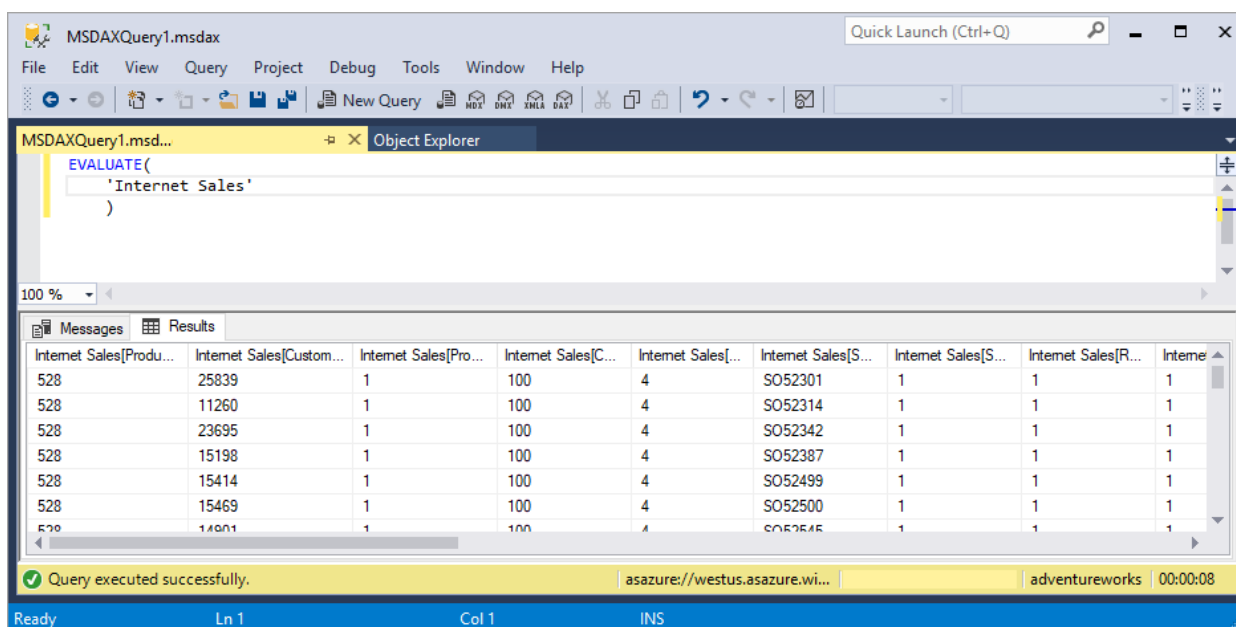
EVALUATE Parameters

| TERM | DEFINITION |
|-------|---------------------|
| table | A table expression. |

EVALUATE Example

```
EVALUATE  
    'Internet Sales'
```

Returns all rows and columns from the Internet Sales table, as a table.



ORDER BY (Optional)

The optional **ORDER BY** keyword defines one or more expressions used to sort query results. Any expression that can be evaluated for each row of the result is valid.

ORDER BY Syntax

```
EVALUATE <table>
[ORDER BY {<expression> [{ASC | DESC}]}[, ...]]
```

ORDER BY Parameters

| TERM | DEFINITION |
|------------|--|
| expression | Any DAX expression that returns a single scalar value. |
| ASC | (default) Ascending sort order. |
| DESC | Descending sort order. |

ORDER BY Example

```
EVALUATE
    'Internet Sales'

ORDER BY
    'Internet Sales'[Order Date]
```

Returns all rows and columns from the Internet Sales table, in ascending order by Order Date, as a table.

The screenshot shows the MSDAX Query tool interface. The query editor contains the following code:

```

EVALUATE(
    'Internet Sales'
)
ORDER BY
    'Internet Sales'[Order Date]
  
```

The results pane displays a table with the following columns: Internet Sales[S...], Internet Sales[T...], Internet Sales[Fr...], Internet Sales[C...], Internet Sales[C...], Internet Sales[Order Date], Internet Sales[D...], Internet Sales[S...], and Internet S... The data rows show various sales figures and dates, with the 'Order Date' column highlighted in yellow.

At the bottom, a status bar indicates: "Query executed successfully. azure://westus.azure.wi... adventureworks 00:00:08".

START AT (Optional)

The optional START AT keyword is used inside an ORDER BY clause. It defines the value at which the query results begin.

START AT Syntax

```

EVALUATE <table>
[ORDER BY {<expression> [{ASC | DESC}]}[, ...]
[START AT {<value>|<parameter>} [, ...]]
  
```

START AT Parameters

| TERM | DEFINITION |
|-----------|--|
| value | A constant value. Cannot be an expression. |
| parameter | The name of a parameter in an XMLA statement prefixed with an @ character. |

START AT Remarks

START AT arguments have a one-to-one correspondence with the columns in the ORDER BY clause. There can be as many arguments in the START AT clause as there are in the ORDER BY clause, but not more. The first argument in the START AT defines the starting value in column 1 of the ORDER BY columns. The second argument in the START AT defines the starting value in column 2 of the ORDER BY columns within the rows that meet the first value for column 1.

START AT Example

```

EVALUATE
    'Internet Sales'

ORDER BY
    'Internet Sales'[Sales Order Number]
START AT "SO7000"
  
```

Returns all rows and columns from the Internet Sales table, in ascending order by Sales Order Number, beginning at SO7000.

The screenshot shows the Microsoft Dynamics 365 Data Explorer interface. The top menu bar includes File, Edit, View, Query, Project, Debug, Tools, Window, and Help. The main window displays a DAX query in the 'Query Editor' tab:

```

EVALUATE(
    'Internet Sales'
)
ORDER BY
    'Internet Sales'[Sales Order Number]
START AT "SO70000"
  
```

Below the query editor, the 'Results' tab shows a table with 10 columns and 5 rows of data. The columns are: Internet Sales[Pr..., Internet Sales[C..., Internet Sales[Pr..., Internet Sales[C..., Internet Sales[S..., Internet Sales[S..., Internet Sales[S..., Internet Sales[R..., and Internet Sales[O... The data rows show various sales order numbers and their corresponding values.

At the bottom of the interface, a status bar indicates 'Query executed successfully.' and provides a link to 'azure://westus.azure.wi...'. The bottom right corner shows 'adventureworks' and a timer '00:00:03'.

DEFINE (Optional)

The optional **DEFINE** keyword introduces one or more calculated entity definitions that exist only for the duration of the query. Definitions precede the **EVALUATE** statement and are valid for all **EVALUATE** statements in the query. Definitions can be variables, measures, tables¹, and columns¹. Definitions can reference other definitions that appear before or after the current definition. At least one definition is required if the **DEFINE** keyword is included in a query.

DEFINE Syntax

```

[DEFINE
    (
        (MEASURE <table name>[<measure name>] = <scalar expression>) |
        (VAR <var name> = <table or scalar expression>) |
        (TABLE <table name> = <table expression>) |
        (COLUMN <table name>[<column name>] = <scalar expression>) |
    ) +
]

(EVALUATE <table expression>) +
  
```

DEFINE Parameters

| TERM | DEFINITION |
|------------|--|
| Entity | MEASURE, VAR, TABLE ¹ , or COLUMN ¹ . |
| name | The name of a measure, var, table, or column definition. It cannot be an expression. The name does not have to be unique. The name exists only for the duration of the query. |
| expression | Any DAX expression that returns a table or scalar value. The expression can use any of the defined entities. If there is a need to convert a scalar expression into a table expression, wrap the expression inside a table constructor with curly braces <code>{ }</code> , or use the <code>ROW()</code> function to return a single row table. |

[1] **Caution:** Query scoped **TABLE** and **COLUMN** definitions are meant for internal use only. While you can define **TABLE** and **COLUMN** expressions for a query without syntax error, they may produce runtime errors and are not recommended.

DEFINE Remarks

- A DAX query can have multiple EVALUATE statements, but can have only one DEFINE statement. Definitions in the DEFINE statement can apply to any EVALUATE statements in the query.
- At least one definition is required in a DEFINE statement.
- Measure definitions for a query override model measures of the same name but are only used within the query. They will not affect the model measure.
- VAR names have unique restrictions. To learn more, see [VAR - Parameters](#).

DEFINE Example

```
DEFINE
    MEASURE 'Internet Sales'[Internet Total Sales] =
        SUM ( 'Internet Sales'[Sales Amount] )

EVALUATE
    SUMMARIZECOLUMNS (
        'Date'[Calendar Year],
        TREATAS (
            {
                2013,
                2014
            },
            'Date'[Calendar Year]
        ),
        "Total Sales", [Internet Total Sales],
        "Combined Years Total Sales",
            CALCULATE (
                [Internet Total Sales],
                ALLSELECTED ( 'Date'[Calendar Year] )
            )
    )
    ORDER BY [Calendar Year]
```

Returns the calculated total sales for years 2013 and 2014, and combined calculated total sales for years 2013 and 2014, as a table. The measure in the DEFINE statement, Internet Total Sales, is used in both Total Sales and Combined Years Total Sales expressions.

The screenshot shows the SQL Server Data Tools (SSDT) interface. The query editor displays the following DAX query:

```
DEFINE
MEASURE 'Internet Sales'[Internet Total Sales] = SUM('Internet Sales'[Sales Amount])
EVALUATE
SUMMARIZECOLUMNS
(
    'Date'[Calendar Year],
    TREATAS({2013, 2014}, 'Date'[Calendar Year]),
    "Total Sales", [Internet Total Sales],
    "Combined Years Total Sales", CALCULATE([Internet Total Sales], ALLSELECTED('Date'[Calendar Year]))
)
ORDER BY [Calendar Year]
```

The Results pane shows the following table:

| Date[Calendar Y... | [Total Sales] | [Combined Years... |
|--------------------|---------------|--------------------|
| 2013 | 16351550.34 | 16397245.06 |
| 2014 | 45694.72 | 16397245.06 |

The status bar at the bottom indicates "Query executed successfully." and shows the file path "asazure://westus.azure.wi..." and the database "adventureworks".

Parameters in DAX queries

A well-defined DAX query statement can be parameterized and then used over and over with just changes in the parameter values.

The [Execute Method \(XMLA\)](#) method has a [Parameters Element \(XMLA\)](#) collection element that allows parameters to be defined and assigned a value. Within the collection, each [Parameter Element \(XMLA\)](#) element defines the name of the parameter and a value to it.

Reference XMLA parameters by prefixing the name of the parameter with an `@` character. Any place in the syntax where a value is allowed, the value can be replaced with a parameter call. All XMLA parameters are typed as text.

IMPORTANT

Parameters defined in the parameters section and not used in the `<STATEMENT>` element generate an error response in XMLA. Parameters used and not defined in the `<Parameters>` element generate an error response in XMLA.

See also

[DAX statements](#)

[SUMMARIZECOLUMNS](#)

[TREATAS](#)

[FILTER](#)

DAX parameter-naming conventions

10/31/2022 • 2 minutes to read • [Edit Online](#)

Parameter names are standardized in DAX reference to facilitate the usage and understanding of the functions.

Parameter names

| TERM | DEFINITION |
|------------|---|
| expression | Any DAX expression that returns a single scalar value, where the expression is to be evaluated multiple times (for each row/context). |
| value | Any DAX expression that returns a single scalar value where the expression is to be evaluated exactly once before all other operations. |
| table | Any DAX expression that returns a table of data. |
| tableName | The name of an existing table using standard DAX syntax. It cannot be an expression. |
| columnName | The name of an existing column using standard DAX syntax, usually fully qualified. It cannot be an expression. |
| name | A string constant that will be used to provide the name of a new object. |
| order | An enumeration used to determine the sort order. |
| ties | An enumeration used to determine the handling of tie values. |
| type | An enumeration used to determine the data type for PathItem and PathItemReverse. |

Prefixing parameter names or using the prefix only

| TERM | DEFINITION |
|-----------|---|
| prefixing | <p>Parameter names may be further qualified with a prefix that is descriptive of how the argument is used and to avoid ambiguous reading of the parameters. For example:</p> <p>Result_ColumnName - Refers to an existing column used to get the result values in the LOOKUPVALUE() function.</p> <p>Search_ColumnName - Refers to an existing column used to search for a value in the LOOKUPVALUE() function.</p> |

| TERM | DEFINITION |
|----------|--|
| omitting | <p>Parameter names will be omitted if the prefix is clear enough to describe the parameter.</p> <p>For example, instead of having the following syntax DATE (Year_Value, Month_Value, Day_Value) it is clearer for the user to read DATE (Year, Month, Day); repeating three times the suffix value does not add anything to a better comprehension of the function and it clutters the reading unnecessarily.</p> <p>However, if the prefixed parameter is Year_columnName then the parameter name and the prefix will stay to make sure the user understands that the parameter requires a reference to an existing column of Years.</p> |

DAX syntax

10/31/2022 • 9 minutes to read • [Edit Online](#)

This article describes syntax and requirements for the DAX formula expression language.

Syntax requirements

A DAX formula always starts with an equal sign (=). After the equals sign, you can provide any expression that evaluates to a scalar, or an expression that can be converted to a scalar. These include the following:

- A scalar constant, or expression that uses a scalar operator (+, -, *, /, >=, <, &&, ...)
- References to columns or tables. The DAX language always uses tables and columns as inputs to functions, never an array or arbitrary set of values.
- Operators, constants, and values provided as part of an expression.
- The result of a function and its required arguments. Some DAX functions return a table instead of a scalar, and must be wrapped in a function that evaluates the table and returns a scalar; unless the table is a single column, single row table, then it is treated as a scalar value.

Most DAX functions require one or more arguments, which can include tables, columns, expressions, and values. However, some functions, such as PI, do not require any arguments, but always require parentheses to indicate the null argument. For example, you must always type PI(), not PI. You can also nest functions within other functions.

- Expressions. An expression can contain any or all of the following: operators, constants, or references to columns.

For example, the following are all valid formulas.

| FORMULA | RESULT |
|--|---|
| = 3 | 3 |
| = "Sales" | Sales |
| = 'Sales'[Amount] | If you use this formula within the Sales table, you will get the value of the column Amount in the Sales table for the current row. |
| = (0.03 *[Amount]) =0.03 * [Amount] | Three percent of the value in the Amount column of the current table. Although this formula can be used to calculate a percentage, the result is not shown as a percentage unless you apply formatting in the table. |
| = PI() | The value of the constant pi. |

Formulas can behave differently depending on how they are used. You must always be aware of the context and how the data that you use in the formula is related to other data that might be used in the calculation.

Naming requirements

A data model often contains multiple tables. Together the tables and their columns comprise a database stored in the in-memory analytics engine (VertiPaq). Within that database, all tables must have unique names. The names of columns must also be unique within each table. All object names are *case-insensitive*; for example, the names **SALES** and **Sales** would represent the same table.

Each column and measure you add to an existing data model must belong to a specific table. You specify the table that contains the column either implicitly, when you create a calculated column within a table, or explicitly, when you create a measure and specify the name of the table where the measure definition should be stored.

When you use a table or column as an input to a function, you must generally *qualify* the column name. The *fully qualified* name of a column is the table name, followed by the column name in square brackets: for examples, 'U.S. Sales'[Products]. A fully qualified name is always required when you reference a column in the following contexts:

- As an argument to the function, VALUES
- As an argument to the functions, ALL or ALLEXCEPT
- In a filter argument for the functions, CALCULATE or CALCULATETABLE
- As an argument to the function, RELATEDTABLE
- As an argument to any time intelligence function

An *unqualified* column name is just the name of the column, enclosed in brackets: for example, [Sales Amount]. For example, when you are referencing a scalar value from the same row of the current table, you can use the unqualified column name.

If the name of a table contains spaces, reserved keywords, or disallowed characters, you must enclose the table name in single quotation marks. You must also enclose table names in quotation marks if the name contains any characters outside the ANSI alphanumeric character range, regardless of whether your locale supports the character set or not. For example, if you open a workbook that contains table names written in Cyrillic characters, such as 'Таблица', the table name must be enclosed in quotation marks, even though it does not contain spaces.

NOTE

To make it easier to enter the fully qualified names of columns, use the AutoComplete feature in the formula editor.

Tables

- Table names are required whenever the column is from a different table than the current table. Table names must be unique within the database.
- Table names must be enclosed in single quotation marks if they contain spaces, other special characters or any non-English alphanumeric characters.

Measures

- Measure names must always be in brackets.
- Measure names can contain spaces.
- Each measure name must be unique within a model. Therefore, the table name is optional in front of a measure name when referencing an existing measure. However, when you create a measure you must always specify a table where the measure definition will be stored.

Columns

Column names must be unique in the context of a table; however, multiple tables can have columns with the same names (disambiguation comes with the table name).

In general, columns can be referenced without referencing the base table that they belong to, except when there might be a name conflict to resolve or with certain functions that require column names to be fully qualified.

Reserved keywords

If the name that you use for a table is the same as an Analysis Services reserved keyword, an error is raised, and you must rename the table. However, you can use keywords in object names if the object name is enclosed in brackets (for columns) or quotation marks (for tables).

NOTE

Quotation marks can be represented by several different characters, depending on the application. If you paste formulas from an external document or Web page, make sure to check the ASCII code of the character that is used for opening and closing quotes, to ensure that they are the same. Otherwise DAX may be unable to recognize the symbols as quotation marks, making the reference invalid.

Special characters

The following characters and character types are not valid in the names of tables, columns, or measures:

- Leading or trailing spaces; unless the spaces are enclosed by name delimiters, brackets, or single apostrophes.
- Control characters
- The following characters that are not valid in the names of objects:

.,':/^*|?&%\$!+=()[]{}<>

Examples of object names

The following table shows examples of some object names:

| OBJECT TYPES | EXAMPLES | COMMENT |
|------------------------------|-----------------------|---|
| Table name | Sales | If the table name does not contain spaces or other special characters, the name does not need to be enclosed in quotation marks. |
| Table name | 'Canada Sales' | If the name contains spaces, tabs or other special characters, enclose the name in single quotation marks. |
| Fully qualified column name | Sales[Amount] | The table name precedes the column name, and the column name is enclosed in brackets. |
| Fully qualified measure name | Sales[Profit] | The table name precedes the measure name, and the measure name is enclosed in brackets. In certain contexts, a fully qualified name is always required. |

| OBJECT TYPES | EXAMPLES | COMMENT |
|---|---------------------|---|
| Unqualified column name | [Amount] | The unqualified name is just the column name, in brackets. Contexts where you can use the unqualified name include formulas in a calculated column within the same table, or in an aggregation function that is scanning over the same table. |
| Fully qualified column in table with spaces | 'Canada Sales'[Qty] | The table name contains spaces, so it must be surrounded by single quotes. |

Other restrictions

The syntax required for each function, and the type of operation it can perform, varies greatly depending on the function. In general, however, the following rules apply to all formulas and expressions:

- DAX formulas and expressions cannot modify or insert individual values in tables.
- You cannot create calculated rows by using DAX. You can create only calculated columns and measures.
- When defining calculated columns, you can nest functions to any level.
- DAX has several functions that return a table. Typically, you use the values returned by these functions as input to other functions, which require a table as input.

DAX operators and constants

The following table lists the operators that are supported by DAX. For more information about the syntax of individual operators, see [DAX operators](#).

| OPERATOR TYPE | SYMBOL AND USE |
|----------------------|---|
| Parenthesis operator | () precedence order and grouping of arguments |
| Arithmetic operators | + (addition) - (subtraction/ sign) * (multiplication) / (division) ^ (exponentiation) |
| Comparison operators | = (equal to) > (greater than) < (less than) >= (greater than or equal to) <= (less than or equal to) <> (not equal to) |

| OPERATOR TYPE | SYMBOL AND USE |
|-----------------------------|-------------------|
| Text concatenation operator | & (concatenation) |
| Logic operators | && (and) |
| | (or) |

Data types

You do not need to cast, convert, or otherwise specify the data type of a column or value that you use in a DAX formula. When you use data in a DAX formula, DAX automatically identifies the data types in referenced columns and of the values that you type in, and performs implicit conversions where necessary to complete the specified operation.

For example, if you try to add a number to a date value, the engine will interpret the operation in the context of the function, and convert the numbers to a common data type, and then present the result in the intended format, a date.

However, there are some limitations on the values that can be successfully converted. If a value or a column has a data type that is incompatible with the current operation, DAX returns an error. Also, DAX does not provide functions that let you explicitly change, convert, or cast the data type of existing data that you have imported into a data model.

IMPORTANT

DAX does not support use of the variant data type. Therefore, when you load or import data into a data model, it's expected the data in each column is generally of a consistent data type.

Some functions return scalar values, including strings, whereas other functions work with numbers, both integers and real numbers, or dates and times. The data type required for each function is described in the section, [DAX functions](#).

You can use tables containing multiple columns and multiple rows of data as the argument to a function. Some functions also return tables, which are stored in memory and can be used as arguments to other functions.

Date and time

DAX stores date and time values using the datetime data type used by Microsoft SQL Server. Datetime format uses a floating-point number where Date values correspond to the integer portion representing the number of days since December 30, 1899. Time values correspond to the decimal portion of a date value where Hours, minutes, and seconds are represented by decimal fractions of a day. DAX date and time functions implicitly convert arguments to datetime data type.

NOTE

The exact maximum DateTime value supported by DAX is December 31, 9999 00:00:00.

Date and time literal

Beginning with the August 2021 version of Power BI Desktop, DAX date and datetime values can be specified as a literal in the format `dt"YYYY-MM-DD"`, `dt"YYYY-MM-DDThh:mm:ss"`, or `dt"YYYY-MM-DD hh:mm:ss"`. When specified as a literal, use of [DATE](#), [TIME](#), [DATEVALUE](#), [TIMEVALUE](#) functions in the expression are not necessary.

For example, the following expression uses DATE and TIME functions to filter on OrderDate:

```
EVALUATE
FILTER (
    FactInternetSales,
    [OrderDate] > (DATE(2015,1,9) + TIME(2,30,0)) && [OrderDate] < (DATE(2015,12,31) + TIME(11,59,59))
)
```

The same filter expression can be specified as a literal:

```
EVALUATE
FILTER (
    FactInternetSales,
    [OrderDate] > dt"2015-1-9T02:30:00" && [OrderDate] < dt"2015-12-31T11:59:59"
)
```

NOTE

The DAX date and datetime-typed literal format is not supported in all versions of Power BI Desktop, Analysis Services, and Power Pivot in Excel. New and updated DAX functionality are typically first introduced in Power BI Desktop and then later included in Analysis Services and Power Pivot in Excel.