Contents

GEOMEANX

MAX

MAXA

MAXX

MEDIAN

MEDIANX

MIN

MINA

MINX

NORM.DIST

NORM.INV

NORM.S.DIST

NORM.S.INV

PERCENTILE.EXC

PERCENTILE.INC

PERCENTILEX.EXC

PERCENTILEX.INC

PERMUT

POISSON.DIST

RANK.EQ

RANKX

ROW

SAMPLE

SELECTCOLUMNS

SIN

SINH

STDEV.S

STDEV.P

STDEVX.S

STDEVX.P

SQRTPI

SUMMARIZE

Data Analysis Expressions (DAX) is a library of functions and operators that can be combined to build formulas and expressions i
Power BI Desktop, Azure Analysis Services, SQL Server Analysis Services, and Power Pivot in Excel.

DAX functions

Get detailed information for any of the over 200 DAX functions.

Learn DAX videos

Learn quickly with introductory and advanced videos.

DAX overview

Get the big picture. This article describes the most essential concepts in DAX.

DAX syntax

Learn about syntax and expression requirements in DAX.

DAX operators

Learn how operators are used in DAX expressions that compare values, perform arithmetic calculations, or work with strings.

DAX queries

Use query tools to create an evaluate statement that queries your data model.

DAX overview

12/18/2018 • 30 minutes to read

Data Analysis Expressions (DAX) is a formula expression language used in Analysis Services, Power BI Desktop, 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 it's particular implementation of DAX.

Calculations

DAX calculation formulas are used in measures, calculated columns, calculated tables, and row filters.

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 the Values window in a PivotTable, and then places the Product Category column from a Product table into the Filters window, the sum of Sales Amount is calculated and displayed for each product category.

Unlike calculated columns and row filters, 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 Field 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

Measures in Power Pivot

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. Because a calculated column is created in a table in the data model, they're not supported in models that retrieve data exclusively from a relational data source using DirectQuery mode.

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 either a DAX query or 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 filters (Row-level security)

In row filters, also known as 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 filters are not available in Power Pivot in Excel.

When defining a row filter 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 filters allow access to that particular row set, the user can view data for that row.

Row filters 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 filters will be intersected with other row filters defined for related tables.

To learn more, see:
Row-level security (RLS) with Power BI
Row filters 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 filters. To create formulas for calculated columns and measures, you will use the formula bar along the top of the model designer window or the DAX Editor. To create formulas for row filters, you will use the Role Manager 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	Description
`=TODAY()`	Inserts today's date in every row of a calculated column.
`=3`	Inserts the value 3 in every row of a calculated column.
`=[Column1] + [Column2]`	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 will be 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

Both the formula bar in the model designer and the formula Row Filters window in the Role Manager dialog box provide an AutoComplete feature. 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

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.

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 the **datetime** data types used by Microsoft SQL Server. For more information, see Date and time functions.

Filter functions

The filter functions in DAX return specific data types, look up values in related tales, 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. For more information, see Filter 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. For more information, 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. For more information, 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. For more information, 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. For more information, see Other functions.

Statistical functions

DAX provides statistical functions that perform aggregations. In addition to creating sums and averages, or finding the minimum and maximum values, in DAX you can also filter a column before aggregating or create aggregations based on related tables. For more information, see Statistical functions.

Text functions

The 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. For more information, 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. For more information, see Time-intelligence functions (DAX).

Table-valued functions

There are DAX functions that output tables, take tables as input, or do both. Because a table can have a single column, table-valued functions also take single columns as inputs. Understanding how to use table-valued

functions is important for fully utilizing DAX formulas. DAX includes the following types of table-valued functions:

- Filter functions return a column, table, or values related to the current row.
- Aggregation functions aggregate any expression over the rows of a table.
- Time-intelligence functions return a table of dates, or use a table of dates to calculate an aggregation.

Examples of table functions include: FILTER, ALL, VALUES, DISTINCT, RELATEDTABLE.

Variables

You can create variables within an expression by using the VAR. VAR is technically not a function, it's a keyword you use 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 (
        TotalQuantity > 1000,
        TotalQuantity * 0.95,
        TotalQuantity * 1.25
        )
```

In this example, TotalQty can then 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]	Real numbers are numbers that can have decimal places. Real numbers cover a wide range of values: Negative values from -1.79E +308 through -2.23E -308 Zero Positive values from 2.23E -308 through 1.79E + 308 However, the number of significant digits is limited to 17 decimal digits.
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	Dates and times in an accepted date-time representation. Valid dates are all dates after March 1, 1900.
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. When a user places a measure or other value field into a PivotTable or into a report based on a tabular model, the engine examines the row and column headers, Slicers, and report filters to determine the context. Then, the necessary queries are run against the data source to get the correct subset of data, make the calculations defined by the formula, and then populate each cell in the PivotTable or report. The set of data that is retrieved is the query context for each cell.

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 PivotTable, context can be changed by adding or removing column and row headers and by adding or removing Slicers. Whenever users add column or row headings to the PivotTable, they change the query context in which the measure is evaluated. Slicing and filtering operations also affect context. 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.

For more information 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). For more information 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 to look up data in another table and use related values to perform complex calculations. For example, you can use a calculated column to look up 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* parameters.

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. For more information, 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 pivot table 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 filter formulas unless the result of a recalculation returns a different value, thus making the row queryable or not queryable by role members.

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 parameters 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

SQL Server Data Tools (SSDT) is an essential tool for creating and deploying Analysis Services data models. The model designer includes a DAX editor for creating DAX calculation formulas.
Analysis Services Projects extension (VSIX) includes the same functionality in SSDT to create Analysis Services modeling projects. Do not install packages if SSDT is already installed.

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.

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:

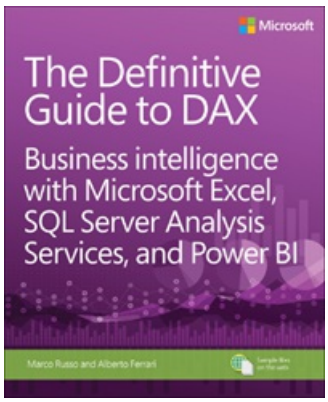
Videos

DAX basics in Power BI Desktop

QuickStart: Learn DAX Basics in 30 Minutes (Power Pivot in Excel)

DAX Resource Center wiki

The Definitive Guide to DAX by Alberto Ferrari and Marco Russo (Microsoft Press). This extensive guide provides basics to innovative high-performance techniques for beginning data modelers and BI professionals.



Videos

12/10/2018 • 2 minutes to read

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 function reference

3/29/2019 • 2 minutes to read

This function reference provides detailed information including syntax, parameters, Return values, and examples for each of the over 200 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.

Examples

Most reference articles contain examples showing formulas and results created in an Excel workbook with a Power Pivot for Excel data model. The data model is connected to an AdventureWorksDW sample database as a datasource. The sample workbook is no longer available.

This reference is not intended to serve as a tutorial or provide in-depth guidance on how to create formulas for data models connected to a specific datasource.

In this section

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

Date and time functions (DAX) - 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.

Time-intelligence functions (DAX) - 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.

Filter functions (DAX) - 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.

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

Logical functions (DAX) - 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 (DAX) - Mathematical functions in DAX are similar to Excel's mathematical and trigonometric functions. However, t99 are some differences in the numeric data types used by DAX functions.

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

Parent and Child functions (DAX) - These Data Analysis Expressions (DAX) functions help users manage data that is presented as a parent/child hierarchy in their data models.

Statistical functions (DAX) - These functions perform aggregations. In addition to creating sums and averages, or finding minimum and maximum values, in DAX you can also filter a column before aggregating or create

aggregations based on related tables.

Text functions (DAX) - 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.

See also

DAX Syntax Reference
DAX Operator Reference
DAX Parameter-Naming Conventions

New DAX functions

4/1/2019 • 2 minutes to read

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. SQL Server Analysis Services and Excel are updated in the next cumulative update. To get the latest, make sure you're using the latest version.

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.

> **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.

New functions

FUNCTION	MONTH
ALLCROSSFILTERED	April 2019
USERCULTURE	April 2019
CONTAINSSTRING	March 2019
CONTAINSSTRINGEXACT	March 2019
DISTINCTCOUNTNOBLANK	March 2019
APPROXIMATEDISTINCTCOUNT (Preview)	Dec. 2018
ISINSCOPE	Nov. 2018
NORM.DIST	August 2018
NORM.INV	August 2018
NORM.S.DIST	August 2018
NORM.S.INV	August 2018
T.DIST	August 2018
T.DIST.2T	August 2018
T.DIST.RT	August 2018

FUNCTION	MONTH
T.INV	August 2018
T.INV.2T	August 2018
DISTINCT (table)	April 2018

Updated functions

FUNCTION	MONTH
ALL	March 2019
LOOKUPVALUE	March 2019
SUMMARIZECOLUMNS function	Nov. 2018

Date and time functions

12/10/2018 • 2 minutes to read

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 section

CALENDAR

12/10/2018 • 2 minutes to read

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.

Examples

The following formula returns a table with dates between January 1st, 2005 and December 31st, 2015.

```
=CALENDAR (DATE (2005, 1, 1), DATE (2015, 12, 31))
```

For a data model which includes actual sales data and future sales forecasts. The following expression returns the date table covering the range of dates in these two tables.

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

CALENDARAUTO

12/10/2018 • 2 minutes to read

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.

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 February 28, 2012.

DATE

12/10/2018 • 4 minutes to read

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

Syntax

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

Parameters

TERM	DEFINITION
year	A number representing the year. 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. Dates beginning with March 1, 1900 are supported. If you enter a number that has decimal places, the number is rounded. For values greater than 9999 or less than zero (negative values), the function returns a **#VALUE!** error. 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.
month	A number representing the month or a calculation according to the following rules: 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. 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. If you enter a negative integer, the following computation occurs: the date is calculated subtracting the value of **month** from **year**. For example, if you have DATE(2008, -6, 15), the function returns a datetime value equivalent to June 15th of 2007, because when 6 months are subtracted from the beginning of 2008 it yields a value of June 2007. See examples below.

TERM	DEFINITION
day	A number representing the day or a calculation according to the following rules: If **day** is a number from 1 to the last day of the given month then it represents a day of the month. 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 `DATE( 2008, 3, 32)`, 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. If you enter a negative integer, the following computation occurs: the date is calculated subtracting the value of **day** from **month**. For example, in the formula `DATE( 2008, 5, -15)`, the DATE function returns a **datetime** value equivalent to April 15th of 2008, because 15 days are subtracted from the beginning of May 2008 yielding a value of April 2008. If **day** contains a decimal portion, it is rounded to the nearest integer value.

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.

This DAX function may return different results when used in a model that is deployed and then queried in DirectQuery mode. For more information about semantic differences in DirectQuery mode, see https://go.microsoft.com/fwlink/?LinkId=219171.

Example: Returning a Simple Date

Description

The following formula returns the date July 8, 2009:

Code

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

Example: Years before 1899

Description

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).

Code

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

Example: Years before 1899

Description

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, 3700: (1900+1800).

Code

```
=DATE(1800,1,2)
```

Example: Years after 1899

Description

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

Code

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

Example: Working with Months

Description

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:

Code

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

Comment

If the **month** value is less than 1, the DATE function subtracts the magnitude of that number of months, plus 1, from the first month in the year specified. The following formula returns September 2, 2007:

```
=DATE(2008,-3,2)
```

Example: Working with Days

Description

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:

Code

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

Comment

If **day** is less than 1, **day** subtracts the magnitude that number of days, plus one, from the first day of the month specified. The following formula returns December 16, 2007:

```
=DATE(2008,1,-15)
```

See also

Date and time functions (DAX)
DAY function (DAX)
TODAY function (DAX)

DATEDIFF

12/10/2018 • 2 minutes to read

Returns the count of interval boundaries crossed between two dates.

Syntax

```
DATEDIFF(<start_date>, <end_date>, <interval>)
```

Parameters

TERM	DEFINITION
start_date	A scalar datetime value.
end_date	A scalar datetime value Return value.
interval	The interval to use when comparing dates. The value can be one of the following: - SECOND - MINUTE - HOUR - DAY - WEEK - MONTH - QUARTER - YEAR

Return value

The count of interval boundaries crossed between two dates.

Remarks

An error is returned if start_date is larger than end_date.

Example

DATE
2012-12-31 23:59:59
2013-01-01 00:00:00

The following all return 1:

```
DATEDIFF(MIN( Calendar[Date] ), MAX( Calendar[Date], second ) )

DATEDIFF(MIN( Calendar[Date] ), MAX( Calendar[Date], minute ) )

DATEDIFF(MIN( Calendar[Date] ), MAX( Calendar[Date], hour ) )

DATEDIFF(MIN( Calendar[Date] ), MAX( Calendar[Date], day ) )

DATEDIFF(MIN( Calendar[Date] ), MAX( Calendar[Date], week ) )

DATEDIFF(MIN( Calendar[Date] ), MAX( Calendar[Date], month ) )

DATEDIFF(MIN( Calendar[Date] ), MAX( Calendar[Date], quarter ) )

DATEDIFF(MIN( Calendar[Date] ), MAX( Calendar[Date], year ) )
```

DATEVALUE

12/10/2018 • 2 minutes to read

Converts a date in the form of text 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

The DATEVALUE function uses the locale and date/time settings of the client computer to understand the text value when performing the conversion. If the current date/time settings represent dates in the format of Month/Day/Year, then the string, "1/8/2009", would be converted to a **datetime** value equivalent to January 8th of 2009. However, if the current date and time settings represent dates in the format of Day/Month/Year, the same string would be converted as a **datetime** value equivalent to August 1st of 2009.

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.

Example

The following example returns a different **datetime** value depending on your computer's 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 (DAX)

DAY

12/10/2018 • 2 minutes to read

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

Description

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

Code

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

Example: Getting the Day from a String Date

Description

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

Code

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

Example: Using a Day Value as a Condition

Description

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.

Code

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

See also

Date and time functions (DAX)
TODAY function (DAX)
DATE function (DAX)

EDATE

12/10/2018 • 2 minutes to read

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 DAX function may return different results when used in a model that is deployed and then queried in DirectQuery mode. For more information about semantic differences in DirectQuery mode, see https://go.microsoft.com/fwlink/?LinkId=219171.

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 (DAX)
Date and time functions (DAX)

EOMONTH

12/10/2018 • 2 minutes to read

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 DAX function may return different results when used in a model that is deployed and then queried in DirectQuery mode. For more information about semantic differences in DirectQuery mode, see https://go.microsoft.com/fwlink/?LinkId=219171.

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 (DAX)
Date and time functions (DAX)

HOUR

12/10/2018 • 2 minutes to read

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

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

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

Example

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 (DAX)
MINUTE function (DAX)

YEAR function (DAX)
SECOND function (DAX)

MINUTE

12/10/2018 • 2 minutes to read

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

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

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

Example

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 (DAX)
HOUR function (DAX)

YEAR function (DAX)
SECOND function (DAX)

MONTH

12/10/2018 • 2 minutes to read

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

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

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 (DAX)
HOUR function (DAX)
MINUTE function (DAX)
YEAR function (DAX)
SECOND function (DAX)

NOW

12/10/2018 • 2 minutes to read

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.

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 (DAX)

SECOND

12/10/2018 • 2 minutes to read

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 of 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

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

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 (DAX)
HOUR function (DAX)
MINUTE function (DAX)
YEAR function (DAX)

TIME

12/10/2018 • 2 minutes to read

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

Syntax

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

Parameters

TERM	DEFINITION
hour	A number from 0 to 23 representing the hour. Any value greater than 23 will be divided by 24 and the remainder will be treated as the hour value.
minute	A number from 0 to 59 representing the minute. Any value greater than 59 will be converted to hours and minutes.
second	A number from 0 to 59 representing the second. Any value greater than 59 will be converted to hours, minutes, and seconds.

Return value

A time (**datetime**).

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. The following restrictions apply:

- Any value for **hours** that is greater than 23 will be divided by 24 and the remainder will be treated as the hour value.
- Any value for **minutes** that is greater than 59 will be converted to hours and minutes.
- Any value for **seconds** that is greater than 59 will be converted to hours, minutes, and seconds.
- For minutes or seconds, a value greater than 24 hours will be divided by 24 and the reminder will be treated

as the hour value. A value in excess of 24 hours does not alter the date portion.

To improve readability of the time values returned by this function, we recommend that you format the column or PivotTable cell that contains the results of the formula by using one of the time formats provided by Microsoft Excel.

This DAX function may return different results when used in a model that is deployed and then queried in DirectQuery mode. For more information about semantic differences in DirectQuery mode, see https://go.microsoft.com/fwlink/?LinkId=219171.

Example

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

```
=TIME(27,0,0)
=TIME(3,0,0)
```

Example

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

```
=TIME(0,750,0)
=TIME(12,30,0)
```

Example

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

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

See also

DATE function (DAX)
Date and time functions (DAX)

TIMEVALUE function

12/10/2018 • 2 minutes to read

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 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 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

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

See also

Date and time functions (DAX)

TODAY

12/10/2018 • 2 minutes to read

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.

> **NOTE**
> 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 (DAX)
NOW function (DAX)

UTCNOW

12/10/2018 • 2 minutes to read

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 (DAX)
UTCTODAY function (DAX)

UTCTODAY

12/10/2018 • 2 minutes to read

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 (DAX)
UTCNOW function (DAX)

WEEKDAY

12/10/2018 • 2 minutes to read

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

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 (DAX)
WEEKNUM function (DAX)
YEARFRAC function (DAX)

WEEKNUM

12/10/2018 • 2 minutes to read

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

Syntax

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

Parameters

TERM	DEFINITION
date	The date in **datetime** format.
return_type	A number that determines the Return value: use 1 when the week begins on Sunday; use 2 when the week begins on Monday. The default is 1. Return type: **1**, week begins on Sunday. Weekdays are numbered 1 through 7. Return type: **2**, week begins on Monday. Weekdays are numbered 1 through 7.

Return value

An integer number.

Remarks

In contrast to Microsoft Excel, which stores dates as serial numbers, DAX uses a **datetime** data type to work with dates and times. If the source data is in a different format, DAX implicitly converts the data to **datetime** to perform calculations.

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 for 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.

Example

The following example returns the week number of the date February 14, 2010.

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

Example

The following example returns the week number of the date stored in the column, **HireDate**, from the table, **Employees**.

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

See also

Date and time functions (DAX)
YEARFRAC function (DAX)
WEEKDAY function (DAX)

YEAR

12/10/2018 • 2 minutes to read

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.

Code

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

See also

Date and time functions (DAX)
HOUR function (DAX)
MINUTE function (DAX)
YEAR function (DAX)
SECOND function (DAX)

YEARFRAC

12/10/2018 • 2 minutes to read

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	(Optional) The type of day count basis to use. All arguments are truncated to integers. Basis - Description 0 - US (NASD) 30/360 1 - Actual/actual 2 - Actual/360 3 - Actual/365 4 - European 30/360

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

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

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 (DAX)
WEEKNUM function (DAX)
YEARFRAC function (DAX)
WEEKDAY function (DAX)

Time-intelligence functions

12/10/2018 • 2 minutes to read

Data Analysis Expressions (DAX) includes time intelligence functions to support the needs of Business Intelligence analysis by enabling you to manipulate data using time periods, including days, months, quarters, and years, and then build and compare calculations over those periods.

In this section

PREVIOUSQUARTER

PREVIOUSYEAR

SAMEPERIODLASTYEAR

STARTOFMONTH

STARTOFQUARTER

STARTOFYEAR

TOTALMTD

TOTALQTD

TOTALYTD

CLOSINGBALANCEMONTH

3/29/2019 • 2 minutes to read

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 the topic, CALCULATE function (DAX).

> **NOTE**
> The **filter** expression has restrictions described in the topic, CALCULATE function (DAX).

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see https://go.microsoft.com/fwlink/?LinkId=219172.

Example

The following sample formula creates a measure that calculates the 'Month End Inventory Value' of the product inventory.

To see how this works, create a PivotTable and add the fields, CalendarYear, MonthNumberOfYear and DayNumberOfMonth, to the **Row Labels** area of the PivotTable. Then add a measure, named **Month End Inventory Value**, using the formula defined in the code section, to the **Values** area of the PivotTable.

Code

```
=CLOSINGBALANCEMONTH(SUMX(ProductInventory,ProductInventory[UnitCost]*ProductInventory[UnitsBalance]),DateTime
[DateKey])
```

See also

Time-intelligence functions (DAX)
CLOSINGBALANCEYEAR function (DAX)
CLOSINGBALANCEQUARTER function (DAX)

CLOSINGBALANCEQUARTER

3/29/2019 • 2 minutes to read

Evaluates the **expression** at the last date of the quarter in the current context.

Syntax

```
CLOSINGBALANCEQUARTER(<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 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 the topic, CALCULATE function (DAX).

> **NOTE**
> The **filter** expression has restrictions described in the topic, CALCULATE function (DAX).

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see https://go.microsoft.com/fwlink/?LinkId=219172.

Example

The following sample formula creates a measure that calculates the 'Quarter End Inventory Value' of the product

inventory.

To see how this works, create a PivotTable and add the fields, CalendarYear, CalendarQuarter and MonthNumberOfYear, to the **Row Labels** area of the PivotTable. Then add a measure, named **Quarter End Inventory Value**, using the formula defined in the code section, to the **Values** area of the PivotTable.

Code

```
=CLOSINGBALANCEQUARTER(SUMX(ProductInventory,ProductInventory[UnitCost]*ProductInventory[UnitsBalance]),DateTi
me[DateKey])
```

See also

Time-intelligence functions (DAX)
CLOSINGBALANCEYEAR function (DAX)
CLOSINGBALANCEMONTH function (DAX)

CLOSINGBALANCEYEAR

3/29/2019 • 2 minutes to read

Evaluates the **expression** at the last date of the year in the current context.

Syntax

```
CLOSINGBALANCEYEAR(<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 at the last 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.

> **NOTE**
> Constraints on Boolean expressions are described in the topic, CALCULATE function (DAX).

> **NOTE**
> The **filter** expression has the restrictions described in the topic, CALCULATE function (DAX).

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 DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see https://go.microsoft.com/fwlink/?LinkId=219172.

Example

The following sample formula creates a measure that calculates the 'Year End Inventory Value' of the product inventory.

To see how this works, create a PivotTable and add the field, CalendarYear, to the **Row Labels** area of the PivotTable. Then add a measure, named **Year End Inventory Value**, using the formula defined in the code section, to the **Values** area of the PivotTable.

Code

```
=CLOSINGBALANCEYEAR(SUMX(ProductInventory,ProductInventory[UnitCost]*ProductInventory[UnitsBalance]),DateTime[
DateKey])
```

See also

Time-intelligence functions (DAX)
CLOSINGBALANCEYEAR function (DAX)
CLOSINGBALANCEQUARTER function (DAX)
CLOSINGBALANCEMONTH function (DAX)

DATEADD

3/29/2019 • 2 minutes to read

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: `year`, `quarter`, `month`, `day`

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 (DAX).

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 DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see https://go.microsoft.com/fwlink/?LinkId=219172.

Example: Shifting a Set of Dates

Description

The following formula calculates dates that are one year before the dates in the current context.

Code

```
=DATEADD(DateTime[DateKey],-1,year)
```

See also

Time-intelligence functions (DAX)
Date and time functions (DAX)

DATESBETWEEN

12/10/2018 • 2 minutes to read

Returns a table that contains a column of dates that begins with the **start_date** and continues until the **end_date**.

Syntax

```
DATESBETWEEN(<dates>,<start_date>,<end_date>)
```

Parameters

Term	Definition
dates	A reference to a date/time column.
start_date	A date expression.
end_date	A date expression.

Return value

A table containing a single column of date values.

Remarks

If **start_date** is a blank date value, then **start_date** will be the earliest value in the **dates** column.

If **end_date** is a blank date value, then **end_date** will be the latest value in the **dates** column.

The dates used as the **start_date** and **end_date** are inclusive: that is, if the sales occurred on September 1 and you use September 1 as the start date, sales on September 1 are counted.

> **NOTE**
> The DATESBETWEEN function is provided for working with custom date ranges. If you are working with common date intervals such as months, quarters, and years, we recommend that you use the appropriate function, such as DATESINPERIOD.

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see https://go.microsoft.com/fwlink/?LinkId=219172.

Example

The following sample formula creates a measure that calculates the 'Summer 2007 sales' for the Internet sales.

To see how this works, create a PivotTable and add the field, CalendarYear, to the **Row Labels** area of the PivotTable. Then add a measure, named **Summer 2007 Sales**, using the formula as defined in the code section, to the **Values** area of the PivotTable.

```
=CALCULATE(SUM(InternetSales_USD[SalesAmount_USD]), DATESBETWEEN(DateTime[DateKey],
    DATE(2007,6,1),
    DATE(2007,8,31)
  ))
```

See also

Time-intelligence functions (DAX)
Date and time functions (DAX)
DATESINPERIOD function (DAX)

DATESINPERIOD

3/29/2019 • 2 minutes to read

Returns a table that contains a column of dates that begins with the **start_date** and continues for the specified **number_of_intervals**.

Syntax

```
DATESINPERIOD(<dates>,<start_date>,<number_of_intervals>,<interval>)
```

Parameters

Term	Definition
dates	A column that contains dates.
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: `year`, `quarter`, `month`, `day`

Return value

A table containing a single column of date values.

Remarks

The **dates** argument can be a reference to a date/time column.

> **NOTE**
> Constraints on Boolean expressions are described in the topic, CALCULATE function (DAX).

If the number specified for **number_of_intervals** is positive, the dates are moved forward in time; if the number is negative, the 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 appear in the values of the underlying table column.

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see https://go.microsoft.com/fwlink/?LinkId=219172.

Example

The following formula returns the Internet sales for the 21 days prior to August 24, 2007.

```
= CALCULATE(SUM(InternetSales_USD[SalesAmount_USD]),DATESINPERIOD(DateTime[DateKey],DATE(2007,08,24),-21,day))
```

See also

Time-intelligence functions (DAX)
Date and time functions (DAX)
DATESBETWEEN function (DAX)

DATESMTD

3/29/2019 • 2 minutes to read

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.

Property Value/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 (DAX).

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see https://go.microsoft.com/fwlink/?LinkId=219172.

Example

The following sample formula creates a measure that calculates the 'Month To Date Total' for the Internet sales.

To see how this works, create a PivotTable and add the fields, CalendarYear, MonthNumberOfYear and DayNumberOfMonth, to the **Row Labels** area of the PivotTable. Then add a measure, named **Month To Date Total**, using the formula defined in the code section, to the **Values** area of the PivotTable.

```
=CALCULATE(SUM(InternetSales_USD[SalesAmount_USD]), DATESMTD(DateTime[DateKey]))
```

See also

Time-intelligence functions (DAX)
Date and time functions (DAX)
DATESYTD function (DAX)
DATESQTD function (DAX)

DATESQTD

3/29/2019 • 2 minutes to read

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.

Property Value/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 (DAX).

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see https://go.microsoft.com/fwlink/?LinkId=219172.

Example

The following sample formula creates a measure that calculates the 'Quarterly Running Total' of the internet sales.

To see how this works, create a PivotTable and add the fields, CalendarYear, CalendarQuarter and MonthNumberOfYear to the **Row Labels** area of the PivotTable. Then add a measure, named **Quarterly Running Total**, using the formula defined in the code section, to the **Values** area of the PivotTable.

```
=CALCULATE(SUM(InternetSales_USD[SalesAmount_USD]), DATESQTD(DateTime[DateKey]))
```

See also

Time-intelligence functions (DAX)
Date and time functions (DAX)
DATESYTD function (DAX)
DATESMTD function (DAX)

DATESYTD

3/29/2019 • 2 minutes to read

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.

Property Value/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 (DAX).

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 DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see https://go.microsoft.com/fwlink/?LinkId=219172.

Example

The following sample formula creates a measure that calculates the 'Running Total' for the Internet sales.

To see how this works, create a PivotTable and add the fields, CalendarYear and CalendarQuarter, to the **Row Labels** area of the PivotTable. Then add a measure named **Running Total**, using the formula defined in the code section, to the **Values** area of the PivotTable.

```
=CALCULATE(SUM(InternetSales_USD[SalesAmount_USD]), DATESYTD(DateTime[DateKey]))
```

See also

Time-intelligence functions (DAX)
Date and time functions (DAX)
DATESMTD function (DAX)
DATESQTD function (DAX)

ENDOFMONTH

12/10/2018 • 2 minutes to read

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.

> **NOTE**
> Constraints on Boolean expressions are described in the topic, CALCULATE function (DAX).

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see https://go.microsoft.com/fwlink/?LinkId=219172.

Example

The following sample formula creates a measure that returns the end of the month, for the current context.

To see how this works, create a PivotTable and add the fields CalendarYear and MonthNumberOfYear to the **Row Labels** area of the PivotTable. Then add a measure, named **EndOfMonth**, using the formula defined in the code section, to the **Values** area of the PivotTable.

```
=ENDOFMONTH(DateTime[DateKey])
```

See also

Date and time functions (DAX)
Time-intelligence functions (DAX)
ENDOFYEAR function (DAX)
ENDOFQUARTER function (DAX)

ENDOFQUARTER

12/10/2018 • 2 minutes to read

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.

> **NOTE**
> Constraints on Boolean expressions are described in the topic, CALCULATE function (DAX).

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see https://go.microsoft.com/fwlink/?LinkId=219172.

Example

The following sample formula creates a measure that returns the end of the quarter, for the current context.

To see how this works, create a PivotTable and add the fields CalendarYear and MonthNumberOfYear to the **Row Labels** area of the PivotTable. Then add a measure, named **EndOfQuarter**, using the formula defined in the code section, to the **Values** area of the PivotTable.

```
=ENDOFQUARTER(DateTime[DateKey])
```

See also

Date and time functions (DAX)
Time-intelligence functions (DAX)
ENDOFYEAR function (DAX)
ENDOFMONTH function (DAX)

ENDOFYEAR

12/10/2018 • 2 minutes to read

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.

> **NOTE**
> Constraints on Boolean expressions are described in the topic, CALCULATE function (DAX).

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 DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see https://go.microsoft.com/fwlink/?LinkId=219172.

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.

To see how this works, create a PivotTable and add the field CalendarYear to the **Row Labels** area of the PivotTable. Then add a measure, named **EndOfFiscalYear**, using the formula defined in the code section, to the

Values area of the PivotTable.

```
=ENDOFYEAR(DateTime[DateKey],"06/30/2004")
```

See also

Date and time functions (DAX)
Time-intelligence functions (DAX)
ENDOFMONTH function (DAX)
ENDOFQUARTER function (DAX)

FIRSTDATE

3/29/2019 • 2 minutes to read

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.

> **NOTE**
> Constraints on Boolean expressions are described in the topic, CALCULATE function (DAX).

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 DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see https://go.microsoft.com/fwlink/?LinkId=219172.

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.

To see how this works, create a PivotTable and add the field CalendarYear to the **Row Labels** area of the PivotTable. Then add a measure, named **FirstSaleDate**, using the formula defined in the code section, to the

Values area of the PivotTable.

```
=FIRSTDATE('InternetSales_USD'[SaleDateKey])
```

See also

Date and time functions (DAX)
Time-intelligence functions (DAX)
LASTDATE function (DAX)
FIRSTNONBLANK function (DAX)

FIRSTNONBLANK

3/29/2019 • 2 minutes to read

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**.

Property Value/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 .

> **NOTE**
> Constraints on Boolean expressions are described in the topic, CALCULATE function (DAX).

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 DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see https://go.microsoft.com/fwlink/?LinkId=219172.

See also

LASTNONBLANK function (DAX)
Statistical functions (DAX)
DAX function reference

LASTDATE

3/29/2019 • 2 minutes to read

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.

> **NOTE**
> Constraints on Boolean expressions are described in the topic, CALCULATE function (DAX).

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 DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see https://go.microsoft.com/fwlink/?LinkId=219172.

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.

To see how this works, create a PivotTable and add the field CalendarYear to the **Row Labels** area of the PivotTable. Then add a measure, named **LastSaleDate**, using the formula defined in the code section, to the **Values** area of the PivotTable.

```
=LASTDATE('InternetSales_USD'[SaleDateKey])
```

See also

Date and time functions (DAX)
Time-intelligence functions (DAX)
FIRSTDATE function (DAX)
LASTNONBLANK function (DAX)

LASTNONBLANK

3/29/2019 • 2 minutes to read

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**.

Property Value/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

> **NOTE**
> Constraints on Boolean expressions are described in the topic, CALCULATE function (DAX).

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 DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see https://go.microsoft.com/fwlink/?LinkId=219172.

See also

FIRSTNONBLANK function (DAX)
Statistical functions (DAX)
DAX function reference

NEXTDAY

12/10/2018 • 2 minutes to read

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.

> **NOTE**
> Constraints on Boolean expressions are described in the topic, CALCULATE function (DAX).

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see https://go.microsoft.com/fwlink/?LinkId=219172.

Example

The following sample formula creates a measure that calculates the 'next day sales' of the internet sales.

To see how this works, create a PivotTable and add the fields, CalendarYear and MonthNumberOfYear, to the **Row Labels** area of the PivotTable. Then add a measure, named **Next Day Sales**, using the formula defined in the code section, to the **Values** area of the PivotTable.

```
=CALCULATE(SUM(InternetSales_USD[SalesAmount_USD]), NEXTDAY('DateTime'[DateKey]))
```

See also

Time-intelligence functions (DAX)
Date and time functions (DAX)
NEXTQUARTER function (DAX)
NEXTMONTH function (DAX)
NEXTYEAR function (DAX)

NEXTMONTH

3/29/2019 • 2 minutes to read

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.

> **NOTE**
> Constraints on Boolean expressions are described in the topic, CALCULATE function (DAX).

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see https://go.microsoft.com/fwlink/?LinkId=219172.

Example

The following sample formula creates a measure that calculates the 'next month sales' for the Internet sales.

To see how this works, create a PivotTable and add the fields, CalendarYear and MonthNumberOfYear, to the **Row Labels** area of the PivotTable. Then add a measure, named **Next Month Sales**, using the formula defined in the code section, to the **Values** area of the PivotTable.

```
=CALCULATE(SUM(InternetSales_USD[SalesAmount_USD]), NEXTMONTH('DateTime'[DateKey]))
```

See also

Time-intelligence functions (DAX)
Date and time functions (DAX)
NEXTDAY function (DAX)
NEXTQUARTER function (DAX)
NEXTYEAR function (DAX)

NEXTQUARTER

3/29/2019 • 2 minutes to read

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.

> **NOTE**
> Constraints on Boolean expressions are described in the topic, CALCULATE function (DAX).

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see https://go.microsoft.com/fwlink/?LinkId=219172.

Example

The following sample formula creates a measure that calculates the 'next quarter sales' for the Internet sales.

To see how this works, create a PivotTable and add the fields, CalendarYear and CalendarQuarter, to the **Row Labels** area of the PivotTable. Then add a measure, named **Next Quarter Sales**, using the formula defined in the code section to the **Values** area of the PivotTable.

```
=CALCULATE(SUM(InternetSales_USD[SalesAmount_USD]), NEXTQUARTER('DateTime'[DateKey]))
```

See also

Time-intelligence functions (DAX)
Date and time functions (DAX)
NEXTDAY function (DAX)
NEXTMONTH function (DAX)
NEXTYEAR function (DAX)

NEXTYEAR

3/29/2019 • 2 minutes to read

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.

> **NOTE**
> Constraints on Boolean expressions are described in the topic, CALCULATE function (DAX).

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 DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see https://go.microsoft.com/fwlink/?LinkId=219172.

Example

The following sample formula creates a measure that calculates the 'next year sales' for the Internet sales.

To see how this works, create a PivotTable and add the fields, CalendarYear and CalendarQuarter, to the **Row Labels** area of the PivotTable. Then add a measure, named **Next Year Sales**, using the formula defined in the code section, to the **Values** area of the PivotTable.

```
=CALCULATE(SUM(InternetSales_USD[SalesAmount_USD]), NEXTYEAR('DateTime'[DateKey]))
```

See also

Time-intelligence functions (DAX)
Date and time functions (DAX)
NEXTDAY function (DAX)
NEXTQUARTER function (DAX)
NEXTMONTH function (DAX)

OPENINGBALANCEMONTH

3/29/2019 • 2 minutes to read

Evaluates the **expression** at the first date of the month in the current context.

Syntax

```
OPENINGBALANCEMONTH(<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 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.

> **NOTE**
> Constraints on Boolean expressions are described in the topic, CALCULATE function (DAX).

> **NOTE**
> The **filter** expression has restrictions described in the topic, CALCULATE function (DAX).
>
> This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see https://go.microsoft.com/fwlink/?LinkId=219172.

Example

The following sample formula creates a measure that calculates the 'Month Start Inventory Value' of the product

inventory.

To see how this works, create a PivotTable and add the fields, CalendarYear, MonthNumberOfYear and DayNumberOfMonth, to the **Row Labels** area of the PivotTable. Then add a measure, named **Month Start Inventory Value**, using the formula defined in the code section, to the **Values** area of the PivotTable.

Code

```
=OPENINGBALANCEMONTH(SUMX(ProductInventory,ProductInventory[UnitCost]*ProductInventory[UnitsBalance]),DateTime
[DateKey])
```

See also

OPENINGBALANCEYEAR function (DAX)
OPENINGBALANCEQUARTER function (DAX)
Time-intelligence functions (DAX)
CLOSINGBALANCEMONTH function (DAX)

OPENINGBALANCEQUARTER

3/29/2019 • 2 minutes to read

Evaluates the **expression** at the first date of the quarter, in the current context.

Syntax

```
OPENINGBALANCEQUARTER(<expression>,<dates>[,<filter>])
```

Parameters

Parameter	Definition
expression	An expression that returns a scalar value.
dates	A column that contains dates.
filte*	(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.

> **NOTE**
> Constraints on Boolean expressions are described in the topic, CALCULATE function (DAX).

> **NOTE**
> The **filter** expression has restrictions described in the topic, CALCULATE function (DAX).
>
> This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see https://go.microsoft.com/fwlink/?LinkId=219172.

Example

The following sample formula creates a measure that calculates the 'Quarter Start Inventory Value' of the product

inventory.

To see how this works, create a PivotTable and add the fields, CalendarYear, CalendarQuarter and MonthNumberOfYear, to the **Row Labels** area of the PivotTable. Then add a measure, named **Quarter Start Inventory Value**, using the formula defined in the code section, to the **Values** area of the PivotTable.

Code

```
=OPENINGBALANCEQUARTER(SUMX(ProductInventory,ProductInventory[UnitCost]*ProductInventory[UnitsBalance]),DateTi
me[DateKey])
```

See also

OPENINGBALANCEYEAR function (DAX)
OPENINGBALANCEMONTH function (DAX)
Time-intelligence functions (DAX)
CLOSINGBALANCEQUARTER function (DAX)

OPENINGBALANCEYEAR

3/29/2019 • 2 minutes to read

Evaluates the **expression** at the first date of the year in the current context.

Syntax

```
OPENINGBALANCEYEAR(<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 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.

> **NOTE**
> Constraints on Boolean expressions are described in the topic, CALCULATE function (DAX).

> **NOTE**
> The **filter** expression has restrictions described in the topic, CALCULATE function (DAX).

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 DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see https://go.microsoft.com/fwlink/?LinkId=219172.

Example

The following sample formula creates a measure that calculates the 'Year Start Inventory Value' of the product inventory.

To see how this works, create a PivotTable and add the field, CalendarYear, to the **Row Labels** area of the PivotTable. Then add a measure, named **Year Start Inventory Value**, using the formula defined in the code section, to the **Values** area of the PivotTable.

Code

```
=OPENINGBALANCEYEAR(SUMX(ProductInventory,ProductInventory[UnitCost]*ProductInventory[UnitsBalance]),DateTime[
DateKey])
```

See also

OPENINGBALANCEQUARTER function (DAX)
OPENINGBALANCEMONTH function (DAX)
Time-intelligence functions (DAX)
CLOSINGBALANCEYEAR function (DAX)

PARALLELPERIOD

3/29/2019 • 2 minutes to read

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: `year`, `quarter`, `month`.

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.

> **NOTE**
> Constraints on Boolean expressions are described in the topic, CALCULATE function (DAX).

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 DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see https://go.microsoft.com/fwlink/?LinkId=219172.

Example

The following sample formula creates a measure that calculates the previous year sales for Internet sales.

To see how this works, create a PivotTable and add the fields, CalendarYear and CalendarQuarter, to the **Row Labels** area of the PivotTable. Then add a measure, named **Previous Year Sales**, using the formula defined in the code section, to the **Values** area of the PivotTable.

```
=CALCULATE(SUM(InternetSales_USD[SalesAmount_USD]), PARALLELPERIOD(DateTime[DateKey],-1,year))
```

See also

Time-intelligence functions (DAX)
Date and time functions (DAX)
DATEADD function (DAX)

PREVIOUSDAY

12/10/2018 • 2 minutes to read

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.

> **NOTE**
> Constraints on Boolean expressions are described in the topic, CALCULATE function (DAX).

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see https://go.microsoft.com/fwlink/?LinkId=219172.

Example

The following sample formula creates a measure that calculates the 'previous day sales' for the Internet sales.

To see how this works, create a PivotTable and add the fields, CalendarYear and MonthNumberOfYear, to the **Row Labels** area of the PivotTable. Then add a measure, named **Previous Day Sales**, using the formula defined in the code section, to the **Values** area of the PivotTable.

```
=CALCULATE(SUM(InternetSales_USD[SalesAmount_USD]), PREVIOUSDAY('DateTime'[DateKey]))
```

See also

Time-intelligence functions (DAX)
Date and time functions (DAX)
PREVIOUSMONTH function (DAX)
PREVIOUSQUARTER function (DAX)
PREVIOUSYEAR function (DAX)

PREVIOUSMONTH

3/29/2019 • 2 minutes to read

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.

> **NOTE**
> Constraints on Boolean expressions are described in the topic, CALCULATE function (DAX).

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see https://go.microsoft.com/fwlink/?LinkId=219172.

Example

The following sample formula creates a measure that calculates the 'previous month sales' for the Internet sales.

To see how this works, create a PivotTable and add the fields, CalendarYear and MonthNumberOfYear, to the **Row Labels** area of the PivotTable. Then add a measure, named **Previous Month Sales**, using the formula defined in the code section, to the **Values** area of the PivotTable.

```
=CALCULATE(SUM(InternetSales_USD[SalesAmount_USD]), PREVIOUSMONTH('DateTime'[DateKey]))
```

See also

Time-intelligence functions (DAX)
Date and time functions (DAX)
PREVIOUSDAY function (DAX)
PREVIOUSQUARTER function (DAX)
PREVIOUSYEAR function (DAX)

PREVIOUSQUARTER

3/29/2019 • 2 minutes to read

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.

> **NOTE**
> Constraints on Boolean expressions are described in the topic, CALCULATE function (DAX).

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see https://go.microsoft.com/fwlink/?LinkId=219172.

Example

The following sample formula creates a measure that calculates the 'previous quarter sales' for Internet sales.

To see how this works, create a PivotTable and add the fields, CalendarYear and CalendarQuarter, to the **Row Labels** area of the PivotTable. Then add a measure, named **Previous Quarter Sales**, using the formula defined in the code section, to the **Values** area of the PivotTable.

```
=CALCULATE(SUM(InternetSales_USD[SalesAmount_USD]), PREVIOUSQUARTER('DateTime'[DateKey]))
```

See also

Time-intelligence functions (DAX)
Date and time functions (DAX)
PREVIOUSMONTH function (DAX)
PREVIOUSDAY function (DAX)
PREVIOUSYEAR function (DAX)

PREVIOUSYEAR

12/10/2018 • 2 minutes to read

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.

> **NOTE**
> Constraints on Boolean expressions are described in the topic, CALCULATE function (DAX).

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 DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see https://go.microsoft.com/fwlink/?LinkId=219172.

Example

The following sample formula creates a measure that calculates the previous year sales for the Internet sales.

To see how this works, create a PivotTable and add the fields, CalendarYear and CalendarQuarter, to the **Row Labels** area of the PivotTable. Then add a measure, named **Previous Year Sales**, using the formula defined in the code section, to the **Values** area of the PivotTable.

```
=CALCULATE(SUM(InternetSales_USD[SalesAmount_USD]), PREVIOUSYEAR('DateTime'[DateKey]))
```

See also

Time-intelligence functions (DAX)
Date and time functions (DAX)
PREVIOUSMONTH function (DAX)
PREVIOUSDAY function (DAX)
PREVIOUSQUARTER function (DAX)

SAMEPERIODLASTYEAR

12/10/2018 • 2 minutes to read

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.

Property Value/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.

> **NOTE**
> Constraints on Boolean expressions are described in the topic, CALCULATE function (DAX).

The dates returned are the same as the dates returned by this equivalent formula:

```
DATEADD(dates, -1, year)
```

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see https://go.microsoft.com/fwlink/?LinkId=219172.

Example

The following sample formula creates a measure that calculates the previous year sales of the Reseller sales.

To see how this works, create a PivotTable and add the fields, CalendarYear to the **Row Labels** area of the PivotTable. Then add a measure, named **Previous Year Sales**, using the formula defined in the code section, to the **Values** area of the PivotTable.

```
=CALCULATE(SUM(ResellerSales_USD[SalesAmount_USD]), SAMEPERIODLASTYEAR(DateTime[DateKey]))
```

See also

Time-intelligence functions (DAX)
Date and time functions (DAX)
PREVIOUSYEAR function (DAX)
PARALLELPERIOD function (DAX)

STARTOFMONTH

12/10/2018 • 2 minutes to read

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.

> **NOTE**
> Constraints on Boolean expressions are described in the topic, CALCULATE function (DAX).

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see https://go.microsoft.com/fwlink/?LinkId=219172.

Example

The following sample formula creates a measure that returns the start of the month, for the current context.

To see how this works, create a PivotTable and add the fields CalendarYear and MonthNumberOfYear to the **Row Labels** area of the PivotTable. Then add a measure, named **StartOfMonth**, using the formula defined in the code section, to the **Values** area of the PivotTable.

```
=STARTOFMONTH(DateTime[DateKey])
```

See also

Date and time functions (DAX)
Time-intelligence functions (DAX)
STARTOFYEAR function (DAX)
STARTOFQUARTER function (DAX)

STARTOFQUARTER

12/10/2018 • 2 minutes to read

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.

> **NOTE**
> Constraints on Boolean expressions are described in the topic, CALCULATE function (DAX).

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see https://go.microsoft.com/fwlink/?LinkId=219172.

Example

The following sample formula creates a measure that returns the start of the quarter, for the current context.

To see how this works, create a PivotTable and add the fields CalendarYear and MonthNumberOfYear to the **Row Labels** area of the PivotTable. Then add a measure, named **StartOfQuarter**, using the formula defined in the code section, to the **Values** area of the PivotTable.

```
=STARTOFQUARTER(DateTime[DateKey])
```

See also

Date and time functions (DAX)

Time-intelligence functions (DAX)
STARTOFYEAR function (DAX)
STARTOFMONTH function (DAX)

STARTOFYEAR

12/10/2018 • 2 minutes to read

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.

> **NOTE**
> Constraints on Boolean expressions are described in the topic, CALCULATE function (DAX).

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see https://go.microsoft.com/fwlink/?LinkId=219172.

Example

The following sample formula creates a measure that returns the start of the year, for the current context.

To see how this works, create a PivotTable and add the fields CalendarYear and MonthNumberOfYear to the **Row Labels** area of the PivotTable. Then add a measure, named **StartOfYear**, using the formula defined in the code section, to the **Values** area of the PivotTable.

```
=STARTOFYEAR(DateTime[DateKey])
```

See also

Date and time functions (DAX)
Time-intelligence functions (DAX)
STARTOFQUARTER function (DAX)
STARTOFMONTH function (DAX)

TOTALMTD

3/29/2019 • 2 minutes to read

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.

> **NOTE**
> Constraints on Boolean expressions are described in the topic, CALCULATE function (DAX).

> **NOTE**
> The **filter** expression has restrictions described in the topic, CALCULATE function (DAX).

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see https://go.microsoft.com/fwlink/?LinkId=219172.

Example

The following sample formula creates a measure that calculates the 'month running total' or 'month running sum'

for the Internet sales.

To see how this works, create a PivotTable and add the fields, CalendarYear, MonthNumberOfYear and DayNumberOfMonth, to the **Row Labels** area of the PivotTable. Then add a measure, named **Month-to-date Total**, using the formula defined in the code section, to the **Values** area of the PivotTable.

Code

```
=TOTALMTD(SUM(InternetSales_USD[SalesAmount_USD]),DateTime[DateKey])
```

See also

ALL function (DAX)
CALCULATE function (DAX)
TOTALYTD function (DAX)
TOTALQTD function (DAX)

TOTALQTD

3/29/2019 • 2 minutes to read

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.

> **NOTE**
> Constraints on Boolean expressions are described in the topic, CALCULATE function (DAX).

> **NOTE**
> The **filter** expression has restrictions described in the topic, CALCULATE function (DAX).
>
> This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see https://go.microsoft.com/fwlink/?LinkId=219172.

Example

The following sample formula creates a measure that calculates the 'quarter running total' or 'quarter running sum' for the Internet sales.

To see how this works, create a PivotTable and add the fields, CalendarYear, CalendarQuarter and MonthNumberOfYear, to the **Row Labels** area of the PivotTable. Then add a measure, named **Quarter-to-date Total**, using the formula defined in the code section, to the **Values** area of the PivotTable.

Code

```
=TOTALQTD(SUM(InternetSales_USD[SalesAmount_USD]),DateTime[DateKey])
```

See also

ALL function (DAX)
CALCULATE function (DAX)
TOTALYTD function (DAX)
TOTALMTD function (DAX)

TOTALYTD

3/29/2019 • 2 minutes to read

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.

> **NOTE**
> Constraints on Boolean expressions are described in the topic, CALCULATE function (DAX).

> **NOTE**
> The **filter** expression has restrictions described in the topic, CALCULATE function (DAX).

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 DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see https://go.microsoft.com/fwlink/?LinkId=219172.

Example

The following sample formula creates a measure that calculates the 'year running total' or 'year running sum' for the Internet sales.

To see how this works, create a PivotTable and add the fields, CalendarYear, CalendarQuarter, and MonthNumberOfYear, to the **Row Labels** area of the PivotTable. Then add a measure, named **Year-to-date Total**, using the formula defined in the code section, to the **Values** area of the PivotTable.

Code

```
=TOTALYTD(SUM(InternetSales_USD[SalesAmount_USD]),DateTime[DateKey])
```

See also

ALL function (DAX)
CALCULATE function (DAX)
DATESYTD function (DAX)
TOTALMTD function (DAX)
TOTALQTD function (DAX)

Filter functions

4/1/2019 • 2 minutes to read

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 section

ADDMISSINGITEMS

12/10/2018 • 2 minutes to read

Adds combinations of items from multiple columns to a table if they do not already exist. The determination of which item combinations to add is based on referencing source columns which contain all the possible values for the columns.

To determine the combinations of items from different columns to evaluate: AutoExist is applied for columns within the same table while CrossJoin is applied across different tables.

The ADDMISSINGITEMS function will return BLANK values for the IsSubtotal columns of blank rows it adds.

Syntax

```
ADDMISSINGITEMS(<showAllColumn>[, <showAllColumn>]…, <table>, <groupingColumn>[, <groupingColumn>]…[, filterTable]…)
```

```
ADDMISSINGITEMS(<showAllColumn>[, <showAllColumn>]…, <table>, [ROLLUPISSUBTOTAL(]<groupingColumn>[, <isSubtotal_columnName>][, <groupingColumn>][, <isSubtotal_columnName>]…[)], [, filterTable]…)
```

Parameters

TERM	DEFINITION
showAllColumn	A column for which to return items with no data for the measures used.
table	A table containing all items with data (NON EMPTY) for the measures used.
groupingColumn	A column which is used to group by in the supplied table argument.
isSubtotal_columnName	A Boolean column in the supplied table argument which contains ISSUBTOTAL values for the corresponding groupingColumn column.
filterTable	A table representing filters to include in the logic for determining whether to add specific combinations of items with no data. Used to avoid having ADDMISSINGITEMS add in item combinations which are not present because they were removed by a filter.

ADDMISSINGITEMS with ROLLUPGROUP

ROLLUPGROUP is used inside the ROLLUPISSUBTOTAL function to reflect ROLLUPGROUPs present in the supplied table argument.

Restrictions

- If ROLLUPISSUBTOTAL was used to define the supplied table argument (or the equivalent rows and

ISSUBTOTAL columns were added by some other means), ROLLUPISSUBTOTAL must be used with the same arguments within ADDMISSINGITEMS. This is also true for ROLLUPGROUP if it was used with ROLLUPISSUBTOTAL to define the supplied table argument.

- The ADDMISSINGITEMS function requires that, if ROLLUPISSUBTOTAL was used to define the supplied table argument, ISSUBTOTAL columns corresponding to each group by column, or ROLLUPGROUP, are present in the supplied table argument. Also, the names of the ISSUBTOTAL columns must be supplied in the ROLLUPISSUBTOTAL function inside ADDMISSINGITEMS and they must match names of Boolean columns in the supplied table argument. This enables the ADDMISSINGITEMS function to identify BLANK values stemming from the fact that a row is a subtotal row from other BLANK values.
- If ROLLUPGROUP was used with ROLLUPISSUBTOTAL to define the supplied table argument, exactly one ISSUBTOTAL column name must be supplied per ROLLUPGROUP and it must match the corresponding ISSUBTOTAL column name in the supplied table argument.

Example

Add blank rows for columns with "show items with no data" turned on. The ADDMISSINGITEMS function will return NULLs/BLANKs for the IsSubtotal columns of blank rows it adds.

```
VAR 'RowHeadersShowAll' =
CALCULATETABLE
(
ADDMISSINGITEMS
(
[Sales Territory Country],
[Sales Territory Region],
'RowHeadersInCrossTab',
ROLLUPISSUBTOTAL
(
[Sales Territory Group],
[Subtotal for Sales Territory Group],
[Sales Territory Country],
[Subtotal for Sales Territory Country],
[Sales Territory Region],
[Subtotal for Sales Territory Region]
),
'RowHeaders'
),
'DateFilter','TerritoryFilter'
)
```

Example with ROLLUPGROUP

```
VAR 'RowHeadersShowAll' =
CALCULATETABLE
(
ADDMISSINGITEMS
(
[Sales Territory Country],
[Sales Territory Region],
'RowHeadersInCrossTab',
ROLLUPISSUBTOTAL
(
ROLLUPGROUP
(
[Sales Territory Group],
[Sales Territory Country]
),
[Subtotal for Sales Territory Country],
[Sales Territory Region],
[Subtotal for Sales Territory Region]
),
'RowHeaders'
)
```

ALL

3/29/2019 • 6 minutes to read

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.

<**Topic Status:** Some information in this topic is pre-release and subject to change in future releases. Pre-release information describes new features or changes to existing features in Microsoft SQL Server 2016.In cases where [Column] is marked as Date column by using Mark as Date table

As described in the following table, 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.
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.

FUNCTION AND USAGE	DESCRIPTION
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.

Example

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.

ALL RESELLER SALES	COLUMN LABELS				
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])
```

Comments

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

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 the 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**.

RESELLER SALES YEAR	COLUMN LABELS				
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]))
```

Comments

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

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**.

RESELLER SALES CATEGORYNAME	COLUMN LABELS				
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]))
```

Comments

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 (DAX)
ALL function (DAX)

ALLEXCEPT function (DAX)
FILTER function (DAX)

ALLCROSSFILTERED

4/1/2019 • 2 minutes to read

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.

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
On Promotion	60398	7390

DIMSALESREASON[SALESREASONNAME]	[TOTALQUANTITY1]	[TOTALQUANTITY2]
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

3/29/2019 • 2 minutes to read

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.

As described in the following table, you can use the ALL and ALLEXCEPT functions 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.

Example

The following example presents a formula that you can use in a measure.

The 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 the PivotTable, the formula will remove any slicer filters, providing a value equal to the sum of SalesAmount_USD for the column label value, as shown in Table 1.

However, if the column CalendarYear is used to slice the PivotTable, 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, as shown in Table 2. The user is encouraged to compare these tables to understand the behavior of ALLEXCEPT().

See also

Filter functions (DAX)
ALL function (DAX)
FILTER function (DAX)

ALLNOBLANKROW

12/10/2018 • 4 minutes to read

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 (DAX).

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 given in the code section following.

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

ROW LABELS	COUNTROWS ALLNOBLANK OF DATETIME	COUNTROWS ALL OF DATETIME
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
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.

```
// 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'))
```

See also

Filter functions (DAX)
ALL function (DAX)
FILTER function (DAX)

ALLSELECTEDAX)

12/10/2018 • 4 minutes to read

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>])
```

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

- This function takes one or no arguments. If there is one argument, the argument is either *tableName* or *columnName*.
- 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.

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 catergory.

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

12/10/2018 • 2 minutes to read

Evaluates an expression in a context that is modified by the specified filters.

Syntax

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

Parameters

TERM	DEFINITION
expression	The expression to be evaluated.
filter1, filter2,…	(optional) A comma separated list of Boolean expression or a table expression that defines a filter.

The expression used as the first parameter is essentially the same as a measure.

The following restrictions apply to Boolean expressions that are used as arguments:

- The expression cannot reference a measure.
- The expression cannot use a nested CALCULATE function.
- The expression cannot use any function that scans a table or returns a table, including aggregation functions.

However, a Boolean expression can use any function that looks up a single value, or that calculate a scalar value.

Return value

The value that is the result of the expression.

Remarks

If the data has been filtered, the CALCULATE function changes the context in which the data is filtered, and evaluates the expression in the new context that you specify. For each column used in a filter argument, any existing filters on that column are removed, and the filter used in the filter argument is applied instead.

Example

To calculate the ratio of current reseller sales to all reseller sales, you add to the PivotTable a measure that calculates the sum of sales for the current cell (the numerator), and then divides that sum by the total sales for all resellers (the denominator). To ensure that the denominator remains the same regardless of how the PivotTable might be filtering or grouping the data, the part of the formula that represents the denominator must use the ALL function to clear any filters and create the correct total.

The following table shows the results when the new measure, named **All Reseller Sales Ratio**, is

created by using the formula 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 display the numbers as percentages, apply percentage number formatting to the area of the PivotTable that contains the new measure, **All Reseller Sales Ratio**.

ALL RESELLER SALES	COLUMN LABELS				
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%

```
=( SUM('ResellerSales_USD'[SalesAmount_USD]))
 /CALCULATE( SUM('ResellerSales_USD'[SalesAmount_USD])
           ,ALL('ResellerSales_USD'))
```

The CALCULATE expression in the denominator enables the sum expression to include all rows in the calculation. This overrides the implicit filters for CalendarYear and ProductCategoryName that exist for the numerator part of the expression.

Related functions

Whereas the CALCULATE function requires as its first argument an expression that returns a single value, the CALCULATETABLE function takes a table of values.

See also

CALCULATETABLE function (DAX)
Filter functions (DAX)

CALCULATETABLE

12/10/2018 • 2 minutes to read

Evaluates a table expression in a context modified by the given filters.

Syntax

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

Parameters

TERM	DEFINITION
Expression**	The table expression to be evaluated
filter1, filter2,…	A Boolean expression or a table expression that defines a filter

The expression used as the first parameter must be a function that returns a table.

The following restrictions apply to Boolean expressions that are used as arguments:

- The expression cannot reference a measure.
- The expression cannot use a nested CALCULATE function.
- The expression cannot use any function that scans a table or returns a table, including aggregation functions.

However, a Boolean expression can use any function that looks up a single value, or that calculates a scalar value.

Return value

A table of values.

Remarks

The CALCULATETABLE function changes the context in which the data is filtered, and evaluates the expression in the new context that you specify. For each column used in a filter argument, any existing filters on that column are removed, and the filter used in the filter argument is applied instead.

This function is a synonym for the RELATEDTABLE function.

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 table shows the results from the following formula.

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

```
=SUMX( CALCULATETABLE('InternetSales_USD', 'DateTime'[CalendarYear]=2006)
     , [SalesAmount_USD])
```

See also

RELATEDTABLE function (DAX)
Filter functions (DAX)

CROSSFILTER

12/10/2018 • 2 minutes to read

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	The cross-filter direction to be used. Must be one of the following: none No cross-filtering occurs along this relationship **one** - Filters on the one or lookup side of the side of the relationship filter the many side. **both** - Filters on either side filter the other **none** - No cross-filtering occurs along this relationship

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

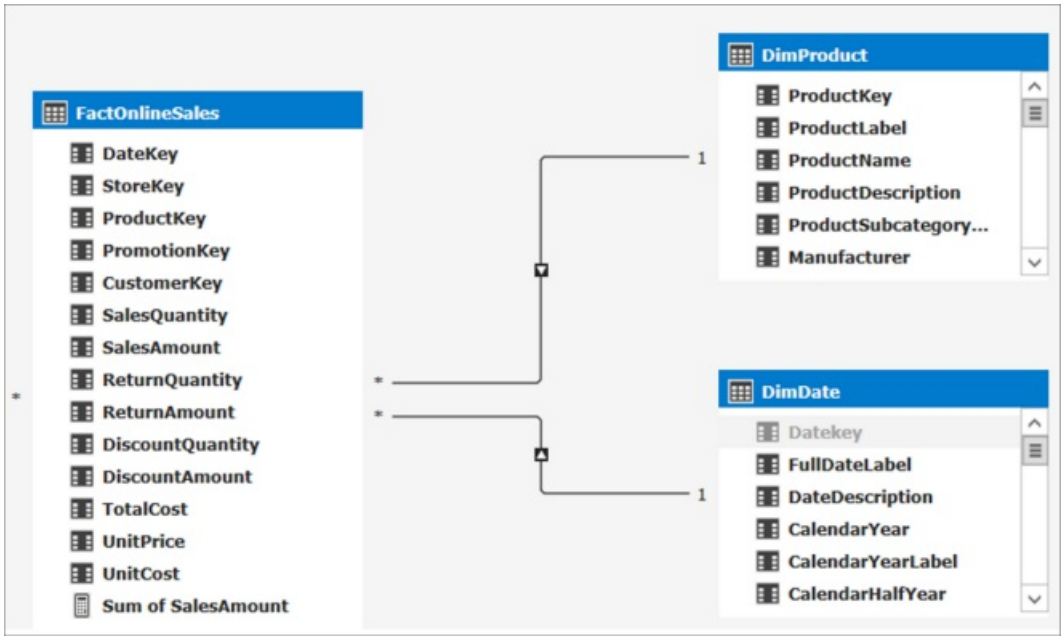
- In the case of a 1:1 relationship, there is no difference between the one and both direction.
- CROSSFILTER can only be used in functions that take a filter as an argument, for example: CALCULATE, CALCULATETABLE, CLOSINGBALANCEMONTH, CLOSINGBALANCEQUARTER, CLOSINGBALANCEYEAR, OPENINGBALANCEMONTH, OPENINGBALANCEQUARTER, 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.

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**

DISTINCT (column)

12/10/2018 • 2 minutes to read

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.

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 (DAX).

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 PivotTable.

```
=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.

UNIQUE INTERNET CUSTOMERS	COLUMN LABELS			
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 (DAX)
FILTER function (DAX)
RELATED function (DAX)
VALUES function (DAX)

DISTINCT (table)

12/10/2018 • 2 minutes to read

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 (DAX)
DISTINCT (column) (DAX)
FILTER function (DAX)
RELATED function (DAX)
VALUES function (DAX)

EARLIER

3/29/2019 • 3 minutes to read

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.

Property Value/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 it 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.

> **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
21	Jerseys	$699,429.78	7
33	Lights		36
34	Locks	$15,059.47	33

PRODUCTSUBCATEGORYKEY	ENGLISHPRODUCTSUBCATEGORYNAME	TOTALSUBCATEGORYSALES	SUBCATEGORYRANKING
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 (DAX)
Filter functions (DAX)

EARLIEST

12/10/2018 • 2 minutes to read

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.

Property Value/Return value

A column with filters removed.

Remarks

The EARLIEST function is similar to EARLIER, but lets you specify one additional level of recursion.

Example

The current sample data does not support this scenario.

```
=EARLIEST(<column>)
```

See also

EARLIER function (DAX)
Filter functions (DAX)

FILTER

12/10/2018 • 2 minutes to read

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, `[Amount] > 0` or `[Region] = "France"`

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.

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 the 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

NON USA INTERNET SALES	COLUMN LABELS			
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 (DAX)
ALL function (DAX)
ALLEXCEPT function (DAX)

FILTERS

12/10/2018 • 2 minutes to read

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

Example

The following example shows how to determine the number of direct filters a column has.

```
=COUNTROWS(FILTERS(ResellerSales_USD[ProductKey]))
```

The example above lets you know how many direct filters on ResellerSales_USD[ProductKey] have been applied to the context where the expression is being evaluated.

HASONEFILTER

12/10/2018 • 2 minutes to read

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

1. This function is similar to HASONEVALUE() with the difference that HASONEVALUE() works based on cross-filters while HASONEFILTER() works by a direct filter.

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

12/10/2018 • 2 minutes to read

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`.

Example

In the following example, you want to create a formula that 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.

Create a measure named [ResellerSales compared to 2007] using the following expression:

```
=IF(HASONEVALUE(DateTime[CalendarYear]),SUM(ResellerSales_USD[SalesAmount_USD])/CALCULATE(SUM(ResellerSales_US
D[SalesAmount_USD]),DateTime[CalendarYear]=2007),BLANK())
```

1. After creating the measure you should have an empty result under [ResellerSales compared to 2007]. The BLANK cell in the result is because you don't have single year filters anywhere in your context.
2. Drag DateTime[CalendarYear] to the **Column Labels** box; your table should look like this:

	COLUMN LABELS			
	2005	**2006**	**2007**	**2008**
ResellerSales compared to 2007	**24.83 %**	**74.88 %**	**100.00 %**	**50.73 %**

3. Drag ProductCategory[ProductCategoryName] to the **Row Labels** box to have something like this:

RESELLERSALES COMPARED TO 2007	COLUMN LABELS			
Row Labels	**2005**	**2006**	**2007**	**2008**
Accessories	6.74 %	31.40 %	100.00 %	55.58 %
Bikes	28.69 %	77.92 %	100.00 %	53.46 %
Clothing	3.90 %	55.86 %	100.00 %	44.92 %
Components	11.05 %	65.99 %	100.00 %	38.65 %
Grand Total	**24.83 %**	**74.88 %**	**100.00 %**	**50.73 %**

Did you notice that **Grand Totals** appeared at the bottom of the columns but not for the rows? This is because the context for Grand Totals on rows implies more than one year; but for columns implies a single year.

4. Drag DateTime[CalendarYear] to the **Horizontal Slicers** box, and drag SalesTerritory[SalesTerritoryGroup] to the **Horizontal Labels** box. You should have an empty results set, because your table contains data for multiple years. Select **2006** in the slicer and your table should now have data again. Try other years to see how the results change.

5. In summary, HASONEVALUE() allows you to identify if your expression is being evaluated in the context of a single value for *columnName*.

ISCROSSFILTERED

12/10/2018 • 2 minutes to read

Returns TRUE when *columnName* or another column in the same or related table is being filtered.

Syntax

```
ISCROSSFILTERED(<columnName>)
```

Parameters

TERM	DEFINITION
columnName	The name of an existing column, using standard DAX syntax. It cannot be an expression.

Return value

TRUE when *columnName* or another column in the same or related table is being filtered. Otherwise returns **FALSE**.

Remarks

- A column is said to be cross-filtered when a filter applied to another column in the same table or in a related table affects *columnName* by filtering it. A column is said to be filtered *directly* when the filter or filters apply over the column.
- The related function ISFILTERED function (DAX) returns TRUE when *columnName* is filtered directly.

See also

ISFILTERED function (DAX)
FILTERS function (DAX)
HASONEFILTER function (DAX)
HASONEVALUE function (DAX)

ISFILTERED

12/10/2018 • 2 minutes to read

Returns TRUE when *columnName* is being filtered directly. If there is no filter on the column or if the filtering happens because a different column in the same table or in a related table is being filtered then the function returns **FALSE**.

Syntax

```
ISFILTERED(<columnName>)
```

Parameters

TERM	DEFINITION
columnName	The name of an existing column, using standard DAX syntax. It cannot be an expression.

Return value

TRUE when *columnName* is being filtered directly.

Remarks

- *columnName* is said to be filtered directly when the filter or filters apply over the column; a column is said to be cross-filtered when a filter applied to another column in the same table or in a related table affects *columnName* the column by filtering it as well.
- The related function ISCROSSFILTERED function (DAX) returns TRUE when *columnName* or another column in the same or related table is being filtered.

See also

ISCROSSFILTERED function (DAX)
FILTERS function (DAX)
HASONEFILTER function (DAX)
HASONEVALUE function (DAX)

KEEPFILTERS

3/29/2019 • 3 minutes to read

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 s 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.

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 (DAX)
CALCULATE function (DAX)
CALCULATETABLE function (DAX)

RELATED

12/10/2018 • 2 minutes to read

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.

> **NOTE**
> 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.

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 table shows the final report that you might get if you used this measure in a PivotTable:

NON USA INTERNET SALES	COLUMN LABELS			
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

NON USA INTERNET SALES	COLUMN LABELS			
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 function (DAX)
Filter functions (DAX)

RELATEDTABLE

12/10/2018 • 2 minutes to read

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 RELATEDTETABLE 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.

Example

The following example uses the RELATEDTABLE function to create a calculated column with the Internet Sales in the Product Category table.

The following table shows the results of using the code shown here.

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
4	4	Accessories	$700,759.96

```
= SUMX( RELATEDTABLE('InternetSales_USD')
     , [SalesAmount_USD])
```

See also

CALCULATETABLE function (DAX)
Filter functions (DAX)

SELECTEDVALUE

12/10/2018 • 2 minutes to read

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>)`.

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

DIMPRODUCT[COLOR]	[IS TOTAL]	[SELECTED COLOR]
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

SUBSTITUTEWITHINDEX

12/10/2018 • 2 minutes to read

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	(Optional) A value that specifies how to sort orderBy_expression values, ascending or descending: 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. Value: **ASC**. Alternative value: **1**/**TRUE**. Ranks in ascending order of orderBy_expression.

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.

USERELATIONSHIP

12/10/2018 • 2 minutes to read

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 USEREALTIONSHIP:

```
=CALCULATE(CALCULATE( CALCULATE( &lt;anyExpression&gt;, USERELATIONSHIP( t1[colA], t2[colB])),
t99[colZ]=999), USERELATIONSHIP( t1[colA], t2[colA]))
```

.

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 you need to create a measure, [InternetSales by ShippingDate] using the following expression:

```
=CALCULATE(SUM(InternetSales[SalesAmount]), USERELATIONSHIP(InternetSales[ShippingDate], DateTime[Date]))
```

Drag your new measure to the Values area in the right pane, drag the InternetSales[ShippingDate] column to the Row Labels area; you now have Internet Sales sliced by shipping date instead of by order date as is usually shown in these examples.

For this example to work the 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.

VALUES

12/10/2018 • 2 minutes to read

Returns a one-column table that contains the distinct values from the specified table or 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 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	The table or column from which unique values are to be returned.

Return value

A column of unique values.

Remarks

When you use the VALUES function in a context that has been filtered, such as in a PivotTable, 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.

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 used the DISTINCT function to return a list of dates from the PivotTable containing these tables, 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:

ROW LABELS	COUNT INVOICES
Accessories	18,208
Bikes	15,205
Clothing	7,461
Grand Total	27,659

```
=COUNTROWS(VALUES('InternetSales_USD'[SalesOrderNumber]))
```

See also

FILTER function (DAX)
COUNTROWS function (DAX)
Filter functions (DAX)

Information functions

4/1/2019 • 2 minutes to read

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 section

CONTAINS

12/10/2018 • 2 minutes to read

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 *columnName*. 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.

Example

The following example creates a calculated measure that tells you whether there were any Internet sales of the product 214 and to customer 11185 at the same time.

```
=CONTAINS(InternetSales, [ProductKey], 214, [CustomerKey], 11185)
```

CUSTOMDATA

12/10/2018 • 2 minutes to read

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.

Exceptions

Remarks

Example

The following DAX code 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")
```

IN Operator / CONTAINSROW function

12/10/2018 • 2 minutes to read

Returns TRUE if a row of values exists or contained in a table, otherwise returns FALSE. Except syntax, the IN operator and CONTAINSROW function are functionally equivalent.

IN Operator

Syntax

```
<scalarExpr> IN <tableExpr>
( <scalarExpr1>, <scalarExpr2>, … ) IN <tableExpr>
```

CONTAINSROW function

Syntax

```
CONTAINSROW(<tableExpr>, <scalarExpr>[, <scalarExpr>, …])
```

Parameters

TERM	DEFINITION
scalarExprN	Any valid DAX expression that returns a scalar value.
tableExpr	Any valid DAX expression that returns a table of data.

Return value

TRUE or FALSE.

Remarks

The number of scalarExprN must match the number of columns in tableExpr.

Unlike the = operator, the IN operator and the CONTAINSROW function perform strict comparison. For example, the BLANK value does not match 0.

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" }.

Examples

Example 1

The following equivalent DAX queries:

```
EVALUATE FILTER(ALL(DimProduct[Color]), [Color] IN { "Red", "Yellow", "Blue" })
ORDER BY [Color]
```

and

```
EVALUATE FILTER(ALL(DimProduct[Color]), ([Color]) IN { "Red", "Yellow", "Blue" })
ORDER BY [Color]
```

and

```
EVALUATE FILTER(ALL(DimProduct[Color]), CONTAINSROW({ "Red", "Yellow", "Blue" }, [Color]))
ORDER BY [Color]
```

Return the following table with a single column:

DIMPRODUCT[COLOR]	
Blue	
Red	
Yellow	

Example 2

The following equivalent DAX queries:

```
EVALUATE FILTER(SUMMARIZE(DimProduct, [Color], [Size]), ([Color], [Size]) IN { ("Black", "L") })
```

and

```
EVALUATE FILTER(SUMMARIZE(DimProduct, [Color], [Size]), CONTAINSROW({ ("Black", "L") }, [Color], [Size]))
```

Return:

DIMPRODUCT[COLOR]	DIMPRODUCT[SIZE]
Black	L

Example 3

The following equivalent DAX queries:

```
EVALUATE FILTER(ALL(DimProduct[Color]), NOT [Color] IN { "Red", "Yellow", "Blue" })
ORDER BY [Color]
```

and

```
EVALUATE FILTER(ALL(DimProduct[Color]), NOT CONTAINSROW({ "Red", "Yellow", "Blue" }, [Color]))
ORDER BY [Color]
```

Return the following table with a single column:

DIMPRODUCT[COLOR]	
Black	
Grey	
Multi	
NA	
Silver	
Silver\Black	
White	

ISBLANK

12/10/2018 • 2 minutes to read

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.

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.

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		

```
//Sales to Previous Year Ratio

=IF( ISBLANK('CalculatedMeasures'[PreviousYearTotalSales])
   , BLANK()
   , ( 'CalculatedMeasures'[Total Sales]-'CalculatedMeasures'[PreviousYearTotalSales] )
      /'CalculatedMeasures'[PreviousYearTotalSales])
```

See also

Information functions (DAX)

ISERROR

12/10/2018 • 2 minutes to read

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.

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 (DAX)
IFERROR function (DAX)
IF function (DAX)

ISEVEN

12/10/2018 • 2 minutes to read

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.

ISINSCOPE

12/10/2018 • 2 minutes to read

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.

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[CATEGORY]	DIMPRODUCT[SUBCATEGORY]	[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 (DAX)
CALCULATE function (DAX)

ISLOGICAL

12/10/2018 • 2 minutes to read

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.

Property Value/Return value

TRUE if the value is a logical value; FALSE if any value other than TRUE OR FALSE.

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 (DAX)

ISNONTEXT

12/10/2018 • 2 minutes to read

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.

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 (DAX)

ISNUMBER

12/10/2018 • 2 minutes to read

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.

Property Value/Return value

TRUE if the value is numeric; otherwise FALSE.

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 (DAX)

ISODD

12/10/2018 • 2 minutes to read

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.

ISONORAFTER

1/16/2019 • 2 minutes to read

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.

This function takes a variable number of triples, the first two values in a triple are the expressions to be compared, and the third parameter indicates the sort order. The sort order can be ascending (default) or descending.

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 or equal to second parameter. If the sort order is descending, the comparison to be done is second parameter less than or equal to 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 (DEC). By default the sort order is ascending.

Return value

True or false.

Example

Table name: 'Info'

COUNTRY	STATE	COUNT	TOTAL
IND	JK	20	800
IND`	MH	25	1000
IND	WB	10	900
USA	CA	5	500
USA	WA	10	900

```
FILTER(Info, ISONORAFTER(Info[Country], “IND”, ASC, Info[State], “MH”, ASC))
```

ISTEXT

12/10/2018 • 2 minutes to read

Checks if a value is text, and returns TRUE or FALSE.

Syntax

```
ISTEXT(<value>)
```

Parameters

TERM	DEFINITION
value	The value you want to check.

Property Value/Return value

TRUE if the value is text; otherwise FALSE

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 (DAX)

LOOKUPVALUE

3/18/2019 • 2 minutes to read

Returns the value in *result_columnName* for the row that meets all criteria specified by *search_columnName* and *search_value*.

Syntax

```
LOOKUPVALUE( <result_columnName>, <search_columnName>, <search_value>[, <search_columnName>, <search_value>]…
[, <alternateResult>])
```

Parameters

TERM	DEFINITION
result_columnName	The name of an existing column that contains the value you want to return. The column must be named using standard DAX syntax, usually, fully qualified. It cannot be an expression.
search_columnName	The name of an existing column, in the same table as result_columnName or in a related table, over which the look-up is performed. The column must be named using standard DAX syntax, usually, fully qualified. It cannot be an expression.
search_value	A scalar expression that does not refer to any column in the same table being searched.
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 a match.

If there is no match that satisfies all the search values, a BLANK or *alternateResult*, if supplied, is returned. In other words, the function will not 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

Example

The following example returns the SafetyStocklLevel for the bike model "Mountain-400-W Silver, 46".

```
=LOOKUPVALUE(Product[SafetyStockLevel], [ProductName], " Mountain-400-W Silver, 46")
```

USERCULTURE

4/1/2019 • 2 minutes to read

Returns the user's Locale Name.

Syntax

```
USERCULTURE()
```

Return value

User's locale name based on the two-letter ISO-639 and ISO-3166 codes for language and country code, detected from the user's application or browser settings. When using Power BI Desktop, this reflects the application language set in **Options** > **Regional Settings**. When viewing a report in the Power BI service, it reflects the browser language.

Example

```
EVALUATE
Row ("Culture", USERCULTURE())
```

Returns

CULTURE
en-US

For a user using an English language browser in the USA.

USERNAME

12/10/2018 • 2 minutes to read

Returns the domain name and username from the credentials given to the system at connection time

Syntax

```
USERNAME()
```

Parameters

Return value

The username from the credentials given to the system at connection time

Example

The following code verifies if the user login is part of the UsersTable.

```
=IF(CONTAINS(UsersTable,UsersTable[login], USERNAME()), "Allowed", BLANK())
```

Logical functions

12/10/2018 • 2 minutes to read

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 section

AND

12/10/2018 • 2 minutes to read

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

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

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".

AND FUNCTION	COLUMN LABELS					
Row Labels	2005	2006	2007	2008		Grand Total
Bib-Shorts						

AND FUNCTION	COLUMN LABELS					
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						
Mountain Frames						
Panniers						

AND FUNCTION	COLUMN LABELS					
Pedals						
Pumps						
Road Bikes						
Road Frames						
Saddles						
Shorts						
Socks						
Tights						
Tires and Tubes				Internet Hit		
Touring Bikes						
Touring Frames						
Vests						
Wheels						
Grand Total						

```
= 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"
     , ""
     )
```

See also

Logical functions (DAX)

FALSE

12/10/2018 • 2 minutes to read

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.

The following table shows the results when the example formula is used with 'ProductCategory'[ProductCategoryName] in Row Labels and 'DateTime'[CalendarYear] in Column Labels.

TRUE-FALSE	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

```
=IF(SUM('InternetSales_USD'[SalesAmount_USD]) >200000, TRUE(), false())
```

See also

TRUE function (DAX)
NOT function (DAX)

IF function (DAX)
DAX function reference

IF

12/10/2018 • 2 minutes to read

Checks if a condition provided as the first argument is met. Returns one value if the condition is TRUE, and returns another value if the condition is FALSE.

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 is returned if the logical test is TRUE. If omitted, TRUE is returned.
value_if_false	The value that is returned if the logical test is FALSE. If omitted, FALSE is returned.

Return value

Any type of value that can be returned by an expression.

Remarks

If the value of **value_if_true** or **value_if_false** is omitted, IF treats it as an empty string value ("").

If the value referenced in the expression is a column, IF returns the value that corresponds to the current row.

The IF function attempts to return a single data type in a column. Therefore, if the values returned by **value_if_true** and **value_if_false** are of different data types, the IF function will implicitly convert data types to accommodate both values in the column. For example, the formula `IF(<condition>,TRUE(),0)` returns a column of ones and zeros and the results can be summed, but the formula `IF(<condition>,TRUE(),FALSE())` returns only logical values. For more information about implicit data type conversion, see Data Types Supported (SSAS Tabular).

Example

The following example uses nested IF functions that evaluate the number in the column, Calls, from the table FactCallCenter. The function assigns a label as follows: **low** if the number of calls is less than 200, **medium** if the number of calls is less than 300 but not less than 200, and **high** for all other values.

```
=IF([Calls]<200,"low",IF([Calls]<300,"medium","high"))
```

Example

The following example gets a list of cities that contain potential customers in the California area by using columns from the table ProspectiveBuyer. Because the list is meant to plan for a campaign that will target married people or people with children at home, the condition in the IF function checks for the value of the columns [MaritalStatus] and [NumberChildrenAtHome], and outputs the city if either condition is met and if the customer is in California. Otherwise, it outputs the empty string.

```
=IF([StateProvinceCode]= "CA" && ([MaritalStatus] = "M" || [NumberChildrenAtHome] >1),[City])
```

Note that parentheses are used to control the order in which the AND (&&) and OR (||) operators are used. Also note that no value has been specified for **value_if_false**. Therefore, the function returns the default, which is an empty string.

See also

TRUE function (DAX)
FALSE function (DAX)
NOT function (DAX)
IF function (DAX)
DAX function reference

IFERROR

12/10/2018 • 2 minutes to read

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)
```

Note that the 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.

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

Logical functions (DAX)

IN

12/10/2018 • 2 minutes to read

Returns True if the scalar value shows up in at least one row of the input relation.

Syntax

```
IN
```

Parameters

TERM	DEFINITION
scalar expression	
table expression	

Return value

Remarks

Example

```
Filtered Sales:=CALCULATE (
        [Internet Total Sales], 'Product'[Color] IN { "Red", "Blue", "Black" }
    )
```

See also

NOT

12/10/2018 • 2 minutes to read

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 (DAX)
FALSE function (DAX)
IF function (DAX)
DAX function reference

OR

12/10/2018 • 2 minutes to read

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.

SALESPERSONFLAG	TRUE					
OR function	Column Labels					
Row Labels	2005	2006	2007	2008		Grand Total
Abbas, Syed E						
Alberts, Amy E						

SALESPERSONFLAG	TRUE					
Ansman-Wolfe, Pamela O						
Blythe, Michael G	Circle of Excellence	Circle of Excellence	Circle of Excellence	Circle of Excellence	Circle of Excellence	Circle of Excellence
Campbell, David R						
Carson, Jillian	Circle of Excellence	Circle of Excellence	Circle of Excellence	Circle of Excellence	Circle of Excellence	Circle of Excellence
Ito, Shu K						
Jiang, Stephen Y						
Mensa-Annan, Tete A						
Mitchell, Linda C	Circle of Excellence	Circle of Excellence	Circle of Excellence	Circle of Excellence	Circle of Excellence	Circle of Excellence
Pak, Jae B	Circle of Excellence	Circle of Excellence	Circle of Excellence	Circle of Excellence	Circle of Excellence	Circle of Excellence
Reiter, Tsvi Michael						
Saraiva, José Edvaldo	Circle of Excellence	Circle of Excellence	Circle of Excellence	Circle of Excellence	Circle of Excellence	Circle of Excellence
Tsoflias, Lynn N						
Valdez, Rachel B						
Vargas, Garrett R						
Varkey Chudukatil, Ranjit R						Circle of Excellence
Grand Total	Circle of Excellence	Circle of Excellence	Circle of Excellence	Circle of Excellence	Circle of Excellence	Circle of Excellence

```
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"
   , ""
   )
```

See also

Logical functions (DAX)

SWITCH

12/10/2018 • 2 minutes to read

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 *expression*.
result	Any scalar expression to be evaluated if the results of *expression* match the corresponding *value*.
else	Any scalar expression to be evaluated if the result of *expression* doesn't match any of the *value* 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

12/10/2018 • 2 minutes to read

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.

The following table shows the results when the example formula is used in a PivotTable, with 'ProductCategory'[ProductCategoryName] in Row Labels and 'DateTime'[CalendarYear] in Column Labels.

TRUE-FALSE	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

```
= IF(SUM('InternetSales_USD'[SalesAmount_USD]) >200000, TRUE(), false())
```

See also

FALSE function (DAX)
NOT function (DAX)

IF function (DAX)
DAX function reference

Math and Trig functions

12/10/2018 • 2 minutes to read

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 section

INT

MROUND

ODD

PI

POWER

PRODUCT

PRODUCTX

QUOTIENT

RADIANS

RAND

RANDBETWEEN

ROUND

ROUNDDOWN

ROUNDUP

SIGN

SQRT

SUM

SUMX

TRUNC

ABS

12/10/2018 • 2 minutes to read

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 (DAX)
SIGN function (DAX)

ACOS

12/10/2018 • 2 minutes to read

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

12/10/2018 • 2 minutes to read

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.

Example

FORMULA	DESCRIPTION	RESULT
=ACOSH(1)	Inverse hyperbolic cosine of 1.	0
=ACOSH(10)	Inverse hyperbolic cosine of 10.	2.993228

ASIN

12/10/2018 • 2 minutes to read

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

12/10/2018 • 2 minutes to read

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.

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

12/10/2018 • 2 minutes to read

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

12/10/2018 • 2 minutes to read

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.

Example

FORMULA	DESCRIPTION	RESULT
=ATANH(0.76159416)	Inverse hyperbolic tangent of 0.76159416	1.00000001
=ATANH(-0.1)		-0.100335348

See also

ATAN function (DAX)

CEILING

12/10/2018 • 2 minutes to read

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

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

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 (DAX)
FLOOR function (DAX)
ISO.CEILING function (DAX)
ROUNDUP function (DAX)

COMBIN

12/10/2018 • 2 minutes to read

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:

$$\binom{n}{k} = \frac{P_{k,n}}{k!} = \frac{n!}{k!(n-k)!}$$

Where

$$P_{k,n} = \frac{n!}{(n-k)!}$$

Example

FORMULA	DESCRIPTION	RESULT
=COMBIN(8,2)	Possible two-person teams that can be formed from 8 candidates.	28

COMBINA

12/10/2018 • 2 minutes to read

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:

$$\binom{N+M-1}{N-1}$$

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

COS

12/10/2018 • 2 minutes to read

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 PI()/180 or use the RADIANS function to convert the angle to radians.

Example

FORMULA	DESCRIPTION	RESULT
=COS(1.047)	Cosine of 1.047 radians	0.5001711
=COS(60*PI()/180)	Cosine of 60 degrees	0.5
=COS(RADIANS(60))	Cosine of 60 degrees	0.5

COSH

12/10/2018 • 2 minutes to read

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:

$$\mathrm{COSH}(z) = \frac{e^{z} + e^{-z}}{2}$$

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

CURRENCY

12/10/2018 • 2 minutes to read

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 3.6666666666666 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 the value $1234.5600.

DEGREES

12/10/2018 • 2 minutes to read

Converts radians into degrees.

Syntax

```
DEGREES(angle)
```

Parameters

TERM	DEFINITION
angle	Required. The angle in radians that you want to convert.

Example

FORMULA	DESCRIPTION	RESULT
=DEGREES(PI())	Degrees of pi radians	180

DIVIDE

12/10/2018 • 2 minutes to read

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.

Example

The following example returns 2.5.

```
=DIVIDE(5,2)
```

Example

The following example returns BLANK.

```
=DIVIDE(5,0)
```

Example

The following example returns 1.

```
=DIVIDE(5,0,1)
```

See also

QUOTIENT function (DAX)
Math and Trig functions (DAX)

EVEN

12/10/2018 • 2 minutes to read

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.

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

12/10/2018 • 2 minutes to read

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 (DAX)
LN function (DAX)
EXP function (DAX)
LOG function (DAX)
LOG function (DAX)

FACT

12/10/2018 • 2 minutes to read

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.

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
5	120
170	7.257415615308E+306

See also

Math and Trig functions (DAX)
TRUNC function (DAX)

FLOOR

12/10/2018 • 2 minutes to read

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.

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],.5)
```

The following table shows the expected results for some sample values.

VALUES	EXPECTED RESULT
10.8423	10.8
8.0373	8
2.9733	2.9

See also

Math and Trig functions (DAX)

GCD

12/10/2018 • 2 minutes to read

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 >=2^53, GCD returns the #NUM! error value.

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

12/10/2018 • 2 minutes to read

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 (DAX)
ROUND function (DAX)
ROUNDUP function (DAX)
ROUNDDOWN function (DAX)
MROUND function (DAX)

ISO.CEILING

12/10/2018 • 2 minutes to read

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

Description

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.

Code

```
=ISO.CEILING(4.42,0.05)
```

Example: Negative Numbers

Description

The following formula returns the ISO ceiling value of -4.40.

Code

```
=ISO.CEILING(-4.42,0.05)
```

See also

Math and Trig functions (DAX)
FLOOR function (DAX)
CEILING function (DAX)
ROUNDUP function (DAX)

LCM

12/10/2018 • 2 minutes to read

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 LCM(a,b) >=2^53, LCM returns the #NUM! error value.

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

12/10/2018 • 2 minutes to read

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 (DAX)
EXP function (DAX)

LOG

12/10/2018 • 2 minutes to read

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 (DAX)
EXP function (DAX)
LOG function (DAX)
LOG function (DAX)

LOG10

12/10/2018 • 2 minutes to read

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 (DAX)
EXP function (DAX)
LOG function (DAX)
LOG function (DAX)

MOD

12/10/2018 • 2 minutes to read

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: MOD(n, d) = n - d*INT(n/d)

Example

The following formula returns 1, the remainder of 3 divided by 2.

```
=MOD(3,2)
```

Example

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 (DAX)
ROUND function (DAX)
ROUNDUP function (DAX)
ROUNDDOWN function (DAX)
MROUND function (DAX)

INT function (DAX)

MROUND

12/10/2018 • 2 minutes to read

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

Description

The following expression rounds 1.3 to the nearest multiple of .2. The expected result is 1.4.

Code

```
=MROUND(1.3,0.2)
```

Example: Negative Numbers

Description

The following expression rounds -10 to the nearest multiple of -3. The expected result is -9.

Code

```
=MROUND(-10,-3)
```

Example: Error

Description

The following expression returns an error, because the numbers have different signs.

Code

```
=MROUND(5,-2)
```

See also

Math and Trig functions (DAX)
ROUND function (DAX)
ROUNDUP function (DAX)
ROUNDDOWN function (DAX)
MROUND function (DAX)
INT function (DAX)

ODD

12/10/2018 • 2 minutes to read

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.

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

12/10/2018 • 2 minutes to read

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 (DAX)

POWER

12/10/2018 • 2 minutes to read

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 (DAX)

PRODUCT

12/10/2018 • 2 minutes to read

Returns the product of the numbers in a column.

To return the product of an expression evaluated for each row in a table, use PRODUCTX function (DAX).

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

Only the numbers in the column are counted. Blanks, logical values, and text are ignored.

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

Example

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

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

See also

PRODUCTX function (DAX)

PRODUCTX

12/10/2018 • 2 minutes to read

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

To return the product of the numbers in a column, use PRODUCT function (DAX).

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

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.

Example

The following computes the future value of an investment:

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

See also

PRODUCT function (DAX)

QUOTIENT

12/10/2018 • 2 minutes to read

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 (DAX)

RADIANS

12/10/2018 • 2 minutes to read

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 3π/2 radians)	4.712389

RAND

12/10/2018 • 2 minutes to read

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 workbook is set to **Manual** or **Automatic** recalculation mode, and whether data has been refreshed. This is different from Microsoft Excel, where you can control when RAND generates a new random number by turning off recalculation.

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.

Moreover, RAND is always recalculated if the function is used in the definition of a measure.

Also, in such contexts the RAND function cannot return a result of zero, to prevent errors such as division by zero.

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see https://go.microsoft.com/fwlink/?LinkId=219172.

Example

To generate a random real number between two other numbers, you can use a formula like the following:

```
= RAND()*(int1-int2)+int1
```

See also

Math and Trig functions (DAX)
Statistical functions (DAX)

RANDBETWEEN

12/10/2018 • 2 minutes to read

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 DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see https://go.microsoft.com/fwlink/?LinkId=219172.

Example

The following formula returns a random number between 1 and 10.

```
=RANDBETWEEN(1,10)
```

See also

Math and Trig functions (DAX)
Statistical functions (DAX)

ROUND

12/10/2018 • 2 minutes to read

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.

You can use the functions TRUNC and INT to obtain the integer portion of the number.

Example

The following formula rounds 2.15 up, to one decimal place. The expected result is 2.2.

```
=ROUND(2.15,1)
```

Example

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 (DAX)
ROUND function (DAX)
ROUNDDOWN function (DAX)
MROUND function (DAX)
INT function (DAX)
TRUNC function (DAX)

ROUNDDOWN

12/10/2018 • 2 minutes to read

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.

Related functions

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

The following example rounds 3.14159 down to three decimal places. The expected result is 3.141.

```
=ROUNDDOWN(3.14159,3)
```

Example

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 (DAX)
ROUND function (DAX)
ROUNDUP function (DAX)
ROUNDDOWN function (DAX)
MROUND function (DAX)
INT function (DAX)

ROUNDUP

12/10/2018 • 2 minutes to read

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

ROUNDUP behaves like ROUND, except that it always rounds a number up.

- 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.

Related functions

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

Description

The following formula rounds 1.3 to the nearest multiple of 0.2. The expected result is 2.

Code

```
=ROUNDUP(1.3,0.2)
```

Example: Negative Number as Second Argument

Description

The following formula rounds the value in the column, **FreightCost**, with the expected results shown in the following table:

Code

```
=ROUNDUP([Values],-1)
```

Comments

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 (DAX)
ROUND function (DAX)
ROUNDDOWN function (DAX)
MROUND function (DAX)
INT function (DAX)

SIGN

12/10/2018 • 2 minutes to read

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 (DAX)

SQRT

12/10/2018 • 2 minutes to read

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 returns 5.

```
=SQRT(25)
```

See also

Math and Trig functions (DAX)

SUM

12/10/2018 • 2 minutes to read

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 function (DAX)
Statistical functions (DAX)

SUMX

12/10/2018 • 2 minutes to read

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.

To see some more complex examples of SUMX in formulas, see ALL function (DAX) and CALCULATETABLE function (DAX).

Example

The following example first filters the table, InternetSales, on the expression, `ShippingTerritoryID = 5`, and then returns the sum of all values in the column, Freight. 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 function (DAX)
Statistical functions (DAX)

TRUNC

12/10/2018 • 2 minutes to read

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

The following formula returns 3, the integer part of pi.

```
=TRUNC(PI())
```

Example

The following formula returns -8, the integer part of -8.9.

```
=TRUNC(-8.9)
```

See also

Math and Trig functions (DAX)
ROUND function (DAX)
ROUNDUP function (DAX)
ROUNDDOWN function (DAX)

MROUND function (DAX)
INT function (DAX)

Other functions

12/10/2018 • 2 minutes to read

These functions perform unique actions that cannot be defined by any of the categories.

In this section

DATATABLE

ERROR

EXCEPT

GENERATESERIES

GROUPBY

INTERSECT

ISEMPTY

NATURALINNERJOIN

NATURALLEFTOUTERJOIN

SUMMARIZECOLUMNS

Table Constructor (DAX)

TREATAS

UNION

VAR (DAX)

DATATABLE

12/10/2018 • 2 minutes to read

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	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 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. 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. A missing value will be treated identically to BLANK(). For example, the following are the same: {1,2,BLANK(),4} {1,2,,4}

Return value

A table declaring an inline set of values.

Remarks

Unlike DATATABLE, Table Constructor allows any scalar expressions as input values.

Example

```
=DataTable("Name", STRING,
               "Region", STRING
               ,{
                        {" User1","East"},
                        {" User2","East"},
                        {" User3","West"},
                        {" User4","West"},
                        {" User4","East"}
                }
          )
```

ERROR

12/10/2018 • 2 minutes to read

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.

Examples

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 and 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.

EXCEPT

12/10/2018 • 2 minutes to read

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.

Example

States1

STATE
A
B
B

STATE
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

GENERATESERIES

12/10/2018 • 2 minutes to read

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.

Examples

Example 1

The following DAX query:

```
EVALUATE GENERATESERIES(1, 5)
```

Returns the following table with a single column:

[VALUE]	
1	
2	
3	

[VALUE]	
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

12/10/2018 • 3 minutes to read

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 attempts to reuse the data that has been grouped making it highly performant.

Syntax

```
GROUPBY (<table>, [<groupBy_columnName1>], [<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	Any DAX expression that returns a single scalar value, where the expression is to be evaluated for each set of GroupBy values. **Note:** The expression used in GroupBy may include any of the "X" aggregation functions, such as SUMX, AVERAGEX, MINX, MAXX, etc. and when one of these function is used in this way, we allow the table argument (which normally must be a table expression) to be replaced by a special CURRENTGROUP() function as described elsewhere in this document. Restrictions on expression: - The CALCULATE function (and therefore measures) are not allowed in the expression. - The CURRENTGROUP function may only be used at the top level of table scans in the expression. That is, SUMX(<table>,SUMX(CURRENTGROUP(…), …)) is not allowed. ABS(SUMX(CURRENTGROUP(), [Column])) is allowed, since ABS does not perform a scan.

Return value

A table with the selected columns for the groupBy_columnName arguments and the grouped by columns designated by the name arguments.

Remarks

The GROUPBY function does the following:

1. Start with the specified table (and all related tables in the “to-one” direction).
2. Create a grouping using all of the GroupBy columns (which are required to exist in the table from step #1.).
3. Each group is one row in the result, but represents a set of rows in the original table.
4. 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.

Notes:

- 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.

Options

CURRENTGROUP()

CURRENTGROUP can only be used in an expression that defines a 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

Assume a data model has four tables: Sales, Customer, Product, Geography where Sales is on the “many” side of a relationship to each of the other three tables.

```
GROUPBY (
Sales,
Geography[Country],
Product[Category],
“Total Sales”, SUMX( CURRENTGROUP(), Sales[Price] * Sales[Qty])
)
```

This will start with the Sales table, extended with all the columns from all the related tables.Then it will build a result with three columns.

- The first column is each of the countries for which there is a sale.
- The second column is each product category for which there is a sale in that country.
- The third column is the sum of sales amount (as calculated from price*qty) for the selected country and product category.

Suppose we’ve built the previous result. We can use GROUPBY again, to find the maximum category sales figure within each country as shown here.

```
DEFINE
VAR SalesByCountryAndCategory =
GROUPBY (
Sales,
Geography[Country],
Product[Category],
“Total Sales”, SUMX( CURRENTGROUP(), Sales[Price] * Sales[Qty])
)

Evaluate GROUPBY (
SalesByCountryAndCategory,
Geography[Country],
 “Max Sales”, MAXX( CURRENTGROUP(), [Total Sales])
)
```

See also

SUMMARIZE function (DAX)
SUMMARIZECOLUMNS function (DAX)

INTERSECT

12/10/2018 • 2 minutes to read

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.

Example

States1

STATE
A
A
B

STATE
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
B
C

STATE
D
D
D

ISEMPTY

12/10/2018 • 2 minutes to read

Checks if a table is empty.

Syntax

```
ISEMPTY(<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.

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(ISEMPTY(FILTER(Info, [Count]>25))))
```

Return value: **FALSE**

NATURALINNERJOIN

12/10/2018 • 2 minutes to read

Performs an inner join of a table with another table. The 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.

Syntax

```
NATURALINNERJOIN(<leftJoinTable>, <rightJoinTable>)
```

Parameters

TERM	DEFINITION
leftJoinTable	A table expression defining the table on the left side of the join.
rightJoinTable	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

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[ProductdID], StoreSales[ProductdID] 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.

NATURALLEFTOUTERJOIN

12/10/2018 • 2 minutes to read

Performs an inner join of a table with another table. The 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.

Syntax

```
NATURALLEFTOUTERJOIN(<leftJoinTable>, <rightJoinTable>)
```

Parameters

TERM	DEFINITION
leftJoinTable	A table expression defining the table on the left side of the join.
rightJoinTable	A table expression defining the table on the right side of the join.

Return value

A table which includes only rows from rightJoinTable for which the values in the common columns specified are also present in leftJoinTable. The table returned will have the common columns from the left table and the other columns from both the tables.

Remarks

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[ProductdID], StoreSales[ProductdID] 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.

SUMMARIZECOLUMNS

12/10/2018 • 6 minutes to read

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

SUMMARIZECOLUMNS 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 formulas are invalid.

```
SUMMARIZECOLUMNS( Sales[StoreId], Sales[StoreId] )
```

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 (i.e.

from Customer table). The filter is not applied to the groupBy columns. The Sales Territory and the Customer table 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.

Options

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.

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

This function does not return a value.

Remarks

IGNORE can be used as an expression argument to SUMMARIZECOLUMNS.

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

CUSTOMERID	TOTALQTY	BLANKIFTOTALQTYISNOT3
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 non-ignored expressions which return blank.

CUSTOMERID	TOTALQTY	BLANKIFTOTALQTYISNOT3
A		5
B		
C		

NONVISUAL

Marks a value filter in SUMMARIZECOLUMNS function as not affecting measure values, but only applying to group-by columns.

Syntax

```
NONVISUAL(<expression>)
```

Return value

A table of values.

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]
```

Result

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]
```

Result

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

ROLLUPADDISSUBTOTAL()

The addition of the ROLLUPADDISSUBTOTAL() syntax modifies the behavior of the SUMMARIZECOUMNS function by adding roll-up/subtotal rows to the result based on the `groupBy_columnName` columns.

Syntax

```
ROLLUPADDISSUBTOTAL ( [<filter> …, ] <groupBy_columnName>, <isSubtotal_columnName>[, <filter> …][,
<groupBy_columnName >, <isSubtotal_columnName>[, <filter> …]…] )
```

Parameters

TERM	DEFINITION
groupBy_columnName	The qualified name of an existing column to be used to create summary groups based on the values found in it. This parameter cannot be an expression.
isSubtotal_columnName	The name of the Boolean column to be added to the result, indicating whether or not a row is a subtotal over the groupBy column (or columns when used with ROLLUPGROUP). This value is calculated using the ISSUBTOTAL function.
filter	A table expression which is added to the filter context at the current grouping level. A filter before the first group-by column is applied at the grand total level.

Return value

The function does not return a value. It only specifies the set of columns to be subtotaled.

Example

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]
```

Result

CATEGORY	SUBCATEGORY	ISCATEGORYSUBTOTAL	ISSUBCATEGORYSUBTOTAL	TOTAL QTY
		True	True	60398
Accessories		False	True	36092
Accessories	Bike Racks	False	False	328
Bikes	Mountain Bikes	False	False	4970
Clothing		False	True	9101

Multiple subtotals

```
SUMMARIZECOUMNS ( 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.

Result

CUSTOMERID	ISCUSTOMERSUBTOTAL	STATE	TOTAL QTY	DATE	ISDATESUBTOTAL
A	FALSE	WA	5	7/10/2014	
B	FALSE	WA	1	7/10/2014	

CUSTOMERID	ISCUSTOMERSUBTOTAL	STATE	TOTAL QTY	DATE	ISDATESUBTOTAL
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

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).

Syntax

```
ROLLUPGROUP(<groupBy_columnName>, <groupBy_columnName>)
```

With ROLLUPADDISSUBTOTAL

```
ROLLUPADDISSUBTOTAL( ROLLUPGROUP(…), isSubtotal_columnName[, <groupBy_columnName>…] )
```

Parameters

TERM	DEFINITION
groupBy_columnName	The qualified name of an existing column to be used to create summary groups based on the values found in it. The set of group by columns supplied to ROLLUPGROUP function will define the granularity to return subtotal rows for (same behavior as when ROLLUP and ROLLUPGROUP are used with the SUMMARIZE function). This parameter cannot be an expression.

Return value

The function does not return a value. It marks a set of columns to be grouped during subtotaling by

ROLLUPADDISSUBTOTAL.

Remarks

ROLLUPGROUP can only be used as an groupBy_columnName argument to ROLLUPADDISSUBTOTAL or the SUMMARIZE function.

Example

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.

Result

STATE	CUSTOMERID	ISCUSTOMERSUBTOTAL	TOTAL QTY	CITY	ISCITYSTATESUBTOTAL
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 function (DAX)

Table Constructor

12/10/2018 • 2 minutes to read

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.

Examples

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	

[VALUE]	
2	
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")
 }
```

[VALUE1]	[VALUE2]	[VALUE3]	[VALUE4]
1.5	1/1/2017	199.99	A
Row2	1/2/2017	249.99	B
Row3	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	

TREATAS

12/10/2018 • 2 minutes to read

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 are "Red" and "Green".

Best for use when a relationship does not exist between the tables.

Examples

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 function

FILTER function

UNION

12/10/2018 • 2 minutes to read

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.

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

COUNTRY	STATE	COUNT	TOTAL
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

VAR

12/10/2018 • 2 minutes to read

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.

Exceptions

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.

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(Sales, 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. In addition, the result of the expression is stored in the variable upon declaration. It doesn’t have to be recalculated each time it is used, as it would without using a variable. This can improve the measure’s performance.

Parent and Child functions

12/10/2018 • 2 minutes to read

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 section

PATH

PATHCONTAINS

PATHITEM

PATHITEMREVERSE

PATHLENGTH

Understanding functions for parent-child hierarchies in DAX

3/29/2019 • 3 minutes to read

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 (DAX) - 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 (DAX) - 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 (DAX) - 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 EmployeKey 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 (DAX) - 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 EmployeKey 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 (DAX) - 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

EMPLOYEEKEY	PARENTEMPLOYEEKEY	PATH	PATHCONTAINS - EMPLOYEE 162
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

12/10/2018 • 2 minutes to read

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 *ID_columnName* must be text or integer, and must also be the same data type as the column referenced in *parent_columnName*.
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 *parent_columnName* data type must be text or integer, and must be the same data type as the value in *ID_columnName*.

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 DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see https://go.microsoft.com/fwlink/?LinkId=219172.

Example

The following example creates a calculated column that lists all the managers for each employee.

```
=PATH(Employee[EmployeeKey], Employee[ParentEmployeeKey])
```

PATHCONTAINS

12/10/2018 • 2 minutes to read

Returns **TRUE** if the specified *item* exists within the specified *path*.

Syntax

```
PATHCONTAINS(<path>, <item>)
```

Parameters

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 DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see https://go.microsoft.com/fwlink/?LinkId=219172.

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

12/10/2018 • 2 minutes to read

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

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:

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 DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see https://go.microsoft.com/fwlink/?LinkId=219172.

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

12/10/2018 • 2 minutes to read

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

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:

Enumeration	**Alternate Enumeration**	**Description**
TEXT	0	Results are returned with the data type of text. (default)
INTEGER	1	Results are returned with the data type of integer.

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 DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see https://go.microsoft.com/fwlink/?LinkId=219172.

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

12/10/2018 • 2 minutes to read

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 DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see https://go.microsoft.com/fwlink/?LinkId=219172.

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]))
```

Statistical functions

3/18/2019 • 2 minutes to read

Data Analysis Expressions (DAX) provides many functions for creating aggregations such as sums, counts, and averages. These functions are very similar to aggregation functions used by Microsoft Excel. This section lists the statistical and aggregation functions provided in DAX.

In this section

MAX

MAXA

MAXX

MEDIAN

MEDIANX

MIN

MINA

MINX

NORM.DIST

NORM.INV

NORM.S.DIST

NORM.S.INV

PERCENTILE.EXC

PERCENTILE.INC

PERCENTILEX.EXC

PERCENTILEX.INC

POISSON.DIST

RANK.EQ

RANKX

ROW

SAMPLE

SELECTCOLUMNS

SIN

SINH

STDEV.P

STDEV.S

STDEVX.P

STDEVX.S

SQRTPI

SUMMARIZE

T.DIST

T.DIST.2T

T.DIST.RT

T.INV

T.INV.2t

TAN

TANH

TOPN

VAR.P

VAR.S

VARX.P

VARX.S

XIRR

XNPV

ADDCOLUMNS

12/10/2018 • 2 minutes to read

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 *table*.

Return value

A table with all its original columns and the added ones.

Remarks

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[ProductCategoryName]	ProductCategory[ProductCategoryAlternateKey]	ProductCategory[ProductCategoryKey]	[Internet Sales]	[Reseller Sales]
Bikes	1	1	25107749.77	63084675.04
Components	2	2		11205837.96
Clothing	3	3	306157.5829	1669943.267

Accessories	4	4	640920.1338	534301.9888

APPROXIMATEDISTINCTCOUNT

12/10/2018 • 2 minutes to read

IMPORTANT

This function is **preview**. It is currently not supported by Microsoft Support. For additional limitations, see Remarks.

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.

This function is **preview**. The following limitations apply:

- This function currently supports **DirectQuery** connections only to the following data sources:
  - Azure SQL Database
  - Azure SQL Data Warehouse
- This feature is not yet available in Intellisense.

AVERAGE

12/10/2018 • 2 minutes to read

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 functions 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.

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.

See also

AVERAGEA function (DAX)
AVERAGEX function (DAX)
Statistical functions (DAX)

AVERAGEA

12/10/2018 • 2 minutes to read

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.

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
0000124	20	Counts as 20

TRANSACTION ID	AMOUNT	RESULT
0000125	n/a	Counts as 0
0000126		Counts as 0
0000126	TRUE	Counts as 1

```
=AVERAGEA([Amount])
```

See also

AVERAGE function (DAX)
AVERAGEX function (DAX)
Statistical functions (DAX)

AVERAGEX

12/10/2018 • 2 minutes to read

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.

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 (DAX)
AVERAGEA function (DAX)
Statistical functions (DAX)

BETA.DIST

12/10/2018 • 2 minutes to read

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 ≤ 0 or beta ≤ 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.

BETA.INV

12/10/2018 • 2 minutes to read

Returns the inverse of the beta cumulative probability density function (BETA.DIST).

If probability = BETA.DIST(x,...TRUE), then BETA.INV(probability,...) = 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 ≤ 0 or beta ≤ 0, BETA.INV returns the #NUM! error value.

If probability ≤ 0 or 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.

CHISQ.INV

12/10/2018 • 2 minutes to read

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 < 1 or deg_freedom > 10^10, CHISQ.INV returns the #NUM! error value.

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

12/10/2018 • 2 minutes to read

Returns the inverse of the right-tailed probability of the chi-squared distribution.

If probability = CHISQ.DIST.RT(x,...), then CHISQ.INV.RT(probability,...) = 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 probability < 0 or probability > 1, CHISQ.INV.RT returns the #NUM! error value.

If deg_freedom is not an integer, it is rounded.

If 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 CHISQ.DIST.RT(x, deg_freedom) = 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.

CONFIDENCE.NORM

12/10/2018 • 2 minutes to read

The confidence interval is a range of values. Your sample mean, x, is at the center of this range and the range is x ± CONFIDENCE.NORM. For example, if x is the sample mean of delivery times for products ordered through the mail, x ± CONFIDENCE.NORM is a range of population means. For any population mean, μ0, in this range, the probability of obtaining a sample mean further from μ0 than x is greater than alpha; for any population mean, μ0, not in this range, the probability of obtaining a sample mean further from μ0 than x is less than alpha. In other words, assume that we use x, standard_dev, and size to construct a two-tailed test at significance level alpha of the hypothesis that the population mean is μ0. Then we will not reject that hypothesis if μ0 is in the confidence interval and will reject that hypothesis if μ0 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*(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 ≤ 0 or alpha ≥ 1, CONFIDENCE.NORM returns the #NUM! error value.

If standard_dev ≤ 0, CONFIDENCE.NORM returns the #NUM! error value.

If size is not an integer, it is rounded.

If 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:

$$\bar{x} \pm 1.96\left(\frac{\sigma}{\sqrt{n}}\right)$$

CONFIDENCE.T

12/10/2018 • 2 minutes to read

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 ≤ 0 or alpha ≥ 1, CONFIDENCE.T returns the #NUM! error value.

If standard_dev ≤ 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.

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

COUNT

12/10/2018 • 2 minutes to read

The COUNT function counts the number of cells in a column that contain numbers.

Syntax

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

Parameters

TERM	DEFINITION
column	The column that contains the numbers 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

If the row contains text that cannot be translated into a number, the row is not counted.

When the function finds no rows to count, it returns a blank. When there are rows, but none of them meet the specified criteria, then the function returns 0.

Example

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

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

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

See also

COUNT function (DAX)
COUNTA function (DAX)
COUNTAX function (DAX)
COUNTX function (DAX)
Statistical functions (DAX)

COUNTA

12/10/2018 • 2 minutes to read

The COUNTA function counts the number of cells in a column that are not empty. It counts not just rows that contain numeric values, but also rows that contain nonblank values, including text, dates, and logical values.

Syntax

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

Parameters

TERM	DEFINITION
column	The column that contains the values to be counted

Return value

A whole number.

Remarks

If you do not need to count cells that contain logical values or text (in other words, if you want to count only cells that contain numbers), use the COUNT or COUNTX functions.

When the function does not find any rows to count, the function returns a blank. When there are rows, but none of them meet the specified criteria, then the function returns 0.

Example

The following example returns all rows in the `Reseller` table that have any kind of value in the column that stores phone numbers. Because the table name does not contain any spaces, the quotation marks are optional.

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

See also

COUNT function (DAX)
COUNTA function (DAX)
COUNTAX function (DAX)
COUNTX function (DAX)
Statistical functions (DAX)

COUNTAX

12/10/2018 • 2 minutes to read

The COUNTAX function counts nonblank 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 nonblank 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 nonblank. Usually the COUNTAX function does not count empty cells but in this case the cell contains a formula, so it is counted.

If you do not need to count logical values or text, use the COUNTX function instead.

Whenever the function finds 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 0 if no rows are found that meet the condition.

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 (DAX)

COUNTA function (DAX)
COUNTAX function (DAX)
COUNTX function (DAX)
Statistical functions (DAX)

COUNTBLANK

12/10/2018 • 2 minutes to read

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 DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see https://go.microsoft.com/fwlink/?LinkId=219172.

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 (DAX)
COUNTA function (DAX)
COUNTAX function (DAX)
COUNTX function (DAX)

Statistical functions (DAX)

COUNTROWS

12/10/2018 • 2 minutes to read

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	The name of the table that contains the rows to be counted, or an expression that returns a table.

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.

Example

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

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

Example

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 (DAX)
COUNTA function (DAX)
COUNTAX function (DAX)
COUNTX function (DAX)
Statistical functions (DAX)

COUNTX

12/10/2018 • 2 minutes to read

Counts the number of rows that contain a number or an expression that evaluates to a number, 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 numeric values, dates, or strings. Arguments that are logical values or text that cannot be translated into numbers are not counted. If the function finds no rows to count, it returns a blank. When there are rows, but none meets the specified criteria, then the function returns 0.

If you want to count logical values, or text, use the COUNTA or COUNTAX functions.

Example

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

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

Example

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 (DAX)
COUNTA function (DAX)
COUNTAX function (DAX)
COUNTX function (DAX)
Statistical functions (DAX)

CROSSJOIN

12/10/2018 • 2 minutes to read

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** column; then, the resulting table has **rA** × **rb** × **rC** rows and **cA** + **cB** + **cC** columns.

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

FONT	PRESENTATION
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

DISTINCTCOUNT

12/10/2018 • 2 minutes to read

The DISTINCTCOUNT function 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.

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

|**Distinct Reseller Orders count**|**Column Labels**||||||
|-|-|-|-|-|-|-|
|**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|
|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 (DAX)](count-function-dax.md)
[COUNTA function (DAX)](counta-function-dax.md)
[COUNTAX function (DAX)](countax-function-dax.md)
[COUNTX function (DAX)](countx-function-dax.md)
[Statistical functions (DAX)](statistical-functions-dax.md)
```
```

EXPON.DIST

12/10/2018 • 2 minutes to read

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 ≤ 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:

$$F(x; \lambda) = 1 - e^{-\lambda x}$$

GENERATE

12/10/2018 • 2 minutes to read

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.

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
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 code 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
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

12/10/2018 • 2 minutes to read

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.

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
Europe	Components	$ 2,214,440.19
NA	Accessories	

SALESTERRITORY[SALESTERRITORYGROUP]	PRODUCTCATEGORY[PRODUCTCATEGORYNAME]	[RESELLER SALES]
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 code 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

PRODUCTCATEGORY[PRODUCTCATEGORYNAME]	[RESELLER SALES]
Components	$ 11,205,837.96
Clothing	$ 1,669,943.27
Accessories	$ 534,301.99

3. 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.

GEOMEAN

12/10/2018 • 2 minutes to read

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 (DAX).

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])

Example

The following computes the geometric mean of the Return column in the Investment table:

```
=GEOMEAN( Investment[Return] )
```

See also

GEOMEANX function (DAX)

GEOMEANX

12/10/2018 • 2 minutes to read

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 (DAX).

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.

Example

The following computes the geometric mean of the ReturnPct column in the Investments table:

```
=GEOMEANX( Investments, Investments[ReturnPct] + 1 )
```

See also

GEOMEAN function (DAX)

MAX

12/10/2018 • 2 minutes to read

Returns the largest numeric 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 numeric value.
expression	Any DAX expression which returns a single numeric value.

Property Value/Return value

A decimal number.

Remarks

When evaluating a single column that contains numeric values, if the column contains no numbers, MAX returns a blank. If you want to evaluate values that are not numbers, use the MAXA function.

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.

Example

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

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

Example

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

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

See also

MAX function (DAX)
MAXA function (DAX)
MAXX function (DAX)
Statistical functions (DAX)

MAXA

12/10/2018 • 2 minutes to read

Returns the largest value in a column. Logical values and blanks are counted.

Syntax

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

Parameters

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

Return value

A decimal number.

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 do not want to include logical values and blanks as part of the calculation, use the MAX function.

Example

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

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 (DAX)
MAXA function (DAX)
MAXX function (DAX)
Statistical functions (DAX)

MAXX

12/10/2018 • 2 minutes to read

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

A decimal number.

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. If the expression does not evaluate to a number, MAXX returns 0 (zero).
- Dates.

Empty cells, logical values, and text values are ignored. If you want to include non-numeric values in the formula, use the MAXA function.

If a blank cell is included in the column or expression, MAXX returns an empty column.

Example

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

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 (DAX)
MAXA function (DAX)
MAXX function (DAX)
Statistical functions (DAX)

MEDIAN

12/10/2018 • 2 minutes to read

Returns the median of numbers in a column.

To return the median of an expresssion evaluated for each row in a table, use MEDIANX function (DAX).

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, logical values, and text are ignored.

MEDIAN(Table[Column]) is equivalent to MEDIANX(Table, Table[Column]).

Example

The following computes the median of a column named Age in a table named Customers:

```
=MEDIAN( Customers[Age] )
```

See also

MEDIANX function (DAX)

MEDIANX)

12/10/2018 • 2 minutes to read

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 (DAX).

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

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 (DAX)

MIN

12/10/2018 • 2 minutes to read

Returns the smallest numeric value in a column, or between two scalar expressions. Ignores logical values and text.

Syntax

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

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

Parameters

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

Return value

A decimal number.

Remarks

The MIN function takes a column or two expressions as an argument, and returns the smallest numeric value. The following types of values in the columns are counted:

- Numbers
- Dates
- Blanks
- If the column contains no numerical data, MIN returns blanks.

When evaluating a column, empty cells, logical values, and text are ignored. If you want to include logical values and text representations of numbers in a reference as part of the calculation, use the MINA function.

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.

Example

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

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

Example

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

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

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

See also

MIN function (DAX)
MINA function (DAX)
MINX function (DAX)
Statistical functions (DAX)

MINA

12/10/2018 • 2 minutes to read

Returns the smallest value in a column, including any logical values and numbers represented as text.

Syntax

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

Parameters

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

Return value

A decimal number.

Remarks

The MINA function takes as argument a column that contains numbers, and determines the smallest value as follows:

- If the column contains no numeric 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 do not want to include logical values and text as part of the calculation, use the MIN function instead.

Example

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

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

Example

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 numeric values, and the formula returns zero (0).

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

See also

MIN function (DAX)

MINA function (DAX)
MINX function (DAX)
Statistical functions (DAX)

MINX

12/10/2018 • 2 minutes to read

Returns the smallest numeric 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 decimal number.

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.

The MINX function evaluates the results of the expression in the second argument according to the following rules:

- Only numbers are counted. If the expression does not result in a number, MINX returns 0 (zero).
- Empty cells, logical values, and text values are ignored. Numbers represented as text are treated as text.

If you want to include logical values and text representations of numbers in a reference as part of the calculation, use the MINA function.

Example

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

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])
```

Comments

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 (DAX)
MINA function (DAX)
MINX function (DAX)
Statistical functions (DAX)

NORM.DIST

12/10/2018 • 2 minutes to read

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 mass function.

Return value

The normal distribution for the specified mean and standard deviation.

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

12/10/2018 • 2 minutes to read

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.

Example

```
EVALUATE { NORM.INV(0.908789, 40, 1.5) }
```

Returns

[VALUE]
42.000002009566287802741 32

See also

NORM.S.INV
NORM.S.DIST function
NORM.DIST function

NORM.S.DIST

12/10/2018 • 2 minutes to read

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 mass function.

Return value

The standard normal distribution (has a mean of zero and a standard deviation of one.

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

12/10/2018 • 2 minutes to read

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.

Example

```
EVALUATE { NORM.S.INV(0.908789) }
```

Returns

[VALUE]
1.33333467304411

See also

NORM.INV
NORM.S.DIST function
NORM.DIST function

PERCENTILE.EXC

12/10/2018 • 2 minutes to read

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 (DAX).

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.

See also

PERCENTILEX.EXC function (DAX)

PERCENTILE.INC

12/10/2018 • 2 minutes to read

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 function (DAX).

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.

See also

PERCENTILEX.INC function (DAX)

PERCENTILEX.EXC

12/10/2018 • 2 minutes to read

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 (DAX).

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.

See also

PERCENTILE.EXC function (DAX)

PERCENTILEX.INC

12/10/2018 • 2 minutes to read

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 function (DAX).

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.

See also

PERCENTILE.INC function (DAX)

PERMUT

12/10/2018 • 2 minutes to read

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:

$$P_{k,n} = \frac{n!}{(n-k)!}$$

Example

FORMULA	DESCRIPTION	RESULT
=PERMUT(3,2)	Permutations possible for a group of 3 objects where 2 are chosen.	6

POISSON.DIST

12/10/2018 • 2 minutes to read

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 mean < 0, POISSON.DIST returns the #NUM! error value.

POISSON.DIST is calculated as follows.

For cumulative = FALSE:

$$POISSON = \frac{e^{-\lambda}\,\lambda^{x}}{x!}$$

For cumulative = TRUE:

$$CUMPOISSON = \sum_{k=0}^{x} \frac{e^{-\lambda}\,\lambda^{k}}{k!}$$

RANK.EQ

12/10/2018 • 2 minutes to read

Returns the ranking of a number in a list of numbers.

Syntax

```
RANK.EQ(<value>, <columnName>[, <order>])
```

Parameters

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 it's 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 *number*, low to high or high to low:

value	**alternate value**	**Description**
0 (zero)	FALSE	Ranks in descending order of *columnName*. If *value* is equal to the highest number in *columnName* then **RANK.EQ** is 1.
1	TRUE	Ranks in ascending order of *columnName*. If *value* is equal to the lowest number in *columnName* then **RANK.EQ** is 1.

Return value

A number indicating the rank of *value* among the numbers in *columnName*.

Exceptions

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).

Example

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

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

12/10/2018 • 2 minutes to read

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	Ranks in descending order of values of expression. If value is equal to the highest number in expression then RANKX returns 1. This is the default value when order parameter is omitted.
1	TRUE	Ranks in ascending order of expression. If value is equal to the lowest number in expression then RANKX returns 1.

ties
(Optional) An enumeration that defines how to determine ranking when there are ties.

enumeration	**Description**

Skip	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). This is the default value when *ties* parameter is omitted.
Dense	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.

Return value

The rank number of *value* among all possible values of *expression* evaluated for all rows of *table* numbers.

Exceptions

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")

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]))
```

ROW function

12/10/2018 • 2 minutes to read

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. *name*.

Return value

A single row table

Remarks

Arguments must always come in pairs of *name* and *expression*.

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]))
```

The code is split in two lines for readability purposes

SAMPLE

12/10/2018 • 2 minutes to read

Returns a sample of N rows from the specified table.

Syntax

```
SAMPLE(<n_value>, <table>, <orderBy_expression>, [<order>[, <orderBy_expression>, [<order>]]…])
```

Parameters

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 *table*.

order
(Optional) A value that specifies how to sort *orderBy_expression* values, ascending or descending:

value	**alternate value**	**Description**
0 (zero)	FALSE	Sorts in descending order of values of *order_by*. This is the default value when *order* parameter is omitted.
1	TRUE	Ranks in ascending order of *order_by*.

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.

SELECTCOLUMNS

12/10/2018 • 2 minutes to read

Adds calculated columns to the given table or table expression.

Syntax

```
SELECTCOLUMNS(<table>, <name>, <scalar_expression> [, <name>, <scalar_expression>]…)
```

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>, <scalar_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.

Example

For the following table named **Info**:

COUNTRY	STATE	COUNT	TOTAL
IND	JK	20	800
IND	MH	25	1000
IND	WB	10	900
USA	CA	5	500
USA	WA	10	900

```
SELECTCOLUMNS(Info, “StateCountry”, [State]&”, ”&[Country])
```

Returns:

STATECOUNTRY
IND, JK
IND, MH
IND, WB
USA, CA
USA, WA

SIN

12/10/2018 • 2 minutes to read

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 your argument is in degrees, multiply it by 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

12/10/2018 • 2 minutes to read

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:

$$SINH(z) = \frac{e^{z} - e^{-z}}{2}$$

Example

FORMULA	DESCRIPTION	RESULT
=2.868*SINH(0.0342*1.03)	Probability of obtaining a result of less than 1.03 seconds.	0.1010491

STDEV.S

12/10/2018 • 2 minutes to read

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

1. 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.
2. 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
3. Blank rows are filtered out from *columnName* and not considered in the calculations.
4. An error is returned if *columnName* contains less than 2 non-blank rows.

This DAX function may return different results when used in a model that is deployed and then queried in DirectQuery mode. For more information about semantic differences in DirectQuery mode, see https://go.microsoft.com/fwlink/?LinkId=219171.

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

12/10/2018 • 2 minutes to read

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.

Exceptions

Remarks

1. 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.
2. STDEV.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
3. Blank rows are filtered out from *columnName* and not considered in the calculations.
4. An error is returned if *columnName* contains less than 2 non-blank rows

This DAX function may return different results when used in a model that is deployed and then queried in DirectQuery mode. For more information about semantic differences in DirectQuery mode, see https://go.microsoft.com/fwlink/?LinkId=219171.

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

12/10/2018 • 2 minutes to read

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

1. 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.

2. 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

3. Blank rows are filtered out from *columnName* and not considered in the calculations.

4. An error is returned if *columnName* contains less than 2 non-blank rows.

This DAX function may return different results when used in a model that is deployed and then queried in DirectQuery mode. For more information about semantic differences in DirectQuery mode, see https://go.microsoft.com/fwlink/?LinkId=219171.

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

12/10/2018 • 2 minutes to read

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

1. 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.
2. 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
3. Blank rows are filtered out from *columnName* and not considered in the calculations.
4. An error is returned if *columnName* contains less than 2 non-blank rows

This DAX function may return different results when used in a model that is deployed and then queried in DirectQuery mode. For more information about semantic differences in DirectQuery mode, see https://go.microsoft.com/fwlink/?LinkId=219171.

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(RELATEDTABLE(InternetSales_USD), InternetSales_USD[UnitPrice_USD] –
(InternetSales_USD[DiscountAmount_USD]/InternetSales_USD[OrderQuantity]))
```

SQRTPI

12/10/2018 • 2 minutes to read

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).

Remarks

xxxxx

Example

FORMULA	DESCRIPTION	RESULT
=SQRTPI(1)	Square root of pi.	1.772454
=SQRTPI(2)	Square root of 2 * pi.	2.506628

SUMMARIZE

12/10/2018 • 5 minutes to read

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 to be 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

1. 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.
2. groupBy_columnName must be either in *table* or in a related table to *table*.
3. Each name must be enclosed in double quotation marks.
4. 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.

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

Advanced SUMMARIZE options

SUMMARIZE with ROLLUP

The addition of the ROLLUP() syntax modifies the behavior of the SUMMARIZE function by adding roll-up rows to the result on the groupBy_columnName columns.

```
SUMMARIZE(<table>, <groupBy_columnName>[, <groupBy_columnName>]…[, ROLLUP(<groupBy_columnName>[,<
groupBy_columnName>…])][, <name>, <expression>]…)
```

ROLLUP parameters

groupBy_columnName
The qualified name of an existing column to be used to create summary groups based on the values found in it. This parameter cannot be an expression.

Note: All other SUMMARIZE parameters are explained before and not repeated here for brevity.

Remarks

- The columns mentioned in the ROLLUP expression cannot be referenced as part of a *groupBy_columnName* columns.

Example

The following example adds roll-up 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])
)
```

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

DATETIME[CALENDARYEAR]	PRODUCTCATEGORY[PRODUCTCATEGORYNAME]	[SALES AMOUNT (USD)]	[DISCOUNT AMOUNT (USD)]
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

ROLLUPGROUP

ROLLUPGROUP() can be used to calculate groups of subtotals. If used in-place of ROLLUP, ROLLUPGROUP will yield the same result by adding roll-up rows to the result on the groupBy_columnName columns. However, the addition of ROLLUPGROUP() inside a ROLLUP syntax can be used to prevent partial subtotals in roll-up rows.

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])
)
```

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

DATETIME[CALENDARYEAR]	PRODUCTCATEGORY[PRODUCTCATEGORYNAME]	[SALES AMOUNT (USD)]	[DISCOUNT AMOUNT (USD)]
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

SUMMARIZE with ISSUBTOTAL

Enables the user to create another column, in the Summarize function, that returns True if the row contains sub-total values for the column given as argument to ISSUBTOTAL, otherwise returns False.

```
SUMMARIZE(<table>, <groupBy_columnName>[, <groupBy_columnName>]…[, ROLLUP(<groupBy_columnName>[,<
groupBy_columnName>…])][, <name>, {<expression>|ISSUBTOTAL(<columnName>)}]…)
```

ISSUBTOTAL parameters

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 sub-total value for the column given as argument, otherwise returns **False**

Remarks

- ISSUBTOTAL can only be used in the expression part of a SUMMARIZE function.
- ISSUBTOTAL must be preceded by a matching *name* column.

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])
)
```

The following table shows a preview of the data as it would be received by any function expecting to receive a

table:

[IS SUB TOTAL FOR DATETIMECALENDARYEAR]	[IS SUB TOTAL FOR PRODUCTCATEGORYNAME]	DATETIME[CALENDARYEAR]	PRODUCTCATEGORY[PRODUCTCATEGORYNAME]	[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
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

T.DIST

12/10/2018 • 2 minutes to read

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.

Example

```
EVALUATE { T.DIST(60, 1, TRUE) }
```

Returns

[VALUE]
0.994695326367377

See also

T.DIST.2T function
T.DIST.RT function
T.INV function
T.INV.2t function

T.DIST.2T

12/10/2018 • 2 minutes to read

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.

Example

```
EVALUATE { T.DIST.2T(1.959999998, 60) }
```

Returns

[VALUE]
0.054644929975921

See also

T.DIST function
T.DIST.RT function
T.INV function
T.INV.2t function

T.DIST.RT

12/10/2018 • 2 minutes to read

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.

Example

```
EVALUATE { T.DIST.RT(1.959999998, 60) }
```

Returns

[VALUE]
0.0273224649879605

See also

T.DIST function
T.DIST.2T function
T.INV function
T.INV.2t function

T.INV

12/10/2018 • 2 minutes to read

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.

Example

```
EVALUATE { T.INV(0.75, 2) }
```

Returns

[VALUE]
0.816496580927726

See also

T.INV.2T function
T.DIST function
T.DIST.2T function
T.DIST.RT function

T.INV.2T

12/10/2018 • 2 minutes to read

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.

Example

```
EVALUATE { T.INV.2T(0.546449, 60) }
```

Returns

[VALUE]
0.606533075825759

See also

T.INV function
T.DIST function
T.DIST.2T function
T.DIST.RT function

TAN

12/10/2018 • 2 minutes to read

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 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

12/10/2018 • 2 minutes to read

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:

$$\text{TANH}(z) = \frac{\text{SINH}(z)}{\text{COSH}(z)}$$

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

TOPN

12/10/2018 • 2 minutes to read

Returns the top N rows of the specified table.

Syntax

```
TOPN(<n_value>, <table>, <orderBy_expression>, [<order>[, <orderBy_expression>, [<order>]]…])
```

Parameters

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).

See the remarks section to understand when the number of rows returned could possible be larger than *n_value*.

See the remarks section to understand when an empty table is returned.

table
Any DAX expression that returns a table of data from where to extract the top 'n' rows.

orderBy_expression
Any DAX expression where the result value is used to sort the table and it is evaluated for each row of *table*.

order
(Optional) A value that specifies how to sort *orderBy_expression* values, ascending or descending:

value	alternate value	Description
0 (zero)	FALSE	Sorts in descending order of values of *order_by*. This is the default value when *order* parameter is omitted.
1	TRUE	Ranks in ascending order of *order_by*.

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 necessarily 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 then TOPN returns an empty table.
- TOPN does not guarantee any sort order for the results.

Example

The following sample creates a measure with the sales of the top 10 sold products.

```
=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]))
```

VAR.S

12/10/2018 • 2 minutes to read

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.

Exceptions

Remarks

1. 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.
2. 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
3. Blank rows are filtered out from *columnName* and not considered in the calculations.
4. An error is returned if *columnName* contains less than 2 non-blank rows.

This DAX function may return different results when used in a model that is deployed and then queried in DirectQuery mode. For more information about semantic differences in DirectQuery mode, see https://go.microsoft.com/fwlink/?LinkId=219171.

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

12/10/2018 • 2 minutes to read

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

1. 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.
2. 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
3. Blank rows are filtered out from *columnName* and not considered in the calculations.
4. An error is returned if *columnName* contains less than 2 non-blank rows

This DAX function may return different results when used in a model that is deployed and then queried in DirectQuery mode. For more information about semantic differences in DirectQuery mode, see https://go.microsoft.com/fwlink/?LinkId=219171.

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

12/10/2018 • 2 minutes to read

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.

Exceptions

Remarks

1. 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.
2. 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
3. Blank rows are filtered out from *columnName* and not considered in the calculations.
4. An error is returned if *columnName* contains less than 2 non-blank rows.

This DAX function may return different results when used in a model that is deployed and then queried in DirectQuery mode. For more information about semantic differences in DirectQuery mode, see https://go.microsoft.com/fwlink/?LinkId=219171.

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

12/10/2018 • 2 minutes to read

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.

Exceptions

Remarks

1. 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.
2. 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
3. Blank rows are filtered out from *columnName* and not considered in the calculations.
4. An error is returned if *columnName* contains less than 2 non-blank rows

This DAX function may return different results when used in a model that is deployed and then queried in DirectQuery mode. For more information about semantic differences in DirectQuery mode, see https://go.microsoft.com/fwlink/?LinkId=219171.

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]))
```

XIRR

12/10/2018 • 2 minutes to read

Returns the internal rate of return for a schedule of cash flows that is not necessarily periodic.

Syntax

```
XIRR(<table>, <values>, <dates>, [guess])
```

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.

Return value

Internal rate of return for the given inputs. If the calculation fails to return a valid result, an error is returned.

Remarks

The value is calculated as the rate that satisfies the following function:

$$\sum_{j=1}^{N} \frac{P_j}{(1+\text{rate})^{\frac{d_j - d_1}{365}}}$$

Where P_j is the j-th payment, d_j is the j-th payment date, and d_1 is the first payment date

The series of cash flow values must contain at least one positive number and one negative number.

Example

The following calculates the internal rate of return of the CashFlows table:

```
Rate of return := 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

12/10/2018 • 2 minutes to read

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, and d_1 is the first payment date

The series of cash flow values must contain at least one positive number and one negative number.

Example

The following calculates the present value of the CashFlows table:

```
Present value := 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

Text functions

3/18/2019 • 2 minutes to read

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 section

UPPER

VALUE

BLANK

12/10/2018 • 2 minutes to read

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. For more information, see Data Types Supported (SSAS Tabular).

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. For details on the behavior of an individual function or operator, see DAX function reference.

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 PivotTable.

RESELLER TO INTERNET SALES RATIO	COLUMN LABELS			
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

Note that, 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 (DAX)
ISBLANK function (DAX)

CODE

12/10/2018 • 2 minutes to read

Returns a numeric code for the first character in a text string. The returned code corresponds to the character set used by your computer.

OPERATING ENVIRONMENT	CHARACTER SET
Macintosh	Macintosh character set
Windows	ANSI

Syntax

```
CODE(text)
```

Parameters

TERM	DEFINITION
text	The text for which you want the code of the first character.

Return value

A numeric code for the first character in a text string.

Example

FORMULA	DESCRIPTION	RESULT
=CODE("A")	Displays the numeric code for A	65
=CODE("!")	Displays the numeric code for !	33

COMBINEVALUES

12/10/2018 • 2 minutes to read

The COMBINEVALUES function 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

The 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(DimDate, "Month", COMBINEVALUES(",", [MonthName], [CalendarYear])))
```

Returns the following single column table:

[MONTH]
January,2007
February,2007
March,2007
April,2007
May,2007
June,2007
July,2007
August,2007
September,2007
October,2007
November,2007
December,2007
January,2008
January,2008
February,2008
March,2008
April,2008
May,2008
June,2008

[MONTH]
July,2008
August,2008
September,2008
October,2008
November,2008
December,2008

CONCATENATE

12/10/2018 • 2 minutes to read

Joins two text strings into one text string.

Syntax

```
CONCATENATE(<text1>, <text2>)
```

Parameters

TERM	DEFINITION
text1, text2	The text strings to be joined into a single text string. Strings can include text or numbers. You can also use column references.

Return value

The concatenated string.

Remarks

The CONCATENATE function joins two text strings into one text string. The joined items can be text, numbers or 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, better, 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 DAX function may return different results when used in a model that is deployed and then queried in DirectQuery mode. For more information about semantic differences in DirectQuery mode, see https://go.microsoft.com/fwlink/?LinkId=219171.

Example: Concatenation of Literals

Description

The sample formula creates a new string value by combining two string values that you provide as arguments.

Code

```
=CONCATENATE("Hello ", "World")
```

Example: Concatenation of Strings in Columns

Description

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.

Code

```
=CONCATENATE(Customer[LastName], CONCATENATE(", ", Customer[FirstName]))
```

Example: Conditional Concatenation of Strings in Columns

Description

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.

Code

```
=CONCATENATE( [FirstName]&" ", CONCATENATE( IF( LEN([MiddleName])>1, LEFT([MiddleName],1)&" ", ""),
[LastName]))
```

Comments

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

Code

```
=CONCATENATE('Products'[Product abbreviation],'Products'[Product number])
```

Comments

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

Text functions (DAX)

CONCATENATEX

12/10/2018 • 2 minutes to read

Concatenates the result of an expression evaluated for each row in a table.

Syntax

```
CONCATENATEX(<table>, <expression>, [delimiter])
```

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.
delimiter	(optional) A separator to use during concatenation.

Return value

A text 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.

Example

Employees table

FIRSTNAME	LASTNAME
Alan	Brewer
Michael	Blythe

CONCATENATEX(Employees, [FirstName] & " " & [LastName], ",")

Returns "Alan Brewer, Michael Blythe"

EXACT

12/10/2018 • 2 minutes to read

Compares two text strings and returns TRUE if they are exactly the same, FALSE otherwise. EXACT is case-sensitive but ignores formatting differences. You can use EXACT to test text being entered into a document.

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.

Property Value/Return value

True or false. (Boolean)

Example

The following formula checks the value of Column1 for the current row against the value of Column2 for the current row, and returns TRUE if they are the same, and returns FALSE if they are different.

```
=EXACT([Column1],[Column2])
```

See also

Text functions (DAX)

FIND

12/10/2018 • 2 minutes to read

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) The value that should be returned when the operation does not find a matching substring, typically 0, -1, or BLANK().

Property Value/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 DAX function may return different results when used in a model that is deployed and then queried in DirectQuery mode. For more information about semantic differences in DirectQuery mode, see https://go.microsoft.com/fwlink/?LinkId=219171.

FIND does not support wildcards. To use wildcards, use SEARCH.

Example

The following formula finds the position of the first letter of the product designation, BMX, in the string that contains the product description.

```
=FIND("BMX","line of BMX racing goods")
```

See also

Text functions (DAX)

FIXED

12/10/2018 • 2 minutes to read

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.

Property Value/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 DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see https://go.microsoft.com/fwlink/?LinkId=219172.

Example

The following example gets the numeric value for the current row in column, PctCost, and returns it as text with 4 decimal places and no commas.

```
=FIXED([PctCost],3,1)
```

Numbers can never have more than 15 significant digits, but decimals can be as large as 127.

See also

Text functions (DAX)
Math and Trig functions (DAX)

FORMAT

12/10/2018 • 2 minutes to read

Converts a value to text according to the specified format.

Syntax

```
FORMAT(<value>, <format_string>)
```

Parameters

TERM	DEFINITION
value	A value or expression that evaluates to a single value.
format_string	A string with the formatting template.

Return value

A string containing **value** formatted as defined by **format_string**.

> **IMPORTANT**
> 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** type).

Remarks

For information on how to use the **format_string** parameter, see the appropriate topic listed below:

TO FORMAT	FOLLOW THESE INSTRUCTIONS
Numbers	Use predefined numeric formats or create user-defined numeric formats.
Dates and times	Use predefined date/time formats or create user-defined date/time formats.

All predefined formatting strings use the current user locale when formatting the result.

Caution

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 does not match any defined format strings. For example, "p" as an abbreviation for "Percent" is not supported. Strings that you provide as an argument to the FORMAT function that are not included in the list of predefined format strings are handled as part of a custom format string, or as a string literal.

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see https://go.microsoft.com/fwlink/?LinkId=219172.

See also

Pre-Defined Numeric Formats for the FORMAT function
Custom Numeric Formats for the FORMAT function
Pre-defined date and time formats for the FORMAT function
Custom date and time formats for the FORMAT function
VALUE function (DAX)

Pre-Defined Numeric Formats for the FORMAT function

12/10/2018 • 2 minutes to read

The following table identifies the predefined numeric format names. These may be used by name as the style argument for the Format function.

FORMAT SPECIFICATION	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.
`"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.

Remarks

Note that format strings are based on Visual Basic (OLE Automation) and therefore might have slightly different behavior than the format strings used by the .NET Framework. Abbreviations such as “P” and “x” are not supported. Any other strings that you provide as an argument to the FORMAT function are interpreted as defining a custom format.

> **IMPORTANT**
>
> If *value* is BLANK() the function returns an empty string.
>
> If *format_string* is BLANK(), the value is formatted with a "General Number" format.

Example

The following samples show the usage of different predefined formatting strings to format a numeric value.

```
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")
```

The above expressions return the following results:

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.

See also

FORMAT function (DAX)
Pre-defined date and time formats for the FORMAT function
Custom Numeric Formats for the FORMAT function

Custom numeric formats for the FORMAT function

12/10/2018 • 6 minutes to read

A user-defined format expression for numbers can have from one to three sections separated by semicolons. If the Style argument of the Format function contains one of the predefined numeric formats, only one section is allowed.

IF YOU USE	THIS IS THE RESULT
One section only	The format expression applies to all values.
Two sections	The first section applies to positive values and zeros; the second applies to negative values.
Three sections	The first section applies to positive values, the second applies to negative values, and the third applies to zeros.

Format specifications

The following table identifies characters you can use to create user-defined number formats.

FORMAT SPECIFICATION	DESCRIPTION
None	Displays the number with no formatting.
0 (zero character)	Digit placeholder. Displays a digit or a zero. If the expression has a digit in the position where the zero appears in the format string, displays the digit; otherwise, displays 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, displays 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, rounds 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, displays the extra digits without modification.
#	Digit placeholder. Displays a digit or nothing. If the expression has a digit in the position where the # character appears in the format string, displays the digit; otherwise, displays 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 fewer digits than there are # characters on either side of the decimal separator in the format expression.

FORMAT SPECIFICATION	DESCRIPTION
. (dot character)	Decimal placeholder. The decimal placeholder determines how many digits are displayed to the left and right of the decimal separator. If the format expression contains only # characters 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 zero as the first digit placeholder to the left of the decimal separator. In some locales, a comma is used as the decimal separator. The actual character used as a decimal placeholder in the formatted output depends on the number format recognized by your system. Thus, you should use the period as the decimal placeholder in your formats even if you are in a locale that uses a comma as a decimal placeholder. The formatted string will appear in the format correct for the locale.
%	Percent placeholder. Multiplies the expression by 100. The percent character (%) is inserted in the position where it appears in the format string.
, (comma character)	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 #). A thousand separator immediately to the left of the decimal separator (whether or not a decimal is specified) or as the rightmost character in the string means "scale the number by dividing it by 1,000, rounding as needed." Numbers smaller than 1,000 but greater or equal to 500 are displayed as 1, and numbers smaller than 500 are displayed as 0. Two adjacent thousand separators in this position scale by a factor of 1 million, and an additional factor of 1,000 for each additional separator. Multiple separators in any position other than immediately to the left of the decimal separator or the rightmost position in the string are treated simply as specifying the use of a thousand separator. In some locales, a period is used as a thousand separator. The actual character used as the thousand separator in the formatted output depends on the Number Format recognized by your system. Thus, you should use the comma as the thousand separator in your formats even if you are in a locale that uses a period as a thousand separator. The formatted string will appear in the format correct for the locale. For example, consider the three following format strings: "#,0.", which uses the thousands separator to format the number 100 million as the string "100,000,000". "#0,.", which uses scaling by a factor of one thousand to format the number 100 million as the string "100000". "#,0,.", which uses the thousands separator and scaling by one thousand to format the number 100 million as the string "100,000".

FORMAT SPECIFICATION	DESCRIPTION
: (colon character)	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.
/ (forward slash character)	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 left 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 left 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. You must also include digit placeholders to the right of this symbol to get correct formatting.
-+$()	Literal characters. These characters are displayed exactly as typed in the format string. To display a character other than one of those listed, precede it with a backslash (\) or enclose it in double quotation marks (" ").
**** (backward slash character)	Displays 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"	Displays the string inside the double quotation marks (" "). To include a string in the style argument from within code, you must use Chr(34) to enclose the text (34 is the character code for a quotation mark (")).

The following table contains some sample format expressions for numbers. (These examples all assume that your system's locale setting is English-U.S.) The first column contains the format strings for the Format function; the other columns contain the resulting output if the formatted data has the value given in the column headings.

FORMAT (STYLE)	"5" FORMATTED AS	"-5" FORMATTED AS	"0.5" FORMATTED AS	"0" FORMATTED AS
Zero-length string ("")	5	-5	0.5	0
0	5	-5	1	0

FORMAT (STYLE)	"5" FORMATTED AS	"-5" FORMATTED AS	"0.5" FORMATTED AS	"0" FORMATTED AS
0.00	5.00	-5.00	0.50	0.00
#,##0	5	-5	1	0
$#,##0;($#,##0)	$5	($5)	$1	$0
$#,##0.00;($#,##0.00)	$5.00	($5.00)	$0.50	$0.00
0%	500%	-500%	50%	0%
0.00%	500.00%	-500.00%	50.00%	0.00%
0.00E+00	5.00E+00	-5.00E+00	5.00E-01	0.00E+00
0.00E-00	5.00E00	-5.00E00	5.00E-01	0.00E00
"$#,##0;;\Z\e\r\o"	$5	$-5	$1	Zero

Remarks

If you include semicolons with nothing between them, the missing section is printed using the format of the positive value.

See also

FORMAT function (DAX)
Pre-Defined Numeric Formats for the FORMAT function
Custom date and time formats for the FORMAT function

Pre-defined date and time formats for the FORMAT function

12/10/2018 • 2 minutes to read

The following table identifies the predefined date and time format names. If you use strings other than these predefined strings, they will be interpreted as a custom date and time format.

FORMAT SPECIFICATION	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.
`"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.

Remarks

The formatting strings are based on Visual Basic (OLE Automation) and not the .NET Framework formatting strings; therefore, your results might be slightly different than what you expect from .NET format strings. Note that abbreviations such as "D" for Long Date and "t" for Short Time are not supported.

> **IMPORTANT**
>
> If *value* is BLANK() the function returns an empty string.
>
> If *format_string* is BLANK(), the value is formatted with a "General Date" format.

See also

Custom date and time formats for the FORMAT function

Custom date and time formats for the FORMAT function

12/10/2018 • 4 minutes to read

The following table shows characters you can use to create user-defined date/time formats.

FORMAT SPECIFICATION	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 that is used as the time separator in formatted output is determined by your application's current culture value.
(/)	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 that is used as the date separator in formatted output is determined by your application's current culture.
(%)	Used to indicate that the following character should be read as a single-letter format without regard to any trailing letters. Also used to indicate that a single-letter format is read as a user-defined format. See what follows for additional details.
d	Displays the day as a number without a leading zero (for example, 1). Use %d if this is the only character in your user-defined numeric format.
dd	Displays the day as a number with a leading zero (for example, 01).
ddd	Displays the day as an abbreviation (for example, Sun).
dddd	Displays the day as a full name (for example, Sunday).
M	Displays the month as a number without a leading zero (for example, January is represented as 1). Use %M if this is the only character in your user-defined numeric format.
MM	Displays the month as a number with a leading zero (for example, 01/12/01).
MMM	Displays the month as an abbreviation (for example, Jan).
MMMM	Displays the month as a full month name (for example, January).
gg	Displays the period/era string (for example, A.D.).

FORMAT SPECIFICATION	DESCRIPTION
h	Displays the hour as a number without leading zeros using the 12-hour clock (for example, 1:15:15 PM). Use %h if this is the only character in your user-defined numeric format.
hh	Displays the hour as a number with leading zeros using the 12-hour clock (for example, 01:15:15 PM).
H	Displays the hour as a number without leading zeros using the 24-hour clock (for example, 1:15:15). Use %H if this is the only character in your user-defined numeric format.
HH	Displays the hour as a number with leading zeros using the 24-hour clock (for example, 01:15:15).
m	Displays the minute as a number without leading zeros (for example, 12:1:15). Use %m if this is the only character in your user-defined numeric format.
mm	Displays the minute as a number with leading zeros (for example, 12:01:15).
s	Displays the second as a number without leading zeros (for example, 12:15:5). Use %s if this is the only character in your user-defined numeric format.
ss	Displays the second as a number with leading zeros (for example, 12:15:05).
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.
y	Displays the year number (0-9) without leading zeros. Use %y if this is the only character in your user-defined numeric format.

FORMAT SPECIFICATION	DESCRIPTION
yy	Displays the year in two-digit numeric format with a leading zero, if applicable.
yyy	Displays the year in four-digit numeric format.
yyyy	Displays the year in four-digit numeric format.
z	Displays the timezone offset without a leading zero (for example, -8). Use %z if this is the only character in your user-defined numeric format.
zz	Displays the timezone offset with a leading zero (for example, -08)
zzz	Displays the full timezone offset (for example, -08:00)

Remarks

Formatting strings are case sensitive. Different formatting can be obtained by using a different case. For example, when formatting a date value with the string "D" you get the date in the long format (according to your current locale). However, if you change the case to "d" you get the date in the short format. Also, unexpected results or an error might occur if the intended formatting does not match the case of any defined format string.

Date/Time formatting uses the current user locale to determine the ultimate format of the string. For example, to format the date March 18, 1995, with the following format string "M/d/yyyy", if the user locale is set to the United States of America (en-us) the result is '3/12/1995', but if the user locale is set to Germany (de-de) the result is '18.03.1995'.

See also

FORMAT function (DAX)
Custom numeric formats for the FORMAT function
Pre-defined date and time formats for the FORMAT function

LEFT

12/10/2018 • 2 minutes to read

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.

Property Value/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 DAX function may return different results when used in a model that is deployed and then queried in DirectQuery mode. For more information about semantic differences in DirectQuery mode, see https://go.microsoft.com/fwlink/?LinkId=219171.

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 (DAX)

LEN

12/10/2018 • 2 minutes to read

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.

Therefore, 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([AddressLin2])
```

LOWER

12/10/2018 • 2 minutes to read

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.

Property Value/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 (DAX)

MID

12/10/2018 • 2 minutes to read

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.

Property Value/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.

Example

The following examples return the same results, the first 5 letters of the column, [ResellerName]. The first example uses the fully qualified name of the column and specifies the starting point; the second example omits the table name and the parameter, **num_chars**.

```
=MID('Reseller'[ResellerName],5,1))
=MID([ResellerName,5])
```

The results are the same if you use the following formula:

```
=LEFT([ResellerName],5)
```

See also

Text functions (DAX)

REPLACE

12/10/2018 • 2 minutes to read

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, *num_chars*, is a blank or references a column that evaluates to a blank, the string for *new_text* is inserted at the position, *start_num*, without replacing any characters. This is the same behavior as in Excel.
new_text	The replacement text for the specified characters in **old_text**.

Property Value/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 DAX function may return different results when used in a model that is deployed and then queried in DirectQuery mode. For more information about semantic differences in DirectQuery mode, see https://go.microsoft.com/fwlink/?LinkId=219171.

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 (DAX)

SUBSTITUTE function (DAX)

REPT

12/10/2018 • 2 minutes to read

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.

Property Value/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

Description

The following example returns the string, 85, repeated three times.

Code

```
=REPT("85",3)
```

Example: Repeating Column Values

Description

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.

Code

```
=REPT([MyText],[MyNumber])
```

Comments

MYTEXT	MYNUMBER	CALCULATEDCOLUMN1
Text	2	TextText
Number	0	
85	3	858585

See also

Text functions (DAX)

RIGHT

12/10/2018 • 2 minutes to read

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.

Property Value/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 DAX function may return different results when used in a model that is deployed and then queried in DirectQuery mode. For more information about semantic differences in DirectQuery mode, see https://go.microsoft.com/fwlink/?LinkId=219171

Example: Returning a Fixed Number of Characters

Description

The following formula returns the last two digits of the product code in the New Products table.

Code

```
=RIGHT('New Products'[ProductCode],2)
```

Example: Using a Column Reference to Specify Character Count

Description

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.

Code

```
=RIGHT('New Products'[ProductCode],[MyCount])
```

See also

Text functions (DAX)
LEFT function (DAX)
MID function (DAX)

SEARCH

12/10/2018 • 2 minutes to read

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) The value that should be returned when the operation does not find a matching substring, typically 0, -1, or BLANK().

Return value

The number of the starting position of the first text string from the first character of the second text string.

Remarks

1. The search function is case insensitive. Searching for "N" will find the first occurrence of 'N' or 'n'.
2. 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'.
3. By using this function, you can locate one text string within a second text string, and return the position where the first string starts.
4. 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.
5. 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 DAX function may return different results when used in a model that is deployed and then queried in DirectQuery mode. For more information about semantic differences in DirectQuery mode, see https://go.microsoft.com/fwlink/?LinkId=219171.

Example: Search within a String

Description

The following formula finds the position of the letter "n" in the word "printer".

Code

```
=SEARCH("n","printer")
```

Comments

The formula returns 4 because "n" is the fourth character in the word "printer."

Example: Search within a Column

Description

You can use a column reference as an argument to SEARCH. The following formula finds the position of the character "-" (hyphen) in the column, [PostalCode].

Code

```
=SEARCH("-",[PostalCode])
```

Comments

The return result is a column of numbers, indicating the index position of the hyphen.

Example: Error-Handling with SEARCH

Description

The formula in the preceding example will fail if the search string is not found in every row of the source column. Therefore, the next example demonstrates how to use IFERROR with the SEARCH function, to ensure that a valid result is returned for every row.

The following formula finds the position of the character "-" within the column, and returns -1 if the string is not found.

Code

```
= IFERROR(SEARCH("-",[PostalCode]),-1)
```

Comments

Note that the data type of the value that you use as an error output must match the data type of the non-error output type. In this case, you provide a numeric value to be output in case of an error because SEARCH returns an integer value.

However, you could also return a blank (empty string) by using `BLANK()` as the second argument to IFERROR.

See also

MID function (DAX)

REPLACE function (DAX)
Text functions (DAX)

SUBSTITUTE

12/10/2018 • 2 minutes to read

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

Property Value/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 DAX function may return different results when used in a model that is deployed and then queried in DirectQuery mode. For more information about semantic differences in DirectQuery mode, see https://go.microsoft.com/fwlink/?LinkId=219171.

Example: Substitution within a String

Description

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.

Code

```
=SUBSTITUTE([Product Code], "NW", "PA")
```

See also

Text functions (DAX)
REPLACE function (DAX)

TRIM

12/10/2018 • 2 minutes to read

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.

Property Value/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 (DAX)

UNICHAR

12/10/2018 • 2 minutes to read

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)
```

UPPER

12/10/2018 • 2 minutes to read

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.

Property Value/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 (DAX)
LOWER function (DAX)

VALUE

12/10/2018 • 2 minutes to read

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 (DAX)

DAX syntax

11/28/2018 • 8 minutes to read

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.

> **NOTE**
>
> Formulas can behave differently depending on whether they are used in a calculated column, or in a measure within a PivotTable. 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.
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)

OPERATOR TYPE	SYMBOL AND USE
Comparison operators	= (equal to) > (greater than) < (less than) >= (greater than or equal to) <= (less than or equal to) <> (not equal to)
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.

DAX operators

12/10/2018 • 6 minutes to read

The Data Analysis Expression (DAX) language uses operators to create expressions that compare values, perform arithmetic calculations, or work with strings. This section describes the use of each operator.

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"
>	Greater than	[Sales Date] > "Jan 2009"
<	Less than	[Sales Date] < "Jan 1 2009"
>=	Greater than or equal to	[Amount] >= 20000
<=	Less than or equal to	[Amount] <= 100

COMPARISON OPERATOR	MEANING	EXAMPLE
<>	Not equal to	[Region] <> "USA"

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
^	Exponentiation
–	Sign (as in –1)
* and /	Multiplication and division
!	NOT (unary operator)
+ and –	Addition and subtraction
&	Connects two strings of text (concatenation)
= < > <= >= <>	Comparison

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. First, both operands are converted to the largest possible common data type.

2. Next, 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. See Data Types Supported (SSAS Tabular) for a complete list of data types supported by DAX in SSAS.

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)
Boolean	Boolean
String	String
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, we recommend you use parentheses to control the order of operations and remove any ambiguity about the result.

See also

DAX queries

3/7/2019 • 4 minutes to read

With DAX queries, you can query and return data defined by a table expression. Reporting clients construct DAX queries whenever a field is placed on a report surface, or a whenever a filter or calculation is applied. DAX queries can also be created and run in SQL Server Management Studio (SSMS) and open-source tools like DAX Studio. DAX queries run in SSMS and DAX Studio return results as a table.

Syntax

```
[DEFINE {  MEASURE <tableName>[<name>] = <expression> }
        {  VAR <name> = <expression>}]
EVALUATE <table>
[ORDER BY {<expression> [{ASC | DESC}]}[, …]
[START AT {<value>|<parameter>} [, …]]]
```

Keywords

EVALUATE (Required)

At the most basic level, a DAX query is an **EVALUATE** statement containing a table expression. However, a query can contain multiple EVALUATE statements.

Syntax

```
EVALUATE <table>
```

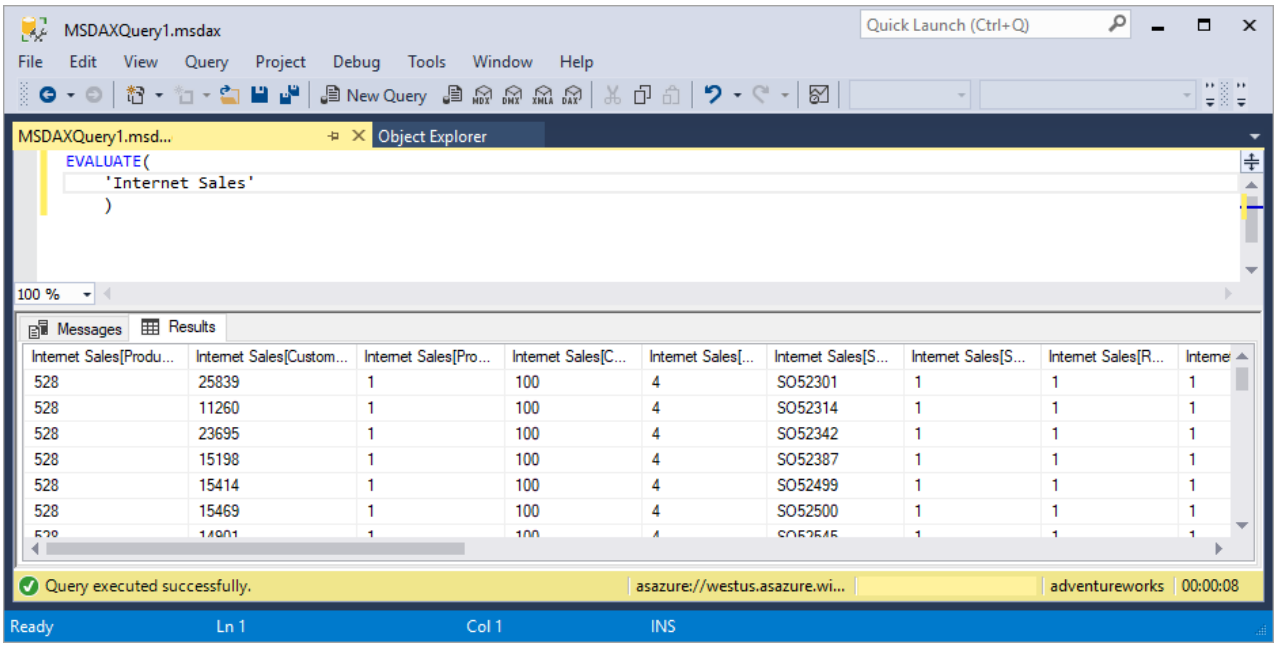
Arguments

TERM	DEFINITION
table	A table expression.

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.

Syntax

```
EVALUATE <table>
[ORDER BY {<expression> [{ASC | DESC}]}[, …]
```

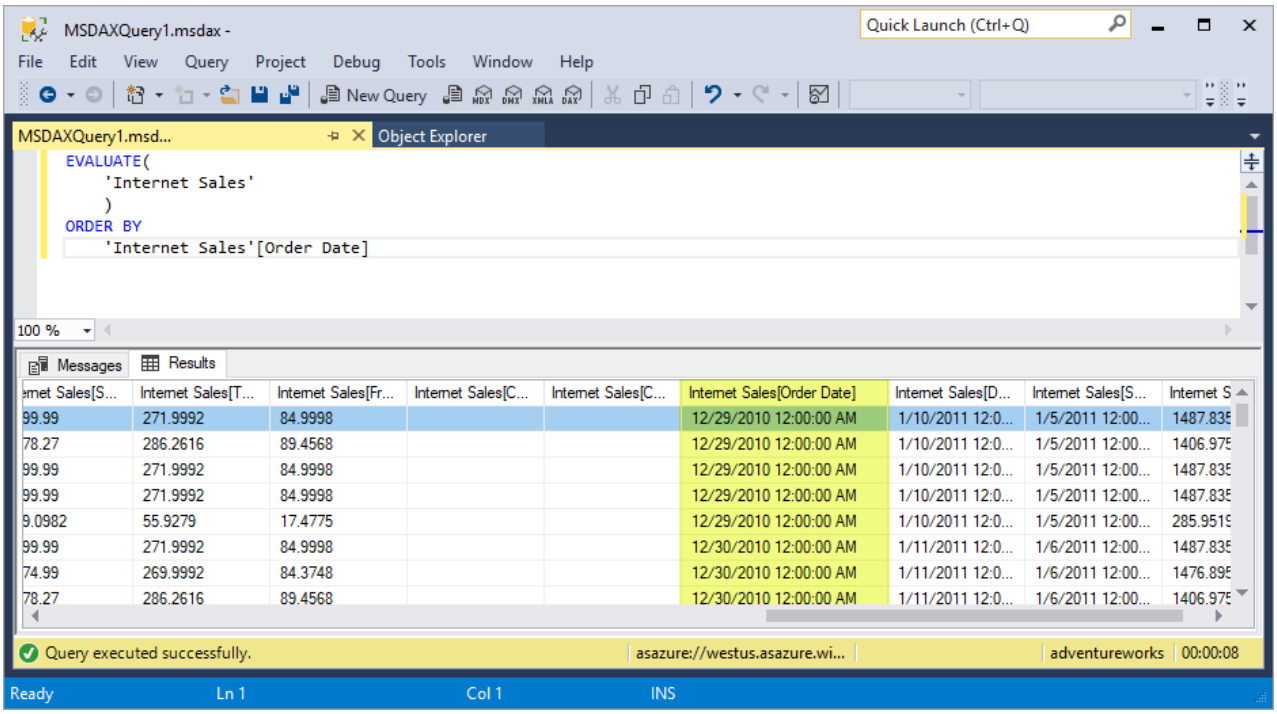
Arguments

TERM	DEFINITION
expression	Any DAX expression that returns a single scalar value.
ASC	(default) Ascending sort order.
DESC	Descending sort order.

Example

```
EVALUATE(
    'Internet Sales'
    )
ORDER BY
 'Internet Sales'[Order Date]
```

Returns all rows and columns from the Internet Sales table, ordered by Order Date, as a table.



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.

Syntax

```
EVALUATE <table>
[ORDER BY {<expression> [{ASC | DESC}]}[, …]
[START AT {<value>|<parameter>} [, …]]]
```

Arguments

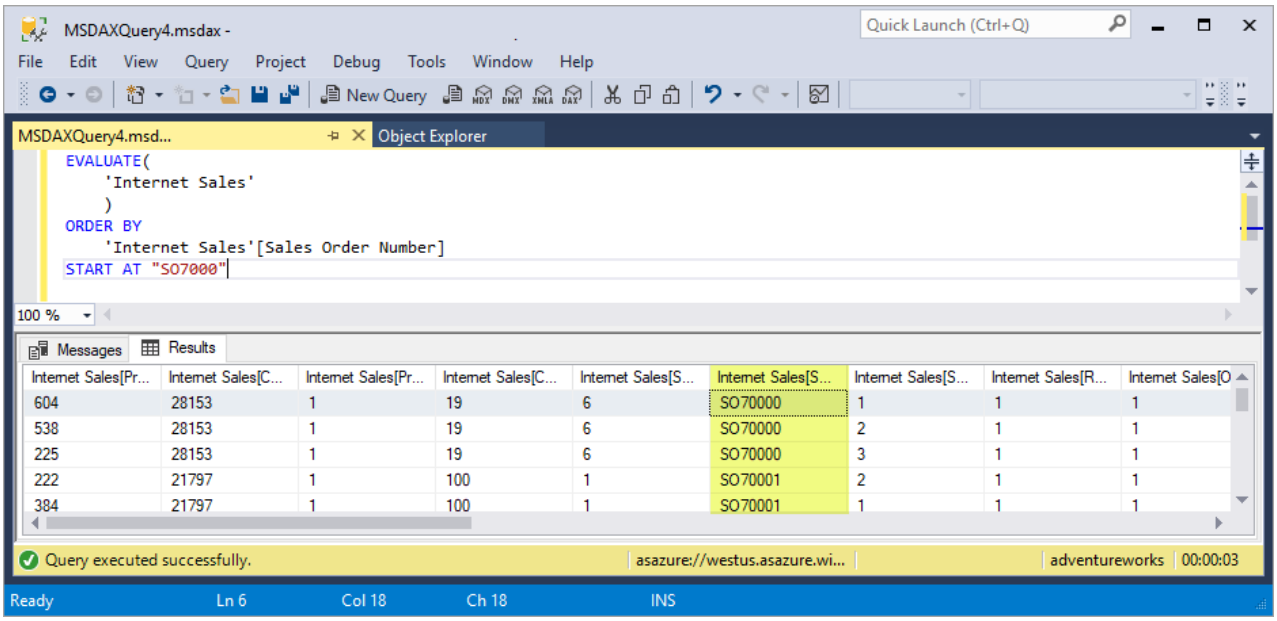
TERM	DEFINITION
value	A constant value. Cannot be an expression.
parameter	The name of a parameter in an XMLA statement prefixed with an `@` character.

START AT 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.

Example

```
EVALUATE(
    'Internet Sales'
    )
ORDER BY
 'Internet Sales'[Sales Order Number]
START AT "SO7000"
```

Returns all rows and columns from the Internet Sales table, ordered by Sales Order Number, beginning at SO7000.



Multiple **EVALUATE/ORDER BY/START AT** clauses can be specified in a single query.

DEFINE (Optional)

The optional **DEFINE** keyword defines entities that exist only for the duration of the query. Definitions are valid for all EVALUATE statements. Entities can be variables, measures, tables, and columns. Definitions can reference other definitions that appear before or after the current definition. Definitions typically precede the EVALUATE statement.

Syntax

```
[DEFINE {  MEASURE <tableName>[<name>] = <expression> }
        {  VAR <name> = <expression>}]
EVALUATE <table>
```

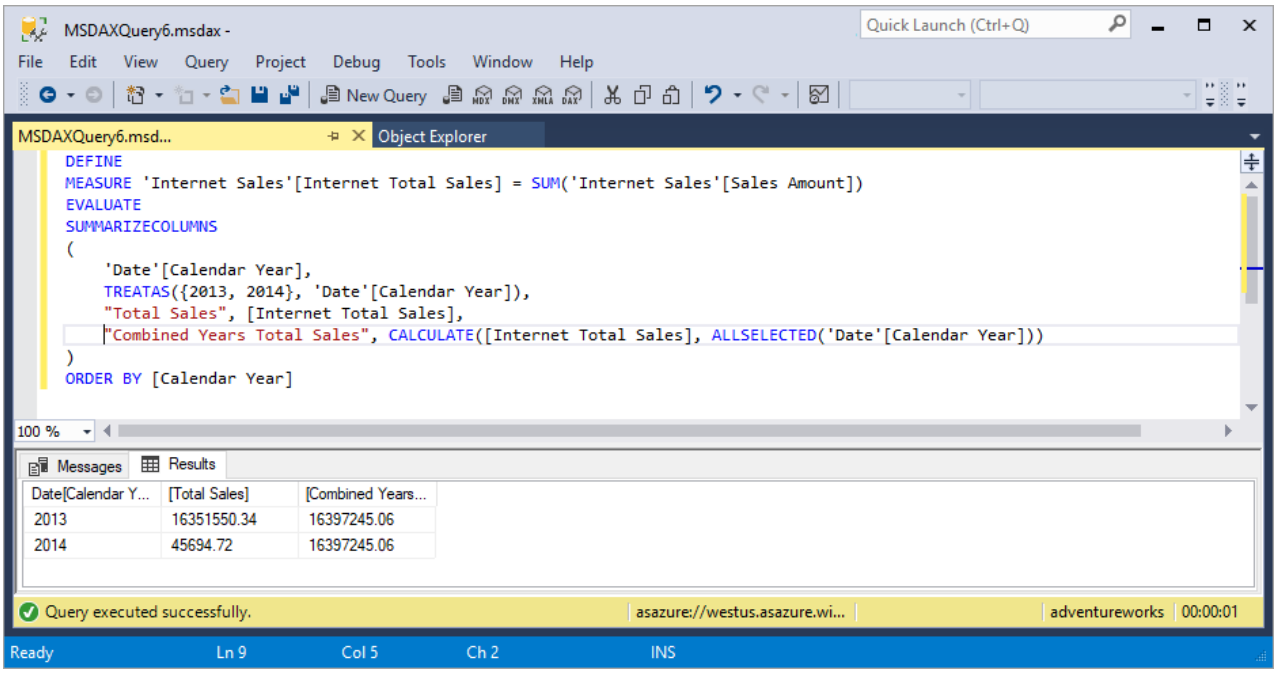
Arguments

TERM	DEFINITION
tableName	The name of an existing table using standard DAX syntax. It cannot be an expression.
name	The name of a new measure. It cannot be an expression.
expression	Any DAX expression that returns a single scalar value. The expression can use any of the defined measures. The expression must return a table. If a scalar value is required, wrap the scalar inside a ROW() function to produce a table.
VAR	An optional expression as a named variable. A VAR can be passed as an argument to other expressions.

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.



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. Hence, any place in the syntax where a value is allowed it 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.

> **IMPORTANT**
>
> Parameters used and not defined in the **<Parameters>** element generate an error response in XMLA.

See also

FILTER
SUMMARIZECOLUMNS
TREATAS
VAR

DAX parameter-naming conventions

12/10/2018 • 2 minutes to read

Parameter names are standardized in DAX reference to facilitate the usage and understanding of the functions.

Parameter names

PARAMTER	DESCRIPTION
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

PARAMTER	DESCRIPTION
prefixing	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: Result_ColumnName - Refers to an existing column used to get the result values in the LOOKUPVALUE() function. Search_ColumnName - Refers to an existing column used to search for a value in the LOOKUPVALUE() function.

PARAMTER	DESCRIPTION
omitting	Parameter names will be omitted if the prefix is clear enough to describe the parameter. 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. 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.