# IMPORTANT DAX FUNCTIONS

# CALENDAR

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

DAX

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

### Parameters

| PARAMETERS | |
|---|---|
| **Term** | **Definition** |
| start_date | Any DAX expression that returns a datetime value. |
| end_date | Any DAX expression that returns a datetime value. |

## Return value

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

## Remarks

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

Source: https://docs.microsoft.com/en-us/dax/calendar-function-dax

# CALENDARAUTO

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

DAX

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

### Parameters

**PARAMETERS**

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

## Return value

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

## Remarks

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

Source: https://docs.microsoft.com/en-us/dax/calendarauto-function-dax

# DATEDIFF

Returns the count of interval boundaries crossed between two dates.

## Syntax

DAX

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

### Parameters

**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:<br><br>- SECOND<br>- MINUTE<br>- HOUR<br>- DAY<br>- WEEK<br>- MONTH<br>- QUARTER<br>- 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.

Source: https://docs.microsoft.com/en-us/dax/datediff-function-dax

# ALL

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

DAX

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

### Parameters

**PARAMETERS**

| Term | Definition |
|---|---|
| table | The table that you want to clear filters on. |
| column | The column that you want to clear filters on. |

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

## Return value

The table or column with filters removed.

## Remarks

- This function is not used by itself, but serves as an intermediate function that can be used to change the set of results over which some other calculation is performed.
- The normal behavior for DAX expressions containing the ALL () function is that any filters applied will be ignored. However, there are some scenarios where this is not the case because of *auto-exist*, a DAX technology that optimizes filtering in order to reduce the amount of processing required for certain DAX queries. An example where auto-exist and ALL () provide unexpected results is when filtering on two or more columns of the same table (like when using slicers), and there is a measure on that same table that uses ALL (). In this case, auto-exist will *merge* the multiple

filters into one and will only filter on existing combinations of values. Because of this merge, the measure will be calculated on the existing combinations of values and the result will be based on filtered values instead of all values as expected. To learn more about auto-exist and its effect on calculations, see Microsoft MVP Alberto Ferrari's Understanding DAX Auto-Exist article on sql.bi.com.

- The following table describes how you can use the ALL and ALLEXCEPT functions in different scenarios.

**TABLE 2**

| 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. |
| ALL (Column[, Column[, …]]) | Removes all filters from the specified columns in the table; all other filters on other columns in the table still apply. All column arguments must come from the same table. The ALL(Column) variant is useful when you want to remove the context filters for one or more specific columns and to keep all other context filters. The second and third examples demonstrate this scenario. |
| ALLEXCEPT(Table, Column1 [,Column2]...) | Removes all context filters in the table except filters that are applied to the specified columns. This is a convenient shortcut for situations in which you want to remove the filters on many, but not all, columns in a table. |

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

Source: https://docs.microsoft.com/en-us/dax/all-function-dax

# AND

Checks whether both arguments are TRUE, and returns TRUE if both arguments are TRUE. Otherwise returns false.

## Syntax

DAX

```
AND(<logical1>,<logical2>)
```

### Parameters

| 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.

Source: https://docs.microsoft.com/en-us/dax/and-function-dax

# COALESCE

Returns the first expression that does not evaluate to BLANK. If all expressions evaluate to BLANK, BLANK is returned.

## Syntax

DAX

```
COALESCE(<expression>, <expression>[, <expression>]…)
```

### Parameters

**PARAMETERS**

| Term | Definition |
|---|---|
| expression | Any DAX expression that returns a scalar expression. |

## Return value

A scalar value coming from one of the expressions or BLANK if all expressions evaluate to BLANK.

## Remarks

Input expressions may be of different data types.

Source: https://docs.microsoft.com/en-us/dax/coalesce-function-dax

# IF

Checks a condition, and returns one value when it's TRUE, otherwise it returns a second value.

## Syntax

DAX

```
IF(<logical_test>, <value_if_true>[, <value_if_false>])
```

### Parameters

**PARAMETERS**

| Term | Definition |
|---|---|
| logical_test | Any value or expression that can be evaluated to TRUE or FALSE. |
| value_if_true | The value that's returned if the logical test is TRUE. |
| value_if_false | (Optional) The value that's returned if the logical test is FALSE. If omitted, BLANK is returned. |

## Return value

Either **value_if_true**, **value_if_false**, or BLANK.

## Remarks

- The IF function can return a variant data type if **value_if_true** and **value_if_false** are of different data types, but the function attempts to return a single data type if both **value_if_true** and **value_if_false** are of numeric data types. In the latter case, the IF function will implicitly convert data types to accommodate both values.
- For example, the formula `IF(<condition>, TRUE(), 0)` returns TRUE or 0, but the formula `IF(<condition>, 1.0, 0)` returns only decimal values even though **value_if_false** is of the whole number data type. For more information about implicit data type conversion, see Data types.

Source: https://docs.microsoft.com/en-us/dax/if-function-dax

# SWITCH

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

## Syntax

DAX

```
SWITCH(<expression>, <value>, <result>[, <value>, <result>]…[, <else>])
```

### Parameters

**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.

Source: https://docs.microsoft.com/en-us/dax/switch-function-dax

# DIVIDE

Performs division and returns alternate result or BLANK() on division by 0.

## Syntax

DAX

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

### Parameters

**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.

Source: https://docs.microsoft.com/en-us/dax/divide-function-dax

# RELATED

Returns a related value from another table.

## Syntax

DAX

```
RELATED(<column>)
```

### Parameters

| | |
|---|---|
| | |

## Return value

A single value that is related to the current row.

## Remarks

- The RELATED function requires that a relationship exists between the current table and the table with related information. You specify the column that contains the data that you want, and the function follows an existing many-to-one relationship to fetch the value from the specified column in the related table. If a relationship does not exist, you must create a relationship.
- When the RELATED function performs a lookup, it examines all values in the specified table regardless of any filters that may have been applied.
- The RELATED function needs a row context; therefore, it can only be used in calculated column expression, where the current row context is unambiguous, or as a nested function in an expression that uses a table scanning function. A table scanning function, such as SUMX, gets the value of the current row value and then scans another table for instances of that value.
- The RELATED function cannot be used to fetch a column across a limited relationship.

Source: https://docs.microsoft.com/en-us/dax/related-function-dax

# AVERAGEX

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

## Syntax

DAX

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

### Parameters

**PARAMETERS**

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

## Return value

A decimal number.

## Remarks

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

Source: https://docs.microsoft.com/en-us/dax/averagex-function-dax

# COUNTAX

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 non-blank result.

## Syntax

DAX

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

### Parameters

**PARAMETERS**

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

## Return value

A whole number.

## Remarks

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

Source: https://docs.microsoft.com/en-us/dax/countax-function-dax

# COUNTX

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

## Syntax

DAX

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

### Parameters

| PARAMETERS | |
|---|---|
| **Term** | **Definition** |
| table | The table containing the rows to be counted. |
| expression | An expression that returns the set of values that contains the values you want to count. |

## Return value

An integer.

## Remarks

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

Source: https://docs.microsoft.com/en-us/dax/countx-function-dax

# MAXX

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

## Syntax

DAX

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

### Parameters

**PARAMETERS**

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

## Return value

The largest value.

## Remarks

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

Source: https://docs.microsoft.com/en-us/dax/maxx-function-dax

# MINX

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

## Syntax

DAX

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

### Parameters

**PARAMETERS**

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

## Return value

A smallest value.

## Remarks

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

Source: https://docs.microsoft.com/en-us/dax/minx-function-dax

# SUMMARIZE

Returns a summary table for the requested totals over a set of groups.

## Syntax

DAX

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

### Parameters

**PARAMETERS**

| Term | Definition |
|---|---|
| table | Any DAX expression that returns a table of data. |
| groupBy_ColumnName | (Optional) The qualified name of an existing column used to create summary groups based on the values found in it. This parameter cannot be an expression. |
| name | The name given to a total or summarize column, enclosed in double quotes. |
| expression | Any DAX expression that returns a single scalar value, where the expression is to be evaluated multiple times (for each row/context). |

## Return value

A table with the selected columns for the *groupBy_columnName* arguments and the summarized columns designed by the name arguments.

## Remarks

- Each column for which you define a name must have a corresponding expression; otherwise, an error is returned. The first argument, name, defines the name of the column in the results. The second argument, expression, defines the calculation performed to obtain the value for each row in that column.
- groupBy_columnName must be either in *table* or in a related table to *table*.
- Each name must be enclosed in double quotation marks.
- The function groups a selected set of rows into a set of summary rows by the values of one or more groupBy_columnName columns. One row is returned for each group.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Source: https://docs.microsoft.com/en-us/dax/summarize-function-dax

# CONCATENATEX

Concatenates the result of an expression evaluated for each row in a table.

## Syntax

DAX

```
CONCATENATEX(<table>, <expression>, [delimiter])
```

### Parameters

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

Source: https://docs.microsoft.com/en-us/dax/concatenatex-function-dax

# VALUE

Converts a text string that represents a number to a number.

## Syntax

DAX

```
VALUE(<text>)
```

### Parameters

**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.

Source: https://docs.microsoft.com/en-us/dax/value-function-dax

# DATESINPERIOD

Returns a table that contains a column of dates that begins with a specified start date and continues for the specified number and type of date intervals.

This function is suited to pass as a filter to the CALCULATE function. Use it to filter an expression by standard date intervals such as days, months, quarters, or years.

## Syntax

DAX

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

### Parameters

PARAMETERS

| Term | Definition |
|---|---|
| dates | A date column. |
| start_date | A date expression. |
| number_of_intervals | An integer that specifies the number of intervals to add to, or subtract from, the dates. |
| interval | The interval by which to shift the dates. The value for interval can be one of the following: DAY, MONTH, QUARTER, and YEAR |

## Return value

A table containing a single column of date values.

## Remarks

- In the most common use case, **dates** is a reference to the date column of a marked date table.
- If the number specified for **number_of_intervals** is positive, dates are moved forward in time; if the number is negative, dates are shifted backward in time.
- The **interval** parameter is an enumeration. Valid values are DAY, MONTH, QUARTER, and YEAR. Because it's an enumeration, values aren't passed in as strings. So don't enclose them within quotation marks.

- The returned table can only contain dates stored in the **dates** column. So, for example, if the **dates** column starts from July 1, 2017, and the **start_date** value is July 1, 2016, the returned table will start from July 1, 2017.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Source: https://docs.microsoft.com/en-us/dax/datesinperiod-function-dax

# DATESMTD

Returns a table that contains a column of the dates for the month to date, in the current context.

## Syntax

DAX

```
DATESMTD(<dates>)
```

### Parameters

| PARAMETERS | |
|---|---|
| **Term** | **Definition** |
| dates | A column that contains dates. |

## Return value

A table containing a single column of date values.

## Remarks

The **dates** argument can be any of the following:

- A reference to a date/time column.
- A table expression that returns a single column of date/time values.
- A Boolean expression that defines a single-column table of date/time values.

  **Note**

  Constraints on Boolean expressions are described in the topic, **CALCULATE function**.

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

Source: https://docs.microsoft.com/en-us/dax/datesmtd-function-dax

# DATESQTD

Returns a table that contains a column of the dates for the quarter to date, in the current context.

## Syntax

DAX

```
DATESQTD(<dates>)
```

### Parameters

| PARAMETERS | |
|---|---|
| **Term** | **Definition** |
| dates | A column that contains dates. |

## Return value

A table containing a single column of date values.

## Remarks

The **dates** argument can be any of the following:

- A reference to a date/time column.
- A table expression that returns a single column of date/time values.
- A Boolean expression that defines a single-column table of date/time values.

  **Note**

  Constraints on Boolean expressions are described in the topic, **CALCULATE function**.

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

Source: https://docs.microsoft.com/en-us/dax/datesqtd-function-dax

# DATESYTD

Returns a table that contains a column of the dates for the year to date, in the current context.

## Syntax

DAX

```
DATESYTD(<dates> [,<year_end_date>])
```

### Parameters

**PARAMETERS**

| Term | Definition |
|---|---|
| dates | A column that contains dates. |
| year_end_date | (optional) A literal string with a date that defines the year-end date. The default is December 31. |

## Return value

A table containing a single column of date values.

## Remarks

The **dates** argument can be any of the following:

- A reference to a date/time column,
- A table expression that returns a single column of date/time values,
- A Boolean expression that defines a single-column table of date/time values.

  **Note**

  Constraints on Boolean expressions are described in the topic, **CALCULATE function**.

- The **year_end_date** parameter is a string literal of a date, in the same locale as the locale of the client where the workbook was created. The year portion of the date is ignored.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Source: https://docs.microsoft.com/en-us/dax/datesytd-function-dax

# ENDOFMONTH

Returns the last date of the month in the current context for the specified column of dates.

## Syntax

DAX

```
ENDOFMONTH(<dates>)
```

### Parameters

**PARAMETERS**

| Term | Definition |
|---|---|
| dates | A column that contains dates. |

## Return value

A table containing a single column and single row with a date value.

## Remarks

- The **dates** argument can be any of the following:
  - A reference to a date/time column.
  - A table expression that returns a single column of date/time values.
  - A Boolean expression that defines a single-column table of date/time values.
- Constraints on Boolean expressions are described in the topic, CALCULATE function.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Source: https://docs.microsoft.com/en-us/dax/endofmonth-function-dax

# ENDOFQUARTER

Returns the last date of the quarter in the current context for the specified column of dates.

## Syntax

DAX

```
ENDOFQUARTER(<dates>)
```

### Parameters

**PARAMETERS**

| Term | Definition |
|---|---|
| dates | A column that contains dates. |

## Return value

A table containing a single column and single row with a date value.

## Remarks

- The **dates** argument can be any of the following:
  - A reference to a date/time column,
  - A table expression that returns a single column of date/time values,
  - A Boolean expression that defines a single-column table of date/time values.
- Constraints on Boolean expressions are described in the topic, CALCULATE function.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Source: https://docs.microsoft.com/en-us/dax/endofquarter-function-dax

# ENDOFYEAR

Returns the last date of the year in the current context for the specified column of dates.

## Syntax

DAX

```
ENDOFYEAR(<dates> [,<year_end_date>])
```

### Parameters

**PARAMETERS**

| Term | Definition |
|---|---|
| dates | A column that contains dates. |
| year_end_date | (optional) A literal string with a date that defines the year-end date. The default is December 31. |

## Return value

A table containing a single column and single row with a date value.

## Remarks

- The **dates** argument can be any of the following:
  - A reference to a date/time column,
  - A table expression that returns a single column of date/time values,
  - A Boolean expression that defines a single-column table of date/time values.
- Constraints on Boolean expressions are described in the topic, CALCULATE function.
- The **year_end_date** parameter is a string literal of a date, in the same locale as the locale of the client where the workbook was created. The year portion of the date is ignored.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Source: https://docs.microsoft.com/en-us/dax/endofyear-function-dax

# FIRSTDATE

Returns the first date in the current context for the specified column of dates.

## Syntax

DAX

```
FIRSTDATE(<dates>)
```

### Parameters

**PARAMETERS**

| Term | Definition |
|---|---|
| dates | A column that contains dates. |

## Return value

A table containing a single column and single row with a date value.

## Remarks

- The **dates** argument can be any of the following:
  - A reference to a date/time column.
  - A table expression that returns a single column of date/time values.
  - A Boolean expression that defines a single-column table of date/time values.
- Constraints on Boolean expressions are described in the topic, CALCULATE function.
- When the current context is a single date, the date returned by the FIRSTDATE and LASTDATE functions will be equal.
- The Return value is a table that contains a single column and single value. Therefore, this function can be used as an argument to any function that requires a table in its arguments. Also, the returned value can be used whenever a date value is required.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Source: https://docs.microsoft.com/en-us/dax/firstdate-function-dax

# LASTDATE

Returns the last date in the current context for the specified column of dates.

## Syntax

DAX

```
LASTDATE(<dates>)
```

### Parameters

| PARAMETERS | |
|---|---|
| **Term** | **Definition** |
| dates | A column that contains dates. |

## Return value

A table containing a single column and single row with a date value.

## Remarks

- The **dates** argument can be any of the following:
  - A reference to a date/time column,
  - A table expression that returns a single column of date/time values,
  - A Boolean expression that defines a single-column table of date/time values.
- Constraints on Boolean expressions are described in the topic, CALCULATE function.
- When the current context is a single date, the date returned by the FIRSTDATE and LASTDATE functions will be equal.
- Technically, the Return value is a table that contains a single column and single value. Therefore, this function can be used as an argument to any function that requires a table in its arguments. Also, the returned value can be used whenever a date value is required.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Source: https://docs.microsoft.com/en-us/dax/lastdate-function-dax

# PARALLELPERIOD

Returns a table that contains a column of dates that represents a period parallel to the dates in the specified **dates** column, in the current context, with the dates shifted a number of intervals either forward in time or back in time.

## Syntax

DAX

```
PARALLELPERIOD(<dates>,<number_of_intervals>,<interval>)
```

### Parameters

**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.
- Constraints on Boolean expressions are described in the topic, CALCULATE function.

- If the number specified for **number_of_intervals** is positive, the dates in **dates** are moved forward in time; if the number is negative, the dates in **dates** are shifted back in time.
- The **interval** parameter is an enumeration, not a set of strings; therefore values should not be enclosed in quotation marks. Also, the values: `year`, `quarter`, `month` should be spelled in full when using them.
- The result table includes only dates that appear in the values of the underlying table column.
- The PARALLELPERIOD function is similar to the DATEADD function except that PARALLELPERIOD always returns full periods at the given granularity level instead of the partial periods that DATEADD returns. For example, if you have a selection of dates that starts at June 10 and finishes at June 21 of the same year, and you want to shift that selection forward by one month then the PARALLELPERIOD function will return all dates from the next month (July 1 to July 31); however, if DATEADD is used instead, then the result will include only dates from July 10 to July 21.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Source: https://docs.microsoft.com/en-us/dax/parallelperiod-function-dax

# SAMEPERIODLASTYEAR

Returns a table that contains a column of dates shifted one year back in time from the dates in the specified **dates** column, in the current context.

## Syntax

DAX

```
SAMEPERIODLASTYEAR(<dates>)
```

### Parameters

**PARAMETERS**

| Term | Definition |
|---|---|
| **dates** | A column containing dates. |

## Return value

A single-column table of date values.

## Remarks

- The **dates** argument can be any of the following:
  - A reference to a date/time column,
  - A table expression that returns a single column of date/time values,
  - A Boolean expression that defines a single-column table of date/time values.
- Constraints on Boolean expressions are described in the topic, CALCULATE.
- The dates returned are the same as the dates returned by this equivalent formula: `DATEADD(dates, -1, year)`
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Source: https://docs.microsoft.com/en-us/dax/sameperiodlastyear-function-dax

# CALCULATE

Evaluates an expression in a modified filter context.

**Note**

There's also the **CALCULATETABLE** function. It performs exactly the same functionality, except it modifies the **filter context** applied to an expression that returns a *table object*.

## Syntax

DAX

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

### Parameters

**PARAMETERS**

| Term | Definition |
|---|---|
| expression | The expression to be evaluated. |
| filter1, filter2,... | (Optional) Boolean expressions or table expressions that defines filters, or filter modifier functions. |

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

Filters can be:

- Boolean filter expressions
- Table filter expressions
- Filter modification functions

When there are multiple filters, they're evaluated by using the AND logical operator. That means all conditions must be TRUE at the same time.

*Boolean filter expressions*

A Boolean expression filter is an expression that evaluates to TRUE or FALSE. There are several rules that they must abide by:

- They can reference only a single column.
- They cannot reference measures.
- They cannot use a nested CALCULATE function.
- They cannot use functions that scan or return a table, including aggregation functions.

*Table filter expression*

A table expression filter applies a table object as a filter. It could be a reference to a model table, but more likely it's a function that returns a table object. You can use the FILTER function to apply complex filter conditions, including those that cannot be defined by a Boolean filter expression.

*Filter modifier functions*

Filter modification functions allow you to do more than simply add filters. They provide you with additional control when modifying filter context.

**FILTER MODIFIER FUNCTIONS**

| Function | Purpose |
|---|---|
| REMOVEFILTERS | Remove all filters, or filters from one or more columns of a table, or from all columns of a single table. |
| ALL [1], ALLEXCEPT, ALLNOBLANKROW | Remove filters from one or more columns, or from all columns of a single table. |
| KEEPFILTERS | Add filter without removing existing filters on the same columns. |
| USERELATIONSHIP | Engage an inactive relationship between related columns, in which case the active relationship will automatically become inactive. |
| CROSSFILTER | Modify filter direction (from both to single, or from single to both) or disable a relationship. |

[1] The ALL function and its variants behave as both filter modifiers and as functions that return table objects. If the REMOVEFILTERS function is supported by your tool, it's better to use it to remove filters.

# Return value

The value that is the result of the expression.

# Remarks

- When filter expressions are provided, the CALCULATE function modifies the filter context to evaluate the expression. For each filter expression, there are two possible standard outcomes when the filter expression is not wrapped in the KEEPFILTERS function:
    - If the columns (or tables) aren't in the filter context, then new filters will be added to the filter context to evaluate the expression.
    - If the columns (or tables) are already in the filter context, the existing filters will be overwritten by the new filters to evaluate the CALCULATE expression.
- The CALCULATE function used *without filters* achieves a specific requirement. It transitions row context to filter context. It's required when an expression (not a model measure) that summarizes model data needs to be evaluated in row context. This scenario can happen in a calculated column formula or when an expression in an iterator function is evaluated. Note that when a model measure is used in row context, context transition is automatic.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Source: https://docs.microsoft.com/en-us/dax/calculate-function-dax

# CALCULATETABLE

Evaluates a table expression in a modified filter context.

**Note**

There's also the **CALCULATE** function. It performs exactly the same functionality, except it modifies the **filter context** applied to an expression that returns a *scalar value*.

## Syntax

DAX

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

### Parameters

**PARAMETERS**

| Term | Definition |
|---|---|
| expression | The table expression to be evaluated. |
| filter1, filter2,... | (Optional) Boolean expressions or table expressions that defines filters, or filter modifier functions. |

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

Filters can be:

- Boolean filter expressions
- Table filter expressions
- Filter modification functions

When there are multiple filters, they're evaluated by using the AND logical operator. That means all conditions must be TRUE at the same time.

*Boolean filter expressions*

A Boolean expression filter is an expression that evaluates to TRUE or FALSE. There are several rules that they must abide by:

- They can reference only a single column.
- They cannot reference measures.
- They cannot use a nested CALCULATE function.
- They cannot use functions that scan or return a table, including aggregation functions.

*Table filter expression*

A table expression filter applies a table object as a filter. It could be a reference to a model table, but more likely it's a function that returns a table object. You can use the FILTER function to apply complex filter conditions, including those that cannot be defined by a Boolean filter expression.

*Filter modifier functions*

Filter modification functions allow you to do more than simply add filters. They provide you with additional control when modifying filter context.

**FILTER MODIFIER FUNCTIONS**

| Function | Purpose |
|---|---|
| REMOVEFILTERS | Remove all filters, or filters from one or more columns of a table, or from all columns of a single table. |
| ALL [1], ALLEXCEPT, ALLNOBLANKROW | Remove filters from one or more columns, or from all columns of a single table. |
| KEEPFILTERS | Add filter without removing existing filters on the same columns. |
| USERELATIONSHIP | Engage an inactive relationship between related columns, in which case the active relationship will automatically become inactive. |
| CROSSFILTER | Modify filter direction (from both to single, or from single to both) or disable a relationship. |

[1] The ALL function and its variants behave as both filter modifiers and as functions that return table objects. If the REMOVEFILTERS function is supported by your tool, it's better to use it to remove filters.

# Return value

A table of values.

# Remarks

- When filter expressions are provided, the CALCULATETABLE function modifies the filter context to evaluate the expression. For each filter expression, there are two possible standard outcomes when the filter expression is not wrapped in the KEEPFILTERS function:
    - If the columns (or tables) aren't in the filter context, then new filters will be added to the filter context to evaluate the expression.
    - If the columns (or tables) are already in the filter context, the existing filters will be overwritten by the new filters to evaluate the CALCULATETABLE expression.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Source: https://docs.microsoft.com/en-us/dax/calculatetable-function-dax

# FILTER

Returns a table that represents a subset of another table or expression.

## Syntax

DAX

```
FILTER(<table>,<filter>)
```

### Parameters

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

Source: https://docs.microsoft.com/en-us/dax/filter-function-dax

# KEEPFILTERS

Modifies how filters are applied while evaluating a CALCULATE or CALCULATETABLE function.

## Syntax

DAX

```
KEEPFILTERS(<expression>)
```

### Parameters

**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.

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

Source: https://docs.microsoft.com/en-us/dax/keepfilters-function-dax