# DAX Language
# Quick Reference Guide

Compiled by

Excelerator BI
Power BI Solutions for Business

**Last updated 29th July 2022 - Version 5.1**

# DAX Functions Quick Reference Guide

This DAX functions quick reference guide has been prepared by Matt Allington from http://exceleratorbi.com.au and contains a list of all the current DAX functions in a summarised and easy to use format. You can print the document and/or use the search features for PDF documents to search for the function you are looking for.

This document is a supplement and is not intended to replace the more detailed documentation that is available online.

When looking for online documentation it is best to do a web search from your favourite search engine by specifying the function name followed by the word DAX i.e. "**FunctionName DAX**".

**Tip:** If you are going to search this document for a function name using search, then type the function name followed by an open bracket. E.g. instead of searching for VALUES you should search **VALUES(**.

## Table of Contents





**Last updated 29th July 2022 - Version 5.1**

## DAX Aggregation Functions (Aggregators)

DAX **Aggregation Functions** (called **aggregators** for short) take a column or a table as the argument and aggregate the values.

| Function | Notes |
|---|---|
| AVERAGE(column) | Returns the average (arithmetic mean) of all the numbers in a column in the current filter context.<br>This function is the equivalent of adding up the values in the column and then dividing by the number of rows. |
| AVERAGEA(column) | The **AVERAGEA** function takes a column and averages the numbers in it, but also handles non-numeric data types according to the following rules:<br>• Values that evaluates to **TRUE** count as 1.<br>• Values that evaluate to **FALSE** count as 0 (zero).<br>• Values that contain non-numeric text count as 0 (zero).<br>• Empty text ("") counts as 0 (zero). |
| COUNT(column) | Counts numbers only in the current filter context. |
| COUNTA(column) | Counts text values as well as numbers in the current filter context. |
| COUNTBLANK(column) | Counts the number of blank cells in a column in the current filter context. |
| COUNTROWS(table) | The **COUNTROWS** function counts the number of rows in the specified table, or in a table defined by an expression in the current filter context. |
| DISTINCTCOUNT(column) | Counts each value in a column once and only once in the current filter context. |
| DISTINCTCOUNTNOBLANK(column) | Counts the number of distinct values in a column.<br>Unlike DISTINCTCOUNT function, DISTINCTCOUNTNOBLANK does not include the BLANK value. |
| MAX(column)<br>MAX(<br>expression1,<br>expression2<br>) | The **MAX** function takes a column or two expressions that return numeric values as argument(s) and returns the largest numeric value.<br>Ignores logical values and text in the current filter context.<br>Can also find the MAX of a text column. |
| MAXA(column) | Like **MAX**, however also considers Dates and Logical values, such as **TRUE** and **FALSE**. Rows that evaluate to **TRUE** count as 1; rows that evaluate to **FALSE** count as 0 (zero) |
| MIN(column)<br>MIN(<br>expression1,<br>expression2<br>) | The **MIN** function takes a column or two expressions that return numeric values as argument(s) and returns the smallest numeric value.<br>Ignores logical values and text in the current filter context. |

| Function | Notes |
|---|---|
| MINA(column) | Like **MIN**, however also considers Dates and Logical Values.<br>Rows that evaluate to **TRUE** count as 1; rows that evaluate to **FALSE** count as 0 (zero) |
| PRODUCT(column) | Multiplies all the values in a column together. Why you would want to? I have no idea. |
| SUM(column) | Adds all the numbers in a column in the current filter context. |
| TOPN(<br>n_value,<br>tablename,<br>orderByexpression1, [Order],<br>[orderByexpression2, [Order]], …<br>) | Returns a table containing the top N rows. Order by expression is typically a measure that you want to rank on. |

## DAX Date and Time Functions

You can use DAX Date and Time Functions in the calculations based on dates and time. DAX Date and Time Functions are like the Excel date and time functions but use a **datetime** data type and can take values from a column as an argument.

| Function | Notes |
|---|---|
| CALENDAR(<br>start date,<br>end date<br>) | 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. |
| CALENDARAUTO(<br>[end month of fiscal year]<br>) | 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. |
| DATE(<br>year,<br>month,<br>day<br>) | Returns the specified date in datetime format.<br>• Dates beginning with March 1, 1900 are supported.<br>• If the year value is between 0 and 1899, the value is added to 1900 to produce the final value. The value of the year argument can include one to four digits.<br>• You should use four digits for the year argument whenever possible to prevent unwanted results. |
| DATEDIFF(<br>start date,<br>end date,<br>interval<br>) | Returns the count of interval boundaries crossed between two dates.<br>The interval to use when comparing dates can be one of the following:<br>• SECOND<br>• MINUTE<br>• HOUR<br>• DAY<br>• WEEK<br>• MONTH<br>• QUARTER<br>• YEAR<br>If start_date is larger than end_date an error is returned. |
| DATEVALUE(date text) | Converts a date in the form of text to a date in datetime format. |
| DAY(date) | Returns the day of the month, a number from 1 to 31. |
| EDATE(<br>start date,<br>months<br>) | Returns the date that is the indicated number of months before or after the start date.<br>You can use **EDATE** to calculate maturity dates or due dates that fall on the same day of the month as the date of issue. You can also use **EDATE** to find the same date prior month or prior year. |

| Function | Notes |
|---|---|
| EOMONTH(<br>start date,<br>months<br>) | Returns the date in datetime format of the last day of the month, before or after a specified number of months.<br>You can use **EOMONTH** to calculate maturity dates or due dates that fall on the last day of the month. |
| HOUR(datetime value) | Returns the hour as a number from 0 (12:00 A.M.) to 23 (11:00 P.M.). |
| MINUTE(datetime value) | Returns the minute as a number from 0 to 59, given a date and time value. |
| MONTH(datetime value) | Returns the month as a number from 1 (January) to 12 (December). |
| NETWORKDAYS(<br>start date,<br>end date<br>[, <weekend>, <holidays>]<br>) | Returns the number of whole workdays between two dates (inclusive).<br>**weekend** indicates the weekend days and they are not included in the calculation of the number of days.<br>weekend can be omitted or can be a number between 1 and 17.<br>if omitted or if it is 1 then the weekend is Saturday & Sunday.<br>the numbers from 2 to 17 represent different combinations of days in the week that are the weekend.<br>**holidays** is a table with a column of one or more dates that are to be excluded from the working day calendar. |
| NOW() | Returns the current date and time in **datetime** format.<br>The **NOW** function is useful when you need to display the current date and time on a report or calculate a value based on the current date and time, and have that value updated each time you open the workbook. |
| QUARTER(datetime value) | New in Power BI Desktop.<br>Returns the quarter as a number from 1 to 4.<br>1 (Jan – Mar) … 4 (Oct – Dec)<br>If the input value is BLANK, the output value is also BLANK. |
| SECOND(datetime value) | Returns the seconds of a time value, as a number from 0 to 59. |
| TIME(<br>hour,<br>minute,<br>second<br>) | Converts hours, minutes, and seconds given as numbers to a time in **datetime** format. |
| TIMEVALUE(time text) | Converts a time in text format to a time in **datetime** format. |

| Function | Notes |
|---|---|
| TODAY() | Returns the current date in **datetime** format.<br>The **TODAY** function is useful when you need to have the current date displayed on a report. It is also useful for calculating intervals. |
| UTCNOW() | Returns the current UTC date and time. |
| UTCTODAY() | Returns the current UTC date. |
| WEEKDAY(<br>date,<br>return type<br>) | Returns a number from 1 to 7 identifying the day of the week of a date.<br>• If return type is 1, and the week begins on Sunday (1) and ends on Saturday (7).<br>• If return type is 2, the week begins on Monday (1) and ends on Sunday (7).<br>• If return type is 3, the week begins on Monday (0) and ends on Sunday (6). |
| WEEKNUM(<br>date,<br>return type<br>) | Returns the week number for the given date in a year according to the **return_type** value. The week number indicates where the week falls numerically within a year.<br>If return type is 1, week begins on Sunday. Weekdays are numbered 1 through 7.<br>If return type is 2, week begins on Monday. Weekdays are numbered 1 through 7. |
| YEAR(datetime value) | Returns the year of a date as a four-digit integer in the range 1900-9999. |
| YEARFRAC(<br>start_date,<br>end_date,<br>[basis]<br>) | Calculates the fraction of the year represented by the number of whole days between two dates.<br>Use the **YEARFRAC** function to identify the proportion of a whole year's benefits or obligations to assign to a specific term.<br>Basis (optional) is the type of day count basis to use:<br>0 - US (NASD) 30/360<br>1 - Actual/actual<br>2 - Actual/360<br>3 - Actual/365<br>4 - European 30/360 |

## DAX Filter Functions

DAX Filter Functions are very different to Excel functions. They are used to (typically) return filtered tables that can be used in your data model. These new "virtual" tables retain lineage with the physical data model and hence they can "filter" the physical data model on the fly. Lookup functions work by using tables and relationships between them. Filtering functions let you manipulate data context to create dynamic calculations.

DAX **FILTER** and **VALUES** functions are the most complex and powerful functions.

| Function | Notes |
|---|---|
| 1.<br>ADDMISSINGITEMS(<br>showAllColumn[, showAllColumn] …,<br>table,<br>groupingColumn[, groupingColumn] …<br>[, filterTable] …<br>)<br>2.<br>ADDMISSINGITEMS(<br>showAllColumn[, showAllColumn] …,<br>table,<br>[ROLLUPISSUBTOTAL(]<br>groupingColumn[, isSubtotal_columnName]<br>[, groupingColumn][, isSubtotal_columnName]<br>…<br>[)],<br>[, filterTable] …<br>) | 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.<br>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.<br>The **ADDMISSINGITEMS** function will return **BLANK** values for the **IsSubtotal** columns of blank rows it adds.<br>See also: The argument functions – **ROLLUPISSUBTOTAL** and **ROLLUPGROUP.** |

| Function | Notes |
|---|---|
| ALL(<br>[TableOrColumn]<br>[, TableOrColumn] …<br>) | Updated in Power BI Desktop to include ALL().<br>Returns all the rows in a table, or all the values in a column, ignoring any filters that might have been applied in the current context.<br>If Table is used as the argument, this function is useful for clearing filters and creating calculations on all the rows in a table.<br>If specific Columns are used as the arguments, this function removes all filters from the specified columns in the table and all other filters on other columns in the table still apply.<br>This is useful when you want to remove the context filters for one or more specific columns and to keep all other context filters.<br>ALL() removes all the filters everywhere. ALL() can only be used to clear filters but not to return a table. |
| ALLCROSSFILTERED(table) | Clear all filters which are applied to a table.<br>ALLCROSSFILTERED can only be used to clear filters but not to return a table. |
| ALLEXCEPT(<br>table,<br>column<br>[, column] …<br>) | Removes all context filters in the table except filters that are applied to the specified columns.<br>This is convenient to use when you want to remove the filters on many, but not all, columns in a table. |
| ALLNOBLANKROW(table \| column) | When the passed parameter was a table, returns all rows but the blank row from the parent table of a relationship and disregards any context filters that might exist.<br>When the passed parameter was a column, returns all distinct values of the column but the blank row, and disregards any context filters that might exist. |

| Function | Notes |
| --- | --- |
| ALLSELECTED([tableName \| columnName]) | 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.<br><br>Keeps filters on Rows and Columns in a pivot table while keeping the filters on slicers and other explicit filters.<br><br>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. |
| CALCULATE(<br>expression<br>[, filter1]<br>[, filter2] …<br>) | Returns a scalar value.<br><br>Modifies the initial filter context prior to calculating the expression. It can do this in 2 ways, by applying the new filters specified in the **CALCULATE** function and/or by converting an existing row context into an equivalent filter context aka context transition.<br><br>expression is same as a measure.<br><br>Filters can be:<br><br>• Boolean filter expressions<br>• Table filter expressions<br>• Filter modification functions<br><br>Boolean filter expressions can contain an aggregation function that returns a scalar value.<br><br>When there are multiple filters, they can be evaluated by using AND (&&) meaning all conditions must be TRUE, or by OR (\|\|), meaning either condition can be true. |

| Function | Notes |
|---|---|
| CALCULATETABLE(<br>expression,<br>filter1,<br>filter2, …<br>) | Returns a table of values.<br>Modifies the filter context prior to returning a table of values. It can do this in 2 ways, by applying the new filters specified in the **CALCULATETABLE** function and/or by converting an existing row context into an equivalent filter context aka context transition.<br>expression must be a table in the data model or a function that returns a table.<br>Filters can be:<br>• Boolean filter expressions<br>• Table filter expressions<br>• Filter modification functions<br>Boolean filter expressions can contain an aggregation function that returns a scalar value.<br>When there are multiple filters, they can be evaluated by using AND (&&) meaning all conditions must be TRUE. |
| CROSSFILTER(<br>columnName1,<br>columnName2,<br>direction<br>) | Sets the cross-filtering direction for the indicated relationship, for the duration of the query. It does not return any value.<br>You can use CROSSFILTER inside CALCULATE.<br>The cross-filter direction to be used is determined by the argument – direction.<br>None - No cross-filtering occurs along this relationship<br>Both - Filters on either side filter the other<br>**OneWay** - Filters on the one or lookup side of the side of the relationship filter the many side.<br>**OneWay_LeftFiltersRight** - Filters on the side of columnName1 filter the side of columnName2.<br>**OneWay_RightFiltersLeft** - Filters on the side of columnName2 filter the side of columnName1. |
| DISTINCT(column) | Returns a 1 column table of all the distinct values in the current filter context. If there are BLANKS then they will be ignored. If you want to return a BLANK as well, then use **VALUES** function instead. |
| DISTINCT(table) | Returns a table by removing duplicate rows from another table or expression.<br>The returned table will contain unique (or distinct) rows. |

| Function | Notes |
|---|---|
| EARLIER(<br>column<br>[, number]<br>) | Used to access a previous row context when more than 1 row context exists in the function.<br>**EARLIER** is useful for nested calculations where you want to use a certain value as an input and produce calculations based on that input. |
| EARLIEST(column) | As above, but returns the absolute first row context. |
| FILTER(<br>table,<br>filter<br>) | Returns a table containing only the filtered rows.<br>You can use **FILTER** to use only specific data in calculations.<br>**FILTER** is not used independently, but as a function that is embedded in other functions such as **CALCULATE**. |
| FILTERS(columnName) | Returns a table containing the list of values that are directly applied as filters. |
| HASONEFILTER(columnName) | Returns TRUE when the number of directly filtered values on columnName is one; otherwise returns FALSE.<br>Used to check if there is one and only one filter on a column in the current filter context. |
| HASONEVALUE(columnName) | Returns TRUE when the context for columnName has been filtered down to one distinct value only. Otherwise is FALSE.<br>Used to check if there is one and only one value visible in a column in the current filter context. |
| ISCROSSFILTERED(columnName) | Returns **TRUE** when columnName or another column in the same or related table is being filtered. Otherwise returns FALSE.<br>Used to check if there is an **indirect filter** on a column in the current filter context. |
| ISFILTERED(columnName) | 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.<br>Used to check if there the column is filtered at all in the current filter context. |

| Function | Notes |
|---|---|
| KEEPFILTERS(expression) | You use **KEEPFILTERS** within the context **CALCULATE** and **CALCULATETABLE** functions, to override the standard behaviour of those functions.<br><br>**CALCULATE filters** replace the current context, while **KEEPFILTERS** adds filters to the current context. |
| RELATED(column) | Returns a related value from another table. A single value that is related to the current row.<br><br>Forces a row context to follow the relationship to a related table and return that value. Can only be used on the many side of the relationship. |
| RELATEDTABLE(tableName) | Returns a table of values from the many side of the relationship.<br><br>Forces a row context to follow the relationship to a related table and return that value. Can only be used on the one side of the relationship. |
| REMOVEFILTERS(<br>[table \| column[, column[, column[,…]]]])<br><br>) | Clears filters from the specified table or columns.<br><br>**REMOVEFILTERS** can only be used to clear filters but not to return a table.<br><br>So it can only be used within **CALCULATE**. |
| SELECTEDVALUE(<br>columnName<br>[, alternateResult]<br><br>) | It can be used as a substitute for **IF(HASONEVALUE())** and will return a scalar value if there is one and only 1 value selected in the current filter context.<br><br>Otherwise returns alternateResult.<br><br>If alternateResult is omitted, the default value is **BLANK().** |

| Function | Notes |
|---|---|
| SUBSTITUTEWITHINDEX(<br>table,<br>indexColumnName,<br>indexColumnsTable,<br>orderBy_expression[, order]<br>[, orderBy_expression[, order]] …<br>) | 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.<br>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.<br>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. |
| TREATAS(<br>table_expression,<br>column1<br>[, column2]<br>[, column3] …<br>) | Returns a table that contains all the rows in column(s) that are also in table_expression.<br>The number of columns specified must be equal to the number of columns in the table expression, and be in the same order.<br>Use when a relationship does not exist between the tables. |
| USERELATIONSHIP(<br>columnName1,<br>columnName2<br>) | You can have more than 1 relationship between 2 tables in DAX, but only 1 can be active at a time. Use this function inside **CALCULATE** to use the inactive relationship instead of the active one. |
| VALUES(TableNameOrColumnName) | Returns a table consisting of a single column of unique values in the current filter context. If there are blanks in the list a blank will be returned.<br>If you want to exclude the blank then use **DISTINCT** instead. |

# DAX Financial Functions

DAX Financial Functions are used in formulas that perform financial calculations. These functions are similar to Excel financial functions.

| Function | Notes |
|---|---|
| ACCRINT(<br>issue,<br>first_interest,<br>settlement,<br>rate,<br>par,<br>frequency<br>[, basis[, calc_method]]<br>) | Returns the accrued interest for a security that pays periodic interest.<br>Formula used:<br>$$\text{ACCRINT} = \text{par} \times \frac{\text{rate}}{\text{frequency}} \times \sum_{i=1}^{\text{NC}} \frac{A_i}{NL_i}$$<br>where:<br>$A_i$= number of accrued days for the $i^{th}$ quasi-coupon period within odd period.<br>NC = number of quasi-coupon periods that fit in odd period. If this number contains a fraction, raise it to the next whole number.<br>$NL_i$ = normal length in days of the quasi-coupon period within odd period. |
| ACCRINTM(<br>issue,<br>maturity,<br>rate,<br>par<br>[, basis]<br>) | Returns the accrued interest for a security that pays interest at maturity.<br>Formula used:<br>$$\text{ACCRINTM} = \text{par} \times \text{rate} \times \frac{A}{D}$$<br>where:<br>A = Number of accrued days counted according to a monthly basis. For interest at maturity items, the number of days from the issue date to the maturity date is used.<br>D = Annual Year Basis. |
| AMORDEGRC(<br>cost,<br>date_purchased,<br>first_period,<br>salvage,<br>period,<br>rate<br>[, basis]<br>) | Returns the depreciation for each accounting period.<br>This function is provided for the French accounting system.<br>If an asset is purchased in the middle of the accounting period, the prorated depreciation is taken into account.<br>The function is similar to **AMORLINC**, except that a depreciation coefficient is applied in the calculation depending on the life of the assets. |

| Function | Notes |
|---|---|
| AMORLINC(<br>cost,<br>date_purchased,<br>first_period,<br>salvage,<br>period,<br>rate<br>[, basis]<br>) | Returns the depreciation for each accounting period.<br>This function is provided for the French accounting system.<br>If an asset is purchased in the middle of the accounting period, the prorated depreciation is taken into account. |
| COUPDAYBS(<br>settlement,<br>maturity,<br>frequency<br>[, basis]<br>) | Returns the number of days from the beginning of a coupon period until its settlement date. |
| COUPDAYS(<br>settlement,<br>maturity,<br>frequency<br>[, basis]<br>) | Returns the number of days in the coupon period that contains the settlement date. |
| COUPDAYSNC(<br>settlement,<br>maturity,<br>frequency<br>[, basis]<br>) | Returns the number of days from the settlement date to the next coupon date. |
| COUPNCD(<br>settlement,<br>maturity,<br>frequency<br>[, basis]<br>) | Returns the next coupon date after the settlement date. |

| Function | Notes |
|---|---|
| COUPNUM(<br>settlement,<br>maturity,<br>frequency<br>[, basis]<br>) | Returns the number of coupons payable between the settlement date and maturity date, rounded up to the nearest whole coupon. |
| COUPPCD(<br>settlement,<br>maturity,<br>frequency<br>[, basis]<br>) | Returns the previous coupon date before the settlement date. |
| CUMIPMT(<br>rate,<br>nper,<br>pv,<br>start_period,<br>end_period,<br>type<br>) | Returns the cumulative interest paid on a loan between start_period and end_period.<br>The units used for specifying rate and nper should be consistent. |
| CUMPRINC(<br>rate,<br>nper,<br>pv,<br>start_period,<br>end_period,<br>type<br>) | Returns the cumulative principal paid on a loan between start_period and end_period.<br>The units used for specifying rate and nper should be consistent. |

| Function | Notes |
|---|---|
| DB(<br>cost,<br>salvage,<br>life,<br>period<br>[, month]<br>) | Returns the depreciation of an asset for a specified period using the fixed-declining balance method.<br>Formulas used:<br>**DB for a period**<br>$$(\text{cost} - \text{total depreciation from prior periods}) \times \text{rate}$$<br>where<br>• $\text{rate} = 1 - ((\frac{\text{salvage}}{\text{cost}})^{(\frac{1}{\text{life}})})$<br>• rate is rounded to 3 decimal places.<br>**DB for the first period**<br>$$\frac{\text{cost} \times \text{rate} \times \text{month}}{12}$$<br>**DB for the last period**<br>$$\frac{(\text{cost} - \text{total depreciation from prior periods}) \times \text{rate} \times (12 - \text{month})}{12}$$ |
| DDB(<br>cost,<br>salvage,<br>life,<br>period<br>[, factor]<br>) | Returns the depreciation of an asset for a specified period using the double-declining balance method or some other method you specify.<br>The parameter ***factor*** specifies the rate at which the balance declines. If factor is omitted, it is assumed to be 2 (the double-declining balance method).<br>Formula used:<br>$$\text{Min}((\text{cost} - \text{total depreciation from prior periods}) \times (\frac{\text{factor}}{\text{life}}),$$<br>$$(\text{cost} - \text{salvage} - \text{total depreciation from prior periods}))$$<br>If you want to switch to the straight-line depreciation method when depreciation is greater than the declining balance calculation use the VDB function. |
| DISC(<br>settlement,<br>maturity,<br>pr,<br>redemption<br>[, basis]<br>) | Returns the discount rate for a security.<br>Formula used:<br>$$\text{DISC} = \frac{\text{redemption} - \text{par}}{\text{redemption}} \times \frac{\text{B}}{\text{DSM}}$$<br>where:<br>• B = number of days in a year, depending on the year basis.<br>• DSM = number of days between settlement and maturity. |
| DOLLARDE(<br>fractional_dollar,<br>fraction<br>) | Converts a dollar price expressed as an integer part and a fraction part, separated by a decimal symbol (e.g. 1.02), into a dollar price expressed as a decimal number.<br>The parameter ***fraction*** is the integer to divide the fraction part of the ***fractional_dollar.*** The choice of integer depends on the dollar precision you want. |

| Function | Notes |
|---|---|
| DOLLARFR(<br>decimal_dollar,<br>fraction<br>) | Converts a dollar price expressed as a decimal number into a dollar price expressed as an integer part and a fraction part, , separated by a decimal symbol (e.g. 1.02.) |
| DURATION(<br>settlement,<br>maturity,<br>coupon,<br>yld,<br>frequency<br>[, basis]<br>) | Returns the Macauley duration for an assumed par value of $100.<br>Duration is defined as the weighted average of the present value of cash flows and is used as a measure of a bond price's response to changes in yield. |
| EFFECT(<br>nominal_rate,<br>npery<br>) | Returns the effective annual interest rate, given the nominal annual interest rate and the number of compounding periods per year.<br>The parameter ***npery*** is used to specify the number of compounding periods per year<br>Formula used:<br>$$\text{EFFECT} = \left(1 + \frac{\text{nominal\_rate}}{\text{npery}}\right)^{\text{npery}} - 1$$ |
| FV(<br>rate,<br>nper,<br>pmt<br>[, pv[, type]]<br>) | Calculates the future value of an investment based on a constant interest rate. You can use FV with either periodic, constant payments, and/or a single lump sum payment.<br>Units used for specifying rate and nper should be consistent. |
| INTRATE(<br>settlement,<br>maturity,<br>investment,<br>redemption<br>[, basis]<br>) | Returns the interest rate for a fully invested security.<br>Formula used:<br>$$\text{INTRATE} = \frac{\text{redemption} - \text{investment}}{\text{investment}} \times \frac{\text{B}}{\text{DIM}}$$<br>where<br>• B = number of days in a year, depending on the year basis.<br>• DIM = number of days from settlement to maturity. |

| Function | Notes |
|---|---|
| IPMT(<br>rate,<br>per,<br>nper,<br>pv<br>[, fv[, type]]<br>) | Returns the interest payment for a given period for an investment based on periodic, constant payments and a constant interest rate.<br>Units used for specifying rate and nper should be consistent. |
| ISPMT(<br>rate,<br>per,<br>nper,<br>pv<br>) | Calculates the interest paid (or received) for the specified period of a loan (or investment) with even principal payments. |
| MDURATION(<br>settlement,<br>maturity,<br>coupon,<br>yld,<br>frequency<br>[, basis]<br>) | Returns the modified Macauley duration for a security with an assumed par value of $100.<br>Formula used:<br>$$\text{MDURATION} = \frac{\text{DURATION}}{1 + (\frac{\text{Market yield}}{\text{Coupon payments per year}})}$$ |
| NOMINAL(<br>effect_rate,<br>npery<br>) | Returns the nominal annual interest rate, given the effective rate and the number of compounding periods per year.<br>The relationship between NOMINAL and EFFECT is shown in the following equation:<br>$$\text{EFFECT} = \left(1 + \frac{\text{nominal\_rate}}{\text{npery}}\right)^{\text{npery}} - 1$$ |
| NPER(<br>rate,<br>pmt,<br>pv<br>[, fv[, type]]<br>) | Returns the number of periods for an investment based on periodic, constant payments and a constant interest rate. |

| Function | Notes |
|---|---|
| ODDFPRICE(<br>settlement,<br>maturity,<br>issue,<br>first_coupon,<br>rate,<br>yld,<br>redemption,<br>frequency<br>[, basis]<br>) | Returns the price per $100 face value of a security having an odd (short or long) first period.<br>Formulas used:<br>**Odd short first coupon:**<br>$$\text{ODDFPRICE} = \left[\frac{\text{redemption}}{(1+\frac{\text{yld}}{\text{frequency}})^{(N-1+\frac{\text{DSC}}{\text{E}})}}\right] + \left[\frac{100 \times \frac{\text{rate}}{\text{frequency}} \times \frac{\text{DFC}}{\text{E}}}{(1+\frac{\text{yld}}{\text{frequency}})^{(\frac{\text{DSC}}{\text{E}})}}\right] + \left[\sum_{k=2}^{N} \frac{100 \times \frac{\text{rate}}{\text{frequency}}}{(1+\frac{\text{yld}}{\text{frequency}})^{(k-1+\frac{\text{DSC}}{\text{E}})}}\right] - \left[100 \times \frac{\text{rate}}{\text{frequency}} \times \frac{\text{A}}{\text{E}}\right]$$<br>where:<br>• A = number of days from the beginning of the coupon period to the settlement date (accrued days).<br>• DSC = number of days from the settlement to the next coupon date.<br>• DFC = number of days from the beginning of the odd first coupon to the first coupon date.<br>• E = number of days in the coupon period.<br>• N = number of coupons payable between the settlement date and the redemption date. (If this number contains a fraction, it is raised to the next whole number.)<br>**Odd long first coupon:**<br>$$\text{ODDFPRICE} = \left[\frac{\text{redemption}}{(1+\frac{\text{yld}}{\text{frequency}})^{(N+N_q+\frac{\text{DSC}}{\text{E}})}}\right] + \left[\frac{100 \times \frac{\text{rate}}{\text{frequency}} \times \left[\sum_{i=1}^{NC} \frac{DC_i}{NL_i}\right]}{(1+\frac{\text{yld}}{\text{frequency}})^{(N_q+\frac{\text{DSC}}{\text{E}})}}\right] + \left[\sum_{k=1}^{N} \frac{100 \times \frac{\text{rate}}{\text{frequency}}}{(1+\frac{\text{yld}}{\text{frequency}})^{(k-N_q+\frac{\text{DSC}}{\text{E}})}}\right] - \left[100 \times \frac{\text{rate}}{\text{frequency}} \times \sum_{i=1}^{NC} \frac{A_i}{NL_i}\right]$$<br>where:<br>• $A_i$ = number of days from the beginning of the $i^{th}$, or last, quasi-coupon period within odd period.<br>• $DC_i$ = number of days from dated date (or issue date) to first quasi-coupon (i = 1) or number of days in quasi-coupon (i = 2, ..., i = NC).<br>• DSC = number of days from settlement to next coupon date.<br>• E = number of days in coupon period.<br>• N = number of coupons payable between the first real coupon date and redemption date. (If this number contains a fraction, it is raised to the next whole number.)<br>• NC = number of quasi-coupon periods that fit in odd period. (If this number contains a fraction, it is raised to the next whole number.)<br>• $NL_i$ = normal length in days of the full $i^{th}$, or last, quasi-coupon period within odd period.<br>• $N_q$ = number of whole quasi-coupon periods between settlement date and first coupon. |

| Function | Notes |
|---|---|
| ODDFYIELD(<br>settlement,<br>maturity,<br>issue,<br>first_coupon,<br>rate,<br>pr,<br>redemption,<br>frequency<br>[, basis]<br>) | Returns the yield of a security that has an odd (short or long) first period.<br>ODDFYIELD is calculated using an iterative method.<br>It uses the Newton method based on the formula used for the function ODDFPRICE.<br>The yield is changed through 100 iterations until the estimated price with the given yield is close to the price. |
| ODDLPRICE(<br>settlement,<br>maturity,<br>last_interest,<br>rate,<br>yld,<br>redemption,<br>frequency<br>[, basis]<br>) | Returns the price per $100 face value of a security having an odd (short or long) last coupon period. |
| ODDLYIELD(<br>settlement,<br>maturity,<br>last_interest,<br>rate,<br>pr,<br>redemption,<br>frequency<br>[, basis]<br>) | Returns the yield of a security that has an odd (short or long) last period.<br>Formula used:<br>$\text{ODDLYIELD} = \left[\frac{(\text{redemption} + ((\sum_{i=1}^{\text{NC}} \frac{\text{DC}_i}{\text{NL}_i}) \times \frac{100 \times \text{rate}}{\text{frequency}})) - (\text{par} + ((\sum_{i=1}^{\text{NC}} \frac{\text{A}_i}{\text{NL}_i}) \times \frac{100 \times \text{rate}}{\text{frequency}}))}{\text{par} + ((\sum_{i=1}^{\text{NC}} \frac{\text{A}_i}{\text{NL}_i}) \times \frac{100 \times \text{rate}}{\text{frequency}})}\right] \times \left[\frac{\text{frequency}}{(\sum_{i=1}^{\text{NC}} \frac{\text{DSC}_i}{\text{NL}_i})}\right]$<br>where:<br>• $A_i$ = number of accrued days for the $i^{th}$, or last, quasi-coupon period within odd period counting forward from last interest date before redemption.<br>• $DC_i$ = number of days counted in the $i^{th}$, or last, quasi-coupon period as delimited by the length of the actual coupon period.<br>• NC = number of quasi-coupon periods that fit in odd period; if this number contains a fraction it will be raised to the next whole number.<br>• $NL_i$ = normal length in days of the $i^{th}$, or last, quasi-coupon period within odd coupon period. |

| Function | Notes |
|---|---|
| PDURATION(<br>rate,<br>pv,<br>fv<br>) | Returns the number of periods required by an investment to reach a specified value.<br>Formula used:<br>$$\text{PDURATION} = \frac{log(\text{fv}) - log(\text{pv})}{log(1 + \text{rate})}$$ |
| PMT(<br>rate,<br>nper,<br>pv<br>[, fv[, type]]<br>) | Calculates the payment for a loan based on constant payments and a constant interest rate. |
| PPMT(<br>rate,<br>per,<br>nper,<br>pv<br>[, fv[, type]]<br>) | Returns the payment on the principal for a given period for an investment based on periodic, constant payments and a constant interest rate. |
| PRICE(<br>settlement,<br>maturity,<br>rate,<br>yld,<br>redemption,<br>frequency<br>[, basis]<br>) | Returns the price per $100 face value of a security that pays periodic interest.<br>Formulas used:<br>• N is the number of coupons payable between the settlement date and redemption date<br>• When N > 1<br>$$\text{PRICE} = \left[\frac{\text{redemption}}{(1 + \frac{\text{yld}}{\text{frequency}})^{(N-1+\frac{\text{DSC}}{\text{E}})}}\right] + \left[\sum_{k=1}^{N} \frac{100 \times \frac{\text{rate}}{\text{frequency}}}{(1 + \frac{\text{yld}}{\text{frequency}})^{(k-1+\frac{\text{DSC}}{\text{E}})}}\right] - \left[100 \times \frac{\text{rate}}{\text{frequency}} \times \frac{\text{A}}{\text{E}}\right]$$<br>• When N = 1<br>$$\text{DSR} = \text{E} - \text{A}$$<br>$$\text{T1} = 100 \times \frac{\text{rate}}{\text{frequency}} + \text{redemption}$$<br>$$\text{T2} = \frac{\text{yld}}{\text{frequency}} \times \frac{\text{DSR}}{\text{E}} + 1$$<br>$$\text{T3} = 100 \times \frac{\text{rate}}{\text{frequency}} \times \frac{\text{A}}{\text{E}}$$<br>$$\text{PRICE} = \frac{\text{T1}}{\text{T2}} - \text{T3}$$<br>where<br>• E = number of days in coupon period in which the settlement date falls.<br>• A = number of days from beginning of coupon period to settlement date. |

| Function | Notes |
|---|---|
| PRICEDISC(<br>settlement,<br>maturity,<br>discount,<br>redemption<br>[, basis]<br>) | Returns the price per $100 face value of a discounted security.<br>Formula used:<br>$$\text{PRICEDISC} = \text{redemption} - \text{discount} \times \text{redemption} \times \frac{\text{DSM}}{\text{B}}$$<br>where<br>• B = number of days in year, depending on year basis.<br>• DSM = number of days from settlement to maturity. |
| PRICEMAT(<br>settlement,<br>maturity,<br>issue,<br>rate,<br>yld<br>[, basis]<br>) | Returns the price per $100 face value of a security that pays interest at maturity.<br>Formula used:<br>$$\text{PRICEMAT} = \frac{100 + (\frac{\text{DIM}}{\text{B}} \times \text{rate} \times 100)}{1 + (\frac{\text{DSM}}{\text{B}} \times \text{yld})} - (\frac{\text{A}}{\text{B}} \times \text{rate} \times 100)$$<br>where:<br>• B = number of days in year, depending on year basis.<br>• DSM = number of days from settlement to maturity.<br>• DIM = number of days from issue to maturity.<br>• A = number of days from issue to settlement. |
| PV(<br>rate,<br>nper,<br>pmt<br>[, fv[, type]]<br>) | Calculates the present value of a loan or an investment, based on a constant interest rate.<br>Formulas used:<br>If rate is not 0, then:<br>$$\text{pv} \times (1 + \text{rate})^{\text{nper}} + \text{pmt}(1 + \text{rate} \times \text{type}) \times \left(\frac{(1 + \text{rate})^{\text{nper}} - 1}{\text{rate}}\right) + \text{fv} = 0$$<br>If rate is 0, then:<br>$$(\text{pmt} \times \text{nper}) + \text{pv} + \text{fv} = 0$$ |
| RATE(<br>nper,<br>pmt,<br>pv<br>[, fv[, type[, guess]]]<br>) | Returns the interest rate per period of an annuity.<br>RATE is calculated by iteration and can have zero or more solutions.<br>If the successive results of RATE do not converge to within 0.0000001 after 20 iterations, an error is returned. |
| RECEIVED(<br>settlement,<br>maturity,<br>investment,<br>discount<br>[, basis]<br>) | Returns the amount received at maturity for a fully invested security.<br>Formula used:<br>$$\text{RECEIVED} = \frac{\text{investment}}{1 - (\text{discount} \times \frac{\text{DIM}}{\text{B}})}$$<br>where<br>• B = number of days in a year, depending on the year basis.<br>• DIM = number of days from issue to maturity. |

| Function | Notes |
|---|---|
| RRI(<br>nper,<br>pv,<br>fv<br>) | Returns an equivalent interest rate for the growth of an investment.<br>Formula used:<br>$$\left(\frac{\text{fv}}{\text{pv}}\right)^{\left(\frac{1}{nper}\right)} - 1$$ |
| SLN(<br>cost,<br>salvage,<br>life<br>) | Returns the straight-line depreciation of an asset for one period. |
| SYD(<br>cost,<br>salvage,<br>life,<br>per<br>) | Returns the sum-of-years' digits depreciation of an asset for a specified period. |
| TBILLEQ(<br>settlement,<br>maturity,<br>discount<br>) | Returns the bond-equivalent yield for a Treasury bill.<br>Formula used:<br>$$\text{TBILLEQ} = \frac{365 \times \text{discount}}{360 - (\text{discount} \times \text{DSM})}$$<br>where:<br>• DSM is the number of days between settlement and maturity computed according to the 360 days per year basis. |
| TBILLPRICE(<br>settlement,<br>maturity,<br>discount<br>) | Returns the price per $100 face value for a Treasury bill.<br>Formula used:<br>$$\text{TBILLPRICE} = 100 \times (1 - \frac{\text{discount} \times \text{DSM}}{360})$$<br>where:<br>• DSM = number of days from settlement to maturity, excluding any maturity date that is more than one calendar year after the settlement date. |
| TBILLYIELD(<br>settlement,<br>maturity,<br>pr<br>) | Returns the yield for a Treasury bill.<br>Formula used:<br>$$\text{TBILLYIELD} = \frac{100 - \text{pr}}{\text{pr}} \times \frac{360}{\text{DSM}}$$<br>where:<br>• DSM = number of days from settlement to maturity, excluding any maturity date that is more than one calendar year after the settlement date. |

| Function | Notes |
|---|---|
| VDB(<br>cost,<br>salvage,<br>life,<br>start_period,<br>end_period<br>[, factor[, no_switch]]<br>) | Returns the depreciation of an asset for any period you specify, including partial periods, using the double-declining balance method or some other method you specify.<br>VDB stands for variable declining balance.<br>The parameter factor is the rate at which the balance declines.<br>If factor is omitted, it is assumed to be 2 (the double-declining balance method). |
| XIRR(<br>table,<br>values,<br>dates<br>[, guess]<br>[, alternateResult]<br>) | Returns the internal rate of return for a schedule of cash flows that is not necessarily periodic.<br>alternateResult is a value returned in place of an error when a solution cannot be determined.<br>Formula used:<br>$\sum_{j=1}^{N} \frac{P_j}{(1+\text{rate})^{\frac{d_j-d_1}{365}}}$<br>Where $P_j$ is the j-th payment, $d_j$ is the j-th payment date, and $d_1$ is the first payment date |
| XNPV(<br>table,<br>values,<br>dates,<br>rate<br>) | Returns the present value for a schedule of cash flows that is not necessarily periodic.<br>Formula used:<br>$\sum_{j=1}^{N} \frac{P_j}{(1+\text{rate})^{\frac{d_j-d_1}{365}}}$<br>Where $P_j$ is the j-th payment, $d_j$ is the j-th payment date, and $d_1$ is the first payment date |

| Function | Notes |
|---|---|
| YIELD(<br>settlement,<br>maturity,<br>rate,<br>pr,<br>redemption,<br>frequency<br>[, basis]<br>) | Returns the yield on a security that pays periodic interest. Use YIELD to calculate bond yield.<br>Formula used:<br>• If there is one coupon period or less until redemption<br>$$\text{YIELD} = \frac{(\frac{\text{redemption}}{100} + \frac{\text{rate}}{\text{frequency}}) - (\frac{\text{par}}{100} + (\frac{\text{A}}{\text{E}} \times \frac{\text{rate}}{\text{frequency}}))}{\frac{\text{par}}{100} + (\frac{\text{A}}{\text{E}} \times \frac{\text{rate}}{\text{frequency}})} \times \frac{\text{frequency} \times \text{E}}{\text{DSR}}$$<br>where:<br>○ A = number of days from the beginning of the coupon period to the settlement date (accrued days).<br>○ DSR = number of days from the settlement date to the redemption date.<br>○ E = number of days in the coupon period.<br>• If there is more than one coupon period until redemption<br>○ YIELD is calculated through a hundred iterations.<br>○ The resolution uses the Newton method, based on the formula used for the function PRICE.<br>○ The yield is changed until the estimated price given the yield is close to price. |
| YIELDDISC(<br>settlement,<br>maturity,<br>pr,<br>redemption<br>[, basis]<br>) | Returns the annual yield for a discounted security. |
| YIELDMAT(<br>settlement,<br>maturity,<br>issue,<br>rate,<br>pr<br>[, basis]<br>) | Returns the annual yield of a security that pays interest at maturity. |

## DAX Information Functions

DAX Information Functions provide required information based on the given argument.

| Function | Notes |
|---|---|
| CONTAINS(<br>table,<br>columnName, value<br>[, columnName, value] …<br>) | Returns **TRUE** if each specified value is contained in the corresponding columnName.Otherwise, the function returns **FALSE**. |
| CONTAINSROW(<br>table,<br>scalar expression1<br>[, scalar expression2, …]<br>) | Returns **TRUE** if a row of values exists or contained in a table.<br>Otherwise returns **FALSE**.<br>Note: The **IN** operator and the **CONTAINSROW** function are functionally the same. Only the syntax is different. |
| CUSTOMDATA() | Returns the content of the **CustomData** property in the connection string.<br>Returns Blank, if **CustomData** property was not defined at connection time. |
| scalar expression IN table<br>OR<br>(scalar expression1, scalar expression2, …) IN table | IN Operator:<br>Returns **TRUE** if a row of values exists or contained in a table.<br>Otherwise returns **FALSE**.<br>Note: The **IN** operator and the **CONTAINSROW** function are functionally the same. Only the syntax is different. |
| ISBLANK(value) | Returns **TRUE** if the value is blank.<br>Otherwise, returns **FALSE**. |
| ISEMPTY(table_expression) | Returns **TRUE** if the table is empty (has no rows),<br>Returns **FALSE** otherwise. |
| ISERROR(value) | Returns **TRUE** if the value is an Error.<br>Otherwise, returns **FALSE**. |
| ISEVEN(number) | Returns **TRUE** if number is even,<br>Returns **FALSE** if number is odd.<br>If number is nonnumeric, **ISEVEN** returns the #VALUE! error value. |
| ISINSCOPE(column) | Returns **TRUE** when the specified column is the level in a hierarchy of levels. |
| ISLOGICAL(value) | Returns **TRUE** if the value is a logical value (**TRUE** OR **FALSE**).<br>Otherwise, returns **FALSE**. |

| Function | Notes |
|---|---|
| ISNONTEXT(value) | Returns **TRUE** if the value is not text or blank. (Blank cells are not text).<br>Returns **FALSE** if the value is text.<br>An empty string is considered as text. |
| ISNUMBER(value) | Returns **TRUE** if the value is numeric.<br>Otherwise, returns **FALSE**. |
| ISODD(number) | Returns **TRUE** if number is odd.<br>Returns **FALSE** if number is even.<br>If number is nonnumeric, **ISODD** returns the #VALUE! error value. |
| ISONORAFTER(<br>scalar expression,<br>scalar expression<br>[, sort order]<br>[, scalar expression,<br>scalar expression<br>[, sort order]<br>], …<br>) | This function takes a variable number of triples, the first two parameters in a triple are the expressions to be compared and the third is the sort order - ascending (default) or descending.<br>Based on the sort order, the first parameter is compared with the second parameter.<br>If the sort order is **ascending**, the comparison to be done is first parameter greater than or equal to second parameter.<br>If the sort order is **descending**, the comparison to be done is first parameter less than or equal to second parameter |
| ISTEXT(value) | Returns **TRUE** if the value is text<br>Otherwise, returns **FALSE**.<br>Empty string is text.<br>**Blank** is not text. |
| LOOKUPVALUE(<br>result_columnName,<br>search_columnName,<br>search_value<br>[, search_columnName,<br>search_value<br>], …<br>[, <alternateResult>]<br>) | Updated in Power BI Desktop to include alternateResult.<br>Returns the value of result_column at the row where all pairs of search_column and search_value have a match.<br>If only some of the criteria match, a **BLANK** or alternateResult (if given) is returned.<br>If multiple rows match the search values and in all cases result_column values are identical, then that value is returned. Otherwise, an error or alternateResult (if given) is returned. |
| USERNAME() | Returns the username from the credentials given to the system at connection time. If this is a Power BI Desktop or Excel file it will be in the format Domain\Username. If it is in the Power BI Service, it will be in the format Username@domain.com |

## DAX Logical Functions

DAX Logical Functions return values based on the conditional results.

| Function | Notes |
|---|---|
| AND(<br>logical_value,<br>logical_value<br>) | Checks whether both arguments are **TRUE**, and returns **TRUE** if both arguments are TRUE. Otherwise returns **FALSE**.<br>If you have more than two arguments, use && (double ampersand) as an alternative and you can have as many "&&" as you like. |
| BITAND(<br>number,<br>number<br>) | Returns a bitwise AND of two numbers.<br>number is any scalar expression that returns number.<br>If not an integer, it is truncated.<br>Supports both positive and negative numbers. |
| BITLSHIFT(<br>number,<br>shift_amount<br>) | Returns a number (integer) shifted left by the specified number of bits.<br>Both number, shift_amount can be any DAX expression that returns an integer expression.<br>If shift_amount is negative, it will shift in the opposite direction.<br>If absolute value of shift_amount is larger than 64, there will be no error but will result in overflow/underflow. |
| BITOR(<br>number,<br>number<br>) | Returns a bitwise OR of two numbers.<br>number is any scalar expression that returns number.<br>If not an integer, it is truncated.<br>Supports both positive and negative numbers. |
| BITRSHIFT(<br>number,<br>shift_amount<br>) | Returns a number (integer) shifted right by the specified number of bits.<br>Both number, shift_amount can be any DAX expression that returns an integer expression.<br>If shift_amount is negative, it will shift in the opposite direction.<br>If absolute value of shift_amount is larger than 64, there will be no error but will result in overflow/underflow. |
| BITXOR(<br>number,<br>number<br>) | Returns a bitwise XOR of two numbers.<br>number is any scalar expression that returns number.<br>If not an integer, it is truncated.<br>Supports both positive and negative numbers. |

| Function | Notes |
|---|---|
| COALESCE(<br>expression,<br>expression<br>[, expression] …<br>) | Returns the scalar value coming from the first expression that does not evaluate to **BLANK**. If all expressions evaluate to **BLANK**, **BLANK** is returned.<br>Input expressions may be of different data types that return a scalar expression.<br>**COALESCE** can be used to convert **BLANK** values of totals to `0`.<br>**COALESCE** simplifies the code that otherwise requires an **IF** condition.<br>i.e. instead of writing **IF(ISBLANK(Result), 0, Result)**, we can write **COALESCE(Result, 0)** |
| FALSE() | Returns the logical value **FALSE**. |
| IF(<br>logical test,<br>value_if_true<br>[, value_if_false]<br>) | Checks if the 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**.<br>Returns blank, if the condition is **FALSE** and value_if_false is omitted. |
| IF.EAGER(<br>logical test,<br>value_if_true<br>[, value_if_false]<br>) | Evaluates both the expressions value_if_true and value_if_false.<br>Then checks if the condition provided as the first argument (logical_test) is met. Returns value_if_true if the condition is **TRUE** and returns value_if_false if the condition is **FALSE**.<br>The difference between IF and IF.Eager is that<br>• IF evaluates the condition first and then selects the branch expression to evaluate based on the result of the condition.<br>• IF-EAGER evaluates both branch expressions first and then evaluates the condition. The return value is based on the condition. |
| IFERROR(<br>value,<br>value_if_error<br>) | Evaluates an expression and returns a specified value if the expression returns an error. Otherwise, returns the value of the expression itself. |
| NOT(logical_value) | Changes **FALSE** to **TRUE**, or **TRUE** to **FALSE**. |
| OR(<br>logical_value,<br>logical_value<br>) | Checks whether one of the arguments is **TRUE** to return **TRUE**. The function returns **FALSE** if both arguments are **FALSE**.<br>If you have more than two arguments, use \|\| (double pipe) as an alternative and you can have as many "\|\|" as you like. |

| Function | Notes |
| --- | --- |
| SWITCH(<br>Expression,<br>value1, expression1<br>[, value2, expression2] …<br>[, else, expression]<br>) | **Expression** is evaluated and the result is matched with the given values. If a match is found, the corresponding expression is evaluated.<br>If the result is not matched with any of the given values, and else is given, the corresponding expression is evaluated.<br>All expressions must be of the same data type. |
| TRUE() | Returns the logical value **TRUE**. |

## DAX Math and Trig Functions

DAX Math and Trig Functions are similar to Excel mathematical and trigonometric functions.

| Function | Notes |
|---|---|
| ABS(number) | Removes the negative sign if it exists. |
| ACOS(number) | 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. |
| ACOSH(number) | 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*. |
| ASIN(number) | 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. |
| ASINH(number) | 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. |
| ATAN(number) | 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. |
| ATANH(number) | 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. |
| CEILING(<br>number,<br>significance<br>) | Rounds a number up, to the nearest integer or to the nearest multiple of significance. |
| COMBIN(<br>number,<br>number_chosen<br>) | Returns the number of combinations for a given number of items.<br>Numeric arguments are truncated to integers.<br>If either argument is nonnumeric, **COMBIN** returns the #VALUE! error value.<br>If number<0, number_chosen<0, or number<number_chosen, **COMBIN** returns the #NUM! error value. |

| Function | Notes |
|---|---|
| COMBINA(<br>number,<br>number_chosen<br>) | Returns the number of combinations (with repetitions) for a given number of items.<br>• number must be greater than or equal to 0, and greater than or equal to number_chosen.<br>• number_chosen must be greater than or equal to 0.<br>If the value of either argument is outside of it's constraints, **COMBINA** returns the #NUM! error value.<br>If either argument is a non-numeric value, **COMBINA** returns the #VALUE! error value.<br>Non-integer values are truncated. |
| COS(number) | Returns the cosine of the given angle. |
| COSH(number) | Returns the hyperbolic cosine of a number. |
| CURRENCY(value) | The value of the expression evaluated and returned as a currency type value. |
| DEGREES(angle) | Converts radians into degrees. |
| DIVIDE(<br>numerator,<br>denominator<br>[, alternate-result]<br>) | Safe divide function that gracefully handles a divide by zero error.<br>Performs division and returns alternate result or **BLANK**() on division by 0. |
| EVEN(number) | Returns number rounded up to the nearest even integer. |
| EXP(number) | Returns e raised to the power of a given number. The constant e equals 2.71828182845904, the base of the natural logarithm. |
| FACT(number) | Returns the factorial of a number, equal to<br>1*2*3*...*number*..<br>*number* should be a non-negative integer. |
| FLOOR(<br>number,<br>significance<br>) | Rounds a number down, toward zero, to the nearest multiple of significance. |
| GCD(<br>number1<br>[, number2] …<br>) | 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.<br>If only one number is given, the function returns the number itself. |
| INT(number) | Rounds a number down to the nearest integer. |

| Function | Notes |
|---|---|
| ISO.CEILING(<br>number<br>[, significance]<br>) | Rounds number up, to the nearest multiple of significance.<br>Rounds number up, to the nearest integer if significance is omitted. |
| LCM(<br>number1<br>[, number2] …<br>) | 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. |
| LN(number) | Returns the natural logarithm of a number.<br>Natural logarithms are based on the constant e (2.71828182845904). |
| LOG(<br>number,<br>[base]<br>) | Returns the logarithm of a number to the base you specify.<br>If base is omitted, it is taken as 10.<br>You might receive an error if the value is too large to be displayed. |
| LOG10(number) | Returns the base-10 logarithm of a number. |
| MOD(<br>number,<br>divisor<br>) | Returns the remainder after a number is divided by a divisor. The result always has the same sign as the divisor.<br>If the divisor is 0 (zero), **MOD** returns an error. You cannot divide by 0. |
| MROUND(<br>number,<br>multiple<br>) | Returns a number rounded to the desired multiple. |
| ODD(number) | Returns number rounded up to the nearest odd integer. |
| PERMUT(<br>number,<br>number_chosen<br>) | Returns the number of permutations for a given number of objects that can be selected from number objects.<br>A permutation is any set or subset of objects or events where internal order is significant. |
| PI() | Returns the value of Pi, 3.14159265358979, accurate to 15 digits. |
| POWER(<br>number,<br>power<br>) | Returns the result of a number raised to a power. |

| Function | Notes |
|---|---|
| QUOTIENT(<br>numerator,<br>denominator<br>) | Performs division and returns only the integer portion of the division result. Use this function when you want to discard the remainder of division. |
| RADIANS(angle) | Converts degrees to radians. |
| RAND() | 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. |
| RANDBETWEEN(<br>bottom,<br>top<br>) | Returns a random number in the range between two numbers you specify. |
| ROUND(<br>number,<br>num_digits<br>) | Rounds a number to the specified number of digits. |
| ROUNDDOWN(<br>number,<br>num_digits<br>) | Rounds a number down, toward zero. |
| ROUNDUP(<br>number,<br>num_digits<br>) | Rounds a number up, away from 0 (zero). |
| SIGN(number) | 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. |
| SIN(number) | Returns the sine of the given angle. |
| SINH(number) | Returns the hyperbolic sine of a number. |
| SQRT(number) | Returns the square root of a number.<br>If the number is negative, the SQRT function returns an error. |
| SQRTPI(number) | Returns the square root of (number * pi). |
| TAN(number) | Returns the tangent of the given angle. |
| TANH(number) | Returns the hyperbolic tangent of a number. |

| Function | Notes |
| --- | --- |
| TRUNC(<br>number,<br>num_digits<br>) | Truncates a number to an integer by removing the decimal, or fractional, part of the number.<br>**num_digits** specifies the precision of the truncation; if omitted, 0 (zero). |

## DAX Other Functions

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

| Function | Notes |
|---|---|
| CONVERT(<br>Expression,<br>Datatype<br>) | Converts an expression of one data type to another.<br>Datatype can be any of the following.<br>• INTEGER (Whole Number)<br>• DOUBLE (Decimal Number)<br>• STRING (Text)<br>• BOOLEAN (True/False)<br>• CURRENCY (Fixed Decimal Number)<br>• DATETIME (Date, Time, etc.) |
| ERROR(text) | Raises an error with an error message. |
| VAR VarName = Expression | 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.<br>VarName is the name of the variable (identifier).<br>• Supported character set: a-z, A-Z, 0-9.<br>• 0-9 are not valid as first character.<br>• __ (double underscore) is allowed as a prefix. No other special characters are supported.<br>• Delimiters are not supported. For example, 'VarName' or [VarName] will result in an error.<br>• Reserved keywords not allowed.<br>• Names of existing tables are not allowed.<br>• Empty spaces are not allowed.<br>Expression is a DAX expression which returns a scalar or table value.<br>• Expression can contain another VAR declaration.<br>When referencing a variable:<br>• Measures cannot refer to variables defined outside the measure expression, but can refer to functional scope variables defined within the expression.<br>• Variables can refer to measures.<br>• Variables can refer to previously defined variables.<br>Columns in table variables cannot be referenced via TableName[ColumnName] syntax. |

## DAX Other Special Functions (X-Functions / Iterators)

These functions perform specific actions that complement the other DAX functions.

**DAX Iterator Functions**, called **iterators** for short, take a column or a table as the argument and aggregate the values just as aggregation functions—but using a different approach. These functions aggregate the values in a row context.

These are **"X-functions"** (i.e., any function that has an X on the end of the name). The iterators given below also include statistical iterator functions.

DAX also has **two financial functions** that got added in Excel 2016.

| Function | Notes |
|---|---|
| AVERAGEX(<br>table,<br>expression<br>) | Calculates the average (arithmetic mean) of a set of expressions evaluated over a table. |
| CONCATENATEX(<br>table,<br>expression<br>[, delimiter]<br>) | Concatenates the result of an expression evaluated for each row in a table. |
| COUNTAX(<br>table,<br>expression<br>) | The **COUNTAX** function counts nonblank results when evaluating the result of an expression over a table.<br>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 result in a nonblank result. |
| COUNTX(<br>table,<br>expression<br>) | Counts the number of rows that contain a number or an expression that evaluates to a number, when evaluating an expression over a table. |
| GEOMEANX(<br>table,<br>expression<br>) | Returns the geometric mean of an expression evaluated for each row in a table. |
| MAXX(<br>table,<br>expression<br>) | Returns the largest numeric value that results from evaluating an expression for each row of a table.<br>• The first argument can be a table name, or an expression that evaluates to a table.<br>• The second argument indicates the expression to be evaluated for each row of the table. |

| Function | Notes |
|---|---|
| MEDIANX(<br>table,<br>expression<br>) | Returns the median of an expression evaluated for each row in a table. |
| MINX(<br>table,<br>expression<br>) | Returns the smallest numeric value that results from evaluating an expression for each row of a table.<br>• The first argument can be a table name, or an expression that evaluates to a table.<br>• The second argument indicates the expression to be evaluated for each row of the table. |
| PERCENTILEX.EXC(<br>table,<br>expression,<br>k<br>) | Returns the percentile number of an expression evaluated for each row in a table. |
| PERCENTILEX.INC(<br>table,<br>expression,<br>k<br>) | Returns the percentile number of an expression evaluated for each row in a table. |
| PRODUCTX(<br>table,<br>expression<br>) | Returns the product of an expression evaluated for each row in a table. |
| RANKX(<br>table,<br>expression,<br>[value],<br>[order],<br>[ties]<br>) | Returns the ranking of a number in a list of numbers for each row in the table argument. |
| STDEVX.P(<br>table,<br>expression<br>) | Returns the standard deviation of the entire population.<br>**expression** is any DAX expression that returns a single scalar value, where the expression is to be evaluated multiple times (for each row/context). |

| Function | Notes |
| --- | --- |
| SUMX(<br>table,<br>expression<br>) | Returns the sum of an expression evaluated for each row in a table. |
| VARX.P(<br>table,<br>expression<br>) | Returns the variance of the entire population. |
| VARX.S(<br>table,<br>expression<br>) | Returns the variance of a sample population. |

## DAX Other Special Functions (Argument Functions)

In DAX, there are also some special functions that have a very specific purpose of usability in other DAX functions only. i.e. they can be used only as arguments to certain DAX functions. These functions cannot be used as standalone functions.

| Function | Notes |
|---|---|
| CURRENTGROUP() | **CURRENTGROUP** can only be used in an expression that defines a column within the GROUPBY function.<br><br>In-effect, **CURRENTGROUP** returns a set of rows from the "table" argument of GROUPBY that belong to the current row of the GROUPBY result.<br><br>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**. |
| IGNORE(expression) | **IGNORE** function does not return a value.<br><br>**IGNORE** can be used as an expression argument to **SUMMARIZECOLUMNS** function. |
| NONVISUAL(expression) | Returns a table of values and used with **SUMMARIZECOLUMNS**.<br><br>Marks a value filter in **SUMMARIZECOLUMNS** function as not affecting measure values, but only applying to group-by columns. |
| ROLLUPADDISSUBTOTAL(<br>groupBy_columnName,<br>isSubtotal_columnName,<br>[groupBy_columnName,<br>isSubtotal_columnName], …<br>) | **ROLLUPADDISSUBTOTAL** function does not return a value. It only specifies the set of columns to be subtotalled.<br><br>**ROLLUPADDISSUBTOTAL** is used with **SUMMARIZECOLUMNS** function. |
| ROLLUPGROUP(<br>groupBy_columnName,<br>groupBy_columnName<br>) | **ROLLUPGROUP** can only be used as a groupBy_columnName argument to the **ROLLUPADDISSUBTOTAL** and / or the **SUMMARIZE** functions.<br><br>**ROLLUPGROUP** is used inside the **ROLLUPISSUBTOTAL** function to reflect ROLLUPGROUPs present in the supplied table argument. |

| Function | Notes |
| --- | --- |
| ROLLUPISSUBTOTAL(<br>groupingColumn,<br>isSubtotal_columnName,<br>[groupingColumn,<br>isSubtotal_columnName], …<br>) | **ROLLUPISSUBTOTAL** is used to define the supplied table argument to the **ADDMISSINGITEMS** function.<br>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. |

## DAX Parent and Child Functions

DAX Parent and Child functions help to manage data that is presented as a parent/child hierarchy in the data model.

For more information read. Understanding Functions for Parent-Child Hierarchies in DAX.

| Function | Notes |
|---|---|
| PATH(<br>ID_columnName,<br>parent_columnName<br>) | Returns a delimited text string with the identifiers of all the parents of the current identifier, starting with the oldest and continuing until current. |
| PATHCONTAINS(<br>path,<br>item<br>) | Returns **TRUE** if the specified ***item*** exists within the specified ***path*****.** |
| PATHITEM(<br>path,<br>position,<br>[type]<br>) | Returns the item at the specified ***position*** from a string resulting from evaluation of a PATH function. Positions are counted from left to right. |
| PATHITEMREVERSE(<br>path,<br>position,<br>[type]<br>) | 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. |
| PATHLENGTH(path) | Returns the number of parents to the specified item in a given **PATH** result, including self. |

## DAX Query Functions

These DAX functions are helpful in writing queries in DAX. These functions return tables that can be used in other DAX functions as input. The resulting tables are virtual and are not materialised.

DAX Studio is a great way to learn about table functions because you can "see" the tables materialised on the screen. Read about DAX Studio as a tool to query your data model here. http://exceleratorbi.com.au/getting-started-dax-studio/.

| Function | Notes |
|---|---|
| ADDCOLUMNS(<br>table,<br>name, expression<br>[, name, expression]<br>…<br>) | Adds calculated columns to the given table or table expression. |
| CROSSJOIN(<br>table,<br>table<br>[, table]<br>…<br>) | 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. |
| DATATABLE(<br>ColumnName1,<br>DataType1,<br>ColumnName2,<br>DataType2, ...,<br>{{Value1, Value2 …},<br>{Value1, Value2 …}, …<br>}<br>) | Returns a table declaring an inline set of values.<br>Each of the columns is given a name and the data type of the column is provided.<br>Then, the set of values is given –<br>{{Row 1 values},<br>{Row 2 values},<br>…<br>} |
| EXCEPT(<br>table_expression1,<br>table_expression2<br>) | Returns a table that contains the rows of one table minus all the rows of another table. |
| GENERATE(<br>table1,<br>table2<br>) | 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. |

| Function | Notes |
|---|---|
| GENERATEALL(<br>table1,<br>table2<br>) | 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. |
| GROUPBY(<br>table,<br>[<groupBy_columnName1>],<br>[<name,<br>expression>]<br>…<br>) | Returns a table with the selected columns for the groupBy_columnName arguments and the grouped by columns designated by the name arguments.<br><br>The **GROUPBY** function is similar to the **SUMMARIZE** function. However, **GROUPBY** does not do an implicit **CALCULATE** and the group isn't placed into the filter context. |
| GENERATESERIES(<br>StartValue,<br>EndValue<br>[, IncrementValue]<br>) | Generates a table of values using the parameters provided |
| INTERSECT(table, table) | Returns the row intersection of two tables, retaining duplicates. |
| NATURALINNERJOIN(<br>leftJoinTable,<br>rightJoinTable<br>) | Performs an inner join of a table with another table.<br>The tables are joined on common columns (by name) in the two tables.<br>If the two tables have no common column names, an error is returned. |
| NATURALLEFTOUTERJOIN(<br>leftJoinTable,<br>rightJoinTable<br>) | Performs an inner join of a table with another table.<br>The tables are joined on common columns (by name) in the two tables.<br>If the two tables have no common column names, an error is returned. |
| ROW(<br>name, expression<br>[, name, expression]<br>…<br>) | Returns a table with a single row containing values that result from the expressions given to each column. |

| Function | Notes |
|---|---|
| SELECTCOLUMNS(<br>table,<br>name, scalar_expression<br>[, name, scalar_expression]<br>…<br>) | Adds calculated columns to the given table or table expression.<br>**SELECTCOLUMNS** starts with an empty table and that is the only difference between **ADDCOLUMNS** and **SELECTCOLUMNS**. |
| SUMMARIZE(<br>table,<br>groupBy_columnName<br>[, groupBy_columnName]<br>…,<br>name, expression,<br>[name, expression]<br>…<br>) | Returns a table of values for use in a query or inside a formula that uses a table. If there are relationships between tables, always specify the table on the many side of the relationship as the table parameter. This function is semantically similar to "group by" in SQL. |
| SUMMARIZECOLUMNS(<br>groupBy_columnName<br>[, groupBy_columnName]<br>…,<br>[filterTable] …<br>[, name, expression] …<br>) | Returns a summary table over a set of groups. A table which includes combinations of values from the supplied columns, based on the grouping specified.<br>Only rows for which at least one of the supplied expressions return a non-blank value are included in the table returned.<br>If all expressions evaluate to BLANK/NULL for a row, that row is not included in the table returned.<br>A column cannot be specified more than once in the groupBy_columnName. |

| Function | Notes |
|---|---|
| Table Constructor<br>{scalar_expression1,<br>scalar_expression2, …<br>}<br><br>{(scalar_expression11,<br>scalar_expression12, …,<br>scalar_expression1N<br>),<br>(scalar_expression21),<br>scalar_expression22, …,<br>scalar_expression2N<br>),<br>…<br>} | Returns a table of one or more columns.<br>• When there is only one column, the name of the column is Value.<br>• When there are N columns where N > 1, the names of the columns from left to right are Value1, Value2, …, ValueN.<br>**Note:**<br>• The number of scalar expressions must be the same for all rows.<br>• When the data types of the values for a column are different in different rows, all values are converted to a common data type. |
| UNION(<br>table_expression1,<br>table_expression2<br>) | Creates a union (join) table from a pair of tables.<br>Returns a table that contains all the rows from each of the two table expressions.<br>The two tables must have the same number of columns.<br>Columns are combined by position in their respective tables.<br>The column names in the return table will match the column names in table_expression1.<br>Duplicate rows are retained.<br>The returned table has lineage where possible. When data types differ, the resulting data type is determined based on the rules for data type coercion.<br>The returned table will not contain columns from related tables. |

## DAX Statistical Functions

Following are the DAX Statistical Functions:

| Function | Notes |
|---|---|
| BETA.DIST(<br>x,<br>alpha,<br>beta,<br>cumulative<br>[, A]<br>[, B]<br>) | 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. |
| BETA.INV(<br>probability,<br>alpha,<br>beta<br>[, A]<br>[, B]<br>) | Returns the inverse of the beta cumulative probability density function (BETA.DIST).<br>If<br>probability = BETA.DIST(x, ...TRUE),<br>then<br>BETA.INV(probability, ...) = x. |
| CHISQ.INV(<br>probability,<br>deg_freedom<br>) | Returns the inverse of the left-tailed probability of the chi-squared distribution.<br>The chi-squared distribution is commonly used to study variation in the percentage of something across samples. |
| CHISQ.INV.RT(<br>probability,<br>deg_freedom<br>) | Returns the inverse of the right-tailed probability of the chi-squared distribution.<br>If<br>probability = CHISQ.DIST.RT(x,...),<br>then<br>CHISQ.INV.RT(probability,...) = x.<br>Use this function to compare observed results with expected ones in order to decide whether your original hypothesis is valid. |
| CONFIDENCE.NORM(<br>alpha,<br>standard_dev,<br>size<br>) | The confidence interval is a range of values.<br>Your sample mean, x, is at the center of this range and the range is x ± CONFIDENCE.NORM. |

| Function | Notes |
|---|---|
| CONFIDENCE.T(<br>alpha,<br>standard_dev,<br>size<br>) | Returns the confidence interval for a population mean, using a Student's t distribution. |
| EXPON.DIST(<br>x,<br>lambda,<br>cumulative<br>) | Returns the exponential distribution. Use EXPON.DIST to model the time between events. |
| GEOMEAN(column) | Returns the geometric mean of the numbers in a column. |
| MEDIAN(column) | Returns the median of numbers in a column. |
| NORM.DIST(X, Mean, Standard_dev, Cumulative) | Returns the normal distribution for the specified mean and standard deviation. |
| NORM.INV(Probability, Mean, Standard_dev) | Returns the inverse of the normal cumulative distribution for the specified mean and standard deviation. |
| NORM.S.DIST(Z, Cumulative) | Returns the standard normal distribution (has a mean of zero and a standard deviation of one). |
| NORM.S.INV(Probability) | Returns the inverse of the standard normal cumulative distribution. The distribution has a mean of zero and a standard deviation of one. |
| PERCENTILE.EXC(<br>column,<br>k<br>) | Returns the k-th percentile of values in a range, where k is in the range 0..1, exclusive. |
| PERCENTILE.INC(<br>column,<br>k<br>) | Returns the k-th percentile of values in a range, where k is in the range 0..1, inclusive. |
| POISSON.DIST(<br>x,<br>mean,<br>cumulative<br>) | Returns the Poisson distribution.<br>A common application of the Poisson distribution is predicting the number of events over a specific time. |

| Function | Notes |
|---|---|
| RANK.EQ(<br>value,<br>columnName,<br>[order]<br>) | Returns the ranking of a number in a list of numbers. |
| SAMPLE(<br>n_value,<br>table,<br>orderBy_expression,<br>[order]<br>[, orderBy_expression, [order], …]<br>) | Returns a sample of N rows from the specified table. |
| STDEV.P(ColumnName) | Returns the standard deviation of the entire population. |
| STDEV.S(ColumnName) | Returns the standard deviation of a sample population. |
| T.DIST(X,Deg_freedom,Cumulative) | Returns the Student's left-tailed t-distribution. |
| T.DIST.2T(X,Deg_freedom) | Returns the two-tailed Student's t-distribution. |
| T.DIST.RT(X,Deg_freedom) | Returns the right-tailed Student's t-distribution. |
| T.INV(Probability,Deg_freedom) | Returns the left-tailed inverse of the Student's t-distribution. |
| T.INV.2T(Probability,Deg_freedom) | Returns the two-tailed inverse of the Student's t-distribution. |
| VAR.P(ColumnName) | Returns the variance of the entire population. |
| VAR.S(ColumnName) | Returns the variance of a sample population. |

## DAX Text Functions

DAX Text Functions are based on the Excel string functions, but have been modified to work with tables and columns.

| Function | Notes |
|---|---|
| BLANK() | Returns a blank.<br>Blanks are not equivalent to nulls. DAX uses blanks for both database nulls and for blank cells in Excel. |
| CODE(text) | Returns a numeric code for the first character in a text string. The returned code corresponds to the character set used by your computer. |
| COMBINEVALUES(<br>delimiter,<br>expression, expression<br>[,expression] …<br>) | Returns the concatenated string where the values of the DAX expressions are joined by the delimiter (which is a constant). |
| CONCATENATE(<br>text1,<br>text2<br>) | Joins two text strings into one text string.<br>If you need to add more than two arguments, use the AND (&) operator. |
| CONTAINSSTRING(<br>within_text,<br>find_text<br>) | Returns **TRUE** or **FALSE** indicating whether one string contains another string.<br>• CONTAINSSTRING is not case-sensitive.<br>• You can use ? and * wildcard characters. Use ~ to escape wildcard characters. |
| CONTAINSSTRINGEXACT(<br>within_text,<br>find_text<br>) | Returns **TRUE** or **FALSE** indicating whether one string contains another string.<br>• CONTAINSSTRING is case-sensitive. |
| FIND(<br>find_text,<br>within_text<br>[, start_num]<br>[, NotFoundValue]<br>) | Returns the starting position of one text string within another text string. **FIND** is case-sensitive. |

| Function | Notes |
|---|---|
| FIXED(<br>number,<br>decimals<br>[, no_commas]<br>) | 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. |
| FORMAT(<br>value,<br>format_string<br>[, <locale_name>]<br>) | Converts a value to text according to the specified format.<br>locale_name is optional and is the name of the locale to be used by the function. Possible values are strings accepted by the Windows API function LocaleNameToLCID(). |
| LEFT(<br>text<br>[,num_chars]<br>) | Returns the specified number of characters from the start of a text string. |
| LEN(text) | Returns the number of characters in a text string. |
| LOWER(text) | Converts all letters in a text string to lowercase. |
| MID(<br>text,<br>start_num,<br>num_chars<br>) | Returns a string of characters from the middle of a text string, given a starting position and length. |
| REPLACE(<br>old_text,<br>start_num,<br>num_chars,<br>new_text<br>) | **REPLACE** replaces part of a text string, based on the number of characters you specify, with a different text string. |
| REPT(<br>text,<br>num_times<br>) | Repeats text a given number of times. Use **REPT** to fill a cell with a number of instances of a text string. |
| RIGHT(<br>text,<br>[num_chars]<br>) | **RIGHT** returns the last character or characters in a text string, based on the number of characters you specify. |

| Function | Notes |
|---|---|
| SEARCH(<br>find_text,<br>within_text<br>[, start_num],<br>[, NotFoundValue]<br>) | Returns the number of the character at which a specific character or text string is first found, reading left to right.<br>**SEARCH** function is case insensitive and accent sensitive. |
| SUBSTITUTE(<br>text,<br>old_text,<br>new_text,<br>instance_num<br>) | Replaces existing text with new text in a text string. |
| TRIM(text) | Removes all spaces from text except for single spaces between words. |
| UNICHAR(number) | Returns the Unicode character referenced by the numeric value. |
| UPPER(text) | Converts a text string to all uppercase letters |
| VALUE(text) | You need not use the **VALUE** function in a formula because Power Pivot implicitly converts text to numbers.<br>The argument text can be a constant or in one of the formats - number, date or time format. Otherwise, an error is returned.<br>You can use a column reference as argument to **VALUE** function. E.g., 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. |

## DAX Time Intelligence Functions

DAX Time Intelligence Functions enable you to manipulate data using time periods, including days, months, quarters and years, and then build and compare calculations over those periods.

There are 3 classes of DAX Time Intelligence functions based on their return value.

- Those that return scalar values (e.g. TOTALYTD). They are stand-alone functions and so can be used as a stand-alone measure/function.
- Those that return a table containing a single column and single row with a date value (e.g. FIRSTNONBLANK). They can be used either as stand-alone functions to define a measure (as a scalar value) or as an argument to any function that requires a table in its arguments. These types of functions therefore behave as both a scalar value and a table, depending on the use case.
- Those that return a table of dates (e.g. DATESYTD) and are designed to be used inside a CALCULATE function.

**DAX Time Intelligence Functions that return Scalar Values**

| Function | Notes |
|---|---|
| CLOSINGBALANCEMONTH(<br>expression,<br>dates,<br>[filter]<br>) | Returns a scalar value that represents the expression evaluated at the last date of the month in the current context.<br>E.g. Use to create a measure that calculates the 'Month End Inventory Value' of the product inventory. |
| CLOSINGBALANCEQUARTER(<br>expression,<br>dates,<br>[filter]<br>) | Returns a scalar value that represents the expression evaluated at the last date of the month in the current context.<br>E.g. Use to create a measure that calculates the 'Quarter End Inventory Value' of the product inventory. |
| CLOSINGBALANCEYEAR(<br>expression,<br>dates,<br>[filter],<br>[year_end_date]<br>) | Returns a scalar value that represents the **expression** at the last date of the year in the current context.<br>E.g. Use to create a measure that calculates the **'Year End Inventory Value'** of the product inventory. |
| OPENINGBALANCEMONTH(<br>expression,<br>dates<br>[, filter]<br>) | Returns a scalar value that represents the expression evaluated at the first date of the month in the current context.<br>E.g. You can create a measure that calculates the **'Month Start Inventory Value'** of the product inventory. |

| Function | Notes |
|---|---|
| OPENINGBALANCEQUARTER(<br>expression,<br>dates<br>[, filter]<br>) | Returns a scalar value that represents the expression evaluated at the first date of the quarter in the current context.<br>E.g. You can create a measure that calculates the '**Quarter Start Inventory Value**' of the product inventory. |
| OPENINGBALANCEYEAR(<br>expression,<br>dates<br>[, filter]<br>) | Returns a scalar value that represents the **expression** evaluated at the first date of the year in the current context.<br>E.g. You can create a measure that calculates the '**Year Start Inventory Value**' of the product inventory. |
| TOTALMTD(<br>expression,<br>dates<br>[, filter]<br>) | Returns a scalar value that represents the expression evaluated for the dates in the current month-to-date, given the dates in dates. |
| TOTALQTD(<br>expression,<br>dates<br>[, filter]<br>) | Returns a scalar value that represents the **expression** evaluated for the dates in the current quarter-to-date, given the dates in **dates**. |
| TOTALYTD(<br>expression,<br>dates<br>[, filter]<br>[, year_end_date]<br>) | Returns a scalar value that represents the **expression** evaluated for the dates in the current year-to-date, given the dates in **dates**. |

**DAX Time Intelligence Functions that return both a Table and a Scalar**

These functions return a table containing a single column and single row with a date value (can be used as a scalar value or a table input to another function).

| | |
|---|---|
| ENDOFMONTH(dates) | Returns a table containing a single column and single row with the last date of the month in the current context for the specified column of dates.<br><br>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. |
| ENDOFQUARTER(dates) | Returns a table containing a single column and single row with the last date of the quarter in the current context for the specified column of dates.<br><br>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. |
| ENDOFYEAR(<br>dates<br>[, year_end_date]<br>) | Returns a table containing a single column and single row with the last date of the year in the current context for the specified column of dates.<br><br>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. |
| FIRSTDATE(dates) | Returns a table containing a single column and single row with the first date in the current context for the specified column of dates.<br><br>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.<br><br>When the current context is a single date, the date returned by the **FIRSTDATE** and **LASTDATE** functions will be equal. |
| FIRSTNONBLANK(<br>column,<br>expression<br>) | Returns a table containing a single column and single row with the first value in the column filtered by the current context, where the expression is not blank.<br><br>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.<br><br>E.g. You could get the first value for which there were sales of a product. |
| FIRSTNONBLANKVALUE(<br>column,<br>expression<br>) | Evaluates an expression filtered by the sorted values of a column and returns the first value of the expression that is not blank.<br><br>This function is different from FIRSTNONBLANK in that the column is added to the filter context for the evaluation of expression. |
| LASTDATE(dates) | Returns a table containing a single column and single row with the last date in the current context for the specified column of dates.<br><br>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. |

| | |
|---|---|
| | When the current context is a single date, the date returned by the **FIRSTDATE** and **LASTDATE** functions will be equal. |
| LASTNONBLANK(<br>column,<br>expression<br>) | Returns a table containing a single column and single row with the last value in the column, filtered by the current context, where the expression is not blank.<br>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.<br>E.g. You could get the last value for which there were sales of a product. |
| LASTNONBLANKVALUE(<br>column,<br>expression<br>) | Evaluates an expression filtered by the sorted values of a column and returns the last value of the expression that is not blank.<br>This function is different from LASTNONBLANK in that the column is added to the filter context for the evaluation of expression. |
| STARTOFMONTH(dates) | Returns a table containing a single column and single row with the first date of the month in the current context for the specified column of dates.<br>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. |
| STARTOFQUARTER(dates) | Returns a table containing a single column and single row with the first date of the quarter in the current context for the specified column of dates.<br>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. |
| STARTOFYEAR(<br>Dates,<br>[year end date]<br>) | Returns a table containing a single column and single row with the first date of the year in the current context for the specified column of dates.<br>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. |

**DAX Time Intelligence Functions that return a Table of Dates**

| | |
|---|---|
| DATEADD(<br>dates,<br>number_of_intervals,<br>interval<br>) | 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.<br>Designed to be used inside a CALCULATE as a table filter.<br>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.<br><br>The result table includes only dates that exist in the **dates** column.<br><br>If the dates in the current context do not form a contiguous interval, the function returns an error. |
| DATESBETWEEN(<br>dates,<br>start_date,<br>end_date<br>) | Returns a table that contains a column of dates that begins with the **start_date** and continues until the **end_date**.<br><br>Designed to be used inside a CALCULATE as a table filter.<br><br>The **DATESBETWEEN** function is provided for working with custom date ranges.<br><br>If you are working with common date intervals such as months, quarters, and years, the more appropriate function is **DATESINPERIOD**. |
| DATESINPERIOD(<br>dates,<br>start_date,<br>number_of_intervals,<br>interval<br>) | Returns a table that contains a column of dates that begins with the **start_date** and continues for the specified **number_of_intervals**.<br><br>Designed to be used inside a CALCULATE as a table filter.<br><br>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.<br><br>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.<br><br>The result table includes only dates that appear in the values of the underlying table column.<br><br>This function can be used as an input to **CALCULATE** to create a rolling total such as Moving Annual Total (**MAT**), rolling 90 days, rolling 90 day average etc. |
| DATESMTD(dates) | Returns a table that contains a column of the dates for the month to date, in the current context.<br><br>Designed to be used inside a CALCULATE as a table filter. |
| DATESQTD(dates) | Returns a table that contains a column of the dates for the quarter to date, in the current context.<br><br>Designed to be used inside a CALCULATE as a table filter. |
| DATESYTD(<br>dates<br>[, year_end_date]<br>) | Returns a table that contains a column of the dates for the year to date, in the current context.<br><br>Designed to be used inside a CALCULATE as a table filter.<br><br>**year_end_date** (optional) is a literal string with a date that defines the year-end date. The default is December 31. |

| | |
|---|---|
| NEXTDAY(dates) | 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.<br>Designed to be used inside a CALCULATE as a table filter.<br>E.g. You can create a measure that calculates the **'next day sales'** of the product sales. |
| NEXTMONTH(dates) | 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.<br>Designed to be used inside a CALCULATE as a table filter.<br>E.g. You can create a measure that calculates the **'next month sales'** of the product sales. |
| NEXTQUARTER(dates) | 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.<br>Designed to be used inside a CALCULATE as a table filter.<br>E.g. You can create a measure that calculates the **'next quarter sales'** of the product sales. |
| NEXTYEAR(<br>dates<br>[, year_end_date]<br>) | 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.<br>Designed to be used inside a CALCULATE as a table filter.<br>E.g. You can create a measure that calculates the **'next year sales**' of the product sales. |

| | |
|---|---|
| PARALLELPERIOD(<br>dates,<br>number_of_intervals,<br>interval<br>) | 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.<br>Designed to be used inside a CALCULATE as a table filter.<br>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.<br>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.<br>The result table includes only dates that appear in the values of the underlying table column.<br>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.<br>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. |
| PREVIOUSDAY(dates) | 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.<br>Designed to be used inside a CALCULATE as a table filter.<br>E.g. You can create a measure '**Previous Day Sales**'. |
| PREVIOUSMONTH(dates) | 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.<br>Designed to be used inside a CALCULATE as a table filter.<br>E.g. You can create a measure '**Previous Month Sales**'. |
| PREVIOUSQUARTER(dates) | 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.<br>Designed to be used inside a CALCULATE as a table filter.<br>E.g. You can create a measure '**Previous Quarter Sales**'. |

| | |
|---|---|
| PREVIOUSYEAR(<br>dates<br>[, year_end_date]<br>) | 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.<br>Designed to be used inside a CALCULATE as a table filter.<br>E.g. You can create a measure '**Previous Year Sales**'. |
| SAMEPERIODLASTYEAR(dates) | 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.<br>Designed to be used inside a CALCULATE as a table filter.<br>The dates returned are the same as the dates returned by this equivalent formula:<br>DATEADD(dates, -1, year) |