IT Real-Time training that work for your career.

PROVIDED TRAINING FOR THOUSANDS OF STUDENTS

SUBJECT, MATERIAL & VIDEOS

VIDEOS PROVIDED

POWER BI

- Introduction to DAX
- DAX Naming Convention and Naming methods
- DAX Parameter naming convention
- DAX Operators and DAX Data Types
- DAX different types of functions

MSBI

IS, AS, RS & MDS

POWER BI

SERVER, DESKTOP & DAX

Trainings:

CLASS ROOM

ONLINE

FAST TRACK

ONE ON ONE

PROJECT TRAINING

Address:

Flat No: 506/B
Nilgiri Block
Aditya Enclave
Mytrivanam Area
Hyderabad.

Website & Blog

www.vinaytechhouse.com

www.msbivinay.blogspot.in

Contact Information

+91 9573168449

040 66638869

WE'VE WORKED WITH A DIVERSE CUSTOMER BASE. HOW CAN WE HELP YOU?

IT Training, Support and Consulting.

DATA ANALYSIS EXPRESSIONS [DAX]

Introduction

What is DAX?and what kind of language it is?

DAX stands for **D**ata **A**nalysis E**x**pressions. DAX is a **formula language** and is a collection of functions, operators, and constants that can be used in a formula or expression to calculate and return one or more values. DAX is the formula language associated with the Data Model of Microsoft Excel Power Pivot and with Microsoft Power BI.

DAX is **a *functional language***, which means the **full executed code is contained inside a function**

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

Who developed DAX and history of DAX?

DAX was developed by the SQL Server Analysis Services team at Microsoft as part of Project Gemini and released in 2009 with the first version of the PowerPivot for Excel 2010 Add-in. Both DAX and MDX can be used to query PowerPivot and Tabular models, however only MDX may be used to query multidimensional SSAS models (cubes) in versions of SSAS up to SQL Server 2012 RTM.

DAX is **not a programming language,** however it is a formula language that allows the users to define custom calculations in calculated columns and calculated fields (also known as measures). DAX helps you create new information from the existing data in your Data Model.

Why do we require DAX?

DAX formulas enable you to perform **data modeling, data analysis, and use the results for reporting** and decision making.

What is DAX Function?

A DAX function is an **inbuilt function** provided in the DAX language to enable you to perform various actions on the data in the tables in your Data Model. DAX functions play an important role in the usage of DAX for data modeling and reporting.

DAX functions enable you to perform commonly used data calculations on the Data Model. Some of the DAX functions have same names and functionality as that of Excel functions but have been modified to use DAX data types and to work with tables and columns, as highlighted in the next section. DAX has additional functions that are designed to work with relational data and perform dynamic aggregation.

What are the differences between Excel Functions and DAX Functions?

There are certain similarities between the Excel functions and the DAX functions and there are certain differences too. Following are the similarities and differences between Excel functions and DAX functions −

Similarities Between Excel Functions and DAX Functions

- Certain DAX functions have the same name and the same general behavior as Excel functions.
- DAX has lookup functions that are similar to the array and vector lookup functions in Excel.

Differences Between Excel Functions and DAX Functions

- DAX functions have been modified to take different types of inputs and some of the DAX functions might return a different data type. Hence, you need to understand the usage of these functions separately though they have the same name.
- You cannot use DAX functions in an Excel formula or use Excel functions in DAX formula, without the required modifications.
- **Excel functions take a cell reference or a range of cells as a reference. DAX functions never take a cell reference or a range of cells as a reference, but instead take a column or table as a reference.**
- Excel date and time functions return an integer that represents a date as a serial number. DAX date and time functions return a datetime data type that is in DAX but not in Excel.
- Excel has no functions that return a table, but some functions can work with arrays. Many of the DAX functions can easily reference complete tables and columns to perform calculations and return a table or a column of values. This ability of DAX adds power to the Power Pivot, Power View and Power BI, where DAX is used.
- DAX lookup functions require that a relationship is established between the respective tables.

DAX Data Types

DAX is designed to work with tables, so it has just two primary data types: **Numeric** and **Other**. **Numeric** can include *integers*, *decimals*, and *currency*. **Other** can include *strings* and *binary objects*.

DAX uses operator overloading, which means you can mix data types in your calculations and the results will change based on the type of data used in the inputs. Conversion happens automatically, which means you don't have to know the data types of the columns you're working with in Power BI, but it also means that sometimes conversion can happen in unexpected ways.

DateTime is stored as a floating point value with both integer and decimal parts. DateTime can be used accurately for calculations of any time period after March 1, 1900.

DAX can compute values for seven data types:

- Integer
- Real
- Currency
- Date (datetime)
- TRUE/FALSE (Boolean)
- String

BLOB (binary large object)quired parameters correctly.

DAX supports the following data types:

Data type in model	Data type in DAX	Description
Whole Number	A 64 bit (eight-bytes) integer value	Numbers that have no decimal places. Integers can be positive or negative numbers, but must be whole numbers between -9,223,372,036,854,775,808 (-2^63) and 9,223,372,036,854,775,807 (2^63-1).
Decimal Number	A 64 bit (eight-bytes) real number [1, 2]	Real numbers are numbers that can have decimal places. Real numbers cover a wide range of values: Negative values from -1.79E +308 through -2.23E -308 Zero Positive values from 2.23E -308 through 1.79E + 308 However, the number of significant digits is limited to 17 decimal digits.
Boolean	Boolean	Either a True or False value.
Text	String	A Unicode character data string. Can be strings, numbers or dates represented in a text format.
Date	Date/time	Dates and times in an accepted date-time representation. Valid dates are all dates after March 1, 1900.
Currency	Currency	Currency data type allows values between -922,337,203,685,477.5808 to 922,337,203,685,477.5807 with four decimal digits of fixed precision.
N/A	Blank	A blank is a data type in DAX that represents and replaces SQL nulls. You can create a blank by using the BLANK function, and test for blanks by using the logical function, ISBLANK.

DAX Operators

The Data Analysis Expression (DAX) language uses operators to create expressions that compare values, perform arithmetic calculations, or work with strings.

There are **four different types** of calculation operators: **arithmetic, comparison, text concatenation, and logical.**

Arithmetic Operators:

To perform basic mathematical operations such as addition, subtraction, or multiplication; combine numbers; and produce numeric results, use the following arithmetic operators.

Arithmetic operator	Meaning	Example
+ (plus sign)	Addition	3+3
– (minus sign)	Subtraction or sign	3–1–1
* (asterisk)	Multiplication	3*3
/ (forward slash)	Division	3/3
^ (caret)	Exponentiation	16^4

Comparison Operators

You can compare two values with the following operators. When two values are compared by using these operators, the result is a logical value, either TRUE or FALSE.

Comparison operator	Meaning	Example
=	Equal to	[Region] = "USA"
==	Strict equal to	[Region] == "USA"
>	Greater than	[Sales Date] > "Jan 2009"
<	Less than	[Sales Date] < "Jan 1 2009"
>=	Greater than or equal to	[Amount] >= 20000
<=	Less than or equal to	[Amount] <= 100
<>	Not equal to	[Region] <> "USA"

All comparison operators except == treat BLANK as equal to number 0, empty string "", DATE(1899, 12, 30), or FALSE. As a result, [Column] = 0 will be true when the value of [Column] is either 0 or BLANK. In contrast, [Column] == 0 is true only when the value of [Column] is 0.

Text Concatenation operator

Use the ampersand (&) to join, or concatenate, two or more text strings to produce a single piece of text.

Text operator	Meaning	Example
& (ampersand)	Connects, or concatenates, two values to produce one continuous text value	[Region] & ", " & [City]

Logical Operators

Use logical operators (&&) and (||) to combine expressions to produce a single result.

Text operator	Meaning	Examples
&& (double ampersand)	Creates an AND condition between two expressions that each have a Boolean result. If both expressions return TRUE, the combination of the expressions also returns TRUE; otherwise the combination returns FALSE.	([Region] = "France") && ([BikeBuyer] = "yes"))
|| (double pipe symbol)	Creates an OR condition between two logical expressions. If either expression returns TRUE, the result is TRUE; only when both expressions are FALSE is the result FALSE.	(([Region] = "France") || ([BikeBuyer] = "yes"))
IN	Creates a logical OR condition between each row being compared to a table. Note: the table constructor syntax uses curly braces.	'Product'[Color] IN { "Red", "Blue", "Black" }

Operators and Precedence Order

It is important while working with operators

Calculation Order

An expression evaluates the operators and values in a specific order. All expressions always begin with an equal sign (=). The equal sign indicates that the succeeding characters constitute an expression.

Following the equal sign are the elements to be calculated (the operands), which are separated by calculation operators. Expressions are always read from left to right, but the order in which the elements are grouped can be controlled to some degree by using parentheses.

Operator Precedence

If you combine several operators in a single formula, the operations are ordered according to the following table. If the operators have equal precedence value, they are ordered from left to right. For example, if an expression contains both a multiplication and division operator, they are evaluated in the order that they appear in the expression, from left to right.

Operator	Description
^	Exponentiation
–	Sign (as in –1)
* and /	Multiplication and division
!	NOT (unary operator)
+ and –	Addition and subtraction
&	Connects two strings of text (concatenation)
=,==,<,>,<=,>=,<>	Comparison

Using Parentheses to control precedence order

To change the order of evaluation, you should enclose in parentheses that part of the formula that must be calculated first. For example, the following formula produces 11 because multiplication is calculated before addition. The formula multiplies 2 by 3, and then adds 5 to the result.

=5+2*3

In contrast, if you use parentheses to change the syntax, the order is changed so that 5 and 2 are added together, and the result multiplied by 3 to produce 21.

=(5+2)*3

In the following example, the parentheses around the first part of the formula force the calculation to evaluate the expression (3 + 0.25) first and then divide the result by the result of the expression, (3 - 0.25).

=(3 + 0.25)/(3 - 0.25)

In the following example, the exponentiation operator is applied first, according to the rules of precedence for operators, and then the sign operator is applied. The result for this expression is -4.

=-2^2

To ensure that the sign operator is applied to the numeric value first, you can use parentheses to control operators, as shown in the following example. The result for this expression is 4.

= (-2)^2

Compatibility

DAX easily handles and compares various data types, much like Microsoft Excel. However, the underlying computation engine is based on SQL Server Analysis Services and provides additional advanced features of a relational data store, including richer support for date and time types. Therefore, in some cases the results of calculations or the behavior of functions may not be the same as in Excel. Moreover, DAX supports more data types than does Excel. This section describes the key differences.

Coercing operands of data types

In general, the two operands on the left and right sides of any operator should be the same data type. However, if the data types are different, DAX will convert them to a common data type to apply the operator in some cases:

1. First, both operands are converted to the largest possible common data type.
2. Next, the operator is applied if possible..

For example, suppose you have two numbers that you want to combine. One number results from a formula, such as =[Price] * .20, and the result may contain many decimal places. The other number is an integer that has been provided as a string value.

In this case, DAX will convert both numbers to real numbers in a numeric format, using the largest numeric format that can store both kinds of numbers. Then DAX will apply the multiplication.

Depending on the data-type combination, type coercion may not be applied for comparison operations. Integer, Real Number, Currency, Date/time and Blank are considered numeric for comparison purposes. Blank evaluates to zero when performing a comparison. The following data-type combinations are supported for comparison operations.

Left Side Data Type	Right Side Data Type
Numeric	Numeric
Boolean	Boolean
String	String

Other mixed data-type comparisons will return an error. For example, a formula such as =”1” > 0 returns an error stating that *DAX comparison operations do not support comparing values of type Text with values of type Integer*.

Data Types used in DAX	Data Types used in Excel
Numbers (I8, R8)	Numbers (R8)
Boolean	Boolean
String	String
DateTime	Variant
Currency	Currency

Differences in Preedence Order

The precedence order of operations in DAX formulas is basically the same as that used by Microsoft Excel, but some Excel operators are not supported, such as percent. Also, ranges are not supported.

Therefore, whenever you copy and paste formulas from Excel, be sure to review the formula carefully, as some operators or elements in the formulas may not be valid. When there is any doubt about the order in which operations are performed, we recommend you use parentheses to control the order of operations and remove any ambiguity about the result.

DAX Important Terminology

a)Measures

Measures are dynamic calculation formulas where the results change depending on context.

Ex: Total Sales:=SUM([Sales Amount])

b)Calculated Columns

A calculated column is a column that you add to an existing table (in the model designer) and then create a DAX formula that defines the column's values. Because a calculated column is created in a table in the data model, they're not supported in models that retrieve data exclusively from a relational data source using DirectQuery mode.

Ex: =[Calendar Year] & " Q" & [Calendar Quarter]

c)Calculated Tables

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

Calculated tables can be helpful in a role-playing dimension

Calculated tables are also useful when configuring a filtered rowset, or a subset or superset of columns from other existing tables. This allows you to keep the original table intact while creating variations of that table to support specific scenarios.

d)Row Filters (Row-Level Security)

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

=Customers[Country] = "USA"

Working with AutoComplete

Both the formula bar in the model designer and the formula Row Filters window in the Role Manager dialog box provide an AutoComplete feature. AutoComplete helps you enter a valid formula syntax by providing you with options for each element in the formula.

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

VARIABLES

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

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

Return

   IF (
      TotalQuantity > 1000,
      TotalQuantity * 0.95,
      TotalQuantity * 1.25
      )
```

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

What are the types of DAX Functions available?

DAX includes functions you can use to perform calculations using dates and times, create conditional values, work with strings, perform lookups based on relationships, and the ability to iterate over a table to perform recursive calculations. If you are familiar with Excel formulas, many of these functions will appear very similar; however, DAX formulas are different in the following important ways:

- A DAX function always references a complete column or a table. If you want to use only particular values from a table or column, you can add filters to the formula.
- If you need to customize calculations on a row-by-row basis, DAX provides functions that let you use the current row value or a related value as a kind of parameter, to perform calculations that vary by context.
- DAX includes many functions that return a table, rather than a value. The table is not displayed in a reporting client, but is used to provide input to other functions. For example, you can retrieve a table and then count the distinct values in it, or calculate dynamic sums across filtered tables or columns.
- DAX functions include a variety of *time-intelligence* functions. These functions let you define or select date ranges, and perform dynamic calculations based on these dates or range. For example, you can compare sums across parallel periods.

DAX supports the following types of functions.

- **DAX Table-Valued Functions**
  - **DAX Filter Functions**
  - **DAX Aggregation Functions**
  - **DAX Time Intelligence Functions**
- **DAX Date and Time Functions**
- **DAX Information Functions**
- **DAX Logical Functions**
- **DAX Math and Trig Functions**
- **DAX Other Functions**
- **DAX Parent and Child Functions**
- **DAX Statistical Functions**
- **DAX Text Functions**
- **DAX Description Structure Functions**

DAX Table-Valued Functions

Many DAX functions take tables as input or output tables or do both. These DAX functions are called DAX table-valued functions. Because a table can have a single column, DAX table-valued functions also take single columns as inputs.

You have the following types of DAX table-valued functions −

- DAX Aggregation functions
- DAX Filter functions
- DAX Time intelligence functions

DAX Aggregation Functions

DAX Aggregation functions aggregate any expression over the rows of a table and are useful in calculations.

DAX Filter Functions

DAX Filter functions return a column or a table or values related to the current row. You can use DAX Filter functions to return specific data types, look up values in related tables and filter by related values. DAX Lookup functions work by using tables and relationships between them. DAX Filter functions enable you to manipulate the data context to create dynamic calculations.

DAX Time Intelligence Functions

DAX Time Intelligence functions return a table of dates or the use a table of dates to calculate an aggregation. These DAX functions help you create calculations that support the needs of Business Intelligence analysis by enabling you to manipulate data using time periods, including days, months, quarters, and years.

DAX Date and Time Functions

DAX Date and Time functions are similar to the Excel date and time functions. However, DAX Date and Time functions are based on the datetime data type of DAX.

DAX Information Functions

DAX Information functions look at the cell or row that is provided as an argument and tell you whether the value matches the expected type.

DAX Logical Functions

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

DAX Math and Trig Functions

DAX Mathematical and Trigonometric functions are very similar to the Excel mathematical and trigonometric functions.

DAX Parent and Child Functions

DAX Parent and Child functions are useful in managing data that is presented as a parent/child hierarchy in the Data Model.

DAX Statistical Functions

DAX Statistical functions are very similar to the Excel Statistical functions.

DAX Text Functions

DAX Text functions work with tables and columns. With DAX Text functions, you can return part of a string, search for text within a string or concatenate string values. You can also control the formats for dates, times, and numbers.

DAX Other Functions

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

DAX Function Description Structure

If you have to use a DAX function in a DAX formula, you need to understand the function in detail. You should know the syntax of the function, the parameter types, what the function returns, etc.

DAX Parameter Naming Conventions

DAX has standard parameter names to facilitate the usage and understanding of the DAX functions. Further, you can use certain prefixes to the parameter names. If the prefix is clear enough, you can use the prefix itself as the parameter name.

To understand the syntax of the DAX functions and to use data values appropriately for the relevant DAX function parameters, you need to understand DAX parameter naming convention.

Following are the DAX standard parameter names −

Sr.No.	Parameter Name & Description
1	**Expression:** Any DAX expression that returns a single scalar value, where the expression is to be evaluated multiple times (for each row/context).
2	**Value:** Any DAX expression that returns a single scalar value where the expression is to be evaluated exactly once before all other operations.
3	**Table:** Any DAX expression that returns a table of data.
4	**TableName:** The name of an existing table using standard DAX syntax. It cannot be an expression.
5	**ColumnName:** The name of an existing column using standard DAX syntax, usually fully qualified. It cannot be an expression.
6	**Name:** A string constant that will be used to provide the name of a new object.
7	**Order:** An enumeration used to determine the sort order.
8	**Ties:** An enumeration used to determine the handling of tie values.
9	**Type:** An enumeration used to determine the data type for PathItem and PathItemReverse.

Prefixing Parameter Names or Using the Prefix Only

You can qualify a parameter name with a prefix −

- The prefix should be descriptive of how the argument is used.
- The prefix should be in such a way that ambiguous reading of the parameter is avoided.

For example,

- **Result_ColumnName** − Refers to an existing column used to get the result values in the DAX LOOKUPVALUE () function.
- **Search_ColumnName** − Refers to an existing column used to search for a value in the DAX LOOKUPVALUE () function.

You can omit the parameter name and use only the prefix, if the prefix is clear enough to describe the parameter. Omitting the parameter name and using only prefix can sometimes help in avoiding the clutter during reading.

For example, Consider **DATE (Year_value, Month_value, Day_value)**. You can omit the parameter name – value, that is repeated thrice and write it as DATE (Year, Month, Day). As seen, by using only the prefixes, the function is more readable. However, sometimes the parameter name and the prefix have to be present for clarity.

For example, Consider **Year_columnName**. The parameter name is ColumnName and the prefix is Year. Both are required to make the user understand that the parameter requires a reference to an existing column of years.

DAX Functions - Description Structure

If you have to use a DAX function in a DAX formula, you need to understand the function in detail. You should know the syntax of the function, the parameter types, what the function returns, etc.

To enable you to understand how to read and interpret the DAX functions, a uniform function description structure is used. The different types of DAX functions are grouped by the type name of the DAX functions as chapters.

- Each of these chapters provides a brief description of the utility of the respective type of DAX functions.
- The brief description will be followed by the list of DAX functions corresponding to that chapter (Type/Category of DAX functions).
- Each DAX function name is hyperlinked to DAX function details that have the following DAX function description structure –
    - Description
    - Syntax
    - Parameters
    - Return Value
    - Remarks
    - Example

  The following sections explain each of these headings that appear in each DAX function explanation.

Description

In the Description section, you will learn what the DAX function is about and where it can be used.

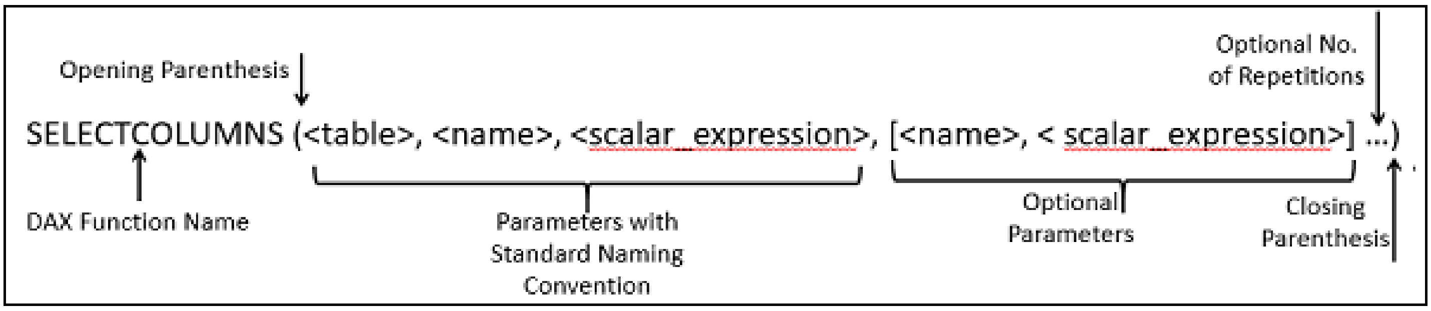
If the DAX function is introduced in Excel 2016, the same will be mentioned here. (Rest of the DAX functions exist in Excel 2013.)

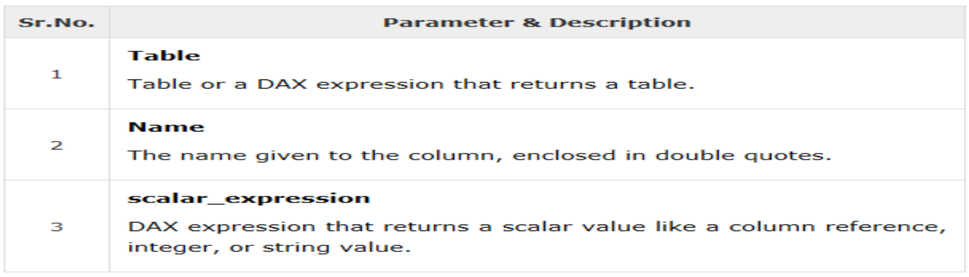
Syntax

In the Syntax section, you will learn the exact function name and the respective parameters.

- DAX function name is given in UPPERCASE letters.
- DAX function name is followed by opening parenthesis.
- Each parameter follows standard DAX parameter naming convention and is enclosed in angle brackets.
- If a parameter is optional, it is further enclosed in square brackets.
- The parameters are separated by commas.
- Ellipses … are used to show an optional number of repetitions of parameters.
- The function syntax ends with closing parenthesis.

Example


Opening Parenthesis
Optional No. of Repetitions
SELECTCOLUMNS (<table>, <name>, <scalar_expression>, [<name>, < scalar_expression>] …)
DAX Function Name
Parameters with Standard Naming Convention
Optional Parameters
Closing Parenthesis



Sr.No.	Parameter & Description
1	**Table** Table or a DAX expression that returns a table.
2	**Name** The name given to the column, enclosed in double quotes.
3	**scalar_expression** DAX expression that returns a scalar value like a column reference, integer, or string value.

Parameters

In the Parameters section, each of the parameters of the specific DAX function is listed in a table with its description. For example, the parameters of the above example DAX function SELECTCOLUMNS is listed in the following table.

Return Value

In the Return Value section, you will learn about what value the DAX function will return and its data type.

Remarks

In the Remarks section, you will learn about any extra information that you need to know about the usage of the DAX function. You will also understand the potential errors and the reasons.

Example

An example of the usage of the DAX function is given in this section.

Note – When you write DAX functions with the data values for the parameters, you will follow the naming conventions as given below –

- A Table name is specified as it appears in the Data Model. E.g. Sales.
- A Column name is specified as it appears in the Data Model with square brackets enclosing it.

  For example: [Sales Amount]

  How many ways we refer a column?

  - It is recommended to use fully qualified names for columns, i.e. a column name is prefixed with the table name that contains it.

    For example: Sales[Sales Amount]

  - If the table name contains spaces, it should be enclosed in single quotes.

    For example: 'East Sales'[Sales Amount]

- A DAX function can return a column or table of values, in which case, it needs to be used as a parameter of another DAX function that requires a column or table.

DAX Logical operators

And→ All conditions to be satisfied

Syn : And(condition1, condition2…)

Or→ Any condition to be satisfied

Syn : Or(condition1, condition2…)

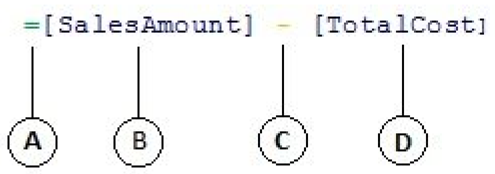
DAX in Data Model [Diffeences between Calculated Column, Measure, and Table?]

We can apply DAX formula in New column and New measure

New column [Calculated field]	New measure [Calculated measure]	New Table
Adding new column	Adding a measure to the query	**One or more columns as new table**
Row by row operation and return Multiple rows	Set of rows operation and return single row	
More memory required due to model operation	More CPU required due to many rows store a Time consuming process	

DAX Formula Syntaxes

Formula 1

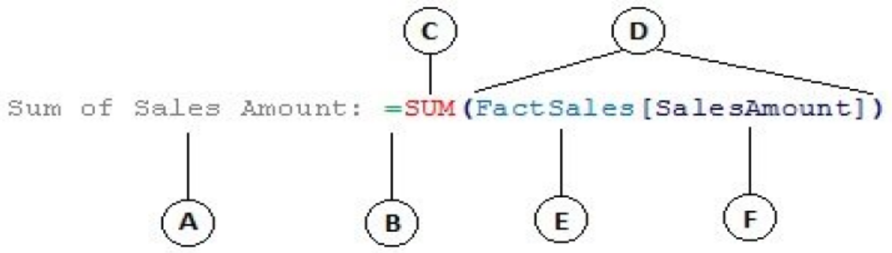


This formula's syntax includes the following elements:
A. The equals sign operator (=) indicates the beginning of the formula, and when this formula is calculated it will return a result or value. All formulas that calculate a value will begin with an equals sign.
B. The referenced column [SalesAmount] contains the values we want to subtract from. A column reference in a formula is always surrounded by brackets []. Unlike Excel formulas which reference a cell, a **DAX formula always references a column.**
C. The subtraction (-) mathematical operator.
D. The referenced column [TotalCost] contains the values we want to subtract from values in the [SalesAmount] column.

You can read this formula as:
In the FactSales table, for each row in the Margin calculated column, calculate (=) a value by subtracting (-) values in the [TotalCost] column from values in the [SalesAmount] column.

Formula 2

Let's take a look at another type of formula, one that is used in a measure:



This formula includes the following syntax elements:

A. The measure name `Sum of Sales Amount`. Formulas for measures can include the measure name, followed by a colon, followed by the calculation formula.
B. The equals sign operator (=) indicates the beginning of the calculation formula. When calculated, it will return a result.
C. The function `SUM` adds up all of the numbers in the [SalesAmount] column. You will learn more about functions later.
D. Parenthesis `()` surround one or more arguments. All functions require at least one argument. An argument passes a value to a function.
E. The referenced table `FactSales`.
F. The referenced column `[SalesAmount]` in the FactSales table. With this argument, the SUM function knows on which column to aggregate a SUM.

Note(Important):
Columns referenced in the same table do not require the table name included in the formula. This can take long formulas that reference many columns shorter and easier to read. However it is good practice to always include the table name in your measure formulas, even when in the same table.

If the name of a table contains spaces, reserved keywords, or disallowed characters, you must enclose the table name in single quotation marks. You must also enclose table names in quotation marks if the name contains any characters outside the ANSI alphanumeric character range, regardless of whether your locale supports the character set or not.
PowerPivot (and SQL Server Data Tools) includes IntelliSense; a feature used to create syntactically correct

Dax syntax

Syntax Naming Conventions

A DAX formula always starts with an equal sign (=). After the equals sign, you can provide any expression that evaluates to a scalar, or an expression that can be converted to a scalar. These include the following:

- A scalar constant, or expression that uses a scalar operator (+,-,*,/,>=,...,&&, ...)
- References to columns or tables. The DAX language always uses tables and columns as inputs to functions, never an array or arbitrary set of values.
- Operators, constants, and values provided as part of an expression.
- The result of a function and its required arguments. Some DAX functions return a table instead of a scalar, and must be wrapped in a function that evaluates the table and returns a scalar; unless the table is a single column, single row table, then it is treated as a scalar value.

  Most DAX functions require one or more arguments, which can include tables, columns, expressions, and values. However, some functions, such as PI, do not require any arguments, but always require parentheses to indicate the null argument. For example, you must always type PI(), not PI. You can also nest functions within other functions.

- Expressions. An expression can contain any or all of the following: operators, constants, or references to columns.

For example, the following are all valid formulas.

Formula	Result
=3	3
="**Sales**"	**Sales**
='Sales'[Amount]	If you use this formula within the Sales table, you will get the value of the column Amount in the Sales table for the current row.
=(0.03 *[Amount]) =0.03 * [Amount]	Three percent of the value in the Amount column of the current table. Although this formula can be used to calculate a percentage, the result is not shown as a percentage unless you apply formatting in the table.
=PI()	The value of the constant pi.

A data model often contains multiple tables. Together the tables and their columns comprise a database stored in the in-memory analytics engine (VertiPaq). Within that database, all tables must have unique names. The names of columns must also be unique within each table. All object names are *case-insensitive*; for example, the names **SALES** and **Sales** would represent the same table.

Each column and measure you add to an existing data model must belong to a specific table. You specify the table that contains the column either implicitly, when you create a calculated column within a table, or explicitly, when you create a measure and specify the name of the table where the measure definition should be stored.

When you use a table or column as an input to a function, you must generally *qualify* the column name. The *fully qualified* name of a column is the table name, followed by the column name in square brackets: for examples, 'U.S. Sales'[Products]. A fully qualified name is always required when you reference a column in the following contexts:

- As an argument to the function, VALUES
- As an argument to the functions, ALL or ALLEXCEPT
- In a filter argument for the functions, CALCULATE or CALCULATETABLE
- As an argument to the function, RELATEDTABLE
- As an argument to any time intelligence function

An *unqualified* column name is just the name of the column, enclosed in brackets: for example, [Sales Amount]. For example, when you are referencing a scalar value from the same row of the current table, you can use the unqualified column name.

If the name of a table contains spaces, reserved keywords, or disallowed characters, you must enclose the table name in single quotation marks. You must also enclose table names in quotation marks if the name contains any characters outside the ANSI alphanumeric character range, regardless of whether your locale supports the character set or not. For example, if you open a workbook that contains table names written in Cyrillic characters, such as 'Таблица', the table name must be enclosed in quotation marks, even though it does not contain spaces.

Note

To make it easier to enter the fully qualified names of columns, use the AutoComplete feature in the formula editor.

Tables

- Table names are required whenever the column is from a different table than the current table. Table names must be unique within the database.
- Table names must be enclosed in single quotation marks if they contain spaces, other special characters or any non-English alphanumeric characters.

Measures

- Measure names must always be in brackets.
- Measure names can contain spaces.
- Each measure name must be unique within a model. Therefore, the table name is optional in front of a measure name when referencing an existing measure. However, when you create a measure you must always specify a table where the measure definition will be stored.

Columns

Column names must be unique in the context of a table; however, multiple tables can have columns with the same names (disambiguation comes with the table name).

In general, columns can be referenced without referencing the base table that they belong to, except when there might be a name conflict to resolve or with certain functions that require column names to be fully qualified.

Reserved keywords

If the name that you use for a table is the same as an Analysis Services reserved keyword, an error is raised, and you must rename the table. However, you can use keywords in object names if the object name is enclosed in brackets (for columns) or quotation marks (for tables).

Note

Quotation marks can be represented by several different characters, depending on the application. If you paste formulas from an external document or Web page, make sure to check the ASCII code of the character that is used for opening and closing quotes, to ensure that they are the same. Otherwise DAX may be unable to recognize the symbols as quotation marks, making the reference invalid.

Special characters

The following characters and character types are not valid in the names of tables, columns, or measures:

- Leading or trailing spaces; unless the spaces are enclosed by name delimiters, brackets, or single apostrophes.
- Control characters
- The following characters that are not valid in the names of objects:

  .,;':/*|?&%$!+=()[]{}<>

Example of Object Names:

Object Types	Examples	Comment
Table name	**Sales**	If the table name does not contain spaces or other special characters, the name does not need to be enclosed in quotation marks.
Table name	**'Canada Sales'**	If the name contains spaces, tabs or other special characters, enclose the name in single quotation marks.
Fully qualified column name	**Sales[Amount]**	The table name precedes the column name, and the column name is enclosed in brackets.
Fully qualified measure name	**Sales[Profit]**	The table name precedes the measure name, and the measure name is enclosed in brackets. In certain contexts, a fully qualified name is always required.
Unqualified column name	**[Amount]**	The unqualified name is just the column name, in brackets. Contexts where you can use the unqualified name include formulas in a calculated column within the same table, or in an aggregation function that is scanning over the same table.
Fully qualified column in table with spaces	**'Canada Sales'[Qty]**	The table name contains spaces, so it must be surrounded by single quotes.

Other restrictions

The syntax required for each function, and the type of operation it can perform, varies greatly depending on the function. In general, however, the following rules apply to all formulas and expressions:

- DAX formulas and expressions cannot modify or insert individual values in tables.
- You cannot create calculated rows by using DAX. You can create only calculated columns and measures.
- When defining calculated columns, you can nest functions to any level.
- DAX has several functions that return a table. Typically, you use the values returned by these functions as input to other functions, which require a table as input.
-

CONTEXT

***Context* is an important concept to understand when creating DAX formulas**. Context is what enables you to perform dynamic analysis, as the results of a formula change to reflect the current row or cell selection and also any related data.

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

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

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

Row Context:

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

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

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

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

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

Multiple Row Context

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

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

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

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

Query Context

Query context refers to the subset of data that is implicitly retrieved for a formula. When a user places a measure or other value field into a PivotTable or into a report based on a tabular model, the engine examines the row and column headers, Slicers, and report filters to determine the context. Then, the necessary queries are run against the data source to get the correct subset of data, make the calculations defined by the formula, and then populate each cell in the PivotTable or report. The set of data that is retrieved is the query context for each cell.

Because context changes depending on where you place the formula, the results of the formula can also change.

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

Filter Context

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

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

Determing context in formulas

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

For example, if you have just imported some data into a new table and it is not related to any other tables (and you have not applied any filters), the *current context* is the entire set of columns in the table. If the table is linked by relationships to other tables, the current context includes th

e related tables. If you add a column from the table to a report that has Slicers and maybe some report filters, the context for the formula is the subset of data in each cell of the report.

DAX usage components

Power BI Desktop

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

Power Pivot in Excel

The Power Pivot in Excel models designer includes a DAX editor for creating DAX calculation formulas.

Visual Studio

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

SQL Server Management Studio

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

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

DAX QUERIES

With DAX queries, you can query and return data defined by a table expression. Reporting clients construct DAX queries whenever a field is placed on a report surface, or a whenever a filter or calculation is applied. DAX queries can also be created and run in SQL Server Management Studio (SSMS) and open-source tools like DAX Studio. DAX queries run in SSMS and DAX Studio return results as a table.

Syntax:

```
[DEFINE {  MEASURE <tableName>[<name>] = <expression> }
    {  VAR <name> = <expression>}]
EVALUATE <table>
[ORDER BY {<expression> [{ASC | DESC}]}[, …]
[START AT {<value>|<parameter>} [, …]]]
```

EVALUATE

At the most basic level, a DAX query is an **EVALUATE** statement containing a table expression. However, a query can contain multiple EVALUATE statements.

Syntax

```
EVALUATE <table>
```

Arguments

TermDefinition

table A table expression.

Example

```
EVALUATE(
   'Internet Sales'
   )
```

Returns all rows and columns from the Internet Sales table, as a table.

ORDER BY

The optional **ORDER BY** keyword defines one or more expressions used to sort query results. Any expression that can be evaluated for each row of the result is valid.

Syntax

```
EVALUATE <table>
[ORDER BY {<expression> [{ASC | DESC}]}[, …]
```

Arguments

Term	Definition
expression	Any DAX expression that returns a single scalar value.
ASC	(default) Ascending sort order.
DESC	Descending sort order.

Example

```
EVALUATE(
   'Internet Sales'
   )
ORDER BY
               'Internet Sales'[Order Date]
```

Returns all rows and columns from the Internet Sales table, ordered by Order Date, as a table.

DEFINE

The optional **DEFINE** keyword defines entities that exist only for the duration of the query. Definitions are valid for all EVALUATE statements. Entities can be variables, measures, tables, and columns. Definitions can reference other definitions that appear before or after the current definition. Definitions typically precede the EVALUATE statement.

Syntax

```
 [DEFINE {  MEASURE <tableName>[<name>] = <expression> }
      {  VAR <name> = <expression>}]
EVALUATE <table>
```

Arguments

Term	Definition
tableName	The name of an existing table using standard DAX syntax. It cannot be an expression.
name	The name of a new measure. It cannot be an expression.
expression	Any DAX expression that returns a single scalar value. The expression can use any of the defined measures. The expression must return a table. If a scalar value is required, wrap the scalar inside a ROW() function to produce a table.
VAR	An optional expression as a named variable. A VAR can be passed as an argument to other expressions.

Example

```
DEFINE
MEASURE 'Internet Sales'[Internet Total Sales] = SUM('Internet Sales'[Sales Amount])
EVALUATE
SUMMARIZECOLUMNS
(
		'Date'[Calendar Year],
		TREATAS({2013, 2014}, 'Date'[Calendar Year]),
		"Total Sales", [Internet Total Sales],
		"Combined Years Total Sales", CALCULATE([Internet Total Sales], ALLSELECTED('Date'[Calendar Year]))
)
ORDER BY [Calendar Year]
```

Returns the calculated total sales for years 2013 and 2014, and combined calculated total sales for years 2013 and 2014, as a table. The measure in the DEFINE statement, Internet Total Sales, is used in both Total Sales and Combined Years Total Sales expressions.

Parameters in DAX Queries

A well-defined DAX query statement can be parameterized and then used over and over with just changes in the parameter values.

The Execute Method (XMLA) method has a Parameters Element (XMLA) collection element that allows parameters to be defined and assigned a value. Within the collection, each Parameter Element (XMLA) element defines the name of the parameter and a value to it.

Reference XMLA parameters by prefixing the name of the parameter with an @ character. Hence, any place in the syntax where a value is allowed it can be replaced with a parameter call. All XMLA parameters are typed as text.

Important

Parameters defined in the parameters section and not used in the **<STATEMENT>** element generate an error response in XMLA.

Important

Parameters used and not defined in the **<Parameters>** element generate an error response in XMLA.