What is**DAX**?

“ Data Analysis Expressions (DAX) is a library of functions and operators combined to create formulas and expressions “

Introductionto DAX

› Whereto find

›Power BI, Power Pivot forExcel, Microsoft AnalysisServices

› Purpose

› DAX was created to enumerate formulas across the data model, where the data is stored in the form of tables, which can be linked together through the sessions. They may have a cardinality of either 1: 1, 1: N, or M: N and your direction, which decides which table filters which. These sessions are either active or inactive. The active session is automatically and participates in the calculation. The inactive is involved in this when it is activated, for example, by a function **USERELATIONSHIP()**

Basic concepts

› Constructsand theirnotation

›Table –‘Table‘

›Column–[Column] -> ‘Table‘[Column]

› Measure – [NameOfMeasure]

› Comments

›Single-line (CTRL + ´) –// or--

›Multi-line –/* */

› Data types

› INTEGER

› DECIMAL

› CURRENCY

› DATETIME

› BOOLEAN

› STRING

›VARIANT (not implemented in Power BI)

› BINARY

›DAX can work very well with some types as well combined as if it were the same type. If so, for example, the DATETIME and INTEGER data types are supported operator "+" then it is possible to use them together.

Example:DATETIME ([Date]) + INTEGER (1) = DATETIME ([Date] + 1)

Operators

› Arithmetic { + , -, / , * , ^ }

› Comparative{ = , == , > , < , >= , <= , <> } ›

Joining text { & }

› Logic { && , II , IN, NOT }

› Prioritization { (,) }

Calculated Columns

› They behave like any other column in the table. Instead of coming from a data source, they are created through a DAX expression evaluated based on the current context line, and we cannot get values of another row directly.

› **Import mode.** Their evaluation and storage is in progress when processing the model.

›**DirectQuery mode.** They are evaluated at runtime, which may slow down the model.

Profit= Trades[Quantity]*Trades[UnitPrice]

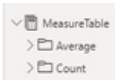
Measures

› They do not compare row-based calculations, but they perform aggregation of row-based values input contexts that the environment passes to the calculation. Because of this, there can be no pre-counting result. **It must be evaluated only at the moment when Measure is called.**

› The condition is that they must always be linked to the table to store their code, which is possible at any time alter. Because their calculation is no longer directly dependent, it is common practice to have one separate **Measure Table**, which groups all Measures into myself. For clarity, they are therefore further divided into **folders**.

Example of **Measure**:

SalesVolume = **SUM**(Trades[Quantity])



Variables

› Variables in DAX calculations allow avoiding repeated recalculations of the same procedure. Which might look like this:

```
NumberSort=
VAR_selectedNumber=
  SELECTEDVALUE( Table[Number] )
RETURN
IF( _selectedNumber< 4, _selectedNumber, 5 )
```

› Their declaration uses the word VAR after followed by the **name**"**=**" and the **expression**. The first using the word VAR creates a section for DAX where possible declare such variables 1 to X. Individual variables always require a comment for their declaration VAR before setting the name. To end this section, the word **RETURN**that it defines is a necessary return point for calculations.

› Variables are localonly.

›If there is a variable in the formula that is not used to get the result, this variable does not evaluate.

(**Lazy Evaluation**)

› Evaluation of variables is **performed based on evaluated context instead of the context in which the variable is used directly.**Within one, The expression can be multiple VAR / RETURN sections that always serve to evaluate the currently evaluated context.

› They can store both the value and the whole table

Calculation contexts

› All calculations are evaluated on a base basis some context that the environment brings to the calculation. (**Evaluation context**)

› Context Filter -

The following calculation calculates the profit forindividualsales.

```
Revenue =
    SUMX( Trades,
      Trades[Quantity]*
      Trades[UnitPrice]
)
```

Country	Revenue
Australia	2.838.077.18
Canada	73.959.95
Germany	340.392.76
Japan	1.833.026.16
Mexico	320.833.27
Nigeria	378.202.44
Total	**5,784,491.77**

If I place this calculation in a table without a **Country**column, then the result will be 5,784,491.77. With this column, we get "**Total**" the same as the previous calculation. Still, the individual records provide us with a **FILTER** context that filters in calculating the input the **SUMX**function's input. They behave the same way, for example, **AXES** in the chart.

› The filter context iscan be adjusted with various functions, such as **FILTER,ALL, ALLSELECTED**

› Row context -Unlike the previous one, this context does not filter the table. It is used to iterate over tables and evaluate values columns. They are typical, but at the same time, specific example calculated columns that are calculated from data that are valid for the table row being evaluated. In particular that, manual creation is not required when creating the line context because **DAX**makes it. Above the mentioned example with the use of **SUMX**also hides in itself line context. Because **SUMX**is the function for that specified, the table in the first argument performs an iterative pass and evaluates the calculation line by line. The line context is possible to use even nested. Or, for each row of the table, evaluates each row of a different table.

Calculate type function

› **CALCULATE,** and**CALCULATETABLE** are functions that can programmatically set the **context filter**. In addition to this feature converts any existing line context to a context filter.

› **Calculate** and **Calculatetable**syntax:

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

› The section filter within the **Calculate**expression is NOT of type **boolean**but **Table**type. Nevertheless, booleancan be used as an argument.

› Example of using the calculate function in a cumulative calculation the sum of sales for the last 12 months:

```
CALCULATE (
   SUM( Trades[Quantity] ),
DATESINPERIOD(
DateKey[Date],
        MAX( DateKey[Date] ),
-1,
YEAR

   ) )
```

› Syntax Sugar:

```
› [TradeVolume](Trades[Dealer] = 1)
=
CALCULATE( [TradeVolume], Trades[Dealer] = 1)
=
CALCULATE( [TradeVolume], FILTER(
ALL(Trades[Dealer] ) ,
Trades[Dealer] = 1) )
```

Calcuation Groups

› They are very similar to **Calculated members**from MDX. In Power BI, it is not possible to create them directly in the **Desktop**application environment, but an External Tool **Tabular Editor**is required.

› This is a set of **Calculation Items**grouped according to their purpose and whose purpose is to prepare an expression, which can be used for different input measures, so it doesn‘t have to write the same expression multiple times. To where she would be, but the input measure is placed **SELECTEDMEASURE()**.

Example:

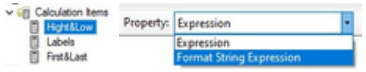
```
CALCULATE ( SELECTEDMEASURE(),
Trades[Dealer] = 1)
```

Name	Ordinal
Hight&Low	0
Labels	1
First&Last	2

› From a visual point of view, the Calculation Group looks like a table with just two columns, "**Name**," "**Ordinal,**" and rows that indicate the individual Calculation Items.

› In addition to facilitating the reusability of the prepared expressions also provide the ability to modify the output format of individual calculations. Within this section, “**Format String Expression**”often uses the **DAX**function **SELECTEDMEASUREFORMATSTRING**(), which returns a format string associated with the **Measures**being evaluated.

Example:




```
VAR _selectedCurrency= SELECTEDVALUE( Trades[Currency] )
RE TURN
  SELECTEDMEASUREFORMATSTRING()& „ “ & _selectedCurrency
```

› In Power BI, they can allbe evaluatedpre-prepared items, or it is possible, for example, to use the cross-section to define items that are currently being evaluated

› Sometimes, however, it is necessary to enable the evaluation of Calculation Items only for Specific Measures. In that case, it is possible to use the **ISSELECTEDMEASURE**()function, whoseoutput is a value of type booleanor the**SELECTEDMEASURENAME**()function that returns the name of the currently inserted measureas a string.

Conditions

Like most languages, DAX uses the IF function. Within this language, it is defined by syntax:

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

Where false, the branch is optional. The IF function explicitly evaluates only a branch that is based on the result of a logical test relevant.

› If both branches need to be evaluated, then there is a function **IF.EAGER()**whose syntax is the same as IF itself but evaluates as:

```
VAR_value_if_true= <value_if_true>
VAR_value_if_false= <value_if_false>
RETURN
IF(<logical_test>, _value_if_true, _value_if_false)
```

› **IF**has an alternative as**IFERROR**. Evaluates the expression and return the output from the <value_if_error> branch only if the expression returns an error. Otherwise, it returns the value of the expression itself.

› **DAX**supports concatenation of conditions, both using submerged ones**IF**, so thanks to the **SWITCH**function. It evaluates the expression against the list values and returns one of several possible result expressions.

Hierarchy

› DAX itself has no capability within the hierarchy to automatically convert your calculations to parent or child levels. Therefore, each level must Prepare Your Measures, which are then displayed based on the **ISINSCOPE**function. She tests which level to go just evaluating. Evaluation takes place from the bottom to the top level.

› The native data model used by DAX does not directly support its **parent/child**hierarchy. On the other hand, DAX contains functions that can convert this hierarchy to separate columns.

› **PATH** -It accepts two parameters, where the first parameter is the key ID column tables. The second parameter is the column that holds the parent ID of the row. The result of this function then looks like this:**1|2|3|4**
Syntax:**PATH**(<ID_columnName>, <parent_columnName>)

› **PATHITEM**–Returns a specific item based on the specified position from the string, resulting from the PATH function. Positions are counted from left to right. The inverted view uses the PATHITEMREVERSE function.
Syntax: **PATHITEM**(<path>, <position>[, <type>])

› **PATHLENGTH** –Returns the number of parent elements to the specified item in given the **PATH**result, including itself.
Syntax: **PATHLENGTH**(<path>)

› **PATHCONTAINS** –Returns **true**if the specified item is specified exists in the specified PATH path.
Syntax: **PATHCONTAINS**(<path>, <item>)

DAX Queries

› The basic building block of DAX queries is the expression EVALUATE followed by any expression whose output is a table.
Example:

```
EVALUATE
ALL(Trades[Dealer] )
```

› The **EVALUATE**statement can be divided into three primary sections. Each section has its specific purpose and its introductory word.

› **Definition –**It always starts with the word **DEFINE**. This section defines local entities such as tables, columns, variables, and measures. There can be one section definition for an entire query, although a query can contain multiple EVALUATEs

› **Query –**It always starts with the word **EVALUATE**. This section contains the table expression to evaluate and return as a result.

› **Result –**This is a section that is optional and starts with the word ORDER BY. It contains the possibility to sort the result based on the inserted inputs.

Example:

```
DEFINE
VAR_tax = 0.79
EVALUATE
ADDCOLUMNS(
Trades,
„AdjustedpProfit“,
      ( Trades[Quantity] * Trades[UnitPrice] ) * _tax
)
ORDER BY[AdjustedpProfit]
```

› This type of notation is used, for example, in DAX Studio (**daxstudio.org**). It is a publicly available tool that provides free access to query validation, code debugging, and query performance measurement.

› DAX studio has the ability to connect directly to **AnalysisServices**, **Power BI** a **Power Pivot forExcel**

Recommendedsources

› Marco Russo& Alberto Ferrari

› Daxpatterns.com

› dax.guide

›TheDefinitiveGuideto DAX

POWER BI CHEATSHEET