**DAX**

Data Analysis Expressions

SQL is a declarative language where DAX is a purely functional language.

DAX stands for Data Analysis Expressions, it is language developed by Microsoft to interact with data in a variety of their platforms like Power BI, PowerPivot and SSAS tabular models. It is designed to be simple and easy to learn while exposing the power and flexibility of tabular models. In a way, you could compare it with Excel formulas on steroids. Using DAX will truly unleash the capabilities of Power BI.

To see a dax formate is correct, we can visit : www.daxformatter.com

**Computed cloumns:**

1.They are computer on data refresh time. Computed columns are used for add a better view to our table.

2. They are stored in the model. They would use your spaces like ram space , disk space. So , for large data we need to consider this point.

Product[price] means-

i)values for the price column from

ii)Product table

iii) for the current row

iv)different for each row

Example :

\*Sales[unit price]\*sales[quantity]

\*Year(today())- year(customer[birthdate])

Here year function will subtract the year from the date.

\*IF(ISBLANK(customer[dob]), BLANK, Year(today())- year(customer[birthdate]))

**Measures:**

They do not work row by rows, instead they use as a table and aggregated functions.

They do not have current rows concept.

Measures are very useful to work with when there is no row by row calculation. And it is very useful as it uses the CPU memory. So , for a huge number of columns, measures are useful.

Measures are as written : [measureName]

**Measure Vs Calculated Columns**

|  |  |
| --- | --- |
| Calculate column | Measure |
| Use it when Need to slice or filter on a value.  Stored in the model . Use disk space, ram space | Use it when calculate percentage , calculate ration, complex aggregated fuctions.  Doesn’t stored in the model. Use CPU space |

**Aggregated Functions:**

These functions helps to aggregate

* SUM()
* AVERAGE()
* MIN()
* MAX()

Only works on numerical values and don’t work on multiple columns. Only on one column.

Like: sum(sales[amount])

Sum(sales[amount] \* sales[quantity]) [wrong]

**X aggregator or Iterator :**

this aggregator scans a table and aggregate row by row and compute an expression for every row.

* SUMX()
* AVERAGEX()
* MINX()
* MAXX()

SUMX( Sales,

Sales[amount] \* sales[quantity]

)

**Variable :**

Basically , its like given a name to a expression. It will evaluated once ,then it will use the values.

If , we don’t call it, it will never evaluate.

i)Measures/ table =

VAR TotalQunatity = sum(sales[quantity])

ii) Margin % =

var sales\_amount = sumx(sales,sales[net price]\*sales[quantity])

var margin = sumx(sales, sales[net price]-sales[unit price])

return margin/sales\_amount

so the margin % will act on these two variables.

Another exam :

(this expression will be written in the Margin % measure)

VAR Margin = sum(sales[margin line])

VAR SalesAmount = sumx(Sales, sales[line amount]\*sales[quantity])

RETURN Margin/SalesAmount

**Table Functions :**

The functions that returns a table. It can also be called calculated column.

* FILTER ()
* ALL()
* VALUES()
* DISTINCT()
* RELATEDTABLE()

**Filter :**

Example :

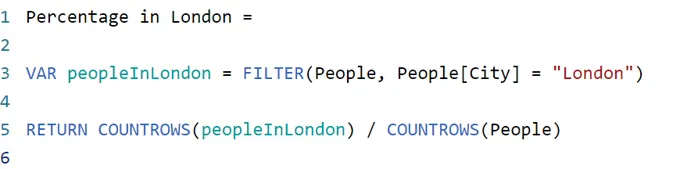
1. ExpensiveProducts = Filter (sales, sales[unit price]>=100)
2. Total sales for expensiveProducts = sumx(filter(sales, sales[unit price]>=100), sales[A]\*sales[B])

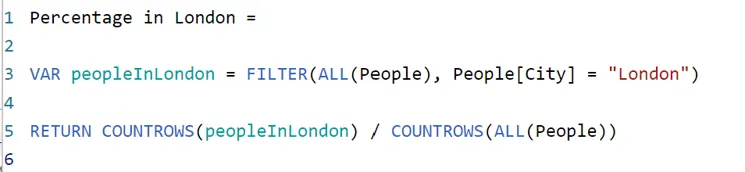
SO , **AggregatedX Function(table function(table name, table function condition),AggregatedX function condition**)

**ALL() :**

All() returns all the values of a table. All removes the filter context. This can be useful for calculating **percentages/ratios.**

I ) Filter(ALL(sales),sales[unit price >=50])







Then the percentage will always show 40.0% because the calculation will ignore any applied filter context and always use the entire dataset.

Distinct() :

|  |  |  |
| --- | --- | --- |
| Azure | 10 | 5 |
| Black | 10 | 2 |
| Red | 10 | 1 |
|  |  |  |
| Gray | 10 | 2 |
|  |  |  |

The DISTINCT operator just returns all the distinct values in a column.

As All() ignores filter context, Distinct() obeys the filter context.

**Example 1:**

i) counting distinct sold products on colors

ii)Select color form product[color]

iii)For all sold values ,create a measure , totalsales= all(sales[product\_key])

iv)For all distinct values ,create a measure , totalsales= countrows(distinct(sales[product\_key]))

|  |  |
| --- | --- |
| London | 10 |
| Paris | 3 |
| Burien | 11 |

Comment : as we need to find out from sales , so we need to count how much the product key arrived in the sales table. That’s why the product\_key arrived.

**Example 2 :** we need to find out the number of people from different cities

1. Select column from customer table customer[city]
2. Create a measure like , num of customers = countrows(distinct(customer[name]))

**Example 3:**

Showing all distinct city values.

This can be very powerful when building up transformations.

We can also use it combined with COUNTROWS to calculate the number of distinct values in a table:

Counting distinct values in a table.



measure to find how many distinct people there are in each city:

Showing number of people aggregated by city.

For each row of this table, a filter context of e.g. People[City] = "London" has been applied.

**Relatedtable():**

RELATEDTABLE () is a function that returns a table and not a scalar value. This means that this function cannot be used by itself to define a calculated column or a measure. Instead this function can only be used to provide an intermediate result that is in turn an argument to another function, such an aggregation function.

RELATEDTABLE () is a table function. Like RELATED, RELATEDTABLE requires a relationship between the two tables. RELATEDTABLE () goes from the One side of a relationship to the many side. As it goes to the Many side, it is unable to return a single value, but instead it returns a table of values. The table of values that it returns can be based on filters.

**Example 1:**

Calculate a customers total sales

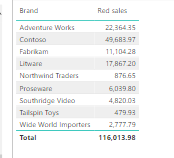
1. Create calculated columns on customer table
2. No of sale = countrows(relatedtable(sales))
3. Total sales price = sumx(relatedtable(sales),

sales[quantity]\*sales[net\_price])

comment : for relatedtable we need to add a column to the current table and use relatedtable with the related table. Simply we countrows to count the number , or to get values ..use relatedtable()

**Related:**

RELATED() follows existing many-to-one relationship(s) from the many side to the one side and returns the single matching value from the other table. In other words, RELATED () can access the one-side from the many-side because there is only one rows exists in the related table and if no matching row exists, RELATED () will return BLANK.

RELATED () works as a lookup and for it to work, there must be a relationship between the two tables.

When the RELATED () function performs a lookup, it examines all values in the specified table regardless of any filters that may have been applied.

Example 1:

1. Add a measure ,

Total amount of red sales = sumx(filter(retated(products), product[color]=”red”) sales[quantity]\*sales[net\_price])

ii) select …sold product/measure and then select the measure.

**Comment : filter context would do the visual, it will show the related products red sales amount.**

Evaluation context :

Useful links:

<https://intelalytics.com/free-training-resources>

https://powerbidocs.com/power-bi-dax-functions/