PROSYS - eMIS

MIS Automation Solutions

Prosys Infotech Private Limited : Newsletter for Private Circulation Only | September 2016 Volume 6 Issue 22

DAX coding style using variables

This article shows how variables in DAX can impact the coding style, simplifying a step-by-step approach and improving the readability of your code.
The new feature of variables in DAX has been available since one year ago in Power BI, Power Pivot for Excel 2016, and Analysis Services 2016. You can find a description of the syntax in the [Variables in DAX] article. The goal here is to focus on how using variables can improve the coding style.

Variables for scalar values

For example, consider the following DAX measure that calculates the taxed amount of the rows in the Sales table.

```
TaxedSales := SUMX (
    Sales,
    Sales[Quantity] * Sales[Unit Price] * ( 1 + Sales[Tax Percentage] ) )


```

VIEW IN DAX FORMATTER

This is certainly an efficient way to perform the calculation, multiplying the quantity by the unit price, and then multiplying such a result by the tax percentage applied to the line (summing one to this value in order to obtain the total taxed value). However, an efficient code might be not the simpler code to write, to read, and to debug. Imagine a more complex calculation than this simple one, and you will recognize the issue.

In order to improve code readability, it would be good to split the calculation in several steps, giving a name to each intermediate calculation. Using the "old" DAX without the variables, you can obtain this result by using ADDCOLUMNS. However, if each term has to use the previous one, you have to use nested ADDCOLUMNS call, otherwise you do not have access to another column added in the same ADDCOLUMNS call.

```
TaxedSalesExplained := SUMX (
    ADDCOLUMNS (
        ADDCOLUMNS (
            ADDCOLUMNS ( Sales, "LineAmount", Sales[Quantity] * Sales[Unit Price] ),
            "Taxes", [LineAmount] * Sales[Tax Percentage]
        ),
        "TaxedAmount", [LineAmount] + [Taxes]
    ),
    [TaxedAmount] )


```

VIEW IN DAX FORMATTER

In the TaxedSalesExplained measure there are three steps in the calculation:

1. LineAmount is the result of quantity multiplied by unit price;
2. Taxes is the value of the taxes that have to be applied to the line;
3. TaxedAmount is the sum of LineAmount and Taxes.

From one point of view, the last measure improves readability and might improve calculation efficiency in case the same intermediate step was used several times in following calculation (which is not the case of this simple example). However, the need of creating multiple nested ADDCOLUMNS is increasing the length of the code and, in this specific example, is affecting performance in a negative way (because part of the large materialization required by the cardinality of the calculation).

By using variables in DAX it is possible to obtain the same efficiency of the initial code and an improved readability obtained by splitting a complex calculation in several smaller steps, giving a name to each one. In the next example, you can see the final result you can obtain by using variables.

```
TaxedSalesVariables := SUMX (
    Sales,
    VAR LineAmount = Sales[Quantity] * Sales[Unit Price]
    VAR Taxes = LineAmount * Sales[Tax Percentage]
    VAR TaxedAmount = [LineAmount] + [Taxes]
    RETURN
        TaxedAmount )

```

VIEW IN DAX FORMATTER

Variables for tables

When you start using variables, you might not realize that a variable can store a table and not only a scalar value. This feature is useful whenever you have the same filter repeated several times in the same DAX expression. While this is certainly not a frequent situation, it could be helpful in complex and long expression. For example, the following formula of the [Time Patterns] has a similar expression in the two branches of the IF statement.

```
[PM Sales] :=
SUMX (
    VALUES ( 'Date'[YearMonthNumber] ),
    IF (
        CALCULATE ( COUNTROWS ( VALUES ( 'Date'[Date] ) ) )
            = CALCULATE ( VALUES ( 'Date'[MonthDays] ) ),
        CALCULATE (
            [Sales],
            ALL ( 'Date' ),
            FILTER (
                ALL ( 'Date'[YearMonthNumber] ),
                'Date'[YearMonthNumber]
                    = EARLIER ( 'Date'[YearMonthNumber] ) - 1
            )
        ),
        CALCULATE (
            [Sales],
            ALL ( 'Date' ),
            CALCULATETABLE ( VALUES ( 'Date'[MonthDayNumber] ) ),
            FILTER (
                ALL ( 'Date'[YearMonthNumber] ),
                'Date'[YearMonthNumber]
                    = EARLIER ( 'Date'[YearMonthNumber] ) - 1
            )
        )
    )
)
```

VIEW IN DAX FORMATTER

By using a variable, you can store the result of the FILTER that is common to the two CALCULATE used in the two branches of the IF statement. The result of the FILTER applied to the YearMonthNumber column of the Date table is the same in both cases, and assigning it to a variable makes the code more readable, as you can see in the following example.

```
[PM Sales] :=
VAR PreviousYearMonth =
    FILTER (
        ALL ( 'Date'[YearMonthNumber] ),
        'Date'[YearMonthNumber]
            = EARLIER ( 'Date'[YearMonthNumber] ) - 1
    )
RETURN
    SUMX (
        VALUES ( 'Date'[YearMonthNumber] ),
        IF (
            CALCULATE ( COUNTROWS ( VALUES ( 'Date'[Date] ) ) )
                = CALCULATE ( VALUES ( 'Date'[MonthDays] ) ),
            CALCULATE (
                [Sales],
                ALL ( 'Date' ),
                PreviousYearMonth
            ),
            CALCULATE (
                [Sales],
                ALL ( 'Date' ),
                CALCULATETABLE ( VALUES ( 'Date'[MonthDayNumber] ) ),
                PreviousYearMonth
            )
        )
    )
```

VIEW IN DAX FORMATTER

I experienced a successful use of variables storing tables in much longer and complex expressions. Even if this could provide a performance improvement in certain conditions, the most important reason for using variables is code readability. Providing a name to intermediate steps of a calculation is also an excellent way to self-document your DAX code.

Read More at: http://www.sqlbi.com/articles/dax-coding-style-using-variables/?utm_source=wysija&utm_medium=email&utm_campaign=17-2016

Contact Information

Prosys Infotech Private Limited, Survey 1/8/1 ,Salunkhe Crystal 2nd floor, Lane Opposite Bank of Maharashtra, NDA-Pashan Road, Pune 411021, INDIA.

Landline—020-65002583, 020-65002584, Sanjay-9422034643 sanjay@prosysinfotech.com