Lesson-9

Topic: Understanding Context in DAX & CALCULATE and Basic Filters, Variables

1.What is row context? Give an example in a calculated column.

Row Context in DAX

Row context is one of the fundamental concepts in DAX (Data Analysis Expressions) that determines how formulas are evaluated in Power BI, Analysis Services, and Power Pivot.

What is Row Context?

Row context is the context that's automatically created when a formula is evaluated row by row in a table. It's essentially the "current row" that's being evaluated in a calculation. This context exists in:

Calculated columns

Iterator functions (like SUMX, AVERAGEX, etc.)

Filter context modifications (like EARLIER)

Example in a Calculated Column

Let's say you have a Sales table with these columns:

Quantity (number of items sold)

UnitPrice (price per item)

You want to create a calculated column for "Line Total" (Quantity × UnitPrice):

Create a new calculated column in the Sales table

Use this formula:

dax

Line Total = Sales[Quantity] * Sales[UnitPrice]

In this example:

When the formula is evaluated for each row, DAX automatically creates a row context

Sales[Quantity] refers to the Quantity value in the current row being calculated

Sales[UnitPrice] refers to the UnitPrice value in the current row

The calculation is performed row by row for the entire table

Key Points About Row Context

It's automatically created in calculated columns

It's temporary and exists only during the evaluation of the formula

It doesn't filter data like filter context does

You can create row context explicitly using iterator functions (X-functions)

2.Write a measure that finds total sales

Dax:
Total Sales = Sumx(Sales[Quantity] * Sales[UnitPrice])

How This Measure Works:

SUMX is an iterator function that:

Creates a row context for each row in the Sales table

Calculates Quantity × UnitPrice for each row

Sums up all these individual line totals

The measure will dynamically respond to any filters applied in your report (like filtering by ProductID, CustomerID, or Category)

3.Use RELATED to fetch the Name from the Customers table into the Sales table.

Creating a Customer Name Measure for Sales Table

Since you want to display customer names without adding a calculated column to your Sales table, you can create a measure that looks up the customer name from the Customers table. Here's how to do it:

Using LOOKUPVALUE in a Measure

dax

```
Customer Name =
LOOKUPVALUE(
    Customers[CustomerName],  // Column to fetch
    Customers[CustomerID],    // Search column
    SELECTEDVALUE(Sales[CustomerID])  // Value to match
)
```

How This Measure Works:

LOOKUPVALUE searches the Customers table for:

The CustomerName where

The CustomerID matches the currently filtered Sales[CustomerID]

SELECTEDVALUE returns the CustomerID in the current filter context (or blank if there are multiple values)

4.What does CALCULATE(SUM(Sales[Quantity]), Sales[Category] = "Electronics") return?

Understanding the CALCULATE Function

The expression CALCULATE(SUM(Sales[Quantity]), Sales[Category] = "Electronics") performs a conditional sum with context transition. Here's what it returns:

The Result

This formula returns the total quantity of all sales where the category is "Electronics", respecting any additional filters that might be applied in your report.

Breaking It Down:

SUM(Sales[Quantity])

The base calculation that sums the Quantity column

CALCULATE(..., Sales[Category] = "Electronics")

Modifies the filter context to only include rows where Category = "Electronics"

This filter overrides any existing filters on the Category column (but combines with other filters)

Context Behavior

If used in a visual with other filters (like by year or region), it will only sum Electronics quantities within those filters

If used without any other filters, it sums all Electronics quantities in the entire table

5.Explain the difference between VAR and RETURN in DAX.

Understanding VAR and RETURN in DAX

VAR and RETURN are essential components of DAX that work together to create more readable and efficient measures. Here's a clear explanation of their differences and how they interact:

VAR (Variable Declaration)

Purpose: Stores intermediate calculations or values

Behavior:

Creates a named variable that holds a value or table

Evaluates immediately when declared (eager evaluation)

Can be used multiple times in the subsequent calculation

Syntax:

VAR VariableName = Expression

RETURN

Purpose: Specifies the final result of the measure

Behavior:

Must appear after all VAR declarations

Can reference any variables declared before it

Contains the expression that becomes the measure's output

Syntax:

RETURN ResultExpression

Key Differences

Feature	VAR	RETURN
Purpose	Stores intermediate values	Defines final output
Position	Before RETURN	Must come last
Quantity	Multiple allowed	Only one per measure
Evaluation	Happens immediately	Evaluates last

Example Showing Both

dax

```
Total Electronics Sales =

VAR ElectronicsCategory = "Electronics"  // VAR stores a value

VAR FilteredSales = FILTER(Sales, Sales[Category] = ElectronicsCategory)  // VAR stores a table

VAR TotalQty = SUMX(FilteredSales, Sales[Quantity] * Sales[UnitPrice])  // VAR stores a calculation

RETURN TotalQty  // RETURN outputs the final result
```

6.Create a calculated column in Sales called TotalPrice using row context (Quantity * UnitPrice).

Row Context: The calculation automatically iterates through each row in the table

UnitPrice Handling:

SUBSTITUTE removes the "$" character from UnitPrice

VALUE converts the text to a numeric value

Multiplication: Each row's Quantity is multiplied by its cleaned UnitPrice

Implementation Steps

In Power BI Desktop:

Right-click your Sales table

Select "New column"

Enter the formula above

Press Enter or click the checkmark to confirm

```
TotalPrice = Sales[Quantity] * Sales[UnitPrice]
```

7.Write a measure Electronics Sales using CALCULATE to sum sales only for the "Electronics" category.

To create a measure in DAX that sums sales only for the "Electronics" category, you can use the CALCULATE function with a filter. Here's the measure:

```
Electronics Sales :=
CALCULATE(
    SUM(Sales[SalesAmount]),
    'Products'[Category] = "Electronics"
)
```

Explanation:

SUM(Sales[SalesAmount]): This part calculates the total sales amount.

'Products'[Category] = "Electronics": This filter ensures only rows where the category is "Electronics" are included.

CALCULATE modifies the context to apply the filter during the summation.

8.Use ALL(Sales[Category]) in a measure to show total sales ignoring category filters.

Creating a Measure That Ignores Category Filters Using ALL()

To create a measure that calculates total sales while ignoring any filters applied to the Category column, you'll use the ALL() function to remove filter context from the Category column.

Solution: Total Sales Ignoring Category Filters

If your UnitPrice is already stored as a numeric value (without "$"):

dax

```
Total Sales All Categories =
CALCULATE(
   SUM(Sales[Quantity] * Sales[UnitPrice]),
   ALL(Sales[Category])
)
```

When to Use This

When you need to show "grand total" values alongside filtered data

In dashboard headers or KPIs that should remain constant

When comparing filtered values to overall totals

Important Notes

This only removes filters from the Category column - other filters still apply

To remove all filters, use ALL(Sales)

The measure will still respect filters on other columns (like Date or Region)

9.Fix this error: A calculated column in Sales uses RELATED(Customers[Region]) but returns blanks.

The issue with RELATED(Customers[Region]) returning blanks in your Sales calculated column typically stems from one of these common problems:

Most Likely Causes and Solutions

1. Missing or Inactive Relationship

Problem: No relationship exists between Sales and Customers tables, or it's inactive

Solution:

dax

```
// First verify relationships
Customer Region =
IF(
   ISBLANK(LOOKUPVALUE(Customers[Region], Customers[CustomerID], Sales[CustomerID])),
   "Relationship issue",
   RELATED(Customers[Region])
)
```

If this shows "Relationship issue", you need to:

Create a relationship between Sales[CustomerID] and Customers[CustomerID]

Ensure the relationship is active (solid line in diagram view)

2. CustomerID Mismatches

Problem: Values in Sales[CustomerID] don't exist in Customers[CustomerID]

Solution:

dax

```
Customer Region =
IF(
   CONTAINS(Customers, Customers[CustomerID], Sales[CustomerID]),
   RELATED(Customers[Region]),
   "Missing customer"
)
```

3. Data Type Mismatch

Problem: CustomerID columns have different data types

Solution:

Check both CustomerID columns are the same type (both text or both numeric)

Convert if needed:

dax

```
Customer Region =
VAR CurrentCustomerID = FORMAT(Sales[CustomerID], "0")
RETURN
RELATED(LOOKUPVALUE(Customers[Region], FORMAT(Customers[CustomerID], "0"), CurrentCustomerID))
```

4. Using RELATED in a Measure

Problem: RELATED only works in row context (calculated columns)

Alternative for measures:

dax

```
Customer Region =
LOOKUPVALUE(
    Customers[Region],
    Customers[CustomerID], SELECTEDVALUE(Sales[CustomerID])
)
```

Complete Working Solution

For a calculated column that handles all cases:

dax

```
Customer Region =
VAR CurrentCustomerID = Sales[CustomerID]
RETURN
IF(
    CONTAINS(Customers, Customers[CustomerID], CurrentCustomerID),
    RELATED(Customers[Region]),
    BLANK()
)
```

Verification Steps

Check relationship exists in Model view

Verify no filter context is removing the relationship

Confirm CustomerID values match exactly (including spaces/case)

Test with LOOKUPVALUE first to isolate the issue

10.Why does CALCULATE override existing filters?

CALCULATE overrides existing filters because it changes the filter context in which a DAX expression is evaluated. This is one of its key features and makes it very powerful in data modeling and analysis.

Here's why and how it works:

CALCULATE creates a new filter context:

When you use CALCULATE, you tell DAX to evaluate the expression using specific filters, even if other filters already exist in the report or pivot table.

Overrides or adds to existing filters:

If there's already a filter on the same column, CALCULATE replaces it.

If no filter exists yet on a column, CALCULATE adds a new one.

Example:

Suppose you have a report filtered to show only "Clothing", but your measure is:

```
Electronics Sales :=
CALCULATE(
   SUM(Sales[SalesAmount]),
   'Products'[Category] = "Electronics"
)
```

Even though the report is filtered to "Clothing", CALCULATE overrides that filter and forces the evaluation to happen only for Electronics.

11.Write a measure that returns average unitprice of products

Here's a DAX measure that returns the average unit price of products:

```
Average Unit Price :=
AVERAGE(Products[UnitPrice])
```

Explanation:

AVERAGE: A DAX function that calculates the arithmetic mean of a column.

Products[UnitPrice]: Refers to the column that holds the unit prices of the products.

Make sure the UnitPrice column is numeric (e.g., Decimal or Currency) for this to work correctly.

12.Use VAR to store a temporary table of high-quantity sales (Quantity > 2), then count rows.

Here's a DAX measure that uses the VAR keyword to store a temporary table of sales where Quantity > 2, and then counts the rows:

```
High Quantity Sales Count :=
VAR HighQuantitySales =
    FILTER(Sales, Sales[Quantity] > 2)
RETURN
    COUNTROWS(HighQuantitySales)
```

Explanation:

VAR HighQuantitySales: Stores a filtered table from the Sales table where Quantity > 2.

RETURN COUNTROWS(...): Counts how many rows (i.e., sales transactions) meet the condition.

This is useful when you want to reuse intermediate results or make your code more readable and efficient.

13.Write a measure % of Category Sales that shows each sale's contribution to its category total.

Here's a DAX measure that calculates % of Category Sales, showing each sale's contribution to its category's total sales:

```
% of Category Sales :=
DIVIDE(
    SUM(Sales[SalesAmount]),
    CALCULATE(
        SUM(Sales[SalesAmount]),
        ALLEXCEPT('Products', Products[Category])
    )
)
```

Explanation:

SUM(Sales[SalesAmount]): The current sale or row’s sales amount.

CALCULATE(..., ALLEXCEPT(...)): Calculates total sales for the category, removing all filters except for the current category.

DIVIDE(...): Safely performs division, avoiding divide-by-zero errors.

Make sure there’s a relationship between the Sales and Products tables via something like ProductID.

14.Simulate a "remove filters" button using ALL in a measure.

Creating a "Remove Filters" Button Using ALL() in a Measure

You can simulate a "remove filters" button functionality by creating a measure that ignores specific filters using the ALL() function. Here's how to implement this:

Basic "Remove Filters" Measure

dax

```
Total Sales Ignoring Filters =
CALCULATE(
   [Total Sales],  // Your base sales measure
   ALL(
      Sales[Category],  // Remove filters from Category
      Sales[CustomerID] // Remove filters from CustomerID
      // Add other columns you want to ignore
   )
)
```

Dynamic Version (Removes All Filters on Sales Table)

dax

```
Total Sales All Data =
CALCULATE(
   [Total Sales],
   ALL(Sales)  // Removes all filters from the Sales table
)
```

How to Use This as a "Button"

Create the measure as shown above

Add a card visual to your report

Place the measure in the card

Add visual interactions:

Go to "Format" pane

Select "Edit interactions"

Set this card to "None" for cross-filtering

Advanced Implementation with Parameter Table

For more control, create a parameter table:

Create a disconnected table:

dax

```
Filter Control = DATATABLE("Action", STRING, {{"Show Filtered"}, {"Show All"}})
```

Create a measure that responds to selection:

dax

```
Dynamic Sales View =
SWITCH(
    SELECTEDVALUE('Filter Control'[Action], "Show Filtered"),
    "Show Filtered", [Total Sales],
    "Show All", CALCULATE([Total Sales], ALL(Sales))
)
```

Add a slicer with the 'Filter Control'[Action] column

Use the 'Dynamic Sales View' measure in visuals

Key Benefits

Visual indication of filtered vs. unfiltered states

User control through slicer interaction

Flexible implementation - can remove filters from specific columns only

No VBA or macros required - pure DAX solution

Important Notes

This doesn't actually remove filters from other visuals - it just shows what the values would be without filters

For a true "reset all filters" button, you would need to use Power BI's built-in buttons or bookmarks

The ALL() function only affects the calculation context of this specific measure

15.Troubleshoot: A CALCULATE measure ignores a slicer. What’s the likely cause?

Common Causes:

☑ 1. Use of ALL or REMOVEFILTERS

These functions remove filters, including those applied by slicers.

DAX

Копировать

Редактировать

CALCULATE(

```
    SUM(Sales[SalesAmount]),
    ALL(Products[Category])  -- This ignores the slicer on Category
)
```

☑ 2. No relationship between slicer table and fact table

If your slicer is based on a table that is not related (or incorrectly related) to the table used in the measure, the slicer won't affect the result.

☑ 3. Slicer is on a column not referenced in the filter context

For example, if your slicer is on Products[Brand], but your measure filters only by Products[Category], and you use ALL(Products), you remove both filters.

☑ Fix Tips:

Avoid using ALL() unless needed, or use ALLEXCEPT() to preserve important slicers:

```
CALCULATE(
    SUM(Sales[SalesAmount]),
    ALLEXCEPT(Products, Products[Category])
)
```

Check relationships between slicer table and fact table.

Use KEEPFILTERS() if you want to add filters without removing slicers.