Lesson 9

Topic: Understanding Context in DAX & CALCULATE and Basic Filters, Variables
Prerequisites: Download DAX_Context_Practice.xlsx file.

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

   Row Context = DAX evaluates each row individually.

   ◆ Applies in:

   - Calculated Columns
   - Iterating functions (like SUMX, FILTER)

   Example (Calculated Column)

   DAX

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

   Key Points to Remember

- Row context happens **automatically** in calculated columns.
- Each row is evaluated **separately**.
- No aggregations or filters are applied unless added manually.

2. **Write a measure that finds total sales**

   Here's a simple DAX measure that calculates the total sales amount:

   DAX Measure:

   DAX

   Total Sales = SUM(Sales[Quantity] * Sales[UnitPrice])

   But this won't work directly, because SUM() cannot multiply two columns row by row.

   So, you should use SUMX, which supports row-by-row calculation:

Correct DAX Measure:

```
DAX
Total Sales =
SUMX(
    Sales,
    Sales[Quantity] * Sales[UnitPrice]
)
```

Explanation:

- SUMX(table, expression): Evaluates the expression for each row in the Sales table, then adds the results.
- Here, Quantity * UnitPrice is calculated per row, then summed.

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

. Use RELATED to Fetch the Customer Name into the Sales Table

If there is a relationship between the Sales table and the Customers table through CustomerID, you can use the RELATED function in a calculated column to bring in the customer's name.

DAX Calculated Column in Sales Table:

```
DAX
Customer Name = RELATED(Customers[Name])
```

How It Works:

- RELATED() fetches the value from the related table (Customers) for the current row in Sales.
- It uses the relationship via CustomerID.

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

   What does this DAX return?

   DAX

   CALCULATE(SUM(Sales[Quantity]), Sales[Category] = "Electronics")

   Explanation:

   This formula:

   - Changes the filter context to include only rows where Category = "Electronics",
   - Then sums the Quantity column for those rows.

   It returns:

   The total quantity of products sold where the category is "Electronics".

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

   VAR

   - Used to define a variable in DAX.
   - It stores the result of an expression so it can be reused multiple times.
   - Improves performance and readability.

   🔹 RETURN

   - Tells DAX what to return as the final result of the expression.
   - It uses the variable(s) defined in VAR.

   Example:

   DAX

   Average Profit =

```
VAR TotalSales = SUM(Sales[Quantity] * Sales[UnitPrice])
VAR TotalCost = SUM(Sales[Quantity] * Sales[UnitCost])
RETURN
    (TotalSales - TotalCost) / TotalSales
```

- VAR TotalSales stores total sales
- VAR TotalCost stores total cost
- RETURN gives back the profit margin using the above variables

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

   Create a Calculated Column: TotalPrice in Sales Table

   You can create this calculated column using row context, which automatically applies in calculated columns.

   DAX Calculated Column:

   DAX

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

   How it Works:

- This uses row context to multiply the Quantity and UnitPrice for each row in the Sales table.
- No need for SUMX or any aggregation here — it's evaluated row by row.

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

   Here's the DAX measure that sums sales amount only for the "Electronics" category using CALCULATE.

DAX Measure:

DAX

CopyEdit

```
Electronics Sales =
CALCULATE(
    SUMX(Sales, Sales[Quantity] * Sales[UnitPrice]),
    Sales[Category] = "Electronics"
)
```

Explanation (English):

- SUMX multiplies Quantity * UnitPrice for each row to get total sale per transaction.
- CALCULATE filters only rows where Category = "Electronics".

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

Use ALL(Sales[Category]) to Ignore Category Filters in a Measure

To show total sales regardless of the selected category in a visual (i.e., ignoring any slicer or filter on Sales[Category]), use the ALL() function inside CALCULATE.

DAX Measure:

DAX

```
Total Sales (Ignore Category) =
CALCULATE(
    SUMX(Sales, Sales[Quantity] * Sales[UnitPrice]),
    ALL(Sales[Category])
)
```

Explanation (English):

- SUMX calculates total sales per row.
- CALCULATE changes the filter context.
- ALL(Sales[Category]) removes any filter applied to Category — so you always get the grand total sales.

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

Fixing the Error: RELATED(Customers[Region]) Returns Blanks in Sales

Problem:

You created a calculated column in the Sales table like this:

DAX

CopyEdit

```
Region = RELATED(Customers[Region])
```

But it's returning blank values.

Root Cause:

Blanks from RELATED() usually mean:

There is no relationship between Sales[CustomerID] and Customers[CustomerID].

Fix:

1. Go to Model view in Power BI.
2. Check that a relationship exists:
   - Sales[CustomerID] → Customers[CustomerID]

3. If not, create a one-to-many relationship:
    - From Customers[CustomerID] (primary)
    - To Sales[CustomerID] (foreign)

   Once the relationship is in place, RELATED(Customers[Region]) will work correctly.

10. Why does CALCULATE override existing filters?

Why Does CALCULATE Override Existing Filters?

In English:

CALCULATE is powerful in DAX because it can modify, replace, or add filters to the current filter context.

Why it overrides existing filters:

- When you use CALCULATE, you pass in new filter expressions.
- These replace any filters on the same columns already in the visual or report.
- This allows you to control the logic of your calculations, regardless of slicers or filters.

Example:

DAX

```
CALCULATE(
   SUM(Sales[Quantity]),
   Sales[Category] = "Electronics"
)
```

Even if a slicer filters for “Clothing,” this formula forces the result to show only Electronics.

11. **Write a measure that returns average unitprice of products**

Write a Measure to Return Average Unit Price of Products

To calculate the average unit price of products in DAX, you can use the AVERAGE() function.

DAX Measure:

DAX

```
Average Unit Price = AVERAGE(Sales[UnitPrice])
```

Explanation :

- This measure calculates the average of the UnitPrice column across all rows in the Sales table.
- Automatically respects slicers and filters (e.g., by product, category, date, etc.)

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

Use VAR to Store a Temporary Table of High-Quantity Sales, Then Count Rows

You can use VAR in a measure to define a temporary table and then apply a DAX function like COUNTROWS() to it.

DAX Measure:

DAX

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

Explanation:

- VAR HighQtySales stores a filtered table where Quantity > 2.
- RETURN uses COUNTROWS() to count how many rows meet that condition.

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

Measure: % of Category Sales — Each Sale's Contribution to Its Category Total

To calculate the percentage of total sales within each category, use the DIVIDE function along with CALCULATE and ALL.

DAX Measure:

DAX

```
% of Category Sales =
DIVIDE(
    SUMX(Sales, Sales[Quantity] * Sales[UnitPrice]),
    CALCULATE(
        SUMX(Sales, Sales[Quantity] * Sales[UnitPrice]),
        ALLEXCEPT(Sales, Sales[Category])
    )
)
```

Explanation:

- The numerator: total sale amount for current row or group.
- The denominator: total sales within the same category, ignoring other filters (thanks to ALLEXCEPT).
- DIVIDE() safely handles division and avoids errors from zero.

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

Simulate a "Remove Filters" Button Using ALL in a Measure

Goal:

You want a measure that ignores user-applied filters (e.g., slicers, visuals) — like clicking a button to show the grand total.

DAX Measure Example:

```
DAX
Total Sales (Ignore Filters) =
CALCULATE(
    SUMX(Sales, Sales[Quantity] * Sales[UnitPrice]),
    ALL(Sales)
)
```

How It Works (English):

- SUMX(Sales, Quantity * UnitPrice) calculates the total sales amount.
- ALL(Sales) removes all filters from the Sales table — as if no slicer or filter was applied.
- CALCULATE forces the measure to ignore visual/report filters.

This behaves like a "Clear Filters" or "Reset View" button.

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

Troubleshoot: A CALCULATE Measure Ignores a Slicer — What's the Likely Cause?

🔍 Likely Cause:

The most common reason is that the measure uses a function like ALL() or REMOVEFILTERS() inside CALCULATE(), which removes slicer filters.

Example:

```
DAX

Total Sales (All) =

CALCULATE(

    SUM(Sales[Amount]),

    ALL(Sales[Category])

)
```

If you use a slicer on Category, this measure will ignore the slicer, because ALL(Sales[Category]) removes the slicer's filter on that column.