

Lesson-8

1.What does DAX stand for?

Key Points About DAX:

It is a formula language used in Power BI, Power Pivot (Excel), and SQL Server Analysis Services (SSAS).

It is designed to work with tabular data models, enabling calculations and aggregations on imported data.

While similar to Excel formulas, DAX is optimized for data modeling and analytics (e.g., handling relationships, time intelligence, and dynamic aggregations).

Common Uses of DAX:

Creating measures (dynamic calculations like sums, averages, or ratios).

Defining calculated columns (static column-level computations).

Implementing business logic (e.g., year-over-year comparisons, moving averages, or custom KPIs).

2.Write a DAX formula to sum the Sales column.

Total Sales = Sum(Sales[Sales])

3.What is the difference between a calculated column and a measure?

1. Calculated Column

Definition: A column added to a table that computes values row by row.

Storage: Physically stored in the data model (increases file size).

Calculation Timing: Computed during data refresh (static until next refresh).

Use Case: Best for:

Row-level calculations (e.g., Profit = [Revenue] - [Cost])

Filtering or grouping in visuals

Creating static categorizations (e.g., "High/Low" flags)

Example:

dax

Profit Margin = DIVIDE([Revenue] - [Cost], [Revenue], 0)

2. Measure

Definition: A dynamic calculation performed at query time (when interacting with visuals).

Storage: Not stored; computed on the fly (minimal impact on file size).

Calculation Timing: Evaluated when a visual is rendered or filtered.

Use Case: Best for:

Aggregations (e.g., sums, averages)

Ratios or percentages (e.g., Total Sales %)

Time intelligence (e.g., YTD Sales, YoY Growth)

Example:

dax

Total Sales = SUM(Sales[Amount])

Key	Differences	Summary
-----	-------------	---------

Feature	Calculated Column	Measure
---------	-------------------	---------

Evaluation	Row-by-row	Aggregated across filters
------------	------------	---------------------------

Storage	Stored in model (increases size)	Not stored (calculated live)
---------	----------------------------------	------------------------------

Performance	Faster in visuals (pre-computed)	Slower (runtime calculation)
-------------	----------------------------------	------------------------------

Usage Filtering, grouping, row logic KPIs, totals, dynamic analysis

Recalculation Only on data refresh With every visual interaction

When to Use Which?

Use a calculated column when you need to:

Filter or slice by a derived value.

Reference row-level values in other calculations.

Use a measure when you need to:

Calculate totals, averages, or ratios.

Respond dynamically to user selections (slicers/filters).

4. Use the DIVIDE function to calculate Profit Margin (Profit/Sales).

To calculate Profit Margin using the DIVIDE function in DAX (which safely handles division by zero), you can use the following measure:

dax

DIVIDE(

[Profit],

[Sales],

0 -- Optional: returns 0 if Sales is 0 instead of error

)

 Explanation:

[Profit] and [Sales] should be existing measures or columns.

DIVIDE(x, y, 0) safely returns 0 when y = 0 (instead of throwing an error).


5. What does COUNTROWS() do in DAX?

The COUNTROWS() function counts the number of rows in a table.

Dax

COUNTROWS(<table>)

<table>: Can be a physical table, filtered table, or a table expression (like VALUES(), FILTER(), etc.).

 Examples:

1. Count rows in a full table

dax

Total Movies :=

COUNTROWS(filtered_movies)


 Returns the total number of movies.

2. Count distinct genres

Dax

Genre Count :=

COUNTROWS(VALUES(genres[genre]))


 Returns the number of unique genres.

3. Count rows with filter

dax

High Rated Movies :=

COUNTROWS(
 FILTER(filtered_movies, filtered_movies[rating] > 8)
)

 Returns the number of movies with a rating above 8.

 Summary:

COUNTROWS() is great for summarizing, filtering, or calculating totals.

Works well with dynamic filters in visuals or slicers.

Common in KPIs like “Number of Customers”, “Number of Orders”, etc.

6. Create a measure: Total Profit that subtracts total cost from total sales

To create a DAX measure for Total Profit (Sales - Cost), use the following:

dax

Total Profit :=

SUM(Sales[Sales]) - SUM(Sales[Cost])

 Explanation:

SUM(Sales[Sales]): Calculates total revenue.

SUM(Sales[Cost]): Calculates total cost.

The difference gives total profit.

Make sure Sales[Sales] and Sales[Cost] are numeric columns in your data model.

7. Write a measure to calculate Average Sales per Product.

To calculate Average Sales per Product in DAX, use this measure:

dax

Average Sales per Product :=

DIVIDE(
 SUM(Sales[Sales]),
 DISTINCTCOUNT(Sales[ProductID])
)

 Explanation:

SUM(Sales[Sales]): Total sales amount.

DISTINCTCOUNT(Sales[ProductID]): Counts how many unique products were sold.

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

8. Use IF() to tag products as "High Profit" if Profit > 1000.

Here's a DAX measure (or calculated column) using IF() to tag products as "**High Profit**" if profit exceeds 1000:

☒ **Option 1: Calculated Column**

If you want to tag each row (product-level):

```
Profit Category :=  
IF(  
    Sales[Profit] > 1000,  
    "High Profit",  
    "Regular Profit"  
)
```

💡 This assumes `Sales[Profit]` is a column (not a measure).

☑ Option 2: Measure-Based Label

If you're using a **measure** for profit (like `Total Profit`), and want to label at aggregated level:

```
Profit Category :=  
IF(  
    [Total Profit] > 1000,  
    "High Profit",  
    "Regular Profit"  
)
```

Use this in a visual (like a matrix or card) where context (like product name) is already applied.

9. What is a circular dependency error in a calculated column?

Circular Dependency Error in Calculated Columns (DAX)

A circular dependency occurs when two or more calculations reference each other in a loop, making it impossible for Power BI to determine the correct evaluation order. This is a common issue in calculated columns (but can also affect measures).

Why It Happens

Calculated Column A depends on Column B.

Column B (or another column) depends back on Column A.

Power BI cannot resolve which should be calculated first.

Example Scenario

dax

// Column 1

`Sales[Profit] = Sales[Revenue] - Sales[Cost]`

// Column 2 (Circular Dependency!)

`Sales[AdjustedCost] = Sales[Cost] + Sales[Profit] * 0.1`

Here, AdjustedCost depends on Profit, but Profit depends on Cost—creating a loop.

How to Fix Circular Dependencies

1. Break the Loop with a Measure

Replace one of the calculated columns with a measure (since measures don't create storage dependencies).

Measures calculate at query time, avoiding the loop.

2. Use DISTINCT() or VALUES() (For Lookup Tables)

If the dependency involves a lookup table, use:

dax

// Instead of direct column reference

Sales[Category] = LOOKUPVALUE(Products[Category], Products[ID], Sales[ProductID])

Ensure no reverse dependencies exist.

3. Rebuild the Data Model

Sometimes, circular dependencies arise from poor table relationships.

Check if bi-directional filtering is causing the issue (switch to single-direction).

Common Causes

Self-referencing columns:

dax

Table[ColumnX] = Table[ColumnX] + 1 // Invalid!

Chained dependencies:

$A \rightarrow B \rightarrow C \rightarrow A$ (indirect loops).

Bi-directional relationships with filtering conflicts.

10.Explain row context vs. filter context.

Understanding these two evaluation contexts is crucial for writing correct DAX formulas in Power BI.

1. Row Context

Definition: A "row-by-row" calculation environment where DAX evaluates expressions for each individual row in a table.

When It Applies:

Automatically created in calculated columns.

Manually created by iterator functions like SUMX(), FILTER(), EARLIER().

Key Behavior:

Does not automatically apply filters from slicers or visuals.

Only "sees" the current row's values.

Example (Row Context)

dax

-- Calculated column (Row Context)

Sales[Profit] = Sales[Revenue] - Sales[Cost]

→ Computes Profit for each row separately.

dax

-- Iterator function (SUMX creates Row Context)

Total Profit = SUMX(Sales, Sales[Revenue] - Sales[Cost])

→ Calculates Revenue - Cost per row, then sums results.

2. Filter Context

Definition: The set of filters applied by slicers, visuals, or DAX functions (like CALCULATE()) that affect aggregation.

When It Applies:

Automatically created by report interactions (slicers, visuals).

Explicitly modified using CALCULATE().

Key Behavior:

Controls which rows are included in calculations.

Overrides Row Context when used with CALCULATE().

Example (Filter Context)

dax

-- Measure (affected by Filter Context)

Total Sales = SUM(Sales[Amount])

→ If a user filters by Year = 2023, only 2023 sales are summed.

dax

-- CALCULATE modifies Filter Context

Sales 2023 = CALCULATE([Total Sales], Sales[Year] = 2023)

→ Forces the calculation to consider only 2023 data, regardless of external filters.

11. Write a measure to calculate YTD Sales using TOTALYTD().

YTD Sales Measure:

dax

YTD Sales :=

```
TOTALYTD(  
    SUM(Sales[Sales]),  
    Sales[Date]  
)
```

 Explanation:

SUM(Sales[Sales]): Calculates total sales.

Sales[Date]: The date column used to track time. This must be a proper date column in your model.

TOTALYTD() automatically aggregates values from the start of the year up to the selected date.

 Requirements:

You must have a date table marked as a Date Table in your model.

Sales[Date] must be related to that date table for proper time intelligence.

12. Create a dynamic measure that switches between Sales, Profit, and Margin.

Step 1: Create a Slicer Table (Disconnected)

Go to Modeling > New Table and add:

Metric Selector =

```
DATATABLE(  
    "Metric", STRING,  
    {  
        {"Sales"},  
        {"Profit"},  
        {"Margin"}  
    }  
)
```


)

This creates a simple table with metric names for the user to choose via a slicer.

☒ Step 2: Create Measures

Make sure you already have these base measures (or define them):

Total Sales := SUM(Sales[Sales])

Total Profit := SUM(Sales[Sales]) - SUM(Sales[Cost])

Profit Margin := DIVIDE([Total Profit], [Total Sales], 0)

☒ Step 3: Create the Dynamic Measure

Now create the final measure:

Selected Metric :=

```
SWITCH(
    SELECTEDVALUE('Metric Selector'[Metric]),
    "Sales", [Total Sales],
    "Profit", [Total Profit],
    "Margin", [Profit Margin],
    BLANK()
)
```

☒ Step 4: Use in a Visual

Add a slicer using the Metric Selector[Metric].

Use the Selected Metric in your visuals — it will update based on the slicer choice.

13. Optimize a slow DAX measure using variables (VAR).

Example: Before Optimization

Here's a slow-performing measure that repeats the same calculation multiple times:

```
High Margin Sales :=
CALCULATE (
    SUM(Sales[Sales]),
    FILTER (
```

```

        Sales,
        (Sales[Sales] - Sales[Cost]) / Sales[Sales] > 0.3
    )
)

```

This recomputes $(\text{Sales}[\text{Sales}] - \text{Sales}[\text{Cost}]) / \text{Sales}[\text{Sales}]$ for each row, which is inefficient.

Optimized Version Using VAR

```

High Margin Sales :=
CALCULATE(
    SUM(Sales[Sales]),
    FILTER(
        Sales,
        VAR SalesAmount = Sales[Sales]
        VAR CostAmount = Sales[Cost]
        VAR Margin = DIVIDE(SalesAmount - CostAmount, SalesAmount)
        RETURN
            Margin > 0.3
    )
)

```

Benefits:

- **Better performance:** Expensive expressions are calculated once per row.
- **Easier to read** and debug.
- Avoids repeated calculations.
- Works great with complex logic, multiple fields, or conditional logic.

14. Use CALCULATE() to override a filter

Example: Override Filter to Always Show Sales for "Electronics" Category

dax

Electronics Sales :=

```

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

```

Even if a report or slicer is filtering for a different category, this measure forces the filter to only include Electronics.

How It Works:

CALCULATE() evaluates the expression (SUM(Sales[Sales])) in a new filter context.

'Product'[Category] = "Electronics" replaces any existing filter on Category.

Another Example: Remove All Category Filters

Dax

Sales All Categories :=

```
CALCULATE(  
    SUM(Sales[Sales]),  
    REMOVEFILTERS('Product'[Category])  
)
```

This measure returns total sales across all categories, ignoring any slicers or visuals filtering Category.

Summary:

CALCULATE() is DAX's filter context modifier.

You can:

Override filters (e.g., 'Region'[Country] = "USA")

Remove filters (with REMOVEFILTERS() or ALL())

Add new filters (e.g., for time periods)

15. Write a measure that returns the highest sales amount

Measure: Highest Sales Amount

Highest Sales Amount :=

```
MAX(Sales[Sales])
```

This returns the maximum individual sale value from the Sales[Sales] column.

If You Want the Highest Total by Product, Region, etc.

You'll need to first aggregate sales and then find the max of those totals.

Example: Highest Total Sales by Product

Highest Sales per Product :=

```
MAXX(  
    VALUES(Sales[ProductID]),  
    CALCULATE(SUM(Sales[Sales]))  
)
```

 Explanation:

VALUES(Sales[ProductID]): Creates a list of distinct products.

CALCULATE(SUM(...)): Computes total sales per product.

MAXX(...): Returns the maximum of those totals.