DAX – Part 4 (Advanced Measures & Filter Functions)

1. CALCULATE

Syntax:

CALCULATE(<expression>, <filter1>, <filter2>, ...

- **Description:** Evaluates an expression in a modified filter context.
- Most important DAX function — used to apply filters dynamically.

Example:

Sales West =
CALCULATE([Total Sales], 'Region'[Name] = "West")

---

2. FILTER

Syntax:

FILTER(<table>, <condition>)

- **Description:** Returns a filtered table based on a condition. Used inside CALCULATE or iterator functions.

Example:

CALCULATE([Total Sales], FILTER('Sales', 'Sales'[Amount] > 1000))

---

3. ALL

Syntax:

ALL(<column/table>)

- **Description:** Removes filters from the specified column or table — useful for calculating grand totals or percentages.

Example:

Sales % of Total =
DIVIDE([Total Sales], CALCULATE([Total Sales], ALL('Region')))

---

4. ALLEXCEPT

Syntax:

ALLEXCEPT(<table>, <column1>, <column2>, ...)

- **Description**: Removes all filters from a table except the specified columns.

Example:

```
CALCULATE([Total Sales], ALLEXCEPT('Sales', 'Sales'[Product]))
```

- Use Case: Keeps filters on key columns (like Product) while ignoring others.

✅ Summary Comparison:

Function	Purpose	Used With
CALCULATE	Changes the filter context	Measures
FILTER	Returns filtered table (row context)	CALCULATE
ALL	Removes all filters	CALCULATE
ALLEXCEPT	Removes all filters except some	CALCULATE

1. VALUES

Syntax:
VALUES(<column>)

- **Description:** Returns a **distinct list of values** in the column, **based on the current filter context**.
- If only one value exists in context, it returns that value; if multiple, returns a table.

Example:
CALCULATE([Total Sales], VALUES('Product'[Category]))

---

2. SELECTEDVALUE

Syntax:
SELECTEDVALUE(<column>, [alternateResult])

- **Description:** Returns the **single value** from the column if **only one** is selected; otherwise returns alternateResult (or BLANK if not provided).
- **Use Case:** Ideal for titles, dynamic cards, or slicer selections.

Example:
SELECTEDVALUE('Customer'[Name], "Multiple Customers")

---

3. HASONEVALUE

Syntax:
HASONEVALUE(<column>)

- **Description:** Returns TRUE if **exactly one value** is in context for the column.
- **Use Case:** Commonly used with IF to control calculations or text display.

Example:
IF(HASONEVALUE('Region'[Name]), [Total Sales], BLANK())

✅ Summary:

Function	Returns	Use Case
VALUES	Table or scalar	Filtering or dynamic context
SELECTEDVALUE	Single value	Slicers, titles, dynamic display

HASONEVALUE	TRUE/FALSE	Conditional logic on single value

✅ What are Nested DAX Statements?

Nested DAX refers to using one DAX function **inside another**, allowing for powerful and dynamic logic — like combining IF, CALCULATE, FILTER, SWITCH, etc., to build complex calculations.

---

1. Nested IF statements

Used for multi-condition logic (like if-else-if):

```
DiscountLevel =
IF([Sales] > 10000, "High",
  IF([Sales] > 5000, "Medium", "Low"))
```

---

2. CALCULATE with FILTER inside

To apply row-level filtering inside a measure:

```
HighValueOrders =
CALCULATE([Total Orders],
   FILTER('Orders', 'Orders'[Amount] > 1000)
)
```

---

3. SWITCH with TRUE()

Used to replace long IF chains with clearer logic:

```
CategoryGroup =
SWITCH(TRUE(),
  [Profit Margin] > 0.5, "Excellent",
  [Profit Margin] > 0.3, "Good",
  [Profit Margin] > 0.1, "Average",
  "Poor")
```

---

4. IF + ISBLANK + CALCULATE

Handle blank values while aggregating data:

```
ValidSales =
IF(ISBLANK([Total Sales]), 0, CALCULATE([Total Sales]))
```

- Always check **filter context** — nesting CALCULATE, ALL, FILTER can drastically change results.

- Avoid deeply nested IF when possible — prefer SWITCH(TRUE()) for clarity.

Performance Considerations

When writing DAX, especially with complex or nested statements, keeping performance in mind is crucial for smooth Power BI reports. Here are key points to consider:

---

1. Minimize Row Context Iterations

- Functions like FILTER, SUMX, AVERAGEX iterate over tables row-by-row and can be expensive.
- Use **simple aggregations** (SUM, COUNT) when possible.
- Avoid unnecessarily large tables in iterators.

---

2. Reduce Use of CALCULATE with Complex Filters

- CALCULATE changes filter context but can slow down if combined with many nested filters or complex FILTER conditions.
- Try to simplify filters or pre-aggregate data when possible.

---

3. Avoid Nested IFs When Possible

- Deeply nested IF statements can become hard to read and slow.
- Use SWITCH(TRUE(), ...) for better clarity and often better performance.

---

4. Use Variables (VAR)

- Use VAR to store intermediate results and reuse them.
- Improves readability and prevents repeated calculations.

```
Measure =

VAR TotalSales = SUM(Sales[Amount])

RETURN

IF(TotalSales > 1000, "High", "Low")
```

---

5. Prefer FILTER with Smaller Tables

- Filtering smaller tables or filtered subsets improves performance.
- Avoid filtering entire large tables if possible.

---

6. Use ALL and ALLEXCEPT Wisely

- Removing filters on entire tables (ALL) can be costly.
- Use ALLEXCEPT or more targeted filter removal instead.

---

7. Date Tables and Relationships

- Ensure a **proper Date Table** is used with continuous dates.
- Use relationships and avoid calculated columns for filtering date ranges when possible.

---

8. Avoid Calculated Columns for Large Data

- Calculated columns run on data refresh and increase data model size.
- Prefer **measures** for on-demand calculations.

Summary:

Tip	**Why?**
Use variables (VAR)	Avoid repeated calculations
Simplify filters in CALCULATE	Avoid expensive filter context changes
Use SWITCH instead of nested IF	Clearer and sometimes faster
Avoid large iterators	Improves calculation speed
Use proper Date Table	Enables efficient time intelligence