DAX Cheat Sheet

1. WhatisDAX?

DAX (Data Analysis Expressions) is a formula language used in Power BI, Excel, and SQL Server Analysis Services (SSAS) to analyze data, create custom calculations, columns and measures, and apply filters efficiently.

Syntax: *[Measure Name] = FunctionName([ColumnName])*

Ex: *Total Sales = SUM(Sales[Amount])*

2. Difference Between Calculated Columns and Measures?

Feature	Calculated Column	Measure
Def inition	A new column added to a table using DAX.	A dynamic calculation used in visuals.
Row Context	Works row-by-row (each row gets a value).	Works over aggregated data (based on filters).
Storage	Stored in the data model (takes space).	Not stored; calculated dynamically based on filters
Usage	Used like a normal column (in slicers, rows, filters).	Used only in visuals (cards, charts, KPIs).
Perfor mance	Slower for large datasets (uses memory).	Faster and optimized (no memory used).
Example	Profit = Sales[Revenue] - Sales[Cost]	Total Sales = SUM(Sales[Revenue])

3. What is a Calculated Table in DAX?

A calculated table is a table that you create using a DAX formula instead of importing it from a data source. It’s generated within Power BI based on existing tables and data. It's stored in the model and updates with your data.

Ex: *RegionRevenue = SUMMARIZE(Sales, Sales[Region], "TotalRevenue", SUM(Sales[Revenue]))*

4. What is Row context and Filter context?

Row Context :

Row context refers to the context in which a DAX expression is evaluated for each individual row in a table. It occurs automatically in calculated columns or in functions that iterate over rows, like SUMX,calculated column.

Ex: *TotalPrice = Sales[Quantity] * Sales[Unit Price]*

Filter Context:

Filter context is created when you apply filters in visuals, slicers, or with DAX functions (like CALCULATE).

Ex: *TotalRevenue = SUM(Sales[Revenue])*

5. Name Some Dax Functions?

1. Aggregation Functions: *SUM( ), AVERAGE( ), MIN( ), MAX( ), COUNT( ), COUNTA( ), COUNTROWS( ), DISTINCTCOUNT( )*

2. Logical Functions : *IF( ), IFERROR( ), AND( ), OR( ), NOT( ), SWITCH( )*

3. Text Functions : *CONCATENATE( ), CONCATENATEX( ), LEFT( ), RIGHT( ), MID( ), LEN( ), SEARCH( ), REPLACE( ), UPPER( ),LOWER( )*

4. Date and Time Functions: *TODAY( ), NOW( ),YEAR( ), MONTH( ), DAY( ), DATEDIFF( ), EDATE( ), EOMONTH( ), DATEADD( ), CALENDAR( ), CALENDARAUTO( )*

5. Filter Functions: *FILTER( ), ALL( ), ALLEXCEPT( ), ALLSELECTED( ), VALUES( ), DISTINCT( ), KEEPFILTERS( ), CALCULATE( )*

6. Iterator Functions: *SUMX( ), AVERAGEX( ), MINX( ), MAXX( ), COUNTX( ), FILTERX( )*

7. Time Intelligence Functions: *TOTALYTD( ), TOTALMTD( ), TOTALQTD( ), DATESYTD( ), DATESMTD( ), DATESQTD( ), SAMEPERIODLASTYEAR( ), PARALLELPERIOD( ), DATEADD(*

8. Table Functions: *SUMMARIZE( ), ADDCOLUMNS( ), SELECTCOLUMNS( ), CROSSJOIN( ), UNION( ), EXCEPT( ), INTERSECT( ),GENERATESERIES( ), DATATABLE( )*

6. Difference between AVERAGE amd AVERAGEX?

AVERAGE( ) *returns the arithmetic mean of all the numeric values in a single column.*

*Ex:AvgPrice = AVERAGE(Sales[Unit Price]) * 2*

- *AVERAGE(Sales[Unit Price])* $\rightarrow$ *(100 + 200 + 150) / 3 = 150*
- *Then: 150 * 2 = 300*

📌 *Result: AvgPrice = 300*

AVERAGEX( ) *evaluates an expression for each row in a table, then returns the average of those row-level results.*

*Ex: AvgLineTotal = AVERAGEX(Sales, Sales[Quantity] * 2)*

Row-by-row Calculation:

*Row A: 2 * 2 = 4*

*Row B: 3 * 2 = 6*

*Row C: 4 * 2 = 8*

Then:

(4 + 6 + 8) / 3 = 6

📌 *Result: AvgLineTotal = 6*

7. What is Calculate function?

CALCULATE( ) *modifies the filter context of a calculation and returns the result of an expression under those new filter conditions.*

Ex:EastRevenue *= CALCULATE(SUM(Sales[Revenue]), Sales[Region] = "East")*

8. What is Summarize function?

SUMMARIZE( ) *creates a summary table by grouping rows based on one or more columns, and optionally adding aggregated columns.*

Ex: ProductRevenueTable *= SUMMARIZE(Sales, Sales[Product],"Total Revenue", SUM(Sales[Revenue]))*

9. What is DAX Studio?

DAX Studio is a free external tool used to write, test, and analyze DAX (Data Analysis Expressions) outside of Power BI.

When to Use It?

- *Use DAX Studio when:*
- *Your DAX is complex and you want to test it independently.*
- *You need to optimize performance.*
- *You're debugging large models or slow visuals.*
- *You want to analyze what your Power BI report is really doing in the background.*

10. How to Apply Multiple filters?

In DAX, multiple filters are applied by combining conditions using logical operators like && (AND) or || (OR), IN inside a FILTER( ) function or within a CALCULATE( ) function. Use FILTER( ) when the conditions are complex or involve expressions.

Ex:

```
CALCULATE(SUM(Sales[Revenue]),
FILTER(Sales, Sales[Region] = "East"
        && Sales[Category] = "Electronics"
        && Sales[Revenue] > 1000))
```

Ex 2: Filter Table for East Region + Revenue > 1000

```
FILTER(Sales, Sales[Region] = "East" && Sales[Revenue] > 1000)
```

11. What are Text functions?

Assume you have a table:

Customer Table

Customer ID	First Name	Last Name	Email
101	John	Deo	john.doe@example.com
102	Mary	Smith	mary.smith@example.com

1. ***CONCATENATE( ):*** *Joins two text strings (use* ***&*** *operator as well)*

Ex: *CONCATENATE(Customer[FirstName],Customer[LastName])*

output: *JohnDeo*

& *- Alternate to CONCATENATE Customer[FirstName] & " " & Customer[LastName])*

output: *John Deo*

2. CONCATENATEX( ):

Joins values from a column or table with a delimiter

Ex: *CONCATENATEX(Customer, Customer[FirstName], ", ")*

output: *John, Mary*

3. LEFT( ):

Gets first n characters from text

Ex: *LEFT(Customer[Email], 4)*

output: *john*

4. RIGHT( ):

Gets last n characters from text

Ex: *RIGHT(Customer[Email], 3)*

output: *com*

5. MID( ):

Returns characters from the middle of a string, based on start and length

Ex: *MID(Customer[Email], 6, 4)*

output: *doe@*

6. LEN( ):

Returns the number of characters in a text string

Ex: LEN(Customer[FirstName])

output: *4 (for John)*

7. UPPER( ):

Converts all characters to uppercase

Ex: *UPPER(Customer[LastName])*

output: *DOE*

8. LOWER( ):

Converts all characters to lowercase

Ex: *LOWER(Customer[LastName])*

output: *doe*

10. REPLACE( ): *Replaces part of a string with new text*
Ex: *REPLACE(Customer[Email], 1, 4, "mark")*
output: *mark.doe@example.com*

11. SUBSTITUTE: *Replaces all instances of one text with another*
Ex: *SUBSTITUTE(Customer[Email], "doe", "walker")*
output: *john.walker@example.com*

12. FORMAT( ): *Converts a value to text in a specific format (e.g., date or number formats)*
Ex: *FORMAT(TODAY( ), "MMMM YYYY")*
output: *June 2025 (example)*

12. Difference between REPLACE and SUBSTITUTE in DAX?

1. ***SUBSTITUTE:*** *Replaces specific text by matching a word or substring, no matter where it appears in the text. By default, it replaces every occurrence in the sentence."*
   - *You don't need to know the position of the word.*
   - *It searches for matching text and replaces it directly.*

EX: *SUBSTITUTE("Power BI is awesome", "awesome", "powerful")*
*-- **Output:** "Power BI is powerful"*

Ex 2: *Mention instance number if need to replace specific occurance.*
- ***SUBSTITUTE(text, old_text, new_text, instance_num)***

SUBSTITUTE("Power BI is powerful. BI is growing.", "BI", "DAX", 1)
Output:
Power DAX is powerful. BI is growing.

2. REPLACE: *Replaces part of a text based on position and number of characters, not based on matching a word.*
You must know:
- *The start position (e.g., 14th character)*
- *The number of characters to replace*

Ex: *REPLACE("Power BI is awesome", 14, 7, "powerful")*
*-- **Output:** "Power BI is powerful"*

13. Difference Between FIND and SEARCH in DAX?

FIND and SEARCH are two DAX functions that look very similar — both are used to find the position of a substring within a text.

But there's one key difference: case sensitivity.

Ex: *Text = "Power BI is Awesome"*

FIND (Case-sensitive): *FIND("A", Text)*

-- Output: 14

SEARCH (Case-insensitive):

SEARCH("A", Text) -- ***Output:*** *14*

SEARCH("a", Text) -- ***Output:*** *14 (case doesn't matter)*

14. What are Aggreagate functions?

Assume this Sales table:

Product	Revenue	Quantity
A	100	2
B	200	3
C	150	1

1.***SUM():*** *Adds up all the values in a numeric column*

Ex: *SUM(Sales[Revenue])*

-- Output: 100 + 200 + 150 = 450

2. AVERAGE(): *Calculates the average of values in a column*

Ex: *AVERAGE(Sales[Revenue])*

-- Output: (100 + 200 + 150) / 3 = 150

3. MIN(): *Returns the smallest value in a column*

Ex: *MIN(Sales[Revenue])*

-- Output: 100

4. MAX(): *Returns the largest value in a column*

Ex: *MAX(Sales[Quantity])*

-- Output: 3

5. DIVIDE( ): *Performs division and handles division by zero errors gracefully.*

Ex: *DIVIDE(SUM(Sales[Revenue]), SUM(Sales[Quantity]))*

- *Calculates average revenue per unit, safely.*
- *If SUM(Sales[Quantity]) = 0, it won't throw an error — it returns blank or alternateResult.*
- *SUM(Sales[Revenue]) = 100 + 200 + 150 = 450*
- *SUM(Sales[Quantity]) = 2 + 3 + 1 = 6* ***--output : 75l-***

6. PRODUCT( ): *Returns the product (multiplication result) of values in a column*

Ex: *PRODUCT(Sales[Quantity])*

-- Output: 2 × 3 × 1 = 6

7. VAR( ): *Returns the statistical variance of values in a column*

15. Difference between COUNT( ), COUNTA( ), DISTINCTCOUNT( )?

COUNT( )- *Counts Non-blank numeric values in a column. Ignores text and blanks.*

Ex: *COUNT(Sales[Amount])*

COUNTA( ) *- Counts all non-blank values — numbers, text, dates, etc. Does not care about data type.*

Ex: *COUNTA(Employees[Name])*

DISTINCTCOUNT( ) *- Counts unique, non-blank values in a column of any data type.*

- *Duplicates are ignored beyond the first occurrence.*
- *Blank values are also ignored.*

Ex: *DISTINCTCOUNT(CustomerData[CustomerID])*

16. What are Logical Functions in DAX?

Logical functions are used to:

- *Test conditions (like if something is true)*
- *Return values based on whether the condition is TRUE or FALSE*
- *Filter or classify data based on logic*

Name	Score	Passed	Email
John	85	TRUE	john@gmail.com
Mary	45	FALSE	(blank)
Alex	60	TRUE	alex1@gmail.com

1. ***IF( ):*** *Returns one value if a condition is TRUE, and another if FALSE*

Ex: *PassStatus = IF(Students[Score] >= 50, "Pass", "Fail")*

2. AND( ): *Returns TRUE if all conditions are TRUE*

Ex: *HighPass = IF(AND(Students[Score] > 70, Students[Passed] = TRUE), "Top Performer", "Normal")*

3. OR( ): *Returns TRUE if any one condition is TRUE*

Ex: *Warning = IF(OR(Students[Score] < 50, Students[Passed] = FALSE), "Pass", "Fail")*

4. NOT( ): *Reverses a condition*

Ex: *IF(NOT(ISBLANK(Customers[Email])), "No", "Yes")*

5. SWITCH( ): *Replaces nested IF statements with cleaner logic*

Ex: *Grade = SWITCH(TRUE( ),*
Students[Score] >= 90, "A",
Students[Score] >= 80, "B",
Students[Score] >= 70, "C",
"D")

17. What are iterator functions?

Iterator functions are a special type of DAX functions that evaluate an expression for each row of a table and then aggregate the results into a single value

Product	Quantity	Price	Region
A	2	100	South
B	5	200	North
C	1	150	South

1.***SUMX( ):*** *evaluates an expression row by row over a table and then sums the results.*
Ex: *SUMX(Sales, Sales[Quantity] * Sales[Price])*

2.AVERAGEX( ): *evaluates an expression row by row and returns the average of the results.*
Ex: *AVERAGEX(Sales, Sales[Quantity] * Sales[Price])*

3.MAXX( ): *returns the maximum value from evaluating an expression row by row.*
Ex: *MAXX(Sales, Sales[Quantity] * Sales[Price])*

4.MINX( ): *returns the minimum value from evaluating an expression row by row.*
Ex: *MINX(Sales, Sales[Quantity] * Sales[Price])*

5.COUNTX( ): *counts the number of non-blank results returned from evaluating an expression row by row.*
COUNTX(<table>, <expression>)
Ex: *COUNTX(FILTER(Sales, Sales[Quantity] * Sales[Price] > 500), Sales[Product])*

6.RANKX( ): *returns the rank of each row based on the evaluation of an expression.*
syntax: *RANKX(<table>, <expression>, [value], [order], [ties])*
Ex: *RANKX(ALL(Sales[Product]), Sales[Quantity] * Sales[Price], , DESC)*

7.ADDCOLUMNS( ): *Adds custom columns to an existing table*
Ex: *ADDCOLUMNS(CALENDAR(...), "Year", YEAR([Date]))*

18. Date Functions in DAX ?

Date functions in DAX are used to extract parts of a date (like year, month, day), perform date calculations (like adding months, finding differences), and control time-based logic in your reports. These functions help you manipulate, format, and analyze date and time values in Power BI.

Function	Purpose	Example
TODAY()	Returns today's date	TODAY() → 23-06-2025
NOW()	Returns current date and time	NOW() → 23-06-2025 10:15 AM
YEAR()	Extracts year from a date	YEAR(Sales[Date]) → 2025
MONTH()	Extracts month number (1-12)	MONTH(Sales[Date]) → 6
DAY()	Extracts day of the month	DAY(Sales[Date]) → 23
WEEKDAY()	Day of week (number) It returns a number	WEEKDAY(Sales[Date], 2) → 1=Mon, 7=Sun
DATEDIFF()	— the count of months or days between the two dates.	DATEDIFF(StartDate, EndDate, DAY)
EDATE()	Returns the same day of the previous months or next months	EDATE([Date], -1) → previous month or EDATE([Date], 4) → next 4th month
EOMONTH()	Returns the last date of the month that [Date]	EOMONTH([Date], 0) → 0 means current month (-1 previous, +1 next month)

19. What is Date Table in Power BI and How to create it?

ADate Table (also called a CalendarTableor Date Dimension) isastandalone table that contains every date in a time period, plus useful columns like Year, Month, Quarter, etc.

<u>Functions Used to Create a Date Table in DAX</u>:

CALENDAR( ): *Creates a continuous range of dates between a start date and end date*

<u>Ex:</u> *CALENDAR(DATE(2024,1,1), DATE(2025,12,31))*

(or)

Use these columns in filters, slicers, axis fields, or time intelligence functions

DateTable =

```
ADDCOLUMNS(
 CALENDAR(DATE(2024,1,1), DATE(2025,12,31)),
 "Year", YEAR([Date]),
 "Month", MONTH([Date]),
 "MonthName", FORMAT([Date], "MMMM"),
 "Quarter", "Q" & QUARTER([Date])
)
```

CALENDARAUTO( ): *Automatically detects min & max dates from your model and builds the table*

<u>Ex:</u>

```
DateTable =
ADDCOLUMNS(
 CALENDARAUTO( ),
 "Year", YEAR([Date]),
 "Month", MONTH([Date]),
 "MonthName", FORMAT([Date], "MMM"),
 "Quarter", "Q" & QUARTER([Date])
)
```

20. What is SELECTCOLUMNS in DAX?

SELECTCOLUMNScreates a new tableby selectingonly the columns you want, optionally renaming them or even calculating new values.

SELECTCOLUMNS( ) is a table function used to:

1. *Pick specific columns from an existing table*
2. *Rename columns as you select them*
3. *Create new calculated columns using DAX expressions*
4. *Reorder or reshape a table for further use*

Example 1: Select & Rename Columns

```
CustomersShort =
SELECTCOLUMNS(
Customers,
"Name", Customers[CustomerName],
"Country", Customers[Country])
```

Example 2: Create New Calculated Columns

```
ProductRevenue =
SELECTCOLUMNS(
    Sales,
    "Product", Sales[Product],
"Revenue", Sales[Price] * Sales[Quantity])
```

21. What are Filter Functions in DAX?

Filter functions areusedtoreturn a subset of a table by applying specific conditions (filters).

They help you create custom filter contexts in measures, calculated columns, or tables.

1.***FILTER( ):*** *Returns a table filtered by a condition*

Syntax: *FILTER(table, condition)*

Ex: *FILTER(Sales, Sales[Region] = "East")*

CALCULATE(SUM(Sales[Revenue]), FILTER(Sales, Sales[Revenue] > 1000))

2. ***ALL( ):*** *When you use ALL( ) inside a measure, it tells DAX to ignore the filter context from visuals like slicers, tables, charts, or filters on the report page.*

Ex:

% of Total Revenue = DIVIDE(

SUM(Sales[Revenue]), CALCULATE(SUM(Sales[Revenue]), ALL(Sales)))

3. ALLEXCEPT( ): *Removes all filters except for specified columns*

Ex: *CALCULATE(SUM(Sales[Revenue]), ALLEXCEPT(Sales, Sales[Product]))*

*4.***ALLSELECTED( ):*** *Used in visuals — removes filters outside the current selection, but keeps slicers/filters in scope.*

Ex: *CALCULATE(SUM(Sales[Revenue]), ALLSELECTED(Sales[Region]))*

5. KEEPFILTERS( ): *When used inside CALCULATE, this tells DAX not to override existing filters, but combine with new ones.*

Ex: *CALCULATE(SUM(Sales[Revenue]),*

KEEPFILTERS(Sales[Region] = "East"))

6. CROSSFILTER( ): *Used only inside CALCULATE to change relationship direction temporarily.*

Ex: *CALCULATE(SUM(Sales[Revenue]),*

CROSSFILTER(Sales[ProductID], Products[ProductID], BOTH))

22. Difference between the VALUE( ) and DISTINCT( )?

VALUES( ) Function:

- *Returns unique values from a column.*
- *Includes blank values (including special blank rows from missing relationships).*
- *Respects filters and slicers.*

Ex: VALUES(Sales[Region])

DISTINCT( ) Function:

- *Also returns unique values from a column.*
- *Does not include blanks*
- *Slightly faster if you just want to remove duplicates.*

Ex: DISTINCT(Sales[Region])

23. What are Time Intelligence Functions?

- *Compare dataacross time periods*
- *Calculate growth, change, and trends*
- *Perform date-based filtering (like year-to-date, last 30 days, etc.)*

1.***DATESYTD( ):*** *Returns all dates from Jan 1 to current row date in the same year.*

Ex: *Sales_YTD = CALCULATE(SUM(Sales[Revenue]), DATESYTD('Date'[Date]))*

- *If today is June 15, 2025, this sums all revenue from Jan 1, 2025 to June 15, 2025.*

2. DATESMTD( ): *Returns all dates from the first day of the current month up to the current row date.*

Ex: *SalesMTD = CALCULATE(SUM(Sales[Revenue]), DATESMTD('Date'[Date]))*

- *On June 15, 2025, returns revenue from June 1 to June 15, 2025.*

3. DATESQTD( ): *Returns dates from the start of the quarter to the current row date.*

Ex: *SalesQTD = CALCULATE(SUM(Sales[Revenue]), DATESQTD('Date'[Date]))*

- *If today is May 10, 2025 (Q2), it sums from April 1 to May 10, 2025.*

4. TOTALYTD(): *Returns cumulative total (running total) from Jan 1 to current date.*

Ex: *SalesYTD = TOTALYTD(SUM(Sales[Revenue]), 'Date'[Date])*

- *On August 31, returns cumulative sales from Jan 1 to Aug 31.*

5. TOTALMTD(): *SalesMTD = TOTALMTD(SUM(Sales[Revenue]), 'Date'[Date])*

- *On March 10, gives running total from March 1 to 10.*

6. TOTALQTD(): *SalesQTD = TOTALQTD(SUM(Sales[Revenue]), 'Date'[Date])*

- *On May 15, gives total from April 1 to May 15 (Q2).*

7. SAMEPERIODLASTYEAR(): *Returns a date range that matches the current context from the same period last year.*

Ex: *LastYearSales = CALCULATE(SUM(Sales[Revenue]), SAMEPERIODLASTYEAR('Date'[Date]))*

- *On March 2025, returns sales for March 2024.*

8. PARALLELPERIOD(): *Returns a period that's n steps before or after current period (can be year, month, quarter).*

Ex: *Sales3MonthsAgo = CALCULATE(SUM(Sales[Revenue]), PARALLELPERIOD('Date'[Date], -3, MONTH))*

- *If row context is 6th April 2025, it returns sales from 1st January 2025 to 31st marc*

9. DATEADD(): *Shifts dates forward or backward by a specified number of intervals.*

Ex: *SalesPrevMonth = CALCULATE(SUM(Sales[Revenue]), DATEADD('Date'[Date], -1, MONTH))*

For 6th June 2025, returns sales from 6th May 2025 to 6th june 2025.

10. PREVIOUSMONTH(): *Returns the entire previous month's dates.*

Ex: *SalesLastMonth = CALCULATE(SUM(Sales[Revenue]), PREVIOUSMONTH('Date'[Date]))*

- *For June 2025, returns all sales from May 2025.*

11. PREVIOUSQUARTER(): *SalesLastQuarter = CALCULATE(SUM(Sales[Revenue]), PREVIOUSQUARTER('Date'[Date]))*

- *For Q2 2025, returns sales from Q1 2025.*
- *It just sees you're in Q1 2024, so it goes back to Q4 2023.*

12. PREVIOUSYEAR(): *PREVIOUSYEAR() is a DAX time intelligence function that returns all dates from the previous year, based on the current date filter context.*

Ex: *SalesLastYear = CALCULATE(SUM(Sales[Revenue]), PREVIOUSYEAR('Date'[Date]))*

- *If year is 2025, returns sales from 2024.*

13. NEXTDAY(), NEXTMONTH(), NEXTQUARTER(), NEXTYEAR():

These return the next date ranges for the current context.

Ex: *SalesNextMonth = CALCULATE(SUM(Sales[Revenue]), NEXTMONTH('Date'[Date]))*

- ***For April 2025, returns sales from May 2025.***

Very Important: All these functions need a proper Date Table. You must:

- *Create a continuous Date table (CALENDAR or CALENDARAUTO)*
- *Mark it as the Date Table in Power BI*
- *Ensure it's linked to your fact table using a relationship on date column*

24. What does modify filter context mean?

Overriding, ignoring, or adding to the existing filters applied by the report visuals.

- *You do this using DAX functions like CALCULATE(), FILTER(), ALL(), etc.*
- *Any filters, slicers, or visuals already applying a filter can be overridden (modified) by the filters you define inside the CALCULATE() function.*

Ex: Suppose your visual is filtered to show only Region = "East".

```
-- This measure respects the visual filter
TotalSales = SUM(Sales[Revenue])
-- This measure *modifies* the filter context to ignore Region filter
TotalSalesAllRegions = CALCULATE(SUM(Sales[Revenue]),
ALL(Sales[Region]))
```

25. ALL() + ALLEXCEPT() + COUNTROWS() + FILTER() + DIVIDE() = "Get % of filtered rows vs total, while keeping only specific grouping (like by Employee Type)"

```
Default Rate by Employee Type =
VAR totalRecords = COUNTROWS(ALL(loan_default))
VAR defaultCases = COUNTROWS(
    FILTER(loan_default, loan_default[default] = "true")
)
RETURN
CALCULATE(
 DIVIDE(defaultCases, totalRecords),
 ALLEXCEPT(loan_default, loan_default[employee_type])
)
```

26. ALL( ) + KEEPFILTERS(VALUES(...)) = "Bring back slicer

All_With_KeepFilters =
CALCULATE(
SUM(TEST_ENV[unit_price]),
ALL(TEST_ENV[product_name]),
KEEPFILTERS(VALUES(TEST_ENV[product_name])))

***Effect**:*

- *ALL(...) removes slicer*
- *But KEEPFILTERS(VALUES(...)) re-applies slicer filter*
- *→ If slicer = Phone Case → still returns 220*

27. Running Total:

cumulativesumover any field like Month, Region, or Product — not just Dates.

***Ex:** Running Total Sales =*
CALCULATE(SUM(Sales[Amount]),
FILTER(
ALLSELECTED(Sales[Month]),
Sales[Month] <= MAX(Sales[Month])))

28. YoY (Year-over-Year) Change :

YoY (Year-over-Year) changemeasures how a value (like sales, revenue, or loan amount) has increased or decreased compared to the same period in the previous year.
So if you're analyzing Jan to Mar 2024, the correct YoY comparison is:

- *Jan to Mar 2024*
- VS
- *Jan to Mar 2023 — NOT the entire 2023 year*

***Ex**: YOY Sales Change (%) =*
DIVIDE(
CALCULATE(
SUM('Sales'[SalesAmount]),
'Sales'[Year] = YEAR(MAX('Sales'[Sale_Date]))) -
CALCULATE(
SUM('Sales'[SalesAmount]),
'Sales'[Year] = YEAR(MAX('Sales'[Sale_Date])) - 1),
CALCULATE(
SUM('Sales'[SalesAmount]),
'Sales'[Year] = YEAR(MAX('Sales'[Sale_Date])) - 1),0)

Ex: Using SAMEPERIODLASTYEAR

YoY Sales Change (%) =

DIVIDE(SUM(Sales[SalesAmount]) -- 🔹 Current year sales

-

CALCULATE(SUM(Sales[SalesAmount]),

SAMEPERIODLASTYEAR('Date'[Date])), -- 🔹 Last year sales

CALCULATE(SUM(Sales[SalesAmount]),

SAMEPERIODLASTYEAR('Date'[Date])), 0)

SAMEPERIODLASTYEAR() gives you a precise, same-calendar-period shift for YoY comparisons — it's fixed to the same length as the current period.

29. How to Calculate Last 12 months sales?

DATESINPERIOD() is greatforcustomrollingwindows — last N months, quarters, or years

Ex: *Last 12 Months Sales = CALCULATE(SUM(Sales[SalesAmount]), DATESINPERIOD('DateTable'[Date], MAX('DateTable'[Date]), -12, MONTH))*

30. MOM % :

MoM (Month-over-Month) compares the performance of a metric (like Sales, Profit, etc.) in the current month vs the previous month.

Formula:

MoM Change (%)= ((Current Month Value−Previous Month Value)/Previous Month Value) ×100

Ex:

MoM Sales Change (%) =

DIVIDE(SUM(Sales[SalesAmount]) -

CALCULATE(

SUM(Sales[SalesAmount]),

DATEADD('Date'[Date], -1, MONTH)),

CALCULATE(

SUM(Sales[SalesAmount]),

DATEADD('Date'[Date], -1, MONTH)),0)

31. Scheduled Refresh:

ScheduledRefresh is a feature in Power BI that automatically reloads the entire dataset from the source on a fixed schedule — daily, hourly, etc.

Key Points:

- *Refreshes all rows in the dataset.*
- *Easy to set up in Power BI Service.*
- *Suitable for small to medium datasets.*

Example:

You have a sales Excel file with 5,000 rows. Every day at 9 AM, Power BI reloads all 5,000 rows from the Excel file, even if only 10 rows changed.

32. Incremental Refresh:

Incremental Refreshupdatesonly the new or changed rows in a dataset while keeping the historical data unchanged.

Key Points:

- *Refreshes only recent data (e.g., last 7 days or last 1 month).*
- *Requires a Date column for filtering.*
- *Improves performance for large datasets.*

Example:

- *Your database has 2 million rows of order data. You set up Incremental Refresh to:*
- *Keep data for 5 years*
- *Refresh only the last 30 days*

Each day, Power BI only checks and loads the rows from the past 30 days — the rest of the 5 years of data stays as-is.