

DAX In PowerBI - Complete Guide

What is DAX?

Imagine Power BI as a classroom...

You have:

- A big register (your dataset)
- A bunch of students (rows)
- And you're the teacher who wants answers (totals, averages, top scorers...)

But here's the twist:

Power BI students don't speak human language. They only understand DAX – the magic math + logic language.

DAX = Data Analysis Expressions

It's like Excel's smart cousin, but for Power BI.

You need DAX when you say things like:

- “Tell me the total sales made this year” → **SUM()**
 - “Who sold more than ₹1000?” → **IF()**
 - “What's the average rating for Chennai?” → **CALCULATE() + AVERAGE()**
-

Why Do We Use DAX?

Power BI without DAX is like a calculator with no buttons. Your data may be beautiful, but without DAX, you can't talk to it.

DAX helps us:

- Create new columns
 - Calculate dynamic measures
 - Build new tables
 - Apply filters, logic, totals, comparisons, and trends
-

Types of DAX Output – with Classroom Analogy

DAX Output Type	Classroom Analogy	What It Does
Calculated Column	Giving each student a separate mark sheet	Creates a new column per row
Measure	Asking class average in a report card	One dynamic result (total, avg)
Calculated Table	Creating a new team with selected students	New table with logic or filter

Matrix of DAX Functions

Categorized by:

- Beginner / Intermediate / Advanced
- Calculated Column / Measure / Calculated Table

Beginner Level DAX Matrix

Function	CC	CM	CT
SUM		✓	
AVERAGE		✓	
MIN		✓	
MAX		✓	
COUNT		✓	
COUNTA		✓	
DISTINCTCOUNT		✓	
IF	✓	✓	
AND	✓	✓	
OR	✓	✓	
NOT	✓	✓	
SWITCH	✓	✓	
CONCATENATE	✓		
LEFT	✓		
RIGHT	✓		

MID	✓		
LEN	✓		
UPPER	✓		
LOWER	✓		
TRIM	✓		
TODAY	✓	✓	✓
NOW	✓	✓	✓
YEAR	✓	✓	✓
MONTH	✓	✓	✓
DAY	✓	✓	✓
WEEKDAY	✓	✓	✓
ROUND	✓	✓	
INT	✓	✓	
MOD	✓	✓	

Intermediate Level DAX Matrix

Function	CC	CM	CT
CALCULATE		✓	✓
FILTER		✓	✓
ALL		✓	✓
ALLEXCEPT		✓	✓
VALUES		✓	✓
SELECTEDVALUE		✓	✓
RANKX		✓	
TOPN		✓	✓
EARLIER	✓		
RELATED	✓	✓	
RELATEDTABLE		✓	✓
SUMX		✓	
AVERAGEX		✓	
COUNTX		✓	
ADDCOLUMNS			✓
SUMMARIZE			✓
SELECTCOLUMNS			✓
UNION			✓
INTERSECT			✓
EXCEPT			✓

Advanced Level DAX Matrix

Function	CC	CM	CT
TREATAS		✓	✓
KEEPFILTERS		✓	
REMOVEFILTERS		✓	✓
ISINSCOPE		✓	
ISFILTERED		✓	
ISCROSSFILTERED		✓	
PATH / PATHITEM	✓	✓	✓
VAR / RETURN	✓	✓	✓
GENERATE			✓
NATURALINNERJOIN			✓
CROSSJOIN			✓

DATASET LINK:

BEGINNER LEVEL DAX:

1. SUM()

In class, students are running a stationery sale. The teacher asks,

“How much total money did we collect from all your sales?”

Arjun adds up each order total from their records. **“We made ₹320, miss!”**

That’s what **SUM()** does – it adds up numbers from a column.

- Type: Measure (CM)

Formula:

Total Sales Amount = SUM(Sales[Total])

- Use: Adds all values in Sales[Total] column – total revenue.
-

2. AVERAGE()

Teacher says,

“Let’s give a prize to students scoring above average. But what’s the average mark?”

Divya sums all marks and divides by number of students. **“It’s 388 marks!”**

AVERAGE() finds the average from a column.

- Type: Measure (CM)

Formula:

Average Marks = AVERAGE(Students[TotalMarks])

- Use: Calculates average marks of all students.
-

3. MIN()

There’s a prize for **“most absent student.”**

Teacher checks: **“Who has the least attendance?”**

Bala points: **“60%! That’s the lowest.”**

MIN() finds the smallest value in a column.

- Type: Measure (CM)

Formula:

Minimum Attendance = MIN(Students[Attendance])

- Use: Shows the lowest attendance from the student data.
-

4. **MAX()**

Now, for the “**Perfect Attendee**” award, teacher asks:

“Who came to school the most?”

Jeeva jumps up: **“98%! That’s mine!”**

MAX() gives the highest value from a column.

- Type: Measure (CM)

Formula:

Highest Attendance = MAX(Students[Attendance])

- Use: Finds the highest attendance among students.
-

5. **COUNT()**

The teacher is checking how many orders were placed during the fest.

“Let’s count the Order IDs.”

Even if some values repeat, **COUNT()** counts how many rows have values (non-blank).

- Type: Measure (CM)

Formula:

No of Orders = **COUNT(Sales[OrderID])**

- Use: Counts how many order IDs exist in the Sales table.
-

6. **COUNTA()**

One student left their name blank on the roll sheet.

Teacher asks,

“How many students filled their name correctly?”

COUNTA() counts all non-blank values, even text in a column.

- Type: Measure (CM)

Formula:

Students with Name = **COUNTA(Students[Name])**

- Use: Counts students with non-empty names.
-

7. **DISTINCTCOUNT()**

The teacher wants to know how many unique cities students came to.

Charan says: “We have students from Chennai, Delhi, Mumbai...”

DISTINCTCOUNT() counts how many different values are in a column.

- Type: Measure (CM)

Formula:

Distinct Cities = DISTINCTCOUNT(Students[City])

- Use: Counts number of unique cities in Students table.
-

8. **IF()**

The teacher wants to separate toppers from average students.

“If marks are 450 or more, call them Topper.”

This is where **IF()** steps in.

- Type: Calculated Column (CC)

Formula:

IsTopper = IF(Students[TotalMarks] >= 450, "Topper", "Average")

- Use: Creates a new column showing who’s a topper based on marks.

You can also use **IF()** in a Measure, like this:

High Sales Count = CALCULATE(COUNTROWS(Sales), Sales[Total] > 150)

9. **AND()**

For the “Star Student” badge, teacher sets 2 rules:

TotalMarks > 450 and BehaviourScore ≥ 4

Only those meeting both criteria qualify.

- Type: Calculated Column (CC)

Formula:

StarStudent = IF(Students[TotalMarks] > 450 && Students[BehaviourScore] >= 4, "Yes", "No")

- Use: Checks two conditions together using AND.
-

10. **OR()**

To identify who needs support, teacher asks:

“Anyone with low attendance or low behaviour score?”

OR() returns true if any one condition is true.

- Type: Calculated Column (CC)

Formula:

NeedsSupport = IF(Students[Attendance] < 70 || Students[BehaviourScore] <= 2, "Yes", "No")

- Use: Flags students needing attention based on either condition.
-

11. NOT()

Teacher says:

“I want to know who is not enrolled in our academic year.”

This is where NOT() helps. It flips TRUE to FALSE and vice versa.

- Type: Calculated Column (CC)

Formula:

NotEnrolled = IF(NOT(Students[Enrolled]), "No", "Yes")

- Use: Shows who is not enrolled in school.
-

12. SWITCH()

Instead of writing many IF()s, teacher asks:

“Let’s give feedback based on BehaviourScore — 1 is Poor, 5 is Excellent.”

SWITCH() acts like a grading machine — picks exact matches and gives results.

- Type: Calculated Column (CC)

Formula:

Feedback = SWITCH(
Students[BehaviourScore],
1, "Poor",
2, "Needs Improvement",
3, "Okay",
4, "Good",
5, "Excellent",
"Unknown")

- Use: Converts BehaviourScore into descriptive feedback.
-

13. **CONCATENATE()**

The teacher wants name tags like “Arjun - 10A” for students.

CONCATENATE() combines two columns into one string.

- Type: Calculated Column (CC)

Formula:

NameTag = **CONCATENATE**(Students[Name], " - " & Students[Class])

- Use: Joins Name and Class into a single text value.
-

14. **LEFT()**

We want to extract just the first letter of student names for ID cards.

LEFT() pulls characters from the start of a string.

- Type: Calculated Column (CC)

Formula:

FirstLetter = **LEFT**(Students[Name], 1)

- Use: Gets the first character of each student's name.
-

15. RIGHT()

In a treasure hunt, the last 2 digits of StudentID matter.

RIGHT() fetches characters from the end of a string.

- Type: Calculated Column (CC)

Formula:

IDCode = RIGHT(Students[StudentID], 2)

- Use: Gets last 2 characters of StudentID.
-

16. MID()

A quiz app wants to extract 3 letters from the middle of product names.

MID() pulls part of a string starting at a given position.

- Type: Calculated Column (CC)

Formula (from Products table):

MidBrand = MID(Products[Brand], 2, 3)

- Use: Grabs part of the brand name starting from character 2.
-

17. LEN()

During the data check, the teacher wants to find who has long names.

LEN() counts how many characters are in a text.

- Type: Calculated Column (CC)

Formula:

NameLength = LEN(Students[Name])

- Use: Returns number of letters in each student's name.
-

18. **UPPER()**

For making ID cards, all names should be in capital letters.

UPPER() changes all letters to uppercase.

- Type: Calculated Column (CC)

Formula:

NameCaps = UPPER(Students[Name])

- Use: Converts names to uppercase.
-

19. **LOWER()**

In email generation, all letters should be in lowercase.

LOWER() changes all letters to small cases.

- Type: Calculated Column (CC)

Formula:

`NameLower = LOWER(Students[Name])`

- Use: Converts names to lowercase.
-

20. **TRIM()**

Sometimes names have extra spaces accidentally entered.

TRIM() removes unnecessary spaces.

- Type: Calculated Column (CC)

Formula:

`CleanedName = TRIM(Students[Name])`

- Use: Cleans up extra spaces before or after names.
-

21. **TODAY()**

Teacher announces:

“We’re giving attendance awards today. What’s today’s date?”

TODAY() simply gives the current system date (no time).

- Type: CC / CM / CT

Formula (as Calculated Column in Students):

`DaysSinceAdmission = TODAY() - Students[AdmissionDate]`

- **Use:** Finds how many days have passed since a student was admitted.
-

22. `NOW()`

Sales Rep wants to log the **exact date and time** when an order was delivered.

`NOW()` gives current date and time together.

- **Type:** CC / CM / CT

Formula (in Sales):

`LoggedAt = NOW()`

- **Use:** Logs current date and time when calculation runs.
-

23. `YEAR()`

The teacher asks:

“How many students will join in 2022?”

`YEAR()` helps extract the year part from a date.

- **Type:** Calculated Column (CC)

Formula:

`AdmissionYear = YEAR(Students[AdmissionDate])`

- **Use:** Pulls out a year from the admission date.
-

24. `MONTH()`

Sales team checks:

“Which month had the most sales?”

`MONTH()` returns month number from a date.

- **Type:** CC / CM

Formula:

`SaleMonth = MONTH(Sales[SaleDate])`

- **Use:** Extracts the month from each sale’s date.
-

25. `DAY()`

Teacher asks:

“How many students joined on the 1st day of any month?”

`DAY()` returns just the day from a full date.

- **Type:** Calculated Column (CC)

Formula:

`JoinDay = DAY(Students[AdmissionDate])`

- **Use:** Extracts day number from a date.
-

26. **WEEKDAY()**

The school gives a “Weekend Joiner” badge.

“Let’s check who joined on Saturdays or Sundays!”

WEEKDAY() gives number for day of the week (1 = Sunday, 7 = Saturday)

- **Type:** Calculated Column (CC)

Formula:

`JoinWeekday = WEEKDAY(Students[AdmissionDate])`

- **Use:** Tells what day of the week each student joined.
-

27. **ROUND()**

For report cards, the teacher rounds off decimal scores to the nearest number.

ROUND() is used to round numbers with specific decimal places.

- **Type:** Calculated Column (CC)

Formula:

`RoundedBehaviour = ROUND(Students[BehaviourScore], 0)`

- **Use:** Rounds off BehaviourScore to the nearest whole number.
-

28. INT()

In some reports, decimal is not allowed – only full units.

INT() removes the decimal and keeps the whole number.

- **Type:** CC / CM

Formula:

WholeUnits = INT(Sales[Total] / Sales[UnitPrice])

- **Use:** Returns the whole number result of a calculation.
-

29. MOD()

In a math game, every 3rd student gets a gift.

“Let’s tag students where row number % 3 = 0.”

MOD() gives remainder after division.

- **Type:** Calculated Column (CC)

Formula:

LuckyDraw = IF(MOD(VALUE(RIGHT(Students[StudentID], 2)), 3) = 0, "Winner", "Try Again")

- **Use:** Tags every 3rd student using remainder logic.

INTERMEDIATE LEVEL DAX

1. CALCULATE()

In class, the teacher says:

“I want to know the average Behaviour Score of students only from Chennai.”

Everyone looks at the full BehaviourScore column... but wait! That includes all cities!

So Bala suggests:

“Let’s first filter to Chennai, then calculate the average.”

That’s what **CALCULATE()** does – it modifies the filter context, then runs the calculation.

- Type: Measure (CM)

Formula:

Chennai Behaviour Avg = CALCULATE(AVERAGE(Students[BehaviourScore]), Students[City] = "Chennai")

- Use: Calculates the average BehaviourScore, but only for Chennai students.
-

2. FILTER()

During a school report, Divya says:

“Let’s create a new list showing only students who scored more than 450 marks.”

FILTER() creates a filtered version of a table using a condition.

- Type: Calculated Table (CT)

Formula:

TopperTable = FILTER(Students, Students[TotalMarks] > 450)

- Use: Creates a new table containing only students with TotalMarks > 450.
-

3. **ALL()**

Teacher says:

“We’re calculating each student’s % score, but use the highest possible marks as 100% – ignore all filters!”

ALL() removes filters and returns the entire column or table.

- Type: Measure (CM)

Formula:

AllStudentAverage = CALCULATE(AVERAGE(Students[TotalMarks]), ALL(Students))

- Use: Calculates average marks considering all rows, even if filters are applied in visuals.
-

4. ALLEXCEPT()

Now teacher wants to know:

“Average marks per class, but keep filters only on Class, ignore others.”

ALLEXCEPT() removes all filters except the one(s) you mention.

- Type: Measure (CM)

Formula:

**ClassWiseAvg = CALCULATE(
AVERAGE(Students[TotalMarks]),ALLEXCEPT(Students, Students[Class]))**

- Use: Keeps only **Class** filter while ignoring everything else – useful in grouped visuals.
-

5. VALUES()

The principal says:

“Tell me the list of unique grades in this class.”

VALUES() returns the distinct values from a column.

- Type: CM or used inside CALCULATE

Formula (for debugging):

UniqueGrades = VALUES(Students[Grade])

- Use: Returns list of unique grades. Often used in slicers or when creating dynamic calculations.

6. **SELECTEDVALUE()**

In the classroom slicer, the teacher picks one city, and wants to see sales only for that city.

If only one city is selected, **SELECTEDVALUE()** returns that.

If multiple selected, it returns blank or default.

- Type: CM

Formula:

SelectedCity = SELECTEDVALUE(Students[City], "Not Selected")

- Use: Gets the selected value in a slicer — used in dynamic titles, filters, KPIs.
-

7. **RANKX()**

A leaderboard is going up on the wall.

“Rank students based on TotalMarks — highest is Rank 1!”

That’s **RANKX()** — it ranks each row compared to others.

- Type: Calculated Column (CC)

Formula:

MarksRank = RANKX(ALL(Students), Students[TotalMarks], DESC)

- Use: Ranks students based on TotalMarks in descending order.
-

8. TOPN()

Only the top 3 students will go to the inter-school quiz.

“Show top 3 students by BehaviourScore.”

TOPN() returns the top N rows based on an expression.

- Type: Calculated Table (CT)

Formula:

Top3Behaviour = TOPN(3, Students, Students[BehaviourScore], DESC)

- Use: Gets the top 3 students by behaviour.
-

9. EARLIER()

In a nested calculation, the teacher wants to calculate the difference from each student's earlier marks.

“Compare every student's marks to their class average.”

EARLIER() helps you refer back to a previous row context – especially useful in calculated columns.

- Type: CC

Formula (basic example):

MarksGap = Students[TotalMarks] - CALCULATE(AVERAGE(Students[TotalMarks]), FILTER(Students, Students[Class] = EARLIER(Students[Class])))

- Use: Compares each student's marks to their own class average.
-

10. RELATED()

In a Sales table, teacher says:

“Show the name of the student who placed each order.”

RELATED() helps pull values from a related table (like a lookup).

- Type: Calculated Column (CC)

Formula:

StudentName = RELATED(Students[Name])

- Use: Pulls name from the Students table into the Sales table using relationship on StudentID.
-

11. RELATEDTABLE()

Teacher looks at a student's name and asks:

“Show me all the orders made by this student.”

RELATEDTABLE() pulls all matching rows from another table that's related.

- Type: CM or used inside CT

Formula (inside calculated table):

StudentOrders = ADDCOLUMNS(Students, "Orders",
COUNTROWS(RELATEDTABLE(Sales)))

- Use: Returns all related rows from another table based on relationship (StudentID here)

12. SUMX()

Teacher says:

“Calculate total revenue – not just unit price * quantity, but row by row.”

SUMX() is a row-by-row SUM. You give it a table + expression.

- Type: Measure (CM)

Formula:

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

- Use: Sums calculated values row-by-row. Better than just SUM([Total]).
-

13. AVERAGEX()

Principal says:

“Give me the average amount paid per order (after discount).”

Use AVERAGEX() to calculate row-wise values first, then average them.

- Type: Measure (CM)

Formula:

AvgNetSales = AVERAGEX(Sales, Sales[Total] - Sales[Discount])

- Use: Averages calculated expression per row.
-

14. COUNTX()

Teacher says:

“Count how many products were sold above ₹100.”

COUNTX() runs row-wise logic and counts only when the condition matches.

- Type: Measure (CM)

Formula:

```
HighPriceOrders = COUNTX( FILTER(Sales, Sales[Total] > 100),  
Sales[OrderID])
```

- Use: Counts rows based on calculated condition.
-

15. ADDCOLUMNS()

The teacher wants a new table that shows the student name and how many orders they placed.

ADDCOLUMNS() adds a calculated column to a table result.

- Type: Calculated Table (CT)

Formula:

```
StudentOrderCount = ADDCOLUMNS( Students, "OrderCount",  
COUNTROWS(RELATEDTABLE(Sales)))
```

- Use: Adds column showing how many orders each student placed.
-

16. SUMMARIZE()

Principal wants a summary:

“Show me total sales per student.”

SUMMARIZE() is like GROUP BY in SQL – creates a table grouped by one or more columns.

- Type: Calculated Table (CT)

Formula:

SalesSummary = SUMMARIZE(Sales, Sales[StudentID], “TotalSales”, SUM(Sales[Total]))

- Use: Creates summary table grouped by StudentID.
-

17. SELECTCOLUMNS()

Teacher says:

“I want only Name and Grade – no other student data.”

SELECTCOLUMNS() creates a new table with only specific columns.

- Type: Calculated Table (CT)

Formula:

StudentBasic = SELECTCOLUMNS(Students, "Name", Students[Name], "Grade", Students[Grade])

- Use: Picks and renames columns for a custom table.

18. UNION()

Two sections A and B have student lists.

“Merge both into one master list.”

UNION() stacks two tables with the same columns.

- Type: Calculated Table (CT)

Formula:

MasterStudentList = UNION(StudentsA, StudentsB)

- Use: Combines two similar tables into one.
-

19. INTERSECT()

Teacher says:

“Show me only the students who are in both Science Club and Sports Club.”

INTERSECT() returns only common rows between two tables.

- Type: Calculated Table (CT)

Formula (conceptual):

CommonStudents = INTERSECT(ScienceClub, SportsClub)

- Use: Gets overlapping rows from two datasets.
-

20. EXCEPT()

Teacher asks:

“Who is in Class A but not in Class B?”

EXCEPT() removes rows from the first table that also exist in the second.

- Type: Calculated Table (CT)

Formula (conceptual):

OnlyClassA = EXCEPT(ClassA, ClassB)

- Use: Returns rows that exist in one table and not in the other.
-

ADVANCED LEVEL DAX

1. TREATAS()

Imagine you have a list of cities from one slicer, and you want to apply that to a completely different table that doesn't directly relate.

Teacher says:

“Even if the Products table has no City column, treat it like it was filtered by the same cities students are from.”

That's what TREATAS() does — applies filters from one column to another table.

- Type: Measure (CM) or inside CALCULATE

Formula (concept):

`CrossCitySales = CALCULATE(SUM(Sales[Total]),
TREATAS(VALUES(Students[City]), Sales[City]))`

- Use: Filters Sales table based on city values selected from Students table.
-

2. `KEEPFILTERS()`

Teacher says:

“Don’t let the visual override filters I manually gave in DAX.”

That’s `KEEPFILTERS()` – it forces a filter to stay, even if a visual or slicer tries to change it.

- Type: CM inside `CALCULATE`

Formula:

`DelhiSales = CALCULATE(SUM(Sales[Total]), KEEPFILTERS(Sales[City] = "Delhi"))`

- Use: Keeps the "Delhi" filter no matter what visuals try to add.
-

3. `REMOVEFILTERS()`

Principal says:

“I want the average marks across the whole school, not just the filtered class.”

Use `REMOVEFILTERS()` to clear filters from a table or column.

- Type: Measure (CM)

Formula:

```
SchoolAverage = CALCULATE(  
AVERAGE(Students[TotalMarks]),REMOVEFILTERS(Students[Class]))
```

- Use: Removes all filters on `Students[Class]` before calculating.
-

4. `ISINSCOPE()`

You want to show details only when a visual is grouped by Class.

This checks:

“Are we currently inside a grouping based on Class?”

- Type: CM (often for formatting or titles)

Formula:

```
IsClassVisible = IF(ISINSCOPE(Students[Class]), "Class Selected", "Not Grouped by  
Class")
```

- Use: Used for dynamic messages or logic inside visuals.
-

5. `ISFILTERED()`

Teacher says:

“If the user has filtered a slicer by Grade, show a warning.”

`ISFILTERED()` checks if a column has any filter applied.

- Type: CM

Formula:

`GradeFilterStatus = IF(ISFILTERED(Students[Grade]), "Grade Filter Active", "All Grades")`

- Use: Helps build interactive reports with custom messages.
-

6. `ISCROSSFILTERED()`

Imagine Students and Sales are linked.

“If the user filtered Sales and it indirectly affects Students, let’s detect that.”

`ISCROSSFILTERED()` checks if a table is filtered because of another table via relationship.

- Type: CM

Formula:

`IndirectFilterCheck = IF(ISCROSSFILTERED(Students[StudentID]), "Cross Filtered", "Not")`

- Use: Detects filter impact across tables.
-

7. `PATH()`

This is for hierarchy. Imagine Principal → Vice Principal → Teacher → Student.

`PATH()` helps store this full parent-child path in one column.

- Type: Calculated Column (CC)

Example: Let's assume StudentID is parent, and another column **MentorID** is the person above them.

HierarchyPath = PATH(Students[StudentID], Students[MentorID])

- Use: Builds hierarchy string like "S005|S003|S001"
-

8. **PATHITEM()**

Now we want to pick specific level from that path, like:

"Who is the top mentor of each student?"

PATHITEM() lets you extract a specific level.

- Type: CC

Formula:

TopMentor = PATHITEM(Students[HierarchyPath], 1)

- Use: Grabs the first ID from the path.
-

9. **VAR**

Used for storing temporary values, like variables in code.

Teacher says:

"I want to store the average, and use it 3 times – don't recalculate again and again!"

- Type: Used inside CM, CC, CT

Formula:

```
AboveAvgFlag = VAR avgMarks = AVERAGE(Students[TotalMarks])  
RETURN IF(Students[TotalMarks] > avgMarks, "Above Avg", "Below Avg")
```

- Use: Stores average once and reuses it.
-

10. RETURN

Always used with **VAR**. It defines the final result you want to return.

Same formula:

```
VAR avgMarks = AVERAGE(Students[TotalMarks])  
RETURN IF(Students[TotalMarks] > avgMarks, "Above Avg", "Below Avg")
```

- Use: Finishes the DAX block by telling what to output.
-

**"Every DAX formula you write is not just a line
of code, it is a step toward mastering the
language of data. Struggle now, so your reports
can speak fluently tomorrow."**

