DAX Expressions Notes

Data Analysis Expressions (DAX) is a formula expression language used in Analysis Services, Power BI, and Power Pivot in Excel. DAX formulas include functions, operators, and values to perform advanced calculations and queries on data in related tables and columns in tabular data models.

Calculations

DAX formulas are used in measures, calculated columns, calculated tables, and row-level security.

Measures

Measures are dynamic calculation formulas where the results change depending on context. Measures are used in reporting that support combining and filtering model data by using multiple attributes such as a Power BI report or Excel PivotTable or PivotChart. Measures are created by using the DAX formula bar in the model designer.

A formula in a measure can use standard aggregation functions automatically created by using the Autosum feature, such as COUNT or SUM, or you can define your formula by using the DAX formula bar. Named measures can be passed as an argument to other measures. For example, using this very simple measure formula: Total Sales = SUM([Sales Amount])

Calculated columns

A calculated column is a column that you add to an existing table (in the model designer) and then create a DAX formula that defines the column's values. When a calculated column contains a valid DAX formula, values are calculated for each row as soon as the formula is entered. Values are then stored in the in-memory data model. For example, in a Date table, when the formula is entered into the formula bar: = [Calendar Year] & " Q" & [Calendar Quarter]

Row-level security

With row-level security, a DAX formula must evaluate to a Boolean TRUE/FALSE condition, defining which rows can be returned by the results of a query by members of a particular role. For example, for members of the Sales role, the Customers table with the following DAX formula: = Customers[Country] = "USA"

Created by Mahesh Babu Pedagadi

Data Analysis Expressions (DAX) is a library of functions and operators that can be combined to build formulas and expressions in Power BI, Analysis Services, and Power Pivot in Excel data models.

Different Function Types:

1. **Aggregation functions**: Calculate scalar values like count, sum, average, min, or max for all rows in a column or table.
2. **Date and time functions**: Similar to Excel's date and time functions but based on datetime data types used by SQL Server.
3. **Filter functions**: Help return specific data types, look up values in related tables, and manipulate data context for dynamic calculations.
4. **Financial functions**: Perform financial calculations like net present value and rate of return.
5. **Information functions**: Determine if values in a table or column match the expected type, such as ISERROR.
6. **Logical functions**: Return information about values in an expression, like TRUE.
7. **Math and Trig functions**: Similar to Excel's math and trig functions, but with differences in numeric data types.
8. **Other functions**: Perform unique actions not covered by other categories.
9. **Parent and Child functions**: Manage data presented in a parent/child hierarchy in data models.
10. **Relationship functions**: Manage and utilize relationships between tables.
11. **Statistical functions**: Calculate values related to statistical distributions and probability.
12. **Table manipulation functions**: Return or manipulate tables.
13. **Text functions**: Return part of a string, search for text, or concatenate string values. Also control formats for dates, times, and numbers.
14. **Time intelligence functions**: Create calculations using built-in knowledge about calendars and dates for meaningful comparisons across time periods.

Aggregation functions: Calculate scalar values like count, sum, average, min, or max for all rows in a column or table.

1. **COUNT**: Counts the number of rows in a column or table.

Syntax: COUNT(<column>)

2. **SUM**: Calculates the sum of values in a column or table.

Syntax: SUM(<column>)

3. **AVERAGE**: Calculates the average of values in a column or table.

Syntax: AVERAGE(<column>)

4. **MIN**: Finds the minimum value in a column or table.

Syntax: MIN(<column>)

5. **MAX**: Finds the maximum value in a column or table.

Syntax: MAX(<column>)

Date and time functions: Similar to Excel's date and time functions but based on datetime data types used by SQL Server.

1. **TODAY**: Returns the current date.

Syntax: TODAY()

2. **NOW**: Returns the current date and time.

Syntax: NOW()

3. **DATE**: Creates a date value from the specified year, month, and day.

Syntax: DATE(<year>, <month>, <day>)

4. **YEAR**: Returns the year component of a date.

Syntax: YEAR(<date>)

5. **MONTH**: Returns the month component of a date.

Syntax: MONTH(<date>)

6. **DAY**: Returns the day component of a date.

Syntax: DAY(<date>)

7. **DATEADD**: Adds or subtracts a specified number of units (days, months, years) to a date.

Syntax: DATEADD(<start_date>, <number>, <interval>)

8. **DATEDIFF**: Calculates the difference between two dates.

Syntax: DATEDIFF(<start_date>, <end_date>, <interval>)

9. **CALENDAR**: Creates a table of dates within a specified range.

Syntax: CALENDAR(<start_date>, <end_date>)

10. **CALENDARAUTO**: Automatically generates a date table based on the data in the model.

Syntax: CALENDARAUTO()

Filter functions: Help return specific data types, look up values in related tables, and manipulate data context for dynamic calculations.

---

1. **FILTER**: Returns a table that contains only the rows that satisfy the specified conditions.

Syntax: FILTER(<table>, <condition>)

2. **RELATED**: Retrieves a related value from another table.

Syntax: RELATED(<column>)

3. **RELATEDTABLE**: Returns a table related to the current table based on a specified relationship.

Syntax: RELATEDTABLE(<table>)

4. **ALL**: Removes all filters from a table or column, or from all columns except specified columns.

Syntax: ALL([<table> [, <column> [, <column> [, ...]]]])

5. **ALLEXCEPT**: Removes all filters from a table except for those specified columns.

Syntax: ALLEXCEPT(<table>, <column>, [<column> [, <column> [, ...]]])

6. **KEEPFILTERS**: Preserves existing filters in the current context while evaluating a calculation.

Syntax: KEEPFILTERS(<expression>)

7. **VALUES**: Returns a single-column table of unique values from a column, considering only the rows that are visible in the current context.

Syntax: VALUES(<column>)

8. **SELECTCOLUMNS**: Returns a table with selected columns from the specified table.

Syntax: SELECTCOLUMNS(<table>, <column1>[, <column2>, ...])

Financial functions: Perform financial calculations like net present value and rate of return.

1. **NPV**: Calculates the net present value of an investment based on a series of cash flows.

Syntax: NPV(<rate>, <value1>, [<value2>, ...])

2. **IRR**: Calculates the internal rate of return for a series of cash flows.

Syntax: IRR(<values>)

3. **XNPV**: Calculates the net present value of cash flows that are not necessarily periodic.

Syntax: XNPV(<rate>, <values>, <dates>)

4. **XIRR**: Calculates the internal rate of return for cash flows that are not necessarily periodic.

Syntax: XIRR(<values>, <dates>)

5. **FV**: Calculates the future value of an investment based on periodic, constant payments and a constant interest rate.

Syntax: FV(<rate>, <nper>, <pmt>, [<pv>, [<type>]])

6. **PV**: Calculates the present value of an investment based on periodic, constant payments and a constant interest rate.

Syntax: PV(<rate>, <nper>, <pmt>, [<fv>, [<type>]])

7. **RATE**: Calculates the interest rate per period of an annuity.

Syntax: RATE(<nper>, <pmt>, <pv>, [<fv>, [<type>]], [<guess>])

8. **DURATION**: Calculates the Macaulay duration of an investment.

Syntax: DURATION(<settlement>, <maturity>, <coupon>, <yld>, <frequency>, [<basis>])

Information functions: Determine if values in a table or column match the expected type, such as ISERROR.

1. **ISERROR**: Checks whether a value is an error.

Syntax: ISERROR(<value>)

2. **ISBLANK**: Checks whether a value is blank.

Syntax: ISBLANK(<value>)

3. **ISNUMBER**: Checks whether a value is a number.

Syntax: ISNUMBER(<value>)

4. **ISTEXT**: Checks whether a value is text.

Syntax: ISTEXT(<value>)

5. **ISLOGICAL**: Checks whether a value is a logical (Boolean) value.

Syntax: ISLOGICAL(<value>)

6. **ISINSCOPE**: Checks whether a column is currently in scope for a calculation.

Syntax: ISINSCOPE(<column>)

7. **HASONEVALUE**: Checks whether a column has only one distinct value in the current context.

Syntax: HASONEVALUE(<column>)

8. **SELECTEDVALUE**: Returns the value if there is only one value in the specified column in the current context; otherwise, returns blank.

Syntax: SELECTEDVALUE(<column>)

9. **USERNAME**: Returns the current user name.

Syntax: USERNAME()

10. **USERPRINCIPALNAME**: Returns the user principal name (UPN) of the current user.

Syntax: USERPRINCIPALNAME()

Logical functions: Return information about values in an expression, like TRUE.

Truth tables are used to represent the outputs of logical operations for all possible combinations of inputs. Here are the truth tables for the basic logical operations AND, OR, and NOT:

1. **AND Truth Table**:

Input A	**Input B**	**Output**
FALSE	FALSE	FALSE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
TRUE	TRUE	TRUE

In the AND operation, the output is TRUE only if both inputs are TRUE; otherwise, the output is FALSE.

2. **OR Truth Table**:

Input A	**Input B**	**Output**
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	TRUE

In the OR operation, the output is TRUE if at least one input is TRUE; the output is FALSE only if both inputs are FALSE.

3. **NOT Truth Table**:

Input	**Output**
FALSE	TRUE
TRUE	FALSE

In the NOT operation, the output is the opposite of the input. If the input is TRUE, the output is FALSE, and vice versa.

1. **IF**: Returns one value if a condition is TRUE and another value if it's FALSE.

Syntax: IF(<condition>, <value_if_true>, <value_if_false>)

2. **AND**: Returns TRUE if all arguments are TRUE, and FALSE otherwise.

Syntax: AND(<logical1>, [<logical2>, ...])

3. **OR**: Returns TRUE if any argument is TRUE, and FALSE otherwise.

Syntax: OR(<logical1>, [<logical2>, ...])

4. **NOT**: Returns the opposite of a logical value - TRUE if the argument is FALSE, and FALSE if the argument is TRUE.

Syntax: NOT(<logical>)

5. **TRUE**: Returns the logical value TRUE.

Syntax: TRUE()

6. **FALSE**: Returns the logical value FALSE.

Syntax: FALSE()

7. **SWITCH**: Evaluates an expression against a list of values and returns the result corresponding to the first matching value.

Syntax: SWITCH(<expression>, <value1>, <result1>, [<value2>, <result2>, ...], [<default_result>])

8. **IFERROR**: Returns a value you specify if a formula evaluates to an error; otherwise, returns the result of the formula.

Syntax: IFERROR(<value>, <value_if_error>)

Math and Trig functions: Similar to Excel's math and trig functions, but with differences in numeric data types.

1. **ABS**: Returns the absolute value of a number.

Syntax: ABS(<number>)

2. **EXP**: Returns e raised to the power of a number.

Syntax: EXP(<number>)

3. **LOG**: Returns the natural logarithm of a number.

Syntax: LOG(<number>, [<base>])

4. **LN**: Returns the natural logarithm of a number.

Syntax: LN(<number>)

5. **SQRT**: Returns the square root of a number.

Syntax: SQRT(<number>)

6. **POWER**: Raises a number to a specified power.

Syntax: POWER(<number>, <power>)

7. **ROUND**: Rounds a number to the specified number of digits.

Syntax: ROUND(<number>, <num_digits>)

8. **TRUNC**: Truncates a number to the specified number of decimal places.

Syntax: TRUNC(<number>, <num_digits>)

9. **SIN**: Returns the sine of an angle given in radians.

Syntax: SIN(<number>)

10. **COS**: Returns the cosine of an angle given in radians.

Syntax: COS(<number>)

11. **TAN**: Returns the tangent of an angle given in radians.

Syntax: TAN(<number>)

12. **PI**: Returns the mathematical constant π (pi).

Syntax: PI()

Other functions: Perform unique actions not covered by other categories.

1. **BLANK**: Returns a blank value.

Syntax: BLANK()

2. **SWITCH**: Evaluates an expression against a list of values and returns the result corresponding to the first matching value.

Syntax: SWITCH(<expression>, <value1>, <result1>, [<value2>, <result2>, ...], [<default_result>])

3. **CONCATENATEX**: Concatenates the result of an expression evaluated for each row in a table.

Syntax: CONCATENATEX(<table>, <expression>, [<delimiter>])

4. **RELATED**: Retrieves a related value from another table.

Syntax: RELATED(<column>)

5. **SELECTEDVALUE**: Returns the value if there is only one value in the specified column in the current context; otherwise, returns blank.

Syntax: SELECTEDVALUE(<column>)

6. **UNICHAR**: Returns the Unicode character that corresponds to the specified numeric value.

Syntax: UNICHAR(<number>)

7. **UNICODE**: Returns the Unicode value of the first character of the text.

Syntax: UNICODE(<text>)

8. **ROW**: Returns a single row table that represents a row from a table.

Syntax: ROW(<column1>, <value1>, [<column2>, <value2>, ...])

9. **DATATABLE**: Creates an in-memory table.

Syntax: DATATABLE(<column1>, <type1>, <column2>, <type2>, ...)

10. **DATATABLESELECTCOLUMNS**: Creates an in-memory table by selecting columns from an existing table.

Syntax: DATATABLESELECTCOLUMNS(<table>, <column1>, [<column2>, ...])

Parent and Child functions: Manage data presented in a parent/child hierarchy in data models.

1. **PATH**: Returns a delimited text string that represents the path from the root node to a specified node in a hierarchy.

Syntax: PATH(<table>, <column>)

2. **PATHCONTAINS**: Checks whether a specified node is in the path of another node in a hierarchy.

Syntax: PATHCONTAINS(<table>, <column>, <target_node>)

3. **PATHITEM**: Returns the name of a node at a specified position in the path of another node in a hierarchy.

Syntax: PATHITEM(<path>, <index>)

4. **PATHLENGTH**: Returns the number of levels in the path from the root node to a specified node in a hierarchy.

Syntax: PATHLENGTH(<path>)

5. **RELATEDHIERARCHY**: Returns a related table filtered by a hierarchy.

Syntax: RELATEDHIERARCHY(<column>)

6. **ISEMPTY**: Checks whether a table or column is empty.

Syntax: ISEMPTY(<table_or_column>)

7. **ISCROSSFILTERED**: Checks whether a column is filtered by a hierarchy.

Syntax: ISCROSSFILTERED(<column>)

8. **HASONEFILTER**: Checks whether there is only one filter applied to a column.

Syntax: HASONEFILTER(<column>)

Relationship functions: Manage and utilize relationships between tables.

1. **RELATED**: Retrieves a related value from another table based on a one-to-many or many-to-one relationship.

Syntax: RELATED(<column>)

2. **RELATEDTABLE**: Returns a table related to the current table based on a specified relationship.

Syntax: RELATEDTABLE(<table>)

3. **CROSSFILTER**: Specifies the direction of filtering propagation across a relationship.

Syntax: CROSSFILTER(<table1>[,<column1>],[<table2>,<column2>],<direction>)

4. **USERELATIONSHIP**: Specifies an alternative relationship to be used in a calculation.

Syntax: USERELATIONSHIP(<column1>, <column2>)

5. **FILTERS**: Returns a table that contains the current filter context.

Syntax: FILTERS([<table>[, <column>[, <column>...]]])

6. **HASONEVALUE**: Checks whether a column has only one distinct value in the current filter context.

Syntax: HASONEVALUE(<column>)

7. **LOOKUPVALUE**: Returns the value in a column that corresponds to the result of a calculation.

Syntax: LOOKUPVALUE(<result_column>, <search_column>, <search_value>)

8. **PATH**: Returns a delimited text string that represents the path from the root node to a specified node in a hierarchy.

Syntax: PATH(<table>, <column>)

Statistical functions: Calculate values related to statistical distributions and probability.

1. **AVERAGEX**: Calculates the average of an expression evaluated for each row in a table.

Syntax: AVERAGEX(<table>, <expression>)

2. **COUNTAX**: Counts the number of rows in a table where the specified expression evaluates to a non-blank value.

Syntax: COUNTAX(<table>, <expression>)

3. **MAXX**: Returns the maximum value of an expression evaluated for each row in a table.

Syntax: MAXX(<table>, <expression>)

4. **MINX**: Returns the minimum value of an expression evaluated for each row in a table.

Syntax: MINX(<table>, <expression>)

5. **STDEV.P**: Calculates the standard deviation based on the entire population given as arguments.

Syntax: STDEV.P(<number1>, [<number2>, ...])

6. **STDEV.S**: Estimates the standard deviation based on a sample of the entire population.

Syntax: STDEV.S(<number1>, [<number2>, ...])

7. **VAR.P**: Calculates the variance based on the entire population given as arguments.

Syntax: VAR.P(<number1>, [<number2>, ...])

8. **VAR.S**: Estimates the variance based on a sample of the entire population.

Syntax: VAR.S(<number1>, [<number2>, ...])

9. **MEDIANX**: Calculates the median of an expression evaluated for each row in a table.

Syntax: MEDIANX(<table>, <expression>)

10. **PERCENTILE.EXC**: Returns the k-th percentile of values in a range, exclusive of 0 and 1.

Syntax: PERCENTILE.EXC(<array>, <k>)

Table manipulation functions: Return or manipulate tables.

1. **FILTER**: Returns a table that contains only the rows that satisfy the specified conditions.

Syntax: FILTER(<table>, <condition>)

2. **SELECTCOLUMNS**: Returns a new table with selected columns from the specified table.

Syntax: SELECTCOLUMNS(<table>, <column1>, [<column2>, ...])

3. **ADDCOLUMNS**: Returns a table with new columns added, calculated from existing columns.

Syntax: ADDCOLUMNS(<table>, <new_column1>, <expression1>, [<new_column2>, <expression2>, ...])

4. **SUMMARIZE**: Returns a summary table with grouped data.

Syntax: SUMMARIZE(<table>, <group_column1>, [<group_column2>, ...], <aggregation_expression1>, [<aggregation_expression2>, ...])

5. **GROUPBY**: Groups the rows of a table based on the values of one or more columns and then performs a calculation on each group.

Syntax: GROUPBY(<table>, <group_column1>, [<group_column2>, ...], <aggregation_expression1>, [<aggregation_expression2>, ...])

6. **DISTINCT**: Returns a table with unique rows based on the specified columns.

Syntax: DISTINCT(<table>)

7. **UNION**: Combines two or more tables into a single table.

Syntax: UNION(<table1>, <table2>, [<table3>, ...])

8. **EXCEPT**: Returns all the rows from one table that are not present in another table.

Syntax: EXCEPT(<table1>, <table2>)

9. **INTERSECT**: Returns all the rows that are common to two tables.

Syntax: INTERSECT(<table1>, <table2>)

10. **DATATABLE**: Creates an in-memory table with the specified columns and values.

Syntax: DATATABLE(<column1>, <type1>, <column2>, <type2>, ...)

Text functions: Return part of a string, search for text, or concatenate string values. Also control formats for dates, times, and numbers.

1. **CONCATENATE**: Concatenates two or more text strings.

Syntax: CONCATENATE(<text1>, <text2>, ...)

2. **LEFT**: Returns the leftmost characters from a text string.

Syntax: LEFT(<text>, <num_chars>)

3. **RIGHT**: Returns the rightmost characters from a text string.

Syntax: RIGHT(<text>, <num_chars>)

4. **MID**: Returns a specific number of characters from a text string, starting at a specified position.

Syntax: MID(<text>, <start_num>, <num_chars>)

5. **LEN**: Returns the length of a text string.

Syntax: LEN(<text>)

6. **LOWER**: Converts all characters in a text string to lowercase.

Syntax: LOWER(<text>)

7. **UPPER**: Converts all characters in a text string to uppercase.

Syntax: UPPER(<text>)

8. **TRIM**: Removes leading and trailing spaces from a text string.

Syntax: TRIM(<text>)

9. **FIND**: Returns the starting position of one text string within another text string.

Syntax: FIND(<find_text>, <within_text>, [<start_num>])

10. **SUBSTITUTE**: Replaces occurrences of a specified text string within another text string with a new text string.

Syntax: SUBSTITUTE(<text>, <old_text>, <new_text>, [<instance_num>])

11. **FORMAT**: Formats a value based on the specified format string.

Syntax: FORMAT(<value>, <format_string>)

12. **FORMATDATETIME**: Formats a datetime value based on the specified format string.

Syntax: FORMATDATETIME(<datetime>, <format_string>)

Time intelligence functions: Create calculations using built-in knowledge about calendars and dates for meaningful comparisons across time periods.

1. **TOTALYTD**: Calculates the year-to-date total for a given expression, up to the specified date.

Syntax: TOTALYTD(<expression>, <dates>)

2. **TOTALMTD**: Calculates the month-to-date total for a given expression, up to the specified date.

Syntax: TOTALMTD(<expression>, <dates>)

3. **TOTALQTD**: Calculates the quarter-to-date total for a given expression, up to the specified date.

Syntax: TOTALQTD(<expression>, <dates>)

4. **DATESYTD**: Returns a table of dates from the start of the year up to the specified date.

Syntax: DATESYTD(<dates>)

5. **DATESMTD**: Returns a table of dates from the start of the month up to the specified date.

Syntax: DATESMTD(<dates>)

6. **DATESQTD**: Returns a table of dates from the start of the quarter up to the specified date.

Syntax: DATESQTD(<dates>)

7. **DATESBETWEEN**: Returns a table of dates between two specified dates.

Syntax: DATESBETWEEN(<dates>, <start_date>, <end_date>)

8. **SAMEPERIODLASTYEAR**: Returns a table of dates for the same time period in the previous year.

Syntax: SAMEPERIODLASTYEAR(<dates>)

9. **PREVIOUSYEAR**: Returns a table of dates for the previous year.

Syntax: PREVIOUSYEAR(<dates>)

10. **DATESINPERIOD**: Returns a table of dates for a specified time period.

Syntax: DATESINPERIOD(<dates>, <start_date>, <number_of_intervals>, <interval>)