# Things to Try on Sample Projects:

1. Calculate total sales from accessories using filter function , Calculate()
2. Create a new measure ‘Previous Year Sales’ using Time Intelligence DAX function, previousyear()
3. Calculate Previous Month Sales as a new measure, in similar lines
4. Create New measures for ‘Quantity sold in last 3 months’ and ‘Profit for last 6 months’
5. Using DAX functions, calculate YTD, MTD and QTD as well as YoY and QoQ and MoM
6. Finally, create a report that includes all new calculated parameters with suitable visuals and capture the insights.
7. While Uploading first Transform Data > to open query editor
8. Use 1st row as Column Header > for promoted header.
9. Check the data type of all columns and do a column profiling. [Hint: Power Query editor ->View->Check Column profile, colum quality, column distribution]
10. Extract Month Name, Month No, Day Name and Year from Date Column. - (Add Column>Date>Yr , Month Name etc)
11. Create a new column ‘ Category- Subcategory’ by merging the respective columns in the Product table, using hyphen (-) as separator (Add Column > Merge Column)
12. Separate Customers first name and last name using customer column from Customer table (Transform > Split Column)
13. Create a new table “category wise sales’ with Product Category(Grouped)and sum of Total sales. Identify the two tables that are to be joined and the common column. Which Join kind is most suitable in this scenario? (Home > Merge Queries>Merge As New, to Merge 2 Tables and select the ‘column name’ & ‘Join Type’ to ensure all number of rows that you need are there in new query table. (Home>Group By> Define Column to group aggregation by & how to Aggregate = ‘Sum’
14. Round the Sales Amount to appropriate decimals - Transform > Rounding > round up
15. Check the query performance using power query diagnostics and comment on it. [Hint: Power Query Editor ->Tools ->Start Diagnostics-Stop Diagnostics].

# 

# 

# 1.What are the steps to navigate to data source settings in PowerBI?

To navigate to data source settings in PowerBI, follow these steps:

First, open PowerBI Desktop.

Second, navigate to the ‘Home’ tab on the ribbon.

Third, click on ‘Transform Data’ to open the Power Query Editor.

Fourth, in the Power Query Editor, select the ‘Home’ tab.

Fifth, click on ‘Data Source Settings’.

# 2.What are the steps to navigate to the power query editor in PowerBI?

To navigate to the Power Query Editor in PowerBI, follow these steps:

First, open PowerBI Desktop.

Second, go to the ‘Home’ tab on the ribbon.

Third, click on ‘Transform Data’ which will open the Power Query Editor.

# 3. How can data be cleaned using the power query editor in PowerBI?

Here are the steps to clean data using the Power Query Editor in Power BI, with numbering added for each step:

1. Access Power Query Editor:

In Power BI Desktop, navigate to the Home tab.

Click on Transform Data in the Queries section.

2. Identify and Remove Unnecessary Columns:

Select the columns you want to remove by clicking on their headers.

Right-click on the selected columns and choose Remove Columns from the context menu.

3. Rename Columns for Clarity:

Right-click on the header of a column you want to rename.

Select Rename from the menu and enter a new descriptive name.

4. Format Data for Consistency:

Select the column containing the data you want to format.

Go to the Transform tab and choose the appropriate formatting option (e.g., Text to Number, Change Type).

5. Handle Missing Values:

Select the column containing missing values.

In the Transform tab, click on Replace Values.

Specify the value you want to find (e.g., blank cells) and the value to replace it with (e.g., zero, specific text).

6. Remove Duplicate Rows (Optional):

Select the table.

Go to the Home tab and click Remove Duplicates.

Choose the columns that define unique rows (if applicable).

7. Cleanse Text Data (Optional):

Select the text column.

In the Transform tab, explore options like Trim, Uppercase, Lowercase, or Remove Extra Spaces to clean the text data.

8. Validate and Apply Changes:

Review the data preview pane to ensure the cleaning steps achieved the desired outcome.

Click Close & Apply in the top right corner to apply the transformations to your data model.

# 4. How does one create both line and stacked column charts in PowerBI?

To create a line and stacked column chart in PowerBI, follow these steps with ordinal numbers:

First, open your report in PowerBI Desktop.

Second, click on the ‘Report’ icon to ensure you are in the report view.

Third, select the ‘Line and Stacked Column Chart’ visual from the Visualizations pane.

Fourth, drag the desired fields to the respective ‘Axis’, ‘Column Values’, ‘Line Values’, and other relevant areas in the Fields pane.

Fifth, adjust the formatting options as needed to customize your chart’s appearance.

This will create a combined visual that displays both line and column data in a single chart.

# 5. How can scatter plot, gauge chart, and doughnut chart visuals be created in PowerBI?

To create scatter plot, gauge chart, and doughnut chart visuals in PowerBI, follow these steps with ordinal numbers:

For a Scatter Plot:

First, open your report in PowerBI Desktop.

Second, click on the ‘Report’ icon to ensure you are in the report view.

Third, select the ‘Scatter Chart’ visual from the Visualizations pane.

Fourth, drag the desired fields to the ‘Axis’, ‘Values’, and ‘Legend’ areas in the Fields pane.

Fifth, adjust the formatting options as needed to customize your scatter plot’s appearance.

For a Gauge Chart:

First, follow steps one and two above to be in the report view.

Second, select the ‘Gauge Chart’ visual from the Visualizations pane.

Third, drag the desired field to the ‘Value’ area and set your target value in the Fields pane.

Fourth, adjust the formatting options to customize your gauge chart’s appearance.

For a Doughnut Chart:

First, follow steps one and two above to be in the report view.

Second, select the ‘Doughnut Chart’ visual from the Visualizations pane.

Third, drag the desired fields to the ‘Values’ and ‘Legend’ areas in the Fields pane.

Fourth, adjust the formatting options as needed to customize your doughnut chart’s appearance.

# 6. How can a star schema be created?

To create a star schema in PowerBI, follow these steps with ordinal numbers:

First, import all the relevant tables into PowerBI Desktop.

Second, navigate to the ‘Model’ view by clicking on the ‘Model’ icon in the left sidebar.

Third, arrange your tables so that the fact table is in the center and dimension tables surround it.

Fourth, create relationships between the fact table and each dimension table by dragging a field from one to a matching field in another.

Fifth, ensure that each relationship is correctly set up as ‘one-to-many’ with the fact table on the ‘one’ side.

Sixth, adjust properties such as ‘Cross filter direction’ and ‘Make this relationship active’ as needed.

# 7. What are the steps to create ‘slicers’ on the reports?

To create slicers on PowerBI reports, follow these steps with ordinal numbers:

First, open your report in PowerBI Desktop.

Second, click on the ‘Report’ icon to ensure you are in the report view.

Third, select the ‘Slicer’ visual from the Visualizations pane.

Fourth, drag the field you want to use as a slicer into the ‘Field’ area in the Fields pane.

Fifth, adjust the slicer settings and formatting options as needed to customize its appearance and behavior.

Here's how to create slicers on specific reports and make them interactive in Power BI without impacting other reports:

1. Utilize Page-Level Filters:

Power BI offers page-level filters specifically designed to filter visuals within a single report page.

While creating your slicer, ensure you're on the desired report page.

In the Visualizations pane, under the Formatting tab, locate the Filters section.

Choose This page only from the Apply to dropdown menu.

2. Leverage Slicers with Slicer Pane (Optional):

Create your slicers as usual.

Select the slicers you want to group together.

In the Formatting pane, under Arrange, choose Create slicer pane.

This groups the slicers into a pane, making them visually distinct and potentially saving space.

You can position the slicer pane anywhere on the report page without affecting other visuals.

3. Employ Bookmarks for Conditional Slicers (Optional):

This approach is useful when you want slicers to be active only under specific conditions.

Create your slicers and any visuals they should filter.

Go to the View tab and select Bookmarks.

Create a bookmark for each desired slicer state (e.g., slicer selection or cleared).

Set the visibility of the slicers based on the selected bookmark. This allows you to control slicer interactivity on a report-by-report basis.

4. Consider Separate Reports for Complex Filtering:

If you have extensive filtering requirements for different reports, creating separate reports might be the best solution.

This ensures complete isolation of filter behavior between reports.

You can leverage features like linking to navigate between reports while maintaining independent filter states.

# 8. How can ‘filters’ be applied on the visuals?

To apply filters on visuals in PowerBI, follow these steps with ordinal numbers:

First, open your report in PowerBI Desktop.

Second, select the visual you want to apply a filter to.

Third, go to the ‘Filters’ pane on the right side of the screen.

Fourth, drag the field you want to filter by into the appropriate filter area (Visual level, Page level, or Report level).

Fifth, set the filter criteria by selecting values or ranges.

Here's how to create slicers on specific reports and make them interactive in Power BI without impacting other reports:

1. Utilize Page-Level Filters:

Power BI offers page-level filters specifically designed to filter visuals within a single report page.

While creating your slicer, ensure you're on the desired report page.

In the Visualizations pane, under the Formatting tab, locate the Filters section.

Choose This page only from the Apply to dropdown menu.

2. Leverage Slicers with Slicer Pane (Optional):

Create your slicers as usual.

Select the slicers you want to group together.

In the Formatting pane, under Arrange, choose Create slicer pane.

This groups the slicers into a pane, making them visually distinct and potentially saving space.

You can position the slicer pane anywhere on the report page without affecting other visuals.

3. Employ Bookmarks for Conditional Slicers (Optional):

This approach is useful when you want slicers to be active only under specific conditions.

Create your slicers and any visuals they should filter.

Go to the View tab and select Bookmarks.

Create a bookmark for each desired slicer state (e.g., slicer selection or cleared).

Set the visibility of the slicers based on the selected bookmark. This allows you to control slicer interactivity on a report-by-report basis.

4. Consider Separate Reports for Complex Filtering:

If you have extensive filtering requirements for different reports, creating separate reports might be the best solution.

This ensures complete isolation of filter behavior between reports.

You can leverage features like linking to navigate between reports while maintaining independent filter states.

1. How can a date dim table be created using calendar auto?

To create a date dimension table using the auto calendar feature in Power BI, follow these steps:

To create a date dimension table using the auto calendar feature in Power BI, follow these steps with ordinal numbers:

First, open Power BI Desktop.

Second, click on the “Home” tab in the ribbon.

Third, select “Enter Data” from the dropdown menu.

Fourth, choose “Auto Date/Time”.

Fifth, Configure Auto Calendar:

Specify the start date and end date for your date dimension.

Customize the granularity (daily, monthly, etc.) and other settings.

Sixth, Create Date Table:

Click “Load” to create the date dimension table.

Power BI will generate a table with date-related columns (e.g., Date, Year, Month, Quarter, etc.).

Seventh, Use the Date Table:

Drag the Date column from the newly created table into your data model.

Establish relationships with other tables using this date column.

2ND SOLUTION:

Here's how to create a date dimension table in Power BI using the CALENDARAUTO function:

**1. Access Modeling Tab:**

* In Power BI Desktop, navigate to the **Modeling** tab. This tab is used to manage tables and relationships within your data model.

**2. Create New Table:**

* Click on **New Table** in the **Modeling** tab. This initiates the creation of a new table using DAX (Data Analysis Expressions).

**3. Utilize CALENDARAUTO Function:**

* In the formula bar that appears, type the following DAX expression:

Code snippet

Calendar = CALENDARAUTO()

Use code with caution.

content\_copy

* Replace "Calendar" with your desired name for the date dimension table.
* The CALENDARAUTO() function automatically generates a table containing all dates present in your data model.

**4. Define Table (Optional):**

* Although optional, it's good practice to define the table as a Date table. Right-click on the newly created table name in the Fields pane and select **Mark as Date Table**. This ensures Power BI recognizes it as a date dimension table, potentially enabling features like time intelligence calculations.

**Additional Considerations:**

* **Calendar vs. CALENDARAUTO:** While similar, the CALENDAR function allows you to specify a start and end date for the generated table, whereas CALENDARAUTO uses the existing date range in your data model.
* **Adding Date Columns:** You can extend your date dimension table by adding additional calculated columns based on the existing date column. These might include Year, Month, Quarter, Day of Week, etc. Use DAX functions like YEAR, MONTH, WEEKNUM, etc., to extract these attributes from the date column.

By following these steps, you can create a dynamic date dimension table in Power BI using the CALENDARAUTO function. This table can then be used to filter, group, and perform time intelligence calculations on your data.

2. What is the process to extract Year, Quarter, Month, Day, and Month name from the created date dim table using the specified functions? Year (use year function)  
Quarter (Use quarter function)  
Month (use Month function)  
Day (use Day function)  
Month name ( use Format function)

To create these date-related columns in Power BI using the appropriate functions:

First: Open your Power BI report.

Second: Go to the Model view.

Third: Create a new calculated column for the year using the formula: YearColumn = YEAR('YourDateColumn').

Fourth: Create another calculated column for the quarter using the formula: QuarterColumn = QUARTER('YourDateColumn').

Fifth: Create a third calculated column for the month using the formula: MonthColumn = MONTH('YourDateColumn').

Sixth: Create yet another calculated column for the day using the formula: DayColumn = DAY('YourDateColumn').

Seventh: Finally, create a calculated column for the month name using the formula: MonthNameColumn = FORMAT('YourDateColumn', "MMMM").

3.How can the Upper, Lower, and Trim functions be applied to a chosen column in the table?1.Upper (Use Upper function)2.Lower (use Lower function)3.Trim(use Trim function)

Here is how I can apply the Upper, Lower, and Trim functions to a chosen column in Power BI:

1. Upper (UPPER function):

Open your Power BI report.

Go to the Model view.

Create a new calculated column with the formula: UpperColumn = UPPER('YourChosenColumn').

Replace 'YourChosenColumn' with the actual name of the column you want to convert to uppercase.

2. Lower (LOWER function):

Create another calculated column: LowerColumn = LOWER('YourChosenColumn').

This will convert the values in the chosen column to lowercase.

3. Trim (TRIM function):

Create yet another calculated column: TrimmedColumn = TRIM('YourChosenColumn').

The TRIM function removes any leading or trailing spaces from the values in the chosen column.

4.What are the steps to create new columns by applying formulas/conditional statements such as If condition, Int(), and round()? If condition (conditional column)  
Int(), round ()

Here is the breakdown the steps for creating new calculated columns in Power BI using formulas and conditional statements. We’ll cover the IF condition, INT(), and ROUND() functions:

First: Open your Power BI report.

Second: Go to the Model view.

Third: Create a new calculated column for the IF condition:

Click on the “New Column” button in the Modeling tab.

Use the following formula:

NewColumn = IF('YourCondition', ValueIfTrue, ValueIfFalse)

Replace 'YourCondition' with your actual condition (e.g., [Sales] > 1000). Specify ValueIfTrue and ValueIfFalse for the desired results.

Fourth: Create a new column using the INT() function:

Still in the Model view, create another calculated column:

IntColumn = INT('YourNumericColumn')

Replace 'YourNumericColumn' with the actual numeric column you want to convert to an integer.

Fifth: Create yet another new column using the ROUND() function:

Create a third calculated column:

RoundedColumn = ROUND('YourNumericColumn', NumDecimalPlaces)

Replace 'YourNumericColumn' with the numeric column you want to round. Adjust NumDecimalPlaces to the desired number of decimal places (e.g., 2 for two decimal places).

5. How can the Min and Max functions be applied to the new custom columns? Min (Use Min function)  
Max ( Use Max function)

Let’s create the calculated columns for the minimum and maximum values in Power BI:

1. Min (Using the Min function):

Open your Power BI report.

Navigate to the Model view.

Create a new calculated column with the following formula:

MinValueColumn = MIN('YourCustomColumn')

Replace 'YourCustomColumn' with the actual name of your custom column.

2. Max (Using the Max function):

Create another calculated column:

MaxValueColumn = MAX('YourCustomColumn')

Again, replace 'YourCustomColumn' with the appropriate column name.

6.Can you explain the ‘all except’ function and provide an example of writing a DAX using the all except function?

In Power BI, the ALLEXCEPT function is a powerful tool within DAX (Data Analysis Expressions) used to temporarily remove filters from specific columns in a table while keeping filters applied to others.

Here's a breakdown of the ALLEXCEPT function:

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

<table>: The table you want to modify filtering for.

<column1>, <column2>, ...: Columns for which you want to retain existing filters.

Functionality: ALLEXCEPT doesn't directly change the existing filters in your data model. Instead, it creates a temporary context within a DAX calculation, allowing you to evaluate an expression while ignoring filters on specified columns.

Example: Analyzing Sales by Product Category Regardless of Date Filters

Let's say you have a table named "Sales" with columns for Product,

7. How can the Time, Weekday, Weeknumber, and Weekday name be extracted from the date dim table in Power BI? Time ( Use time function) Weekday ( Use Weekday function) Weeknumber (use Weeknum function) Weekday name ( use Format function)

Here's how to extract Time, Weekday, Weeknumber, and Weekday name from a date dimension table in Power BI using the functions you mentioned:

**1. Time:**

* Use the TIME function to extract the time portion from the date column.

**DAX Example:**

Code snippet

Time = TIME(DimDate[Date])

* This formula creates a new calculated column named "Time" that extracts the time component (hours, minutes, seconds) from the "Date" column in your date dimension table named "DimDate".

**2. Weekday:**

* Use the WEEKDAY function to determine the day of the week (as a number) from the date column.

**DAX Example:**

Code snippet

Weekday = WEEKDAY(DimDate[Date])

* This formula creates a new calculated column named "Weekday" that assigns a numerical value (1-7) to each day of the week, with 1 representing Sunday and 7 representing Saturday, based on the "Date" column in "DimDate".

**3. Weeknumber:**

* Use the WEEKNUM function to extract the week number of the year from the date column.

**DAX Example:**

Code snippet

WeekNumber = WEEKNUM(DimDate[Date])

* This formula creates a new calculated column named "WeekNumber" that shows the ISO 8601 week number (1-52 or 1-53) for each date in "DimDate", considering the date's position within the year.

**4. Weekday Name:**

* Use the FORMAT function to convert the numeric weekday value (extracted earlier) into a textual representation of the day name.

**DAX Example:**

Code snippet

WeekdayName = FORMAT(WEEKDAY(DimDate[Date]), "ddd")

* This formula creates a new calculated column named "WeekdayName" that displays the abbreviated weekday name (e.g., "Mon", "Tue", "Wed") based on the corresponding numeric weekday value obtained from the "WEEKDAY" function. You can adjust the format code within the FORMAT function to get different weekday name representations (e.g., "dddd" for full names).

**Remember to replace "DimDate" with the actual name of your date dimension table.**

By implementing these DAX expressions, you can enrich your date dimension table with additional columns containing the desired temporal components. These new columns can then be used for further analysis and visualisation in your Power BI reports.

8.Which column would you like to choose to perform the following functions (Mid, Len, Left, Right) in Power BI by creating new custom columns?Mid(Use Upper function)Len(use Lower function)Left(use Trim function)Right( Use right function)

Let’s create new custom columns in Power BI using the specified functions:

1. Mid (Using UPPER function):

Suppose you have a column called Description containing text data. You want to extract a portion of the text from the middle (e.g., characters 3 to 6). Create a new calculated column:

MidColumn = MID(UPPER('YourColumn'), 3, 4)

Replace 'YourColumn' with the actual name of your column.

2. Len (Using LOWER function):

If you want to calculate the length of each value in a column (e.g., the length of product names), create another calculated column:

LenColumn = LEN(LOWER('YourColumn'))

Again, replace 'YourColumn' with the appropriate column name.

3. Left (Using TRIM function):

To extract the leftmost part of a text value (e.g., the first 10 characters), create a new calculated column:

LeftColumn = LEFT(TRIM('YourColumn'), 10)

Replace 'YourColumn' with the actual column name.

4. Right (Using RIGHT function):

If you want to extract the rightmost part of a text value (e.g., the last 5 characters), create one more calculated column:

RightColumn = RIGHT('YourColumn', 5)

Once again, replace 'YourColumn' with the appropriate column name.

9. How to create custom columns in Power BI to perform logical functions such as AND, OR, and Switch based on the specified requirements? AND / OR / SWITCH

To create custom columns in Power BI to perform logical functions using DAX expressions:

1. AND Function:

Suppose you have two columns: IsHighValue (a boolean column) and IsUrgent (another boolean column). You want to create a new column that checks if both conditions are true. Create a calculated column:

HighAndUrgent = AND('YourTable'[IsHighValue], 'YourTable'[IsUrgent])

Replace 'YourTable' with the actual name of your table.

2. OR Function:

If you want to check if either condition is true, create another calculated column:

HighOrUrgent = OR('YourTable'[IsHighValue], 'YourTable'[IsUrgent])

3. SWITCH Function:

The SWITCH function allows you to perform conditional logic based on specific values. For example, let’s say you have a column called Status with values like “Open,” “In Progress,” and “Closed.” You want to create a new column that assigns a numerical value based on the status:

StatusValue =

SWITCH(

'YourTable'[Status],

"Open", 1,

"In Progress", 2,

"Closed", 3,

0

)

Replace 'YourTable' with your actual table name and adjust the status values and corresponding numerical values as needed.

# 10. Which custom columns would you like to create in Power BI to perform Count and AVG functions on specific data? Count(Use Count function) AVG( Use AVG function)

To create custom columns in Power BI to perform the Count and AVG (average) functions:

1. Count (Using COUNT function):

Suppose you have a table called Sales with a column named SalesAmount. You want to create a new calculated column that shows the count of sales transactions. Here’s how you can do it:

TotalSalesCount = COUNT('Sales'[SalesAmount])

Replace 'Sales' with the actual name of your table.

2. AVG (Using AVERAGE function):

If you want to calculate the average sales amount, create another calculated column:

AverageSalesAmount = AVERAGE('Sales'[SalesAmount])

Again, adjust the table and column names as needed.

# 11.Explore Calculate and Keepfilters functions in Power BI, and provide an example of creating new custom columns using these functions?

To create CALCULATE and KEEPFILTERS functions in Power BI, along with examples of creating new custom columns use these functions:

1. CALCULATE Function:

The CALCULATE function is a powerful tool for modifying filter context within a DAX expression. It allows you to create dynamic calculations by temporarily altering filters.

Syntax: CALCULATE(<expression>, <filter1>, <filter2>, ...)

Example: Suppose you have a sales table with columns SalesAmount, ProductCategory, and Region. You want to create a new column that calculates the total sales amount for the “Electronics” category, regardless of any other filters applied (such as date or region).

TotalSalesElectronics =

CALCULATE(

SUM('Sales'[SalesAmount]),

'Sales'[ProductCategory] = "Electronics"

)

This custom column will always show the total sales for “Electronics,” regardless of other slicer selections.

2. KEEPFILTERS Function:

The KEEPFILTERS function preserves existing filters while applying new ones. It’s useful when you want to override some filters but keep others intact.

Syntax: KEEPFILTERS(<expression>)

Example: Suppose you have a table with a column OrderStatus (values: “Pending,” “Shipped,” “Cancelled”). You want to create a new column that counts the number of “Shipped” orders, considering only the “Shipped” status.

ShippedOrdersCount =

COUNTROWS(

KEEPFILTERS('Orders'[OrderStatus] = "Shipped")

)

This custom column will count only the “Shipped” orders, regardless of other filters applied.

# 12. How can YTD, QTD, and MTD be calculated?

Here are the steps to calculate Year-to-Date (YTD), Quarter-to-Date (QTD), and Month-to-Date (MTD) values in Power BI:

1. Year-to-Date (YTD):

YTD represents the cumulative total from the beginning of the year up to a specific date. To calculate YTD, follow these steps:

Create a measure that sums the relevant metric (e.g., sales amount) for the selected period.

Use the DATESYTD function to filter the data from the start of the year up to the current date:

YTD Sales =

CALCULATE(

SUM('Sales'[SalesAmount]),

DATESYTD('Date'[Date])

)

Replace 'Sales' with your actual table name and 'Date' with your date column.

2. Quarter-to-Date (QTD):

QTD represents the cumulative total from the beginning of the quarter up to a specific date. To calculate QTD:

Create a measure that sums the relevant metric (e.g., profit) for the selected period.

Use the DATESQTD function to filter the data from the start of the quarter up to the current date:

QTD Profit =

CALCULATE(

SUM('Profit'[ProfitAmount]),

DATESQTD('Date'[Date])

)

3. Month-to-Date (MTD):

MTD represents the cumulative total from the beginning of the month up to a specific date. To calculate MTD:

Create a measure that sums the relevant metric (e.g., expenses) for the selected period.

Use the DATESMTD function to filter the data from the start of the month up to the current date:

MTD Expenses =

CALCULATE(

SUM('Expenses'[ExpenseAmount]),

DATESMTD('Date'[Date])

)

# 13. In Power BI, What are the steps to using the DATESBETWEEN function?

The DATESBETWEEN function in Power BI allows you to filter a table or column within a specified date range. Here are the steps to use it:

Syntax:

The basic syntax of DATESBETWEEN is as follows:

DATESBETWEEN(<date\_column>, <start\_date>, <end\_date>)

<date\_column>: The date column you want to filter.

<start\_date>: The beginning of the date range.

<end\_date>: The end of the date range.

Example:

Suppose you have a table named Sales with a date column called OrderDate. You want to calculate the total sales amount for orders placed between January 1, 2022, and March 31, 2022.

Create a new measure (calculated column) using DATESBETWEEN:

TotalSalesQ1 =

CALCULATE(

SUM('Sales'[SalesAmount]),

DATESBETWEEN('Sales'[OrderDate], DATE(2022, 1, 1), DATE(2022, 3, 31))

)

Replace 'Sales' with your actual table name and adjust the date range as needed.

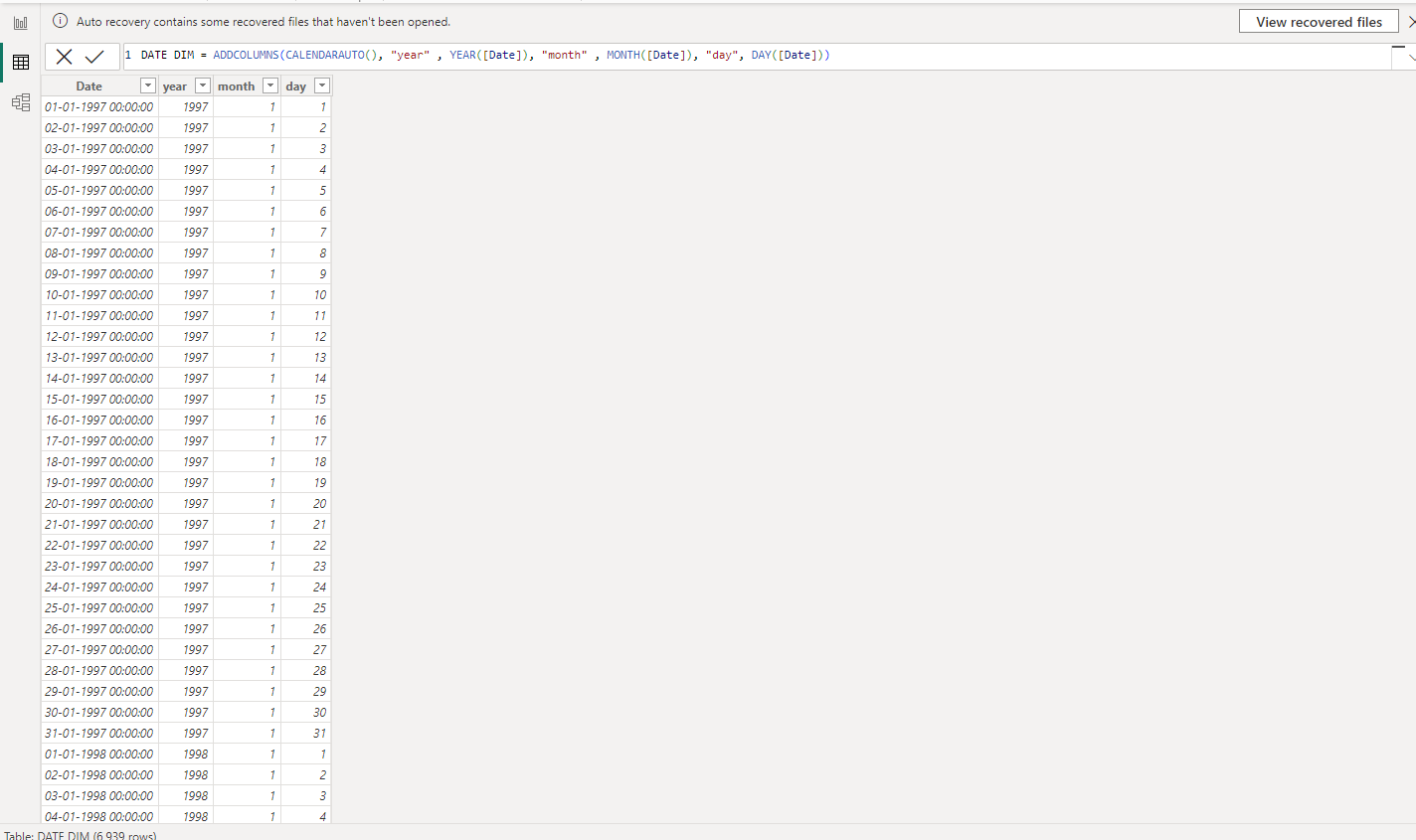
Usage Notes:

DATESBETWEEN considers both the start and end dates, so the range is inclusive.

You can use any valid DAX expression for the start and end dates (e.g., referencing other measures or using relative dates).

==================

Task1: How can a date dim table be created using calendar auto?



Task 2: What is the process to extract Year, Quarter, Month, Day, and Month name from the created date dim table using the specified functions?

Year (use year function)

Quarter (Use quarter function)

Month (use Month function)

Day (use Day function)

Month name ( use Format function)

a. Year (use year function)



b. Quarter (Use quarter function)



c. Month (use Month function)



d. Day (use Day function)



e. Month name ( use Format function)



Replace [CET] with your columnname.

Task 3: How can the Upper, Lower, and Trim functions be applied to a chosen column in the table?

1.Upper (Use Upper function)

2.Lower (use Lower function)

3.Trim(use Trim function)

f. Upper (Use Upper function)

Converts the whole column in upper case



g. Lower (use Lower function)

Converts the whole column in lower case



h. Trim(use Trim function)

Removes the space between the words. Single space is ignored. Multiple spaces are removed



Task 4: What are the steps to create new columns by applying formulas/conditional statements such as If condition, Int(), and round()?

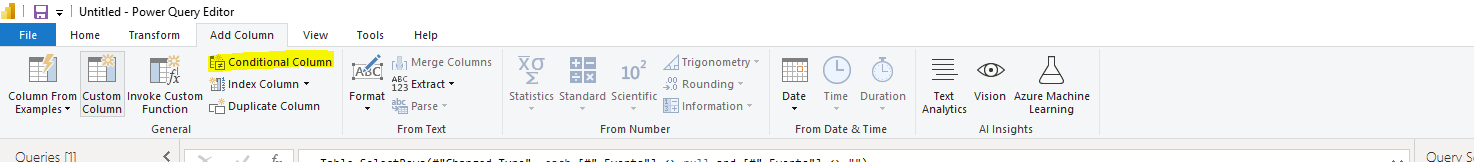
i. If condition ( conditional columns)

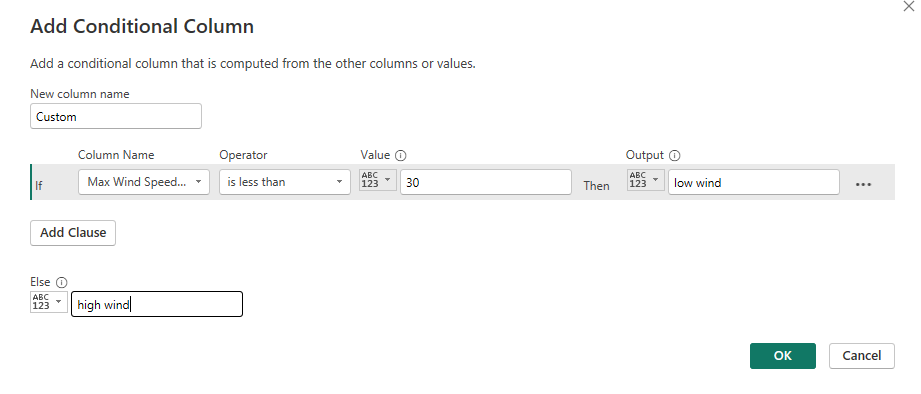
table view



You can do this in table view or in transform query as well.

Power Query





j. Int(), round ()

This will convert any numerical form of number to integer.



The ROUND function is used to round a number to a specified number of digits.



Task 5:How can the Min and Max functions be applied to the new custom columns?

Min (Use Min function)

Max ( Use Max function)

k. Min (Use Min function)

It retrieves the smallest numerical value from the specified column.



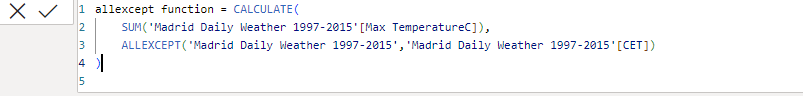
l. Max ( Use Max function)

It retrieves the highest numerical value from the specified column.



Task6: Can you explain the all except function and provide an example of writing a DAX using the all except function?

The ALLExcept function is used to remove filters from all columns in a table except for the specified columns. This function is particularly useful when you want to retain filters on certain columns while removing filters on others.



# 14. How are the Sum and Sumx functions utilized in Power BI for data aggregation and calculations?

Here are the steps to use the SUM and SUMX functions in Power BI for data aggregation and calculations:

1. SUM Function:

The SUM function is straightforward and commonly used for aggregating numeric values within a column.

It calculates the sum of a specific column for the entire dataset or within a filtered context.

To use SUM:

Create a new measure (calculated column) using the SUM function:

TotalSales = SUM('Sales'[SalesAmount])

Replace 'Sales' with your actual table name and 'SalesAmount' with the appropriate column.

2.SUMX Function:

The SUMX function is more versatile and allows you to perform calculations row by row before aggregating the results.

It iterates through each row of a table, evaluates an expression, and then sums up the individual results.

Common use cases include multiplying quantity and price per transaction before calculating total sales.

To use SUMX:

Create a new measure (calculated column) using the SUMX function:

TotalSalesX = SUMX('Sales', 'Sales'[Quantity] \* 'Sales'[Price])

Here, we multiply the quantity and price for each transaction and then sum up the results.

Steps to Create Measures:

Open Power BI Desktop.

Go to the Modeling tab.

Click on the New Measure icon.

Name your measure and enter the appropriate DAX expression.

Remember to adjust the column names and expressions based on your specific data model.

# 15. Can you explain the functions Distinct, ISBLANK, RANKX, and concatenate in Power BI?

Here is an explaination of the 4 different functions:

DISTINCT Function:

The DISTINCT function returns a table with unique values from a specified column.

Example: If you have a column called ProductCategory and want to create a new table with distinct product categories, you can use:

UniqueCategories = DISTINCT('Sales'[ProductCategory])

ISBLANK Function:

The ISBLANK function checks if a value is blank (empty) or null.

It returns TRUE if the value is blank, otherwise FALSE.

Example: To create a calculated column that flags empty order descriptions, use:

EmptyDescriptionFlag = IF(ISBLANK('Orders'[Description]), TRUE(), FALSE())

RANKX Function:

The RANKX function calculates the rank of a value within a column based on a specified measure.

It’s useful for ranking data, such as sales reps by total sales.

Example: To rank products by revenue, create a measure like this:

ProductRank = RANKX('Products', [TotalRevenue], , DESC)

CONCATENATEX Function:

The CONCATENATEX function concatenates values from a column into a single text string.

It’s often used for creating summary text or labels.

Example: To concatenate distinct product names, use:

AllProducts = CONCATENATEX(VALUES('Products'[ProductName]), 'Products'[ProductName], ", ")

# 16. How to compare the Sameperoid last year sales (Use SAMEPERIODLASTYEAR function)

To compare sales for the same period last year using the SAMEPERIODLASTYEAR function in Power BI, follow these steps:

Create a Measure:

Open your Power BI report.

Go to the Modeling tab.

Click on the New Measure icon.

Name your measure (e.g., “Revenue Last Year”).

Write the DAX Formula:

Use the SAMEPERIODLASTYEAR function to calculate the previous year’s sales.

For example:

[Revenue Last Year] = CALCULATE(SUM(Sales[Revenue]), SAMEPERIODLASTYEAR(Calendar[Date]))

Replace 'Sales' with your actual table name and 'Revenue' with the appropriate column.

Apply to Visualizations:

Apply the newly created measure ([Revenue Last Year]) to your visualizations (charts, tables, etc.).

This will allow you to compare the current revenue with the same period last year.

++++++=++

DAX FUNCTIONS :

DAX ExpressionsNotes

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"

# (DAX) Data Analysis Expressions 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.

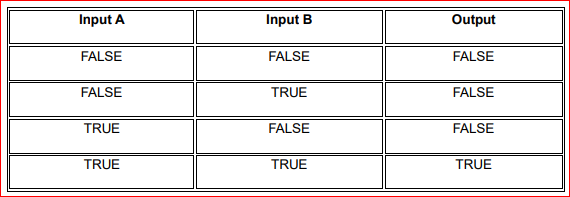
Syntax: USERPRINCIPALNAME()

# **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

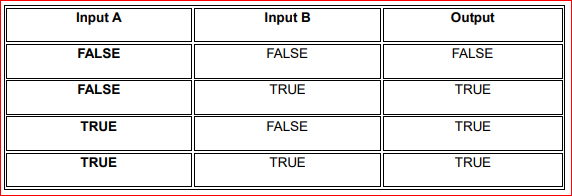
tables for the basic logical operations AND, OR, and NOT:

1. AND Truth Table:



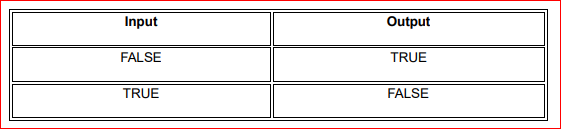
In the AND operation, the output is TRUE only if both inputs are TRUE; otherwise, the output is FALSE.

2. OR Truth Table:



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:



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>)