Introduction

Filter functions are a central element of the data analysis process in DAX and Power BI. These functions allow analysts to manipulate data by applying filters to tables and columns, enabling more precise and granular data analysis. Filter functions are normally used within the **CALCULATE** function of DAX.

In this reading, you'll explore some common examples of filter functions used in **CALCULATE**, like Boolean and table filter expressions. You'll also review some examples of filter modification functions.

Syntax

Let's briefly recap the **CALCULATE** syntax with the following example:

```
CALCULATE(<expression>[, <filter1> [, <filter2>])
```

The syntax of this DAX formula begins with the **CALCULATE** keyword, followed by the argument in parentheses. The argument contains an expression and one or more filters. These filters are one of the key elements of the **CALCULATE** function. As you learned earlier in this course, filters can be used to retrieve greater levels of granularity from your data.

Let's review the different types of filters and how they work inside the **CALCULATE** function.

Boolean filter expressions

A Boolean filter expression evaluates as either **TRUE** or **FALSE**. In other words, the statement returns a positive or negative value.

There are several rules that Boolean filter expressions must abide by:

- They can reference columns from a single table.
- They cannot reference measures.
- They cannot use a nested **CALCULATE** function.
- They cannot use functions that scan or return a table, including aggregation functions.

For example, Adventure Works wants to calculate the total sales of high-end products, defined as products whose unit price is equal to or greater than **$500**. This calculation can be performed using a Boolean filter with **CALCULATE** in DAX as follows:

```
Sales of high-end products =
CALCULATE (
    SUM ( Sales[Total Sales] ),
    FILTER ( Products, Products[Unit Price] >= 500 )
)
```

The measure makes use of a **FILTER** function to apply its second argument. This argument checks the **Unit Price** column of the **Products** table and evaluates all records as either True or False. Any records with a value greater than,

or equal to, **$500** are evaluated as **TRUE**. So, this formula filters all **TRU**E records and returns a table listing all products whose unit price is greater than or equal to **$500**.

Table filter expressions

Table filter expressions implement a table as a filter. Such an expression could reference a table in the data model, but more likely, it is a function that returns a table object.

You can also use the **FILTER** function to apply complex filter conditions, including those that a Boolean filter expression cannot define.

For example, Adventure Works wants to generate its total sales in **Germany** for 2019. To calculate this formula, Adventure Works must use two table filters. The first filter is from the **Region** table, and the second is from the **Date** table as follows:

```
2019 Sales in Germany =
CALCULATE (
    SUM ( [Total Sales] ),
    FILTER ( Region[Country] = "Germany", Date[Year] = "2019" )
)
```

In this **CALCULATE** statement, the **FILTER** function identifies all instances of **Germany** in the **Country** column of the **Region** table. It also identifies all instances of **2019** in the **Year** column of the **Date** table. It then returns these results as a calculated table that computes the total sales for Germany in 2019.

Filter modification functions

The filter modifier functions are used for more than adding filters. They provide additional control when modifying a filter context. For example, when you add a filter to the **CALCULATE** statement, **CALCULATE** overrides any existing filters previously created in the column. You must use filter modifier functions to ensure that **CALCULATE** adds the new filter to any previous filter.

Here's an overview of some of the most common filter modification functions used within the **CALCULATE** function of DAX.

REMOVEFILTERS

The **REMOVEFILTERS** function is used to remove all filters or remove filters from one or more columns of a table or all columns of a single table.

For example, a **Product** and **Region** filter has been applied to the tables in theAdventure Worksdatabase.You want to analyze the company's total sales without any filters applied to the **Sales** table. You can write a DAX expression as follows:

```
Total Sales =
CALCULATE
(
    [Total Sales],
    REMOVEFILTERS ( Product ),
    REMOVEFILTERS ( Region)
)
```

REMOVEFILTERS eliminates all filters from the tables specified within the parentheses. In this case, it removes

filters from the **Product** and **Region** dimensions.

KEEPFILTERS

KEEPFILTERS adds a filter without removing existing filters on the same columns. In other words, you can keep the existing filters and still evaluate the expression.

For example, Adventure Works must calculate its total sales while keeping the existing color filter on the **Product** table. The required DAX expression is as follows:

```
Blue Products Sale =
CALCULATE
(
    [Total Sales],
    KEEPFILTERS ( Products[Color] = "Blue" )
)
```

If an existing active filter exists on **Product[Color]**, then **KEEPFILTERS** ensures the filter is not overridden. Instead, it is merged with the new filter.

ALL

The **ALL** family of functions (**ALLEXCEPT, ALLSELECTED, ALLNOBLANKROW**) removes filters from one or more columns. They can also be used to remove all columns of a single table.

This function acts in an equivalent way to **REMOVEFILTERS**.

For example, Adventure Works can use the **ALL** function to calculate total sales for every region as follows:

```
Total Sales =
CALCULATE
(
    [Total Sales],
    ALL ( Product ),
    ALL ( Region)
)
```

In this formula, the **ALL** function returns the required results as a table or a column, which can be used in other DAX functions. **REMOVEFILTERS** does not return a table or column. Instead, it just removes filters from the specified tables or columns.

CROSSFILTER

The **CROSSFILTER** function modifies the filter direction (from **Both** to **Single**, or from **Single** to **Both**). It can also be used to disable a relationship between tables.

Adventure Works wants to analyze its total number of products by a specific year. However, the default cross-filter direction of the data model does not allow the company to compute the value. So, Adventure Works can create a measure using DAX without changing the default cross-filter direction in the data model as follows:

```
Product by Year =
CALCULATE
(
    DISTINCTCOUNT ( Products[ProductKey] ),
    CROSSFILTER
```

```
        ( Sales[ProductKey],
        Products[ProductKey],
        BOTH )
)
```

The formula analyzes the measure based on the **Year** column from the **Date** table and returns accurate results according to the analytical needs of the business.

USERELATIONSHIP

The **USERELATIONSHIP** function engages an inactive relationship between related columns. This means that the active relationship will automatically become inactive.

By default, Power BI utilizes the active relationship for all analysis and visualization. However, there may be times when your analysis requires using the inactive relationship. In this instance, you can use **USERELATIONSHIP**.

For example, Adventure Works wants to calculate its total sales using the **Shipping Date** column from the **Sales** table and the **Date** column from the **Date** table. This relationship is currently inactive. So, Adventure Works can use the **USERELATIONSHIP** function within the **CALCULATE** function as follows:

```
Sales by Shipping date =
CALCULATE (
    SUM ( Sales[SalesAmount] ),
    USERELATIONSHIP ( Sales[ShippingDate], Date[Date] )
)
```

In this formula, the **USERELATIONSHIP** function only switches the default relationship between related tables for the current measures. Remember that this approach typically handles inactive relationships within role-playing dimensions.

Points to remember

Making use of logical operators

When you use multiple filters in the **CALCULATE** function, they can be executed using logical operators.

You can use the **AND** (**&&**) logical operator, which means all conditions must be true. Or you can use the **OR** (**||**) logical operator, meaning either condition can be true. Which logical operator you use depends on your analytical requirements.

ALL filter modification functions

The **ALL** filter modification functions work as filter modifiers and as a function that returns a table object.

CALCULATE filter results

The results of a filter argument in the **CALCULATE** function are always presented as a table with one or more columns. You can use a filter function inside the **CALCULATE** function to create virtual tables dynamically based on filter conditions. The result of the filter argument is always a virtual table with one or more columns. The filter reduces the number of rows in the table.