How to Use the “Not Equal” Operator in Power BI Measures

Authored by

mohammed looti

January 10, 2026

RECOMMENDED CITATION

mohammed looti (2026). *How to Use the “Not Equal” Operator in Power BI Measures*. PSYCHOLOGICAL SCALES. Retrieved from https://scales.arabpsychology.com/?p=125282

The Not Equal operator is a fundamental tool within DAX (Data Analysis Expressions) used extensively in Power BI for comparative logic. Its primary function is to compare two distinct values or expressions, yielding a definitive Boolean result--either true or false. This capability is paramount for creating complex calculated fields that require identifying data points that deviate from a specified condition, such as sales revenue not equal to a specific target or customer satisfaction ratings not equal to a certain threshold. Utilizing this operator, analysts can efficiently flag discrepancies and focus precisely on outliers. This core functionality significantly enhances Data analysis by allowing users to filter and visualize data based on explicit inequalities, thereby providing crucial insights into areas that may require further attention or improvement.

Defining the Not Equal Operator in DAX

In the DAX language environment, the syntax employed to represent the "not equal to" condition is the less than and greater than symbols used consecutively, forming the expression **<>**. This symbol is critical for writing concise and effective conditional logic within measures, calculated columns, or calculated tables. It acts as an inequality constraint, evaluating whether the left operand holds a value that is different from the right operand.

Understanding the role of the **<>** operator is essential for any advanced Power BI user. When incorporated into functions like `IF`, `FILTER`, or `CALCULATETABLE`, it allows for highly selective data manipulation. For example, instead of filtering for all items equal to "A," using **<>** enables the inclusion of all items *except* "A." This inversed filtering capability is often more efficient and straightforward when dealing with large datasets where the exclusion criterion is simple, but the inclusion criteria are numerous and complex.

Core Applications of Inequality Operators

When working with the **<>** operator in DAX, there are two common and powerful ways to apply this inequality logic to transform and analyze data within your Power BI model. These methods address distinct needs: one focusing on row-level classification, and the other on modifying the dataset structure itself.

You can use **<>** as the "not equal" operator in DAX.

There are two common ways to use this operator in practice:

Method 1: Use <> to Create New Column (Row-Level Context)

Team Classification =
IF('my_data' <> "B", "Not on Team B", "On Team B")

This first method involves utilizing the operator within a calculated column, typically combined with the `IF` function. Since calculated columns operate row-by-row, the **<>** operator checks the value in a specific column for the current row against a static value. If the values differ (i.e., they are not equal), a specific result is returned; otherwise, an alternative result is generated. This is essential for classification and tagging data at the lowest granular level.

Method 2: Use <> to Create New Table (Filter Context)

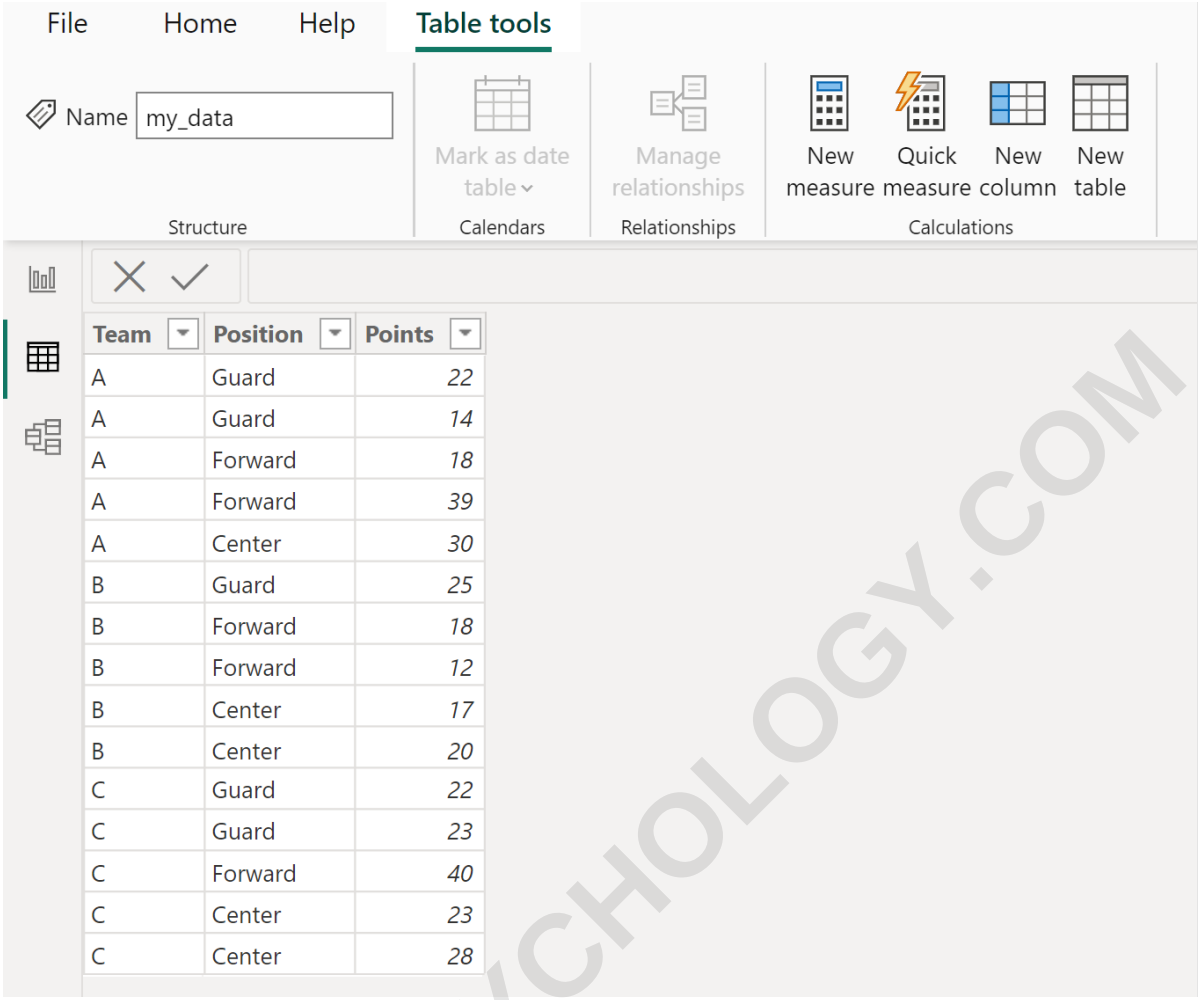
```
filtered_data =
CALCULATETABLE('my_data', 'my_data' <> "B")
```

The second primary application involves using **<>** within functions designed to return a filtered table, such as `FILTER` or `CALCULATETABLE`. In this scenario, the inequality condition defines the filter context for the entire table creation process. Only those rows where the specified column value is not equal to the benchmark value are included in the resulting table. This method is invaluable for subsetting data, creating dimension tables based on specific exclusions, or preparing data for subsequent detailed analysis.

Setting Up the Sample Dataset

To demonstrate the practical implementation of the "not equal" operator, we will utilize a sample dataset. This table, named **my_data**, is structured within Power BI and contains information about various basketball players. This simple dataset allows us to clearly illustrate how both row-level classification and table-level filtering are achieved using the **<>** operator.

The following examples show how to use each of these methods in practice with the following table named **my_data** in Power BI that contains information about various basketball players:



Team	Position	Points
A	Guard	22
A	Guard	14
A	Forward	18
A	Forward	39
A	Center	30
B	Guard	25
B	Forward	18
B	Forward	12
B	Center	17
B	Center	20
C	Guard	22
C	Guard	23
C	Forward	40
C	Center	23
C	Center	28

As illustrated in the image above, the **my_data** table includes columns such as Player Name, Position, and Team assignment. Our goal in the upcoming examples is to isolate or classify players who are **not** members of 'Team B'. This process is foundational to many data analysis tasks where specific subsets of data must be excluded from certain calculations or reports, making the identification of unequal values critically important.

Example 1: Using <> for Row-Level Classification via a New Column

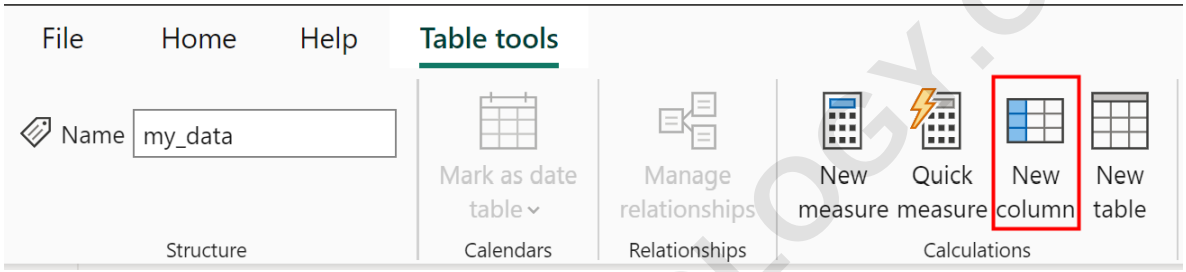
In many analytical scenarios, it is necessary to tag or categorize each row based on whether a certain condition is met or, in this case, not met. Our specific objective is to create a new column that explicitly labels players based on their team affiliation, flagging whether they are "Not on Team B" or "On Team B." This classification is achieved using the DAX `IF` function combined with the **<>** inequality check.

The conditional logic required here is straightforward: if the value in the **Team** column for the current row is not equal to "B", then return the string "Not on Team B." Otherwise, if the value is

equal to "B," return the string "On Team B." This approach uses the row context of a calculated column to perform a precise evaluation for every single player record in the table. The list below outlines the desired outcomes for this new classification column:

"Not on Team B" if the value in the **Team** column is not equal to "B"
"On Team B" if the value in the **Team** column is equal to "B"

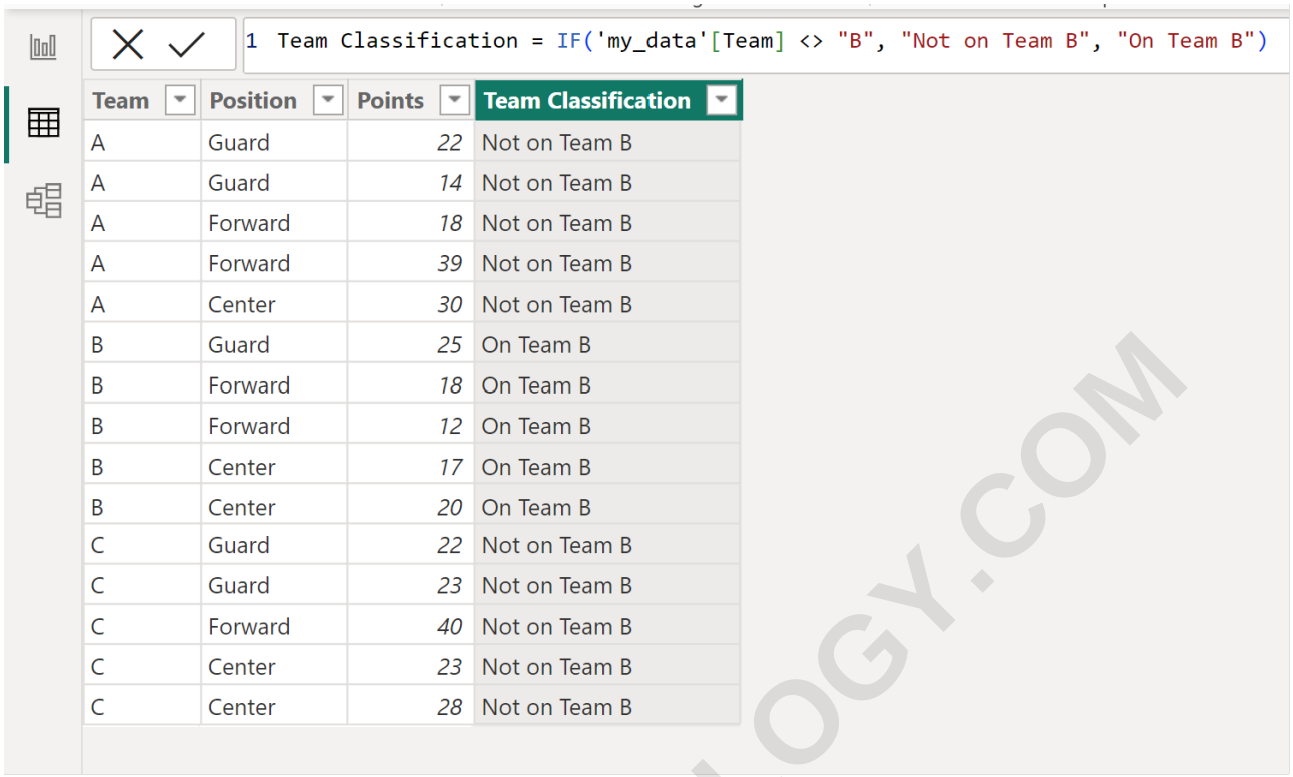
To implement this in Power BI Desktop, navigate to the **Table tools** tab in the ribbon interface and select the **New column** icon. This action prepares the environment to accept the DAX formula that defines the logic for the new calculated field, enabling the row-level evaluation required for the classification task.



The following formula must then be entered into the formula bar. Note how the **<>** operator dictates the primary conditional test, determining the flow of the `IF` statement:

Team Classification =
IF('my_data' <> "B", "Not on Team B", "On Team B")

Upon execution, Power BI generates a new column named **Team Classification**. This column accurately reflects the specified inequality logic, returning one of the two defined strings for every row, making it easy to filter, sort, and analyze data based on whether a player is excluded from Team B.



```
1 Team Classification = IF('my_data'[Team] <> "B", "Not on Team B", "On Team B")
```

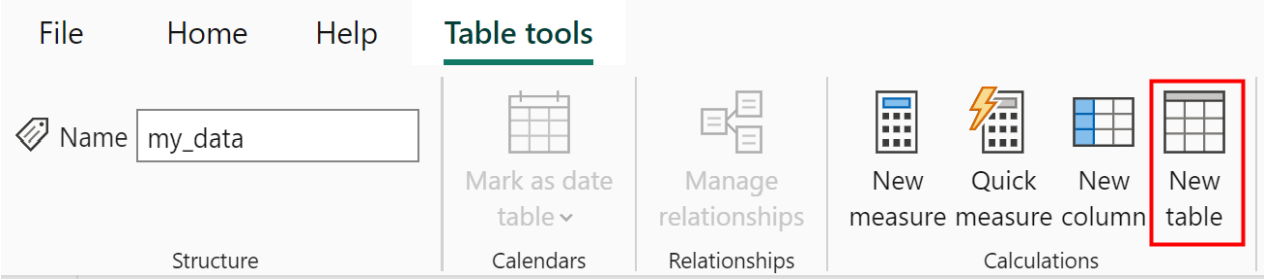
Team	Position	Points	Team Classification
A	Guard	22	Not on Team B
A	Guard	14	Not on Team B
A	Forward	18	Not on Team B
A	Forward	39	Not on Team B
A	Center	30	Not on Team B
B	Guard	25	On Team B
B	Forward	18	On Team B
B	Forward	12	On Team B
B	Center	17	On Team B
B	Center	20	On Team B
C	Guard	22	Not on Team B
C	Guard	23	Not on Team B
C	Forward	40	Not on Team B
C	Center	23	Not on Team B
C	Center	28	Not on Team B

Example 2: Using <> for Filtering Data via a New Table

The second major application of the Not Equal operator is creating a new, filtered table. Unlike a calculated column, which adds metadata to existing rows, a calculated table returns a completely new table object to the data model. This is particularly useful when you need to isolate a subset of data for dedicated reporting or when performing complex intermediate calculations that require a specific filter context.

Our objective in this example is to generate a new table that contains only those rows from the source **my_data** table where the value in the **Team** column is explicitly **not equal** to "B." This necessitates using a DAX function designed for table manipulation, such as `CALCULATETABLE`, which allows us to impose a strong filter condition across the entire dataset before the resulting table is constructed.

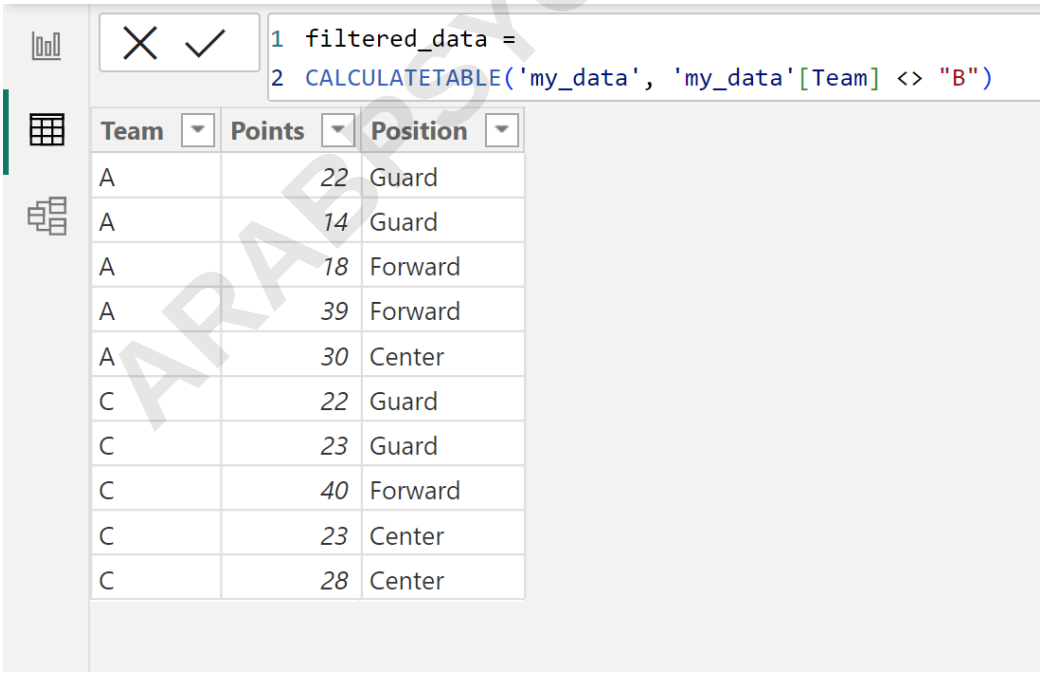
To begin this process in Power BI Desktop, navigate to the **Table tools** tab and click the **New table** icon. This prepares the environment for a DAX expression that will return a filtered set of rows, thereby constructing the new table based on our defined inequality constraint.



The following DAX formula leverages the `CALCULATETABLE` function. The inequality condition `'my_data' <> "B"` serves as the filter argument, ensuring that only rows satisfying the "not equal to B" criterion are passed through to the resulting table object:

filtered_data =
CALCULATETABLE('my_data', 'my_data' <> "B")

Executing this formula results in the creation of a new table named **filtered_data**. A quick inspection confirms that this new table contains only the rows from the original **my_data** table where the **Team** column value is not equal to B. This powerful technique effectively segments the data, offering a clean view of the records that fall outside the specified criterion, which is highly beneficial for targeted data analysis and visualization.



```
1 filtered_data =
2 CALCULATETABLE('my_data', 'my_data'[Team] <> "B")
```

Team	Points	Position
A	22	Guard
A	14	Guard
A	18	Forward
A	39	Forward
A	30	Center
C	22	Guard
C	23	Guard
C	40	Forward
C	23	Center
C	28	Center

Conclusion and Further Exploration of DAX Operators

The utilization of the Not Equal operator (**<>**) in DAX is fundamental for robust data modeling and analytical precision within Power BI. By enabling straightforward inequality checks, it allows analysts to move beyond simple inclusion criteria and focus on exclusions, outliers, or deviations, whether through row-level classification or table-level filtering.

Note: You can find a complete list of operators you can use in DAX by consulting the official Microsoft documentation.

Mastering this operator, alongside other comparison and logical operators available in DAX, unlocks the full potential of conditional logic, allowing for the creation of dynamic, responsive, and highly accurate reports and visualizations. Continuous learning regarding the full spectrum of DAX operators is highly recommended for optimizing data transformations.

The following tutorials explain how to perform other common tasks in Power BI: