# How to Use the BETWEEN Operator in SAS (With Examples)

Authored by

**stats writer**

November 19, 2025

The BETWEEN operator in SAS is a fundamental logical operator designed specifically for range comparison. Its core function is to efficiently check if a variable's value falls within a specified, inclusive range defined by two endpoints. This functionality is crucial for data manipulation and subsetting tasks, allowing analysts to quickly isolate records that meet precise numerical or chronological criteria.

Unlike complex nested comparisons using multiple AND conditions (e.g., `variable >= a AND variable <= b`), the **BETWEEN** operator simplifies the syntax significantly, enhancing both the readability and maintainability of SAS code. It is an indispensable tool when working with date ranges, numerical scores, financial thresholds, or any other data where boundaries are clearly defined. When employing this operator, it is essential to remember that the comparison is always inclusive, meaning the boundary values themselves are included in the resulting subset.

In the context of the SAS programming environment, the **BETWEEN** operator is typically utilized within PROC SQL statements or within the `WHERE` clauses of other procedures like `PROC PRINT` or `PROC REPORT`. Its availability across various modules ensures consistent and powerful data filtering capabilities across the entire data processing workflow. Mastering this operator is a key step towards efficient data management and analysis within SAS.

## Understanding the Syntax and Mechanics of BETWEEN

The syntax for the **BETWEEN** operator is straightforward yet powerful, following the standard form: `variable BETWEEN value1 AND value2`. The variable being tested can be numeric, character, or date/datetime, though it is most commonly applied to numerical and chronological data types. The operator checks two conditions simultaneously: first, that the variable is greater than or equal to `value1`, and second, that the variable is less than or equal to `value2`. Both `value1` and `value2` must be constants or expressions that evaluate to the same data type as the variable being tested.

A significant advantage of using **BETWEEN** is its intrinsic clarity. Instead of writing verbose code that explicitly states the equality and inequality conditions, this operator provides a semantic shortcut that clearly conveys the intention--selecting values within a defined range. For instance, to select records where a score is between 50 and 100, the statement `score BETWEEN 50 AND 100` is far more readable than the equivalent logical operator combination. This increased clarity is crucial for team environments and long-term project maintenance.

It is important to note the strict inclusivity of the operator. If you intend to exclude the boundary points, you must use traditional comparison operators (e.g., `>` and `<`). When applied to character variables, the comparison uses the collating sequence defined by the operating environment and checks if the character value falls alphabetically between the two specified strings, including those boundary strings themselves. However, its most robust and frequent use remains within numerical analysis.

## Applying BETWEEN in PROC SQL Queries

The **BETWEEN** operator finds its most natural home within the PROC SQL procedure, where it aligns perfectly with standard SQL syntax. Using PROC SQL allows users to perform declarative data queries, simplifying the process of filtering large datasets based on range criteria. The operator is placed directly within the `WHERE` clause, specifying the column and the range boundaries required for inclusion.

For instance, if we have a dataset named `my_data` containing athletic performance statistics, we might want to isolate only those athletes who scored within a specific point range. The following exemplary PROC SQL code demonstrates how to achieve this filtering efficiently:

```
proc sql;
select *
from my_data
where points between 15 and 35;
quit;
```

This particular query is designed to select all variables (indicated by `select *`) from the source dataset called `my_data`, retaining only the rows where the value in the `points` column satisfies the range condition. Specifically, it includes records where the points total is greater than or equal to **15** and less than or equal to **35**. This concise structure avoids the redundancy inherent in non-range comparisons and is a hallmark of efficient SQL programming.

## Setting Up the Demonstration Dataset

To fully illustrate the utility of the **BETWEEN** operator, we will use a sample dataset containing hypothetical statistics for several basketball teams. This data setup phase is crucial, as it provides the tangible context against which the filtering logic will be tested. The dataset, named `my_data`, includes two critical variables: `team` (a character variable indicating the team name) and `points` (a numeric variable representing a player's score).

We use the SAS `DATA` step and the `DATALINES` statement to create this foundational dataset. This method is often employed for quick examples and testing scenarios, allowing the analyst to define the data structure and populate it with sample observations directly within the program code. The structure ensures that the numeric variable `points` is readily available for our range filtering operation.

The code below executes the dataset creation and then immediately displays the complete data using `PROC PRINT`, allowing us to inspect the initial observations before applying any filtering:

```
/*create dataset: player scores across different teams*/
data my_data;
input team $ points;
datalines;
Cavs 12
Cavs 14
Warriors 15
Hawks 18
Mavs 31
Mavs 32
Mavs 35
Celtics 36
Celtics 40
;
run;

/*view the unfiltered dataset contents*/
proc print data=my_data;
run;
```

Upon execution of the `PROC PRINT` statement, the following output table is generated, which shows all nine observations in the original dataset. Note the range of scores, from the lowest (12 points) to the highest (40 points). This full set serves as our starting point for the subsequent filtering process, clearly demonstrating which records should be retained or excluded by the **BETWEEN** operator.

| Obs | team | points |
|---|---|---|
| 1 | Cavs | 12 |
| 2 | Cavs | 14 |
| 3 | Warriors | 15 |
| 4 | Hawks | 18 |
| 5 | Mavs | 31 |
| 6 | Mavs | 32 |
| 7 | Mavs | 35 |
| 8 | Celtics | 36 |
| 9 | Celtics | 40 |

## Detailed Example: Filtering Numeric Data with BETWEEN

Now that we have established our base data, we can implement the primary filtering objective: selecting only those players whose scores fall inclusively between 15 and 35 points. This is accomplished using a direct application of the **BETWEEN** operator within a PROC SQL block. The goal is to subset the data efficiently without needing to explicitly list all comparison operators.

We use the `WHERE` clause to define the condition, specifying `points BETWEEN 15 AND 35`. This statement instructs SAS to retain records where the `points` variable matches 15, 35, or any value in between. Records with scores of 12, 14, 36, and 40 should therefore be excluded from the resulting table, demonstrating the precise control offered by the range operator.

The resulting PROC SQL code structure is concise and focuses purely on the filtering mechanism:

```
/*select all rows where value in points column is between 15 and 35, inclusive*/
proc sql;
select *
from my_data
where points between 15 and 35;
quit;
```

Executing this query yields a filtered dataset, presented below. Observe carefully that the only observations returned are those where the score in the `points` column adheres strictly to the defined range . This confirms that values like 15 (Warriors), 31 (Mavs), 32 (Mavs), and 35 (Mavs) were included, while 12, 14, 36, and 40 were successfully excluded. This demonstration confirms the operator's effectiveness in range-based subsetting.

| team | points |
|---|---|
| Warriors | 15 |
| Hawks | 18 |
| Mavs | 31 |
| Mavs | 32 |
| Mavs | 35 |

## Combining BETWEEN with Other Logical Operators

While the **BETWEEN** operator is powerful for single-variable range checks, its true versatility shines when combined with other logical operators such as **AND** and **OR** within a complex `WHERE`

clause. This allows analysts to impose multiple, layered criteria on the data, leading to highly specific subsets. For example, we might need to filter the data not only by points but also by a specific categorical variable, like the team name.

To demonstrate this, suppose we want to refine our subset to include only those players who scored between 15 and 35 points **AND** who specifically play for the 'Mavs' team. This requires nesting the range condition and linking it to the categorical condition using the logical operator **AND**. Using parentheses around the **BETWEEN** clause is often a good practice to ensure clarity and correct order of operations, although in this simple case, it is not strictly required by SQL precedence rules.

The revised PROC SQL statement below illustrates this combined filtering strategy:

```
/*select rows where points is between 15 and 35 and team is Mavs*/
proc sql;
select *
from my_data
where (points between 15 and 35) and team='Mavs';
quit;
```

The output of this refined query is highly restricted, yielding only the records that satisfy both criteria simultaneously. In our basketball dataset, this results in the isolation of the three observations belonging to the 'Mavs' team that fall within the defined point range. Combining **BETWEEN** with other conditions is a standard procedure for advanced analytical filtering in SAS.

| team | points |
|---|---|
| Mavs | 31 |
| Mavs | 32 |
| Mavs | 35 |

## Using the NOT BETWEEN Operator

In scenarios where an analyst needs to exclude values that fall within a specific range, SAS provides the **NOT BETWEEN** operator. This operator serves as the logical inverse of **BETWEEN**. Instead of selecting values `>= A AND <= B`, **NOT BETWEEN** selects all values that are strictly less than A or strictly greater than B. This is particularly useful for identifying outliers, records that fall outside a desired operational threshold, or records that violate business rules.

The syntax remains similar, simply prefixing the operator with **NOT**: `variable NOT BETWEEN value1 AND value2`. For example, if we wanted to find all players who scored either very low (below 15) or very high (above 35), we would modify the query used in the previous examples:

```
/*select all rows where value in points column is NOT between 15 and 35*/
proc sql;
select *
from my_data
where points not between 15 and 35;
quit;
```

Executing this query on our sample data would return the observations for the Cavs (12 and 14 points) and the Celtics (36 and 40 points). The use of **NOT BETWEEN** provides a concise, alternative method to write complex exclusionary logic, maintaining the high standard of code readability that the **BETWEEN** operator encourages.

## Conclusion and Further Resources

The **BETWEEN** operator is a cornerstone of efficient data subsetting in SAS, particularly when working with numeric or date ranges. By providing a streamlined syntax for inclusive range checking, it significantly simplifies the development of `WHERE` and `IF` statements within both the Data step and PROC SQL. Its equivalence to combined greater-than-or-equal-to and less-than-or-equal-to conditions ensures that boundary points are always included in the result set.

Effective utilization of the **BETWEEN** operator, especially when combined with powerful conditional operators like **AND**, allows for the creation of sophisticated and highly targeted data filters necessary for complex reporting and analysis. Whether you are filtering based on salary, transaction dates, or, as demonstrated here, athletic scores, **BETWEEN** remains an essential tool in the SAS programmer's toolkit.

For detailed technical specifications, syntax variations, and performance considerations regarding the **BETWEEN** operator in SAS, analysts are strongly encouraged to consult the official documentation provided by SAS Institute. Understanding the nuances of this logical operator ensures optimal data handling.

**Note**: You can find the complete documentation for the **BETWEEN** operator in SAS.

## Related SAS Tutorials and Guides

To further enhance your data manipulation skills in SAS, explore these related tutorials that cover common data management and analytical tasks:

How to Use the **IN** Operator in SAS for Membership Checking.
Implementing Complex Conditional Logic with **IF-THEN/ELSE** Statements in the Data Step.
Detailed Guide to Filtering and Subsetting Data using **PROC SQL**.