# Learning Efficient Data Filtering: A Guide to the SAS BETWEEN Operator

Authored by

**Mohammed looti**

November 14, 2025

## Introduction to Efficient Range Filtering in SAS

The BETWEEN operator in SAS is recognized as an exceptionally powerful and intuitive feature specifically engineered to simplify the critical task of data retrieval based on defined ranges. This elegant utility allows data analysts and programmers to select records where a specific column's value falls inclusively within a predetermined lower and upper boundary. Utilizing this operator is paramount for enhancing the overall efficiency, readability, and long-term maintainability of complex data manipulation scripts. This efficiency gain is particularly noticeable when developers are handling extensive datasets or navigating filtering requirements involving continuous variables such as numerical scores, calculated metrics, temporal elements like dates, or even specific alphabetical sequences.

In the vast majority of analytical contexts within the SAS ecosystem, the BETWEEN operator is seamlessly deployed within a PROC SQL statement. PROC SQL serves as the primary and most robust interface in SAS for executing standard SQL commands, providing a standardized, high-performance method for querying, modifying, and managing tabular data structures. The BETWEEN operator fits naturally into this declarative framework, acting as a crucial component for defining precise, range-based constraints that are highly optimized for execution. Its adoption offers a significant advantage over manually constructing conditions using multiple comparison operators, inevitably leading to cleaner, more concise, and significantly less error-prone code.

The core value proposition inherent in using the BETWEEN keyword lies in its exceptional ability to condense verbose and repetitive conditional logic into a single, highly straightforward expression. Consider, for example, a fundamental data selection requirement: finding values equal to or greater than a minimum and equal to or less than a maximum. A condition specifying this boundary manually would require the explicit syntax: `data_column >= minimum_value AND data_column <= maximum_value`. Conversely, employing the SAS operator allows this logic to be written much more concisely and intuitively as: `data_column BETWEEN minimum_value AND maximum_value`. This simplification not only drastically reduces the volume of code necessary but, more importantly, substantially improves the overall clarity of the data filtering logic, ensuring that the precise intent of the query is instantly recognizable to any programmer or analyst reviewing the SAS program.

## Understanding the Core Syntax and Inclusive Nature

To effectively harness the capabilities of the BETWEEN operator, it is absolutely essential to grasp its fundamental structure and, perhaps more crucially, its inherent **inclusive** behavior within the SAS processing environment. The standard structure is straightforward: it involves listing the target column name, followed by the keyword **BETWEEN**, then specifying the lower boundary value, followed by the keyword **AND**, and finally, the upper boundary value. It is paramount to internalize that the **BETWEEN** operator in SAS, mirroring its function in standard SQL, is always inclusive.

This means that both the starting value (the lower boundary) and the ending value (the upper boundary) are considered valid matches, and any record containing these exact values will be successfully included in the resulting data subset.

This operator is almost universally incorporated into the WHERE clause of a PROC SQL step, which is the dedicated location for applying filtering conditions prior to data retrieval. The placement of the **WHERE** statement dictates precisely which records are selected from the source table before they are returned or manipulated further. Understanding this functional context is vital, as the WHERE clause is fundamentally the workhorse of conditional data selection in SAS programming. Furthermore, the boundaries specified for the range must be logically consistent with the underlying data types of the column being filtered; this entails defining numeric ranges for numeric columns, and character or date ranges for their respective types, often requiring special formatting for dates.

The general and precise syntax for utilizing **BETWEEN** for range specification is clearly illustrated in the following structured example. This query is designed to filter a large dataset based on a single numerical column named `points`, aiming to pull only records within a specific score bracket:

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

In executing this specific instruction, SAS is systematically directed to retrieve all columns (indicated by `select *`) from the designated source dataset, which is named **my_data**. The filtering logic, meticulously contained within the WHERE clause, strictly guarantees that only those rows where the value in the **points** column is 15, 35, or any numerical value falling precisely between these two bounds are selected for inclusion. This practical demonstration solidifies the concept of the operator's inclusive nature, treating both the minimum and maximum boundaries as valid data points for the final output.

## Practical Application: Filtering Numerical Data

To move beyond theoretical understanding and fully appreciate the immediate utility and efficiency of the **BETWEEN** operator, we will now examine a tangible, real-world scenario involving common data analysis tasks. For this demonstration, we will employ a sample basketball dataset within the SAS environment. This dataset tracks essential player performance metrics, specifically recording the team affiliation and the total points scored by each individual player. This provides a clear, numerical context ideal for applying and verifying the effectiveness of range filtering techniques.

Our initial step requires the establishment of this sample data structure. The following SAS Data Step code is utilized to create and populate the temporary dataset named **my_data**. This is immediately followed by a standard `PROC PRINT` step, which serves to display the initial, unfiltered contents of the newly created table, allowing us to visualize the complete data landscape before filtering:

```
/*create dataset*/
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 dataset*/
proc print data=my_data;
```

| 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 |

The core objective in this demonstration is to accurately isolate players whose scores fall within a predefined average performance tier, specifically those who scored a minimum of 15 points and a maximum of 35 points, ensuring that both boundaries are inclusively selected. Employing the **BETWEEN** operator within a PROC SQL step provides the single most efficient, concise, and highly readable solution for achieving this filtering task. The subsequent code snippet clearly demonstrates the exact implementation required to filter the data precisely according to these established numerical range criteria, showcasing the operator's simplicity and power:

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

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

Upon the successful execution of this PROC SQL statement, the resulting output table confirms the perfect application of the inclusive range filter. Only players whose score in the **points** column falls within the spectrum from 15 up to and including 35 are present in the final result set. This output effectively and precisely excludes the outliers: both the lower scores (12 and 14) and the higher scores (36 and 40) are filtered out. This straightforward, yet powerful, example fundamentally underscores how the **BETWEEN** operator streamlines the process of filtering numerical data, transforming what could be a complex, multi-comparison selection task into a manageable and highly intuitive procedure requiring minimal code.

## Constructing Complex Filters with Logical Operators

The inherent functionality of the **BETWEEN** operator can be significantly augmented and extended by strategically integrating it with other fundamental logical operators available in SQL, notably **AND**, **OR**, and the inverse operator, **NOT**. Combining these distinct elements grants developers the power to construct highly specific, multi-faceted filtering criteria directly within the WHERE

clause of their queries. A critical best practice when concatenating multiple complex conditions is the consistent utilization of parentheses. Parentheses explicitly define the intended order of evaluation, thereby eliminating potential ambiguity in execution and ensuring that the query processes the data exactly according to the logical hierarchy intended by the programmer.

A frequent and practical complex filtering requirement encountered in data analysis is the need to select records that satisfy a quantitative range condition **and** concurrently meet an additional categorical criterion. For instance, in our basketball example, we might refine our goal to retrieve only those players who scored between 15 and 35 points, but who exclusively belong to the 'Mavs' team affiliation. This necessary refinement demands the combination of the numerical range filter (defined by **BETWEEN**) with a separate equality filter (`team='Mavs'`). These two conditions are linked using the **AND** logical operator. This robust, two-step filtering process rapidly narrows down the search space, focusing the output on a highly relevant and specialized subset of the original data.

The following syntax provides a clear demonstration of how to implement this powerful combined conditional logic successfully within the SAS environment. This code filters the existing **my_data** dataset based simultaneously on the required points range and the categorical team name criterion:

```
/*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;
```

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

Execution of this sophisticated query yields a precise result set, containing only those records where the points scored are confirmed to be within the specified inclusive range (15 to 35) and the team name field is exactly 'Mavs'. Beyond the conjunction capabilities illustrated here, analysts should also be aware of the **NOT BETWEEN** operator. This inverse function provides an immediate selection capability for retrieving all rows that fall **outside** the defined range, making it

exceptionally useful for outlier analysis or exclusion filtering. This fluidity in combining **BETWEEN** with other logical operators is fundamental to achieving truly sophisticated and high-precision data retrieval using PROC SQL queries.

## Handling Diverse Data Types and Boundary Conditions

When implementing the **BETWEEN** operator within SAS, sophisticated analysts must remain acutely aware of how this function interacts specifically with different data types. While its application to standard numerical columns is inherently straightforward, **BETWEEN** is equally effective and necessary for filtering time-based date values and character strings. For filtering dates, it is critical to remember the internal representation used by SAS: dates are stored as numeric values, quantifying the number of days elapsed since the arbitrary zero point of January 1, 1960. Consequently, date boundaries within a **BETWEEN** clause must be correctly specified using standard SAS date literals, such as `'01JAN2023'd`. When the operator is applied to character values, the comparison relies entirely on the system's collating sequence (typically ASCII or EBCDIC), resulting in the selection of strings that fall alphabetically between the two specified character bounds.

A critical, recurring point of emphasis for all users of the BETWEEN keyword is its strict and unyielding **inclusive** definition. The selection criteria are hard-coded to always include the exact values defined by both the lower bound and the upper bound. For example, filtering a date column using `BETWEEN '01FEB2024'd AND '29FEB2024'd` will unequivocally include records dated precisely on February 1st and records dated exactly on February 29th. If the specific analytical requirement strictly calls for an **exclusive** range (i.e., values must be strictly greater than the lower bound and strictly less than the upper bound), the **BETWEEN** operator must be consciously avoided. In such cases, developers should revert to explicit comparison operators like `>` and `<`, ensuring absolute precision in range definition and effectively preventing the inadvertent inclusion of boundary data points.

Furthermore, it is highly beneficial for advanced troubleshooting and optimization to recognize the functional logical equivalence between the concise **BETWEEN** operator and the combined use of standard comparison operators. The expression `column BETWEEN A AND B` is mathematically and functionally identical to the lengthier expression `column >= A AND column <= B`. Although both constructs yield the exact same result set when applied correctly, **BETWEEN** is overwhelmingly preferred in the majority of SAS programming scenarios due to its superior readability, inherent conciseness, and reduced complexity. This simplification drastically lowers the cognitive load required to instantly understand the filtering intent of the query. Nevertheless, understanding this underlying equivalence remains crucial for developers who need to implement nuanced, non-inclusive ranges or who are tasked with troubleshooting complex WHERE clause logic, allowing for maximum control over all boundary conditions.

## Summary of Best Practices and Conclusion

The **BETWEEN** operator stands as a cornerstone utility within the SAS programming toolkit, facilitating efficient, reliable, and highly readable range-based data filtering operations. When deployed within PROC SQL, its elegantly straightforward syntax allows analysts to define inclusive ranges for numerical scores, chronological dates, or alphabetical character fields, thereby dramatically reducing the complexity often associated with constructing verbose, multi-part conditional statements within the WHERE clause. Adhering to best practices, such as verifying data consistency and using parentheses in complex logical constructions, ensures the stability of the analytical process.

Throughout this comprehensive guide, we have systematically established the critical importance of understanding the operator's inclusive nature, demonstrated its fundamental application for isolating numerical data subsets, and explored the advanced techniques for combining it with powerful logical operators (such as **AND** and **NOT**) to construct highly refined, multi-criteria queries. Mastery of **BETWEEN** across the spectrum of data types--especially dates--is essential for achieving accurate and robust data processing, while simultaneously minimizing the persistent risk of boundary inclusion or exclusion errors that plague manual comparison logic.

In conclusion, the strategic use of **BETWEEN** not only optimizes code structure but also enhances the collaborative nature of data projects by making selection criteria immediately self-evident. For those seeking to delve further into its specialized applications, particularly regarding nuanced date formats, time series analysis, or system-specific character collating sequences, we strongly advise consulting the official SAS documentation for the **BETWEEN** operator. This authoritative resource provides the most comprehensive and up-to-date technical details necessary to ensure correct, compliant, and optimal implementation in all your future data management and programming tasks.

## Additional Resources

To further enhance your SAS programming skills and explore other common data manipulation techniques essential for robust analysis, consider exploring the following related tutorials and documentation:

Understanding the WHERE clause in PROC SQL.
Filtering datasets using conditional logic in Data Step.
Working with SAS data types: Numeric, Character, and Date.
Introduction to Logical Operators (AND, OR, NOT) in SAS.