How to Easily Filter Rows Containing Specific Strings in SAS

Authored by

stats writer

December 1, 2025

In the context of data analysis using SAS, the ability to efficiently filter datasets based on textual content is fundamental for generating meaningful reports and focused analysis. You can filter rows that contain a specific string or pattern by leveraging powerful conditional logic within your processing steps. While this is commonly accomplished using the WHERE statement, it is critical to select the correct operator--such as **CONTAINS** or **IN**--depending on whether you are looking for a partial match or an exact match against a list of discrete values.

The initial approach often involves utilizing the PROC SQL query environment, which natively supports the standard SQL syntax, including the **LIKE** operator for pattern matching. However, within the native DATA step environment, SAS provides dedicated operators that are often more straightforward for simple substring identification. The WHERE statement defines a crucial condition: only rows satisfying this condition, such as those where a specific column contains the desired text sequence, will be included in the resulting output dataset or procedure execution.

Mastering these filtering techniques ensures that your data preparation is precise and streamlined. By properly using operators like **CONTAINS**, you can easily identify rows that include the specific string you are searching for, regardless of its position within the column's value. This is particularly useful in cleaning messy text data or segmenting records based on embedded descriptive information. The two most effective methods for achieving this filtering goal in a DATA step are demonstrated below, offering both flexibility for partial matches and efficiency for filtering against multiple predetermined values.

Mastering String Filtering in SAS: An Overview

Data filtering is arguably the most essential task in any analytical workflow. In SAS, filtering operations are typically executed using the WHERE statement, a highly efficient component that processes conditions before the data is fully loaded, thereby improving performance, especially with very large datasets. When dealing with character variables, the core challenge is often matching a string that is only partially present within the data field, such as finding "management" within "Project Management Department".

The standard SAS approach provides two distinct, yet highly effective, methods for this purpose. The first method focuses on identifying a specific substring within a variable using the **CONTAINS** operator. This operator is highly intuitive and directly addresses the need for partial matching. The second method, utilizing the **IN** operator, is optimized for scenarios where you need to select rows whose variable value exactly matches one of several specified strings. Understanding the subtle differences between these methods is key to writing concise and performant SAS code.

Although the examples provided here utilize the SAS DATA step, it is important to note that similar functionality exists when using PROC SQL. In SQL syntax, the LIKE operator, often combined with percent sign (%) wildcards, performs the same function as the **CONTAINS** operator in the DATA

step. However, for most fundamental filtering tasks within the traditional SAS processing environment, the methods outlined below are the most common and robust choices.

Method 1: Filtering Rows that Contain a Specific String

The simplest and most direct way to filter a SAS dataset for rows containing a specific substring is by using the **CONTAINS** operator (or the equivalent mnemonic `?`) within the WHERE statement. This method is highly effective because it searches the entire character variable for the presence of the specified text fragment, regardless of where that fragment occurs within the variable's value. This capability is invaluable when dealing with titles, descriptions, or names where you only know a part of the overall string.

When using **CONTAINS**, the syntax is straightforward: specify the variable name, followed by the operator `CONTAINS`, and then the target string enclosed in quotation marks (single or double). It is crucial to remember that, by default, SAS string comparisons are **case-sensitive**. If your data contains variations in capitalization (e.g., "String1" vs. "string1"), you must ensure you either match the case exactly or apply a function like `UPCASE` to the variable before comparison to achieve case-insensitive matching.

The following example illustrates how to define a new dataset, `specific_data`, by selecting only those rows from `original_data` where the variable `var1` includes the literal substring "string1". This is an extremely common pattern in data extraction and preparation phases.

```
/*filter rows where var1 contains "string1"*/
data specific_data;
set original_data;
where var1 contains 'string1';
run;
```

Method 2: Filtering Rows that Match One of Several Specific Strings using IN

While the **CONTAINS** operator is perfect for partial matches, sometimes the requirement is to filter a dataset based on whether a variable's value exactly matches one of a predetermined list of potential values. For this scenario, the **IN** operator offers superior clarity and efficiency compared to chaining multiple `OR` conditions. The **IN** operator allows you to specify a concise list of acceptable character values for a given variable.

Using the **IN** operator significantly cleans up the code, especially when the list of target strings grows large. Instead of writing `WHERE var1 = 'string1' OR var1 = 'string2' OR var1 = 'string3'`, you can consolidate this logic into a single, highly readable statement. This method

explicitly requires that the entire value of the variable matches one of the values provided in the parenthetical list.

It is important to emphasize the distinction: **IN** performs an exact match against the entire variable value, whereas **CONTAINS** performs a partial match (substring search). When using the **IN** operator with strings, each target value must be enclosed in quotes and separated by commas, as demonstrated in the following code block. This method is ideal for selecting specific categories, known identifiers, or complete names from a list of possibilities.

```
/*filter rows where var1 contains "string1", "string2", or "string3"*/
data specific_data;
set original_data;
where var1 in ('string1', 'string2', 'string3');
run;
```

Practical Setup: Creating the Example SAS Dataset

To illustrate these powerful filtering techniques, we will utilize a sample dataset containing information about NBA teams and their scores. This dataset, named `nba_data`, provides a clean, small sample suitable for demonstrating how the WHERE statement selectively processes records based on string content in the `team` variable.

The following SAS DATA step code uses the `DATALINES` statement to input the team names (character variable, denoted by `$`) and their corresponding points (numeric variable). Creating this temporary dataset ensures that our filtering examples are reproducible and easy to follow.

After creating the dataset, a simple `PROC PRINT` is used to display the contents, allowing us to visualize the data before the filtering is applied. This initial step is critical for verifying the input data structure and ensuring the subsequent filtering logic yields the expected results.

```
/*create dataset*/
data nba_data;
input team $ points;
datalines;
Mavs 95
Spurs 99
Warriors 104
Rockets 98
Heat 95
Nets 90
```

```
Magic 99
Cavs 106
;
run;

/*view dataset*/
proc print data=nba_data;
```

The dataset `nba_data` contains eight rows, each representing a unique team and its score. This data serves as the foundation for the demonstrations that follow.

Obs	team	points
1	Mavs	95
2	Spurs	99
3	Warriors	104
4	Rockets	98
5	Heat	95
6	Nets	90
7	Magic	99
8	Cavs	106

Implementation Demonstration: Filtering for a Partial String Match

Our first practical demonstration uses **Method 1 (CONTAINS)** to extract rows where the `team` name includes the specific substring "avs". Notice that "avs" is not a complete team name, but a fragment present in two different team entries: "Mavs" and "Cavs". This partial matching capability showcases the power and utility of the **CONTAINS** operator when exact naming conventions are unknown or variable.

In the code below, we create a new dataset called `specific_data`. Within the DATA step, the WHERE statement specifies the condition `team CONTAINS 'avs'`. SAS iterates through the `nba_data` rows, checking if the text 'avs' appears anywhere within the `team` variable for that record. Only the records that satisfy this condition are passed to the output dataset.

Upon executing the DATA step and running the subsequent `PROC PRINT`, we confirm that only "Mavs" and "Cavs" remain in the filtered dataset. This perfectly demonstrates how **CONTAINS** allows for flexible text searches crucial for data mining and categorization.

```
/*filter rows where team contains the string 'avs'*/
data specific_data;
set nba_data;
where team contains 'avs';
run;

/*view resulting rows*/
proc print data=specific_data;
```

The resulting dataset, `specific_data`, successfully filters the original eight rows down to two:

Obs	team	points
1	Mavs	95
2	Cavs	106

As expected, the only two rows shown are the ones where the **team** column contains 'avs' in the name, namely "Mavs" and "Cavs". This confirms the effective use of the **CONTAINS** operator for partial string matching.

Implementation Demonstration: Filtering for Multiple Discrete Strings

Next, we apply **Method 2 (IN)** to filter the dataset based on an exact match against a predefined list of team names: "Mavs", "Nets", and "Rockets". This approach is structurally cleaner than using multiple `OR` conditions and is preferred when selecting specific, whole values.

The code specifies the condition `team IN ('Mavs', 'Nets', 'Rockets')` in the WHERE statement. SAS compares the value of the `team` variable in each row against every string provided within the parentheses. If an exact match is found for any of the listed strings, that row is included in the new `specific_data` dataset.

Unlike the **CONTAINS** example, which matched fragments like 'avs', the **IN** operator requires the column value to match the listed string completely. For instance, if the list had contained 'Net', the row for 'Nets' would not have been included because 'Net' is not an exact match for 'Nets'. This precision makes the **IN** operator ideal for working with known categorical data or identifiers.

```
/*filter rows where team contains the string 'Mavs', 'Nets', or 'Rockets'*/
data specific_data;
set nba_data;
where team in ('Mavs', 'Nets', 'Rockets');
```

```
run;

/*view resulting rows*/
proc print data=specific_data;
```

The resulting output confirms that only the three specified teams are retained:

Obs	team	points
1	Mavs	95
2	Rockets	98
3	Nets	90

Advanced Considerations: Case Sensitivity and Alternatives

A critical factor when filtering strings in SAS is case sensitivity. By default, both the **CONTAINS** and **IN** operators perform case-sensitive comparisons. If the data is inconsistent (e.g., 'heat' vs. 'Heat'), a search for 'Heat' will miss 'heat'. To overcome this limitation and ensure comprehensive results, it is best practice to standardize the case of both the variable being filtered and the filter value.

The standard method for achieving case-insensitive matching is to use the `UPCASE` or `LOWCASE` function within the WHERE statement. For example, to search for 'nets' regardless of case, you would write: `WHERE UPCASE(team) CONTAINS 'NETS'`. This converts the variable content to uppercase on the fly for comparison purposes, ensuring that "Nets", "nets", and "NETS" are all successfully identified.

Another powerful alternative for substring searching, especially if you need to determine the exact position of the string, is the `INDEX` function. The `INDEX` function returns the starting position of the first occurrence of a substring within a string; if the substring is not found, it returns 0. Therefore, a condition like `WHERE INDEX(team, 'avs') > 0` is logically equivalent to `WHERE team CONTAINS 'avs'`, offering analysts slightly more flexibility in complex conditional logic.

Finally, for users preferring SQL syntax, the PROC SQL environment provides the LIKE operator. This operator uses the percent sign (%) as a wildcard for any sequence of zero or more characters. For instance, to find teams containing 'avs', the SQL syntax would be: `WHERE team LIKE '%avs%'`. While this provides identical functionality to the **CONTAINS** operator, choosing between the DATA step and PROC SQL often comes down to personal preference or the broader context of the SAS program being written.

Summary of String Filtering Techniques

Effective filtering is the cornerstone of robust data analysis in SAS. The techniques explored here provide powerful and efficient mechanisms for subsetting data based on character variables. Whether you need the flexibility of partial matching using **CONTAINS** or the precision of discrete value selection using **IN**, the SAS WHERE statement provides the necessary tools.

The primary distinction to remember is that **CONTAINS** looks for a substring, making it ideal for exploratory filtering or finding embedded codes, while **IN** looks for an exact match against a predefined list of values, which is best suited for filtering categories or specific identifiers. Always be mindful of case sensitivity and employ functions like `UPCASE` when necessary to ensure complete and accurate data capture.

For further exploration of SAS capabilities, particularly in the realm of data manipulation and procedure usage, consider reviewing additional tutorials that explain how to perform other common tasks in SAS:

Detailed methods for handling missing data in large datasets.

Techniques for merging and appending SAS datasets efficiently.

Advanced usage of `PROC REPORT` for customized output generation.