SAS: Use CONTAINS in PROC SQL

Authored by

Mohammed looti

March 29, 2026

Filtering data based on partial string matches is an absolutely fundamental skill in data manipulation and preparation. Within the SAS environment, the PROC SQL procedure provides a highly efficient and intuitive method for executing this task through the use of the **`CONTAINS`** operator. This feature is indispensable when analysts need to retrieve records where a specific column value includes a certain string pattern, rather than relying solely on exact, monolithic matches. The **`CONTAINS`** operator significantly streamlines the process of pattern identification, allowing for flexible searching across character variables.

This comprehensive guide is designed to thoroughly explore the practical applications and nuances of the **`CONTAINS`** operator specifically within PROC SQL. We will systematically cover a range of scenarios, starting with simple filtering for a single pattern, advancing to complex selections involving multiple criteria, and concluding with techniques for explicitly excluding records that contain unwanted strings using **`NOT CONTAINS`**. By mastering these techniques, users will significantly enhance their ability to write precise and powerful SQL queries in SAS, leading to more accurate and effective data selection processes essential for high-quality data analysis.

Preparing the Data: Establishing Our Example Dataset

To effectively illustrate the practical functionality of the **`CONTAINS`** operator, we must first establish a reliable sample dataset. This example dataset, named `my_data`, simulates real-world data by containing information about various basketball players. Crucially, the dataset includes two primary variables that we will manipulate throughout the examples: `team` (a character variable essential for string pattern matching) and `points` (a numeric variable representing player scores). Having a clear, consistent data structure ensures that the results of our **`CONTAINS`** demonstrations are easily verifiable and understandable.

The following SAS code snippet details the creation of this sample data structure. We employ a foundational DATA step to input the team names and corresponding points, followed immediately by PROC PRINT. This standard procedure ensures we can visualize the raw data, confirming its structure and contents before we begin applying complex PROC SQL filtering logic using the partial match operators.

```
/*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

Example 1: Selecting Rows Based on a Single String Pattern

Our initial and most essential example focuses on the core use case of the **`CONTAINS`** operator: retrieving only those rows where a designated character variable includes a specific string pattern. This is the simplest form of partial matching. For this demonstration, we aim to isolate all records within the `my_data` dataset where the `team` variable contains the substring 'avs' at any point within the full string. The power of **`CONTAINS`** lies in its ability to find this pattern regardless of where it appears (beginning, middle, or end) of the value.

The structure of the PROC SQL statement below is straightforward yet powerful. It applies the **`CONTAINS`** operator directly within the WHERE clause to execute the desired filtering. It is important to remember a key behavior of character comparisons in SAS: this operation is **case-sensitive** by default, meaning the literal string 'avs' must match exactly in terms of capitalization for the record to be selected.

```
/*select all rows where team contains 'avs'*/
proc sql;
select *
from my_data
where team contains 'avs';
quit;
```

Upon execution, the resulting output clearly demonstrates the efficacy of the operator. As shown below, only the rows corresponding to 'Cavs' and 'Mavs' are selected, precisely because they are the only team names that satisfy the partial match criteria ('avs'). This confirms that the **`CONTAINS`** operator is highly effective for rapidly isolating relevant data based on embedded substrings, serving as a vital function when performing initial data exploration or cleaning tasks.

team	points
Cavs	12
Cavs	14
Mavs	31
Mavs	32
Mavs	35

Example 2: Filtering Rows by Multiple String Patterns

Data analysis frequently requires filtering based on the presence of one or more possible characteristics simultaneously. Fortunately, the PROC SQL **`CONTAINS`** operator is fully compatible with standard logical connectors. By utilizing the **`OR`** operator, we can significantly expand our search criteria to match multiple patterns within the same query. Our objective here is to retrieve all records where the `team` variable contains either the pattern 'avs' or the pattern 'ics', targeting two distinct subsets of the data simultaneously.

The following SQL query illustrates how to construct this compound condition. We link two separate **`CONTAINS`** clauses using the **`OR`** operator, instructing the procedure to return a row if it satisfies *at least one* of the specified criteria. This logical structure is crucial for combining results from different partial matches, allowing us to group related but non-identical records in a single output set.

```
/*select all rows where team contains 'avs' or 'ics'*/
proc sql;
```

```
select *
from my_data
where team contains 'avs' or team contains 'ics';
quit;
```

The output confirms the success of the combined logic: it includes rows for 'Cavs' and 'Mavs' (matching 'avs') alongside 'Celtics' (matching 'ics'). This strategic combination of the **`CONTAINS`** operator and logical operators is a cornerstone of advanced filtering techniques, demonstrating how complex selection requirements can be efficiently managed within the WHERE clause. This flexibility is what makes PROC SQL a powerful tool for data conditioning.

team	points
Cavs	12
Cavs	14
Mavs	31
Mavs	32
Mavs	35
Celtics	36
Celtics	40

Example 3: Excluding Rows That Contain a Specific Pattern

In addition to selecting records that match a specific string pattern, data preparation often requires the inverse action: explicitly excluding records that contain undesirable or irrelevant strings. This technique is known as inverse pattern matching. In PROC SQL, this powerful negation is achieved using the **`NOT CONTAINS`** operator, which functions as the exact inverse complement of the standard **`CONTAINS`** operator.

The subsequent code demonstrates how to implement **`NOT CONTAINS`** within the WHERE clause. Our goal here is to retrieve all rows from the `my_data` dataset where the `team` variable definitively does not include the string 'avs'. This exclusion technique is invaluable for data cleansing operations, quality control, or when focusing exclusively on subsets of data that lack particular characteristics.

```
/*select all rows where team does not contain 'avs'*/
proc sql;
select *
```

```
from my_data
where team not contains 'avs';
quit;
```

As expected, observing the output confirms that only the teams 'Warriors', 'Hawks', and 'Celtics' remain, as 'Cavs' and 'Mavs' have been successfully filtered out due to the presence of 'avs'. The **`NOT CONTAINS`** operator is therefore an indispensable tool in the analyst's arsenal for performing precise inverse pattern matching and ensuring targeted data isolation within the SQL environment. Mastering both the inclusionary and exclusionary forms of the **`CONTAINS`** operator allows for complete command over string-based data retrieval.

team	points
Warriors	15
Hawks	18
Celtics	36
Celtics	40

Further Learning and Resources

The **`CONTAINS`** and **`NOT CONTAINS`** operators within PROC SQL are fundamental components for robust data management, enabling fine-grained control over filtering based on partial string matches. Mastery of these operators ensures more flexible and accurate data retrieval, which is critical for effective reporting and data analysis. Whether the requirement is to search for specific substrings, combine searches for multiple patterns, or exclude unwanted entries, these tools provide reliable, SQL-based solutions in the SAS environment. They are essential techniques for anyone working extensively with character data.

To continue developing your proficiency in SQL programming and data manipulation within SAS, we recommend exploring the following related tutorials that delve deeper into advanced dataset operations and query construction:

How to Use PROC SQL to Count Unique Values in SAS
A Guide to Joining Datasets in PROC SQL
Understanding the SELECT statement and FROM clause in PROC SQL