SAS: Use a “NOT IN” Operator

Authored by

Mohammed looti

April 10, 2026

Introduction: Understanding the `NOT IN` Operator in SAS

In the realm of SAS programming, efficiently manipulating and filtering data is paramount for any analytical task. One of the most fundamental operations involves selecting data based on specific criteria, and often, this means excluding records that match a certain set of values. The **`NOT IN` operator** in SAS provides a powerful and intuitive way to achieve this. It allows users to return only those rows where a specified variable's value does not exist within a given list of values.

This operator is particularly useful when you need to exclude multiple categories or specific identifiers from your analysis without writing complex chains of `AND` and `OR` conditions. Instead of explicitly stating `WHERE variable NE 'value1' AND variable NE 'value2' AND variable NE 'value3'`, the **`NOT IN` operator** offers a concise and readable alternative: `WHERE variable NOT IN ('value1', 'value2', 'value3')`. This simplification not only improves code clarity but also enhances maintainability.

Throughout this guide, we will delve into the practical applications of the **`NOT IN` operator**. We will explore its use within different SAS contexts, specifically focusing on its integration with PROC SQL for direct data querying and with the `SET` statement within a Data Step for creating new, filtered datasets. By the end, you will have a comprehensive understanding of how to effectively leverage this operator for precise data exclusion.

Core Functionality: Filtering Data with `NOT IN`

The primary purpose of the **`NOT IN` operator** is to facilitate robust data filtering by excluding observations where a specified variable's value is present in a list. This list can consist of literal values, or it can be generated from a subquery, offering significant flexibility. When SAS processes a `WHERE` clause containing **`NOT IN`**, it evaluates each row to determine if the variable's value matches any element within the provided list. If no match is found, the row is included in the result set; otherwise, it is excluded.

Understanding the distinction between **`NOT IN`** and other exclusion operators like `NE` (Not Equal) or `NOT EQ` is crucial. While `NE` is used to check for inequality against a single value (e.g., `WHERE team NE 'Cavs'`), **`NOT IN`** is designed for comparison against a collection of values. This makes it inherently more efficient and readable for scenarios requiring exclusion based on multiple criteria. For instance, to exclude both 'Cavs' and 'Celtics', `WHERE team NOT IN ('Cavs', 'Celtics')` is superior to `WHERE team NE 'Cavs' AND team NE 'Celtics'`, particularly as the list of excluded values grows.

The versatility of the **`NOT IN` operator** extends beyond simple lists. It can also be combined with subqueries, enabling dynamic filtering based on values retrieved from another table or a derived result set. This advanced application allows for more complex data exclusion logic, where the list of

values to exclude is not static but rather determined by the current state of other data. Such capabilities are invaluable for intricate data preparation tasks and relational database operations within SAS.

Practical Application: Using `NOT IN` in PROC SQL

One of the most common and powerful contexts for using the **`NOT IN` operator** in SAS is within PROC SQL. This procedure provides a powerful, industry-standard SQL interface for managing and querying SAS datasets, making it familiar to anyone with a background in relational databases. Let's consider a practical example involving a dataset of basketball players, where we want to select players from specific teams while excluding others.

Suppose we have the following dataset in SAS, containing information about various basketball players, including their team and points scored:

```
/*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;
```

This code snippet first creates a SAS dataset named `my_data` with two variables: `team` (character) and `points` (numeric). The `datalines` statement then populates this dataset with sample records. Finally, `PROC PRINT` is used to display the contents of the newly created dataset, allowing us to verify its structure and 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

Now, to demonstrate the effectiveness of the **`NOT IN` operator**, let's use PROC SQL to select only the rows where the team is not 'Cavs' or 'Celtics'. This operation will effectively filter out all players belonging to these two specific teams, leaving us with data from the remaining teams.

```
/*select all rows where team is not 'Cavs' or 'Celtics'*/
proc sql;
select *
from my_data
where team not in ('Cavs', 'Celtics');
quit;
```

In this PROC SQL block, the `SELECT * FROM my_data` statement initiates a query on our `my_data` dataset, aiming to retrieve all columns. The crucial part is the `WHERE team NOT IN ('Cavs', 'Celtics')` clause. Here, the **`NOT IN` operator** checks the `team` variable for each record. If the team name is either 'Cavs' or 'Celtics', that row is excluded from the result set. All other rows, where the team is not one of the specified values, are included.

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

As you can observe from the output, the query successfully returned only the rows where the team variable did not match 'Cavs' or 'Celtics'. This demonstrates the clean and efficient way the **`NOT IN` operator** handles multiple exclusion criteria within a single, readable conditional statement, providing a clear subset of your original dataset.

Advanced Usage: Integrating `NOT IN` with the `SET` Statement

Beyond direct querying with PROC SQL, the **`NOT IN` operator** is also incredibly valuable within a SAS Data Step, particularly when combined with the `SET` statement. This approach is ideal when the goal is not just to view filtered data, but to create a new, permanent dataset that contains only the records satisfying your exclusion criteria. The `SET` statement reads observations from an existing SAS dataset, and when paired with a `WHERE` clause, it allows for selective inclusion of these observations into a new dataset.

Using the same `my_data` dataset from our previous example, let's illustrate how to create a new dataset, named `new_data`, that includes all players except those from the 'Cavs' or 'Celtics'. This process involves a standard Data Step, where the `WHERE` statement--containing our **`NOT IN` operator**--acts as the gatekeeper for which records are written to the output dataset.

```
/*create new dataset that only contains rows where team is not Cavs or Celtics*/
data new_data;
set my_data;
where team not in ('Cavs', 'Celtics');
run;

/*view new dataset*/
proc print data=new_data;
```

In this Data Step, `data new_data;` declares the creation of a new dataset. The `set my_data;` statement reads each observation from the original `my_data`. The subsequent `where team NOT`

`IN ('Cavs', 'Celtics');` clause then filters these observations. Only those observations where the `team` variable is not 'Cavs' and not 'Celtics' are passed through to be written to the `new_data` dataset. This effectively creates a derivative dataset based on your exclusion criteria.

Obs	team	points
1	Warriors	15
2	Hawks	18
3	Mavs	31
4	Mavs	32
5	Mavs	35

Upon viewing the `new_data` dataset using `PROC PRINT`, you will observe that it exclusively contains the rows from the original `my_data` where the team is neither 'Cavs' nor 'Celtics'. This method is particularly useful for creating analytical subsets, preparing data for specific reports, or refining datasets before further complex transformations, ensuring that your subsequent analyses are based on precisely the data you intend to include.

Considerations and Best Practices

While the **`NOT IN` operator** is a powerful tool for data exclusion in SAS, it's essential to be aware of certain considerations to ensure robust and efficient code. One critical aspect is the handling of missing values. If the variable being evaluated by **`NOT IN`** contains a missing value, SAS treats the comparison as unknown. For instance, if a team value is missing, `team NOT IN ('Cavs', 'Celtics')` will not evaluate to true, and the row will be excluded from the result set, which might not always be the desired behavior. To explicitly include missing values, you would need to add an additional condition, such as `WHERE (team NOT IN ('Cavs', 'Celtics') OR team IS NULL)`.

Another important consideration is performance, especially when dealing with very large datasets or extensive lists within the **`NOT IN` operator**. For extremely long lists of values, or when the list is derived from a complex subquery on a massive table, the processing time can increase. In such scenarios, alternative approaches might offer better performance. For example, creating a separate lookup dataset containing the values to be excluded and then performing a merge or join operation (e.g., using `PROC SQL` with a `LEFT JOIN` and checking for missing join keys) could be more efficient.

When working with the **`NOT IN` operator**, ensure consistency in data types. If the variable is numeric, the list of values should also be numeric. If the variable is character, the values in the list

should be enclosed in single quotes. Mismatched data types can lead to errors or unexpected results. Furthermore, for optimal readability and maintenance, especially when your exclusion list is extensive, consider storing the list of values in a macro variable or a separate dataset that can be dynamically accessed. This practice promotes modularity and makes your code easier to update.

Conclusion: Mastering Data Exclusion in SAS

The **`NOT IN` operator** is an indispensable component of the SAS programmer's toolkit, offering a streamlined and effective method for excluding specific values from your data analysis. Its ability to simplify complex exclusion criteria into a single, elegant statement significantly enhances code clarity and reduces the potential for errors that might arise from extensive `AND` and `OR` conditions. Whether you are performing ad-hoc queries with PROC SQL or constructing new, refined datasets within a Data Step, the **`NOT IN` operator** proves invaluable for precise data filtering.

By understanding its core functionality, leveraging it in various SAS procedures, and applying best practices regarding missing values and performance, you can ensure your data manipulation tasks are both accurate and efficient. The examples provided have demonstrated its straightforward application in filtering observations based on a list of excluded team names, showcasing its immediate utility in real-world scenarios.

Ultimately, mastering the **`NOT IN` operator** empowers you to take greater control over your data, allowing you to focus your analytical efforts on the most relevant subsets of information. This precision is critical for generating accurate insights and making informed decisions based on clean, carefully curated data. Integrate this powerful operator into your SAS programming workflow to streamline your data preparation and analysis processes.

Additional Resources

To further enhance your SAS programming skills, explore these tutorials that explain how to perform other common data manipulation tasks:

SAS Official Documentation on the `NOT IN` Operator: SAS Documentation

Understanding SAS Data Steps: SAS Data Step Basics

An Introduction to PROC SQL in SAS: PROC SQL Overview