How to Create Frequency Tables with WHERE Statements in SAS PROC FREQ

Authored by

stats writer

November 22, 2025

Introduction to PROC FREQ and Data Subset Filtering

The SAS programming language provides powerful tools for statistical analysis, chief among them the procedures (PROCs). The **PROC FREQ** procedure is foundational, utilized extensively by analysts to generate frequency tables and contingency tables, providing essential descriptive summaries of categorical data within a dataset. Understanding the distribution of variables is often the crucial first step in any analytical project, confirming data quality and informing subsequent modeling decisions.

While PROC FREQ is highly effective at summarizing entire datasets, real-world analysis frequently demands granularity. Analysts often need to examine frequencies for specific subsets of the data--for instance, checking the distribution of product categories only for customers in a certain region, or analyzing job roles exclusively within one department. This necessity for targeted analysis requires an efficient mechanism for row-level filtering.

In SAS, the most efficient and recommended method for performing this conditional filtering directly within a procedure is through the use of the **WHERE** statement. Integrating the WHERE statement ensures that only the relevant observations are processed by the procedure, leading to optimized performance, reduced processing time, and cleaner analytical results, especially when dealing with very large datasets.

Understanding the Basic Syntax of WHERE in PROC FREQ

The WHERE statement is a global SAS statement, meaning it can be used across various procedures, including PROC FREQ, to select observations that meet a specified condition. When placed immediately after the procedure call and before the required procedure-specific statements (like `TABLES` in PROC FREQ), it acts as a filter on the input data before the frequency calculations are performed.

The basic syntax is straightforward and involves three primary components: the procedure declaration, the filtering condition, and the variable to be tabulated. The condition specified in the **WHERE** statement must evaluate to true for the observation to be included in the analysis. For character variables, the comparison value must be enclosed in single or double quotes, as shown in the template below.

The fundamental structure for applying a conditional filter using the **WHERE** statement within **PROC FREQ** follows this powerful design:

```
proc freq data=my_data;
where var1 ='A';
tables var2;
```

run;

This executed code instructs SAS to generate a frequency table specifically for the variable **var2**, but crucially, it only processes rows where the variable **var1** satisfies the condition of being equal to 'A'. This targeted approach significantly streamlines analytical output and ensures that the resulting summary statistics are derived exclusively from the targeted subset of the data.

Setting Up the Sample Dataset for Demonstration

To effectively demonstrate the application of the WHERE statement, we utilize a simulated dataset named `my_data`. This dataset contains observations related to a hypothetical sports league, allowing us to generate specific frequency distributions based on team membership and player performance metrics. The dataset includes three variables: **team** (character), **position** (character), and **points** (numeric).

The creation of this dataset is handled using a standard DATA step, where the `INPUT` statement defines the variables and the `DATALINES` statement provides the raw input records. The use of the dollar sign ($) after the variable name in the input statement signifies that **team** and **position** are character variables, a distinction that is important when formulating filtering criteria in the subsequent **WHERE** statement.

The following code snippet creates and displays the complete dataset used throughout our examples. This ensures reproducibility and provides a clear baseline for understanding which rows are included or excluded by our subsequent filtering logic:

/*create dataset*/
data my_data;
input team $ position $ points;
datalines;
A Guard 22
A Guard 20
A Guard 30
A Forward 14
A Forward 11
B Guard 12
B Guard 22
B Forward 30
B Forward 9
B Forward 12
B Forward 25

```
;
run;

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

Obs	team	position	points
1	A	Guard	22
2	A	Guard	20
3	A	Guard	30
4	A	Forward	14
5	A	Forward	11
6	B	Guard	12
7	B	Guard	22
8	B	Forward	30
9	B	Forward	9
10	B	Forward	12
11	B	Forward	25

Practical Example 1: Filtering by a Single Condition

Our initial practical application involves using a simple, single condition to isolate the data associated with Team 'A'. Suppose the goal is to analyze the distribution of player positions exclusively for Team 'A', ignoring all records related to Team 'B' and any other teams that might be present in a larger dataset. This selective focusing is essential when conducting team-specific performance reviews or ensuring equitable positional distribution within a subset.

To achieve this, we insert the `WHERE team='A'` clause into the PROC FREQ step. This instruction tells the SAS engine to filter the input dataset `my_data` row by row, retaining only those observations where the value in the **team** variable is precisely 'A'. The subsequent `TABLES position;` statement then performs the frequency count only on this filtered subset.

Executing this code demonstrates how efficiently the WHERE statement limits the scope of the procedure, yielding results that are highly relevant to the specific analytical question:

```
/*calculate frequency of position where team is equal to 'A'*/
proc freq data=my_data;
where team='A';
```

tables position;
run;

The resulting output clearly indicates the distribution of positions solely for Team A, confirming that the filter was correctly applied. This filtered frequency table shows exactly five observations were processed, corresponding to the five rows belonging to Team A in our original dataset:

The FREQ Procedure

position	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Forward	2	40.00	2	40.00
Guard	3	60.00	5	100.00

Interpreting this focused output, we can observe the following frequency counts for Team A:

The position "Forward" occurs **2** times for Team A.
The position "Guard" occurs **3** times for Team A.

Advanced Filtering: Combining Conditions with AND and OR Operators

The analytical utility of the WHERE statement is significantly enhanced by its support for complex Boolean logic. By employing logical operators such as **AND**, **OR**, and **NOT**, analysts can construct highly specific filtering criteria that target observations based on multiple variable values simultaneously. This capability moves beyond simple subsetting to enable truly nuanced data exploration.

The **AND** operator is used when an observation must satisfy two or more conditions concurrently (e.g., records where Team is 'A' **AND** Points are greater than 20). Conversely, the **OR** operator includes an observation if it satisfies at least one of the specified conditions (e.g., records where Position is 'Guard' **OR** Team is 'B'). Mastering these operators is critical for defining precise analytical cohorts within your data.

Furthermore, parentheses can be used to control the order of evaluation and group complex logical expressions, ensuring that the filtering logic is executed exactly as intended. For example, if you wanted to analyze players who are either 'Guard' or 'Forward', but only if they belong to team 'A' or 'B', you would use grouping to clearly delineate the desired subsets. This flexibility allows for the immediate investigation of highly specific hypotheses without the need for preparatory DATA step manipulation.

Practical Example 2: Implementing Complex Boolean Logic

To illustrate the power of combined conditions, let us refine our analysis. Instead of finding the frequencies of all positions for Team A, we aim to find the frequency of positions only for players who are designated as 'Guard' *and* who belong to 'Team A'. This double-constraint filtering isolates a very specific subgroup--the Guards of Team A--for frequency counting.

The WHERE statement is modified to include the **AND** operator, linking the two necessary conditions: `WHERE team='A' AND position='Guard'`. This logic dictates that a row will only be counted if **both** the team is 'A' and the position is 'Guard'. All other rows, including Forwards on Team A and all players on Team B, will be skipped during processing.

The resulting code and output effectively demonstrate this precision:

```
/*calculate frequency of position where team is 'A' and position is 'Guard'*/
proc freq data=my_data;
where team='A' and position='Guard';
tables position;
run;
```

The output displays the frequency of the values for the position variable, confirming that only the three rows meeting both criteria were included in the calculation. Since the analysis was filtered only to include Guards, the resulting table shows a frequency count of 3 for the 'Guard' position, and naturally, no other positions appear in the filtered results:

The FREQ Procedure

position	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Guard	3	100.00	3	100.00

Efficiency and Scope: The Power of WHERE vs. Data Step Filtering

A critical consideration for expert SAS programming involves choosing the most efficient method for data subsetting. While filtering can be achieved during a DATA step using a subsetting IF statement (which creates a new, filtered dataset), utilizing the WHERE statement directly within a procedure like **PROC FREQ** often offers superior performance, especially with large datasets.

The fundamental advantage of the procedural WHERE statement lies in its interaction with the

SAS data engine. When a **WHERE** clause is specified, SAS can apply the filter directly when reading the data from storage (I/O optimization). Only the records that satisfy the condition are brought into the procedure's memory space for processing. This avoids the overhead of reading the entire dataset and creating a potentially large intermediate dataset on disk or in memory, which is what happens when using a subsetting IF statement in a DATA step.

Therefore, if the specific filtered subset is only needed for a single procedural analysis (such as calculating a frequency distribution using PROC FREQ), the **WHERE** statement is the most resource-conservative approach. However, if the exact same subset of data will be used repeatedly across multiple procedures (e.g., first running PROC FREQ, then PROC MEANS, then PROC REG), it may be more pragmatic to create a permanent or temporary filtered dataset using a DATA step to avoid redundant filtering operations across different procedures.

Expanding Filtering Capabilities: Numeric Conditions

While our examples focused on filtering character variables (Team and Position), the WHERE statement is equally vital for handling numeric filtering. This is particularly useful in frequency analysis when you are interested in the distribution of one categorical variable conditional on the quantitative range of another variable.

For instance, using our sample data, we might want to analyze the distribution of player positions, but only for high-scoring players--those who achieved more than 20 points. In this case, the condition would be applied to the **points** variable using a comparison operator:

```
proc freq data=my_data;
where points > 20;
tables position;
run;
```

This approach allows analysts to easily isolate specific performance quartiles, outliers, or groups based on metric thresholds. Beyond simple greater-than or less-than comparisons, the **WHERE** statement supports sophisticated range filtering using the **BETWEEN AND** operator (e.g., `WHERE points BETWEEN 10 AND 20`) or checking membership in a list using the **IN** operator (e.g., `WHERE position IN ('Guard', 'Forward')`). These numeric and list-based capabilities significantly enhance the precision available in PROC FREQ analysis.

Key Considerations for Using the WHERE Statement

To ensure successful and error-free execution of the **WHERE** statement within PROC FREQ, several key programming considerations must be kept in mind regarding data type handling and

missing values.

First, always adhere to **data type consistency**. Comparisons involving character variables must use quotation marks (single or double), while comparisons involving numeric variables must be performed without quotes. Mismatching data types (e.g., putting quotes around a numeric comparison value) will generally result in an error or unexpected results, as SAS attempts complex type conversions.

Second, understand how the WHERE statement handles **missing values**. By default, an observation is included in the analysis only if the comparison condition evaluates to true. If the variable used in the WHERE statement has a missing value (represented by a blank for character data or a period for numeric data), the condition is treated as unknown, and the observation is consequently excluded from the procedure. If you need to include missing values in your subset, you must explicitly add a clause like: `OR var IS MISSING`.

In summary, the integration of the **WHERE** statement into **PROC FREQ** is an essential skill for any intermediate or advanced SAS user. It provides a highly efficient, syntax-driven method for achieving analytical precision by ensuring that frequency distributions are calculated only for the specific observations relevant to the current research objective. This capability streamlines code, saves computational resources, and produces highly relevant output.

For complete documentation regarding all available syntax options, constraints, and advanced usage of **PROC FREQ**, please refer to the official SAS documentation.

The following tutorials explain how to perform other common tasks in SAS: