SAS: Use HAVING Clause Within PROC SQL

Authored by

Mohammed looti

April 8, 2026

In the demanding environment of statistical analysis and large-scale data manipulation, the **PROC SQL** procedure in **SAS** stands out as an indispensable tool for data professionals. This procedure offers the efficiency and flexibility of standard **SQL** syntax applied directly within the **SAS** environment. A core feature enabling advanced filtering is the **HAVING clause**, designed specifically to impose conditions on grouped data after aggregation has occurred. Mastering the application of the **HAVING clause** is essential for anyone seeking to extract highly specific and precise insights from complex **datasets**.

While often confused with the **WHERE clause** due to its conditional nature, the **HAVING clause** fulfills a distinctly different and specialized role within **SQL** queries. The power of **HAVING** lies in its timing: it allows filters to be applied to the results of aggregate calculations, such as sums or averages, which is impossible using the initial row-based filtering provided by **WHERE**. This article provides a comprehensive guide to understanding and utilizing the **HAVING clause** within **PROC SQL**, illustrating its utility through a practical, real-world example and clearly defining its operational differences from the **WHERE clause**.

The Critical Distinction: WHERE vs. HAVING

A foundational concept in efficient **SQL** programming, particularly within **PROC SQL**, revolves around differentiating between filtering individual data rows and filtering summarized groups of rows. This distinction dictates whether you should employ the **WHERE clause** or the **HAVING clause**. Understanding this divergence in application is critical for constructing logically sound and performant queries that yield accurate results.

The **WHERE clause** is executed early in the query process, operating exclusively on individual rows **before** any grouping or aggregate calculations are performed. Its purpose is to act as an initial sieve, discarding rows that fail to meet specified criteria before they reach the aggregation stage. Consequently, the **WHERE clause** cannot reference any calculated aggregate values. Attempting to use an **aggregate function** (like SUM, AVG, or COUNT) directly within the **WHERE clause** will invariably result in a processing error because those summary values simply do not exist at that stage of execution.

Conversely, the **HAVING clause** is executed much later, specifically **after** the data has been grouped using the **GROUP BY clause** and after all aggregate functions have been computed. The **HAVING clause** uses these newly calculated summary values (e.g., total sales, average temperature, count of records) to filter the resulting groups. This capability makes the **HAVING clause** indispensable when the analytical requirement is to filter based on the results of the calculation itself--for example, identifying teams whose combined statistics exceed a certain threshold.

The **WHERE clause** filters individual rows based on non-aggregated column values, acting **before**

grouping.
The **HAVING clause** filters groups of rows based on aggregated column values, acting **after** grouping and calculation.

Establishing the Sample Dataset

To clearly demonstrate the practical utility of the **HAVING clause**, we will work with a specific, relatable scenario. We need to create a sample **dataset** representing basketball player statistics. This dataset, which we will name `my_data`, contains essential metrics such as the player's team affiliation, their position, and the total points they scored. This structure allows us to perform meaningful aggregations--such as summing up points per team or position--and then apply group-level filtering.

We begin by utilizing a standard **SAS data step** to construct and populate this sample data table. This process involves defining the variables `team` (character), `position` (character), and `points` (numeric), and then inputting the records detailing the performance of various players across three teams (A, B, and C). This initial setup is crucial as it provides the foundation upon which all subsequent **PROC SQL** operations will be performed.

```
/*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
C Guard 22
C Guard 19
C Guard 10
;
run;
```

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

After executing the data step, a simple `PROC PRINT` command is used to display the contents of the `my_data` table. Viewing the raw data confirms that all records have been loaded correctly and allows us to visualize the starting point for our subsequent group-based analysis using **PROC SQL**.

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
12	C	Guard	22
13	C	Guard	19
14	C	Guard	10

Practical Implementation of HAVING in PROC SQL

With our sample **dataset** prepared, we can now proceed to the core demonstration of the **HAVING clause**. Our analytical objective is twofold: first, we must calculate the total points scored only by players in the 'Guard' position for every team; and second, we must filter these calculated totals to display only those teams where the collective points scored by their guards exceed 50. This scenario perfectly showcases the combined filtering capabilities of the **WHERE clause** (for initial row selection) and the **HAVING clause** (for final group selection).

The following **PROC SQL query** structures this multi-stage filtering process efficiently. The **WHERE clause** filters out all non-Guard players initially, reducing the dataset. The **GROUP BY clause** then aggregates the remaining Guard records by team, calculating `sum(points)` for each

team. Finally, the **HAVING clause** applies the critical filter, ensuring that only those teams whose sum of guard points is greater than 50 are included in the final report, thereby fulfilling our specific analytical requirement.

```
proc sql;
select team, sum(points) as sum_points
from my_data
where position='Guard'
group by team
having sum_points>50;
quit;
```

The execution of this query produces a highly focused result set. Team A (72 points) and Team C (51 points) meet the criteria, while Team B (34 points) is excluded. This successful outcome demonstrates the precise control offered by the **HAVING clause**, enabling analysts to quickly isolate groups that satisfy complex quantitative conditions based on summarized data.

team	sum_points
A	72
C	51

Understanding the Logical Execution Order

To truly appreciate the power and necessity of the **HAVING clause**, it is essential to understand the internal logical processing order that **PROC SQL** follows. Although the clauses in an **SQL** query are typically written in the order SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY, the actual sequence of execution differs significantly. This logical sequence determines when filters are applied and when aggregations are calculated.

Understanding this flow is crucial for predicting query behavior and for debugging issues where aggregation or filtering seems incorrect. The distinct timing of the **WHERE clause** versus the **HAVING clause** is the key differentiator, ensuring that individual row filters are applied before grouping, and group filters are applied after the summary calculations are complete.

The **FROM clause** is processed first, identifying the source table (`my_data`) to be used as the input for the entire query operation.

Next, the **WHERE clause** (`where position='Guard'`) is applied, filtering individual rows and

retaining only those records where the position is 'Guard'.

The **GROUP BY clause** (`group by team`) then takes the remaining rows and consolidates them into distinct groups based on the unique values in the `team` column.

The **SELECT statement**'s aggregate function (`sum(points) as sum_points`) is calculated for each of the groups established in the previous step.

Finally, the **HAVING clause** (`having sum_points>50`) filters these newly formed groups based on the calculated aggregate value, eliminating any group whose sum of points does not meet the condition.

The Impact of Omitting Group Filtering

To provide compelling evidence of the **HAVING clause**'s specific role, it is instructive to run the same query while intentionally omitting the group filter. By doing so, we gain visibility into the intermediate result set--the aggregated data that exists just before the **HAVING clause** would have executed. This comparison clearly highlights exactly which groups the **HAVING clause** is responsible for excluding.

The following **PROC SQL query** retains the initial row filtering (`where position='Guard'`) and the grouping and aggregation (`group by team` and `sum(points)`), but removes the group-level condition. The output will show the total points scored by guards for all teams, regardless of whether that total exceeds 50.

```
proc sql;
select team, sum(points) as sum_points
from my_data
where position='Guard'
group by team;
quit;
```

When reviewing the results of this modified query, we observe the inclusion of Team B, whose guards totaled 34 points. This group was correctly excluded in the previous example but is now displayed because the group filter was removed. This outcome unequivocally confirms that the **HAVING clause** acts as the sole mechanism for filtering aggregated results based on conditions applied to those summarized values, providing fine-grained control over the final presentation of data.

team	sum_points
A	72
B	34
C	51

Conclusion

The **HAVING clause** represents an indispensable feature within the **PROC SQL** environment in **SAS**, offering a robust and necessary method for filtering data post-aggregation. The fundamental difference between the **WHERE clause** and the **HAVING clause** centers entirely on the execution timing: **WHERE** operates on individual rows prior to grouping, whereas **HAVING** operates on calculated groups after aggregation has been completed.

By strategically employing the **HAVING clause**, data analysts gain the ability to impose granular conditions on summarized data, allowing for the extraction of highly targeted and meaningful insights. This capability is paramount in complex analytical tasks where identifying groups that meet specific quantitative thresholds--such as minimum performance benchmarks or maximum expenditure limits--is required.

A comprehensive understanding of when and how to apply both the **WHERE** and **HAVING** clauses is critical for any professional utilizing **PROC SQL** in **SAS**, empowering them to write sophisticated, accurate, and highly efficient data processing queries.

Additional Resources

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