Learning Data Sorting with ORDER BY in SAS PROC SQL: A Comprehensive Guide

Authored by

Mohammed looti

November 14, 2025

In the dynamic world of data analysis and business intelligence, the capacity to structure and present information in a logical, coherent manner is paramount. For professionals utilizing SAS, the industry-leading statistical software suite, the PROC SQL procedure provides a powerful, familiar interface for executing sophisticated SQL-like queries directly against their datasets. While many clauses control data selection and filtering, the **ORDER BY** statement is the fundamental mechanism that governs the final output presentation, transforming raw results into actionable reports.

The core function of the **ORDER BY** clause is to impose a specific sequence on the resulting rows of a query, based on the values found within one or more designated columns. Without this explicit instruction, the arrangement of the resulting data is essentially arbitrary and highly unreliable, potentially obscuring crucial trends or relationships that analysts rely upon. Therefore, mastering the efficient application of **ORDER BY** is not merely a refinement; it is a critical skill for generating clean, consistent, and interpretable analytical output.

This comprehensive guide aims to solidify your expertise in data sorting within the PROC SQL environment. We will systematically explore the required syntax and practical use cases of the **ORDER BY** statement, starting with simple single-variable sorting--both ascending and descending order--and progressing to complex, multi-variable hierarchical sorting techniques. By the end of this tutorial, you will possess the precise expertise needed to ensure your SAS output is organized exactly according to your analytical specifications, significantly boosting both efficiency and decision-making quality.

The Necessity of Deterministic Data Ordering in PROC SQL

PROC SQL is one of the most versatile and widely adopted procedures within the SAS ecosystem. Its primary strength lies in integrating the structured querying power of standard SQL--a language familiar to database experts globally--with the specialized data management capabilities inherent to the SAS environment. This procedure allows users to efficiently extract, aggregate, transform, and join data from multiple datasets using core declarative constructs such as `SELECT`, `FROM`, and `WHERE`.

It is essential to distinguish the roles of these clauses: the `SELECT` statement defines which variables are retrieved, and the `WHERE` clause filters which observations (rows) are included in the result set. In contrast, the **ORDER BY** clause dictates the final sequence or presentation layer of the output. This distinction is critical because sorting is a display concern, not a data manipulation task. Crucially, without an explicit **ORDER BY** clause, the sequence of rows in your output is entirely dependent upon the internal storage mechanisms, indexing, or optimization decisions made by the PROC SQL engine.

Relying on arbitrary output order means that running the exact same query twice might produce

results in different physical sequences, rendering the output unreliable for direct comparison, reporting, or auditing. By explicitly including **ORDER BY**, you enforce a deterministic order on the results, guaranteeing consistency regardless of underlying system variations or query optimization paths. This reliability is foundational for critical tasks such as quickly identifying maximum or minimum values, generating Top-N reports, or preparing structured input for visualization tools. Therefore, the **ORDER BY** clause should be regarded as a mandatory component whenever the sequence of data holds analytical or reporting significance.

Understanding the ORDER BY Syntax and Directives

The syntax for the **ORDER BY** clause is highly intuitive. It must be positioned after the `FROM` and any optional `WHERE` or `GROUP BY` clauses within the PROC SQL statement. You specify the variables you wish to sort by, separating them with commas. To control the direction of the sort, two main directives are used:

ASC: Specifies ascending order. For character data, this means alphabetical order (A-Z); for numeric data, it means lowest to highest values. This is the **default behavior** in PROC SQL; if you omit both `ASC` and `DESC`, the system sorts in ascending order.
DESC: Specifies descending order. This reverses the sort: Z-A for character data and highest to lowest for numeric data. This keyword must be explicitly added immediately following the variable name.

These directives are the key to precisely manipulating data presentation. For numerical variables, ascending order is commonly used for temporal or sequential data views. Conversely, descending order is indispensable for ranking, such as when displaying the top sales figures, highest scores, or most recent records, ensuring the most significant values are prioritized at the top of the report. The following practical demonstrations will utilize these principles to illustrate real-world sorting scenarios.

Preparing the Sample Dataset in SAS

To provide clear and reproducible examples of the **ORDER BY** clause in action, we must first establish a sample dataset. We will construct a table named **my_data**, which simulates player statistics for a fictional basketball league. This dataset contains four key variables: **team** (character), **position** (character), **points** (numeric), and **assists** (numeric). Creating this structure in SAS is most efficiently accomplished using a `DATA step` combined with the `DATALINES` statement to input the raw data directly.

The code block below generates the **my_data** table, populating it with 12 observations. Note how the input statement defines the type of each variable: the dollar sign (`$`) designates character variables (`team` and `position`), while the absence of the dollar sign defaults to numeric variables

(`points` and `assists`). This careful setup ensures that our subsequent PROC SQL procedures interact correctly with both text and numeric data types.

```
/*create dataset*/
data my_data;
input team $ position $ points assists;
datalines;
A Guard 14 4
B Guard 22 6
B Guard 24 9
A Forward 13 8
C Forward 13 9
A Guard 10 5
B Guard 24 4
C Guard 22 6
D Forward 34 2
D Forward 15 5
B Forward 23 5
B Guard 10 4
;
run;

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

The output displayed below, generated by the `PROC PRINT` statement, represents the initial, unsorted state of **my_data**. This view serves as our essential analytical baseline. We can clearly observe that the current sequence of rows simply reflects the order in which the data was entered via the `DATALINES` statement--an order that is rarely suitable for formal analytical reporting.

Obs	team	position	points	assists
1	A	Guard	14	4
2	B	Guard	22	6
3	B	Guard	24	9
4	A	Forward	13	8
5	C	Forward	13	9
6	A	Guard	10	5
7	B	Guard	24	4
8	C	Guard	22	6
9	D	Forward	34	2
10	D	Forward	15	5
11	B	Forward	23	5
12	B	Guard	10	4

Method 1: Single-Variable Ascending Sort (Default)

The most straightforward application of the **ORDER BY** clause involves sorting the entire dataset based on the values of a single variable in ascending order. As previously established, PROC SQL defaults to ascending order, meaning we can omit the **ASC** keyword, although including it can enhance clarity, especially in complex scripts.

In our basketball example, sorting by the **team** variable in ascending order will arrange the output alphabetically (A, B, C, D). This immediate organization provides structure to the data, making it effortless to locate all players associated with a specific team. This technique is indispensable for generating organized directories, listings, or any report where alphabetical or chronological sequence is required.

```
/*display results in ascending order by value in team column*/
proc sql;
select *
from my_data
order by team;
quit;
```

In the execution above, the `SELECT *` command retrieves all columns from our source table, **my_data**. The subsequent `ORDER BY team` clause instructs the PROC SQL engine to sort the

resulting rows based solely on the alphabetical sequence of the team names. The `QUIT` statement is necessary to terminate the procedure and finalize the result set generation.

team	position	points	assists
A	Forward	13	8
A	Guard	14	4
A	Guard	10	5
B	Guard	24	4
B	Guard	24	9
B	Guard	10	4
B	Forward	23	5
B	Guard	22	6
C	Forward	13	9
C	Guard	22	6
D	Forward	15	5
D	Forward	34	2

As confirmed by the output, the rows are now perfectly arranged by team, starting with A and ending with D. This simple yet powerful technique forms the indispensable foundation for all subsequent, more complex sorting operations we will explore.

Method 2: Prioritizing Values with Descending Order

While ascending order (lowest to highest) is the default, data analysts frequently need to prioritize the highest values, such as identifying top-performing entities, the largest transaction volumes, or the most recently recorded data. For these essential scenarios, the **DESC** keyword must be utilized to explicitly enforce a descending order sort.

To demonstrate this functionality, let us sort our basketball data by the **points** variable in descending order. This action will immediately highlight the highest individual scores, effectively creating a simple leaderboard across all teams. The explicit use of the **DESC** keyword is mandatory here, as it overrides the default ascending behavior, ensuring that the maximum values appear first.

```
/*display results in descending order by value in team column*/
proc sql;
select *
```

```
from my_data
order by points desc;
quit;
```

In this revised code block, the addition of the **DESC** directive after the **points** variable reverses the numerical sequence, placing the highest scores (34, 24, 24, etc.) at the beginning of the output. This method proves invaluable for quickly pivoting data presentation to focus on the elements with the greatest magnitude or importance. If we had applied **DESC** to a character variable, such as **team**, the sort would begin with Team D and proceed backward alphabetically.

team	position	points	assists
D	Forward	34	2
D	Forward	15	5
C	Forward	13	9
C	Guard	22	6
B	Guard	24	4
B	Guard	10	4
B	Forward	23	5
B	Guard	24	9
B	Guard	22	6
A	Guard	10	5
A	Guard	14	4
A	Forward	13	8

The resulting output clearly validates the effectiveness of the **DESC** keyword. When sorting character variables in descending order, PROC SQL adheres to the collation sequence defined by the SAS system options, ensuring accurate reverse alphabetical sorting across all characters.

Method 3: Hierarchical Sorting with Multiple Variables

In many real-world analytical scenarios, simple sorting is insufficient. Data often requires hierarchical ordering, meaning the data must first be sorted based on a primary criterion, and then, for all records that share the same primary value, sorted again based on a secondary criterion. PROC SQL expertly handles this complex ordering by permitting multiple variables to be listed in the **ORDER BY** clause, separated by commas. The sequence in which the variables appear dictates the hierarchy of the sort keys.

Furthermore, this multi-variable approach allows for mixed sorting specifications. For instance, we might want to list teams alphabetically (the primary sort, ascending) and simultaneously rank players within each team by their performance (the secondary sort, descending order of **points**). This powerful combination is achieved by applying the **DESC** keyword only to the secondary variable, **points**, while leaving the primary variable, **team**, to default to ascending order.

We will execute a query that first sorts by **team** and then by **points** in descending order. This structure provides immediate insight into the internal ranking of players within their respective teams, yielding a report structure far more informative than a flat sort based on a single variable.

```
/*display results in ascending order by team, then descending order by points*/
proc sql;
select *
from my_data
order by team, points desc;
quit;
```

The successful execution of this statement demonstrates true hierarchical sorting. All rows belonging to Team A are rigorously grouped together first. Only within that group is the secondary sorting criterion applied, ensuring the player with the highest points appears at the top of the Team A section. This sequential, layered process is then repeated independently for Team B, Team C, and Team D. This approach is critical for detailed reporting where sub-group analysis, such as regional sales aggregation or departmental performance rankings, is required.

team	position	points	assists
A	Guard	14	4
A	Forward	13	8
A	Guard	10	5
B	Guard	24	9
B	Guard	24	4
B	Forward	23	5
B	Guard	22	6
B	Guard	10	4
C	Guard	22	6
C	Forward	13	9
D	Forward	34	2
D	Forward	15	5

The resulting output clearly validates the mixed sort criteria: Team A is sorted first, and within Team A, the players are ranked by points (14, 13, 10). This confirms that PROC SQL executes sorting operations sequentially, applying the secondary criterion only when the values of the primary criterion are identical (a tie).

Conclusion

The ability to reliably sort and structure output is a cornerstone of effective data analysis, and the **ORDER BY** statement provides SAS users with the precise control needed over their query results. We have explored the fundamental necessity of sorting for ensuring consistent data presentation, moving beyond the arbitrary order of raw data to an organized, readable structure.

Whether you are implementing a simple ascending sort using default behavior, ranking data using the **DESC** keyword, or executing sophisticated, multi-variable hierarchical sorting to rank sub-groups, the principles remain constant: clarity, consistency, and precision. By integrating these **ORDER BY** techniques into your workflow, you enhance not only the visual appeal of your reports but also the analytical depth of your findings, ensuring that your PROC SQL outputs are always fit for purpose.

Mastering this essential SQL clause within the PROC SQL environment is a critical step toward becoming a proficient SAS programmer and data analyst. Continue practicing these methods with your own datasets to solidify your expertise.

Additional Resources

To further your understanding and explore more advanced functionalities in SAS and PROC SQL, consider the following tutorials and documentation:

SAS PROC SQL Documentation: Official documentation for comprehensive details on all aspects of PROC SQL.
Introduction to SAS for Statistical Analysis: General resources for getting started with SAS.
SQL Tutorial (W3Schools): A general SQL reference that can complement your PROC SQL learning.