Do you use ORDER BY when you do PROC SQL in SAS?

Authored by

stats writer

November 19, 2025

Yes, the ORDER BY clause is used when doing PROC SQL in SAS to sort the output of the query result set in either ascending or descending order based on the specified column(s). This fundamental SQL operation is critical for presentation and subsequent data processing within the SAS environment.

1. Understanding the ORDER BY Clause in PROC SQL

The use of the ORDER BY statement within the PROC SQL procedure is foundational for any effective data analysis and reporting task performed in SAS. When executing a query, SAS retrieves data records according to the criteria defined in the `SELECT` and `WHERE` clauses. However, without explicit instructions, the order of the resulting rows is not guaranteed and often reflects the physical storage order of the data file, which is typically arbitrary and non-analytical.

The primary purpose of the ORDER BY clause is to impose a specific, logical sequence on the output. This sorting mechanism allows analysts to quickly identify trends, check for outliers, or prepare data for visual presentation where a defined flow (e.g., chronological, alphabetical, or numerical) is necessary. By using this statement, you instruct the PROC SQL engine to arrange the output based on the values contained in one or more specified columns, ensuring reproducibility and clarity in your results.

It is crucial to remember that the ORDER BY clause is always the final step in the logical processing of a SQL query execution, after filtering (`WHERE`), grouping (`GROUP BY`), and column selection (`SELECT`). Its placement must therefore be at the end of the `SELECT` statement before the `QUIT` or semicolon, guaranteeing that the results are sorted only after all necessary transformations and aggregations have taken place.

2. The Mechanics of the ORDER BY Statement

The syntax for incorporating the ORDER BY statement is straightforward, yet highly flexible, allowing for complex sorting requirements. You specify the variable(s) (columns) by which you wish to sort the dataset. By default, if no direction is specified, SAS, like most SQL implementations, defaults to ascending order (A to Z, 0 to 9, or oldest to newest dates). The keywords used to control this direction are `ASC` (for Ascending) and `DESC` (for Descending).

When sorting character variables, the sort order depends on the collation sequence defined by your SAS session options, typically following standard alphabetical rules. For numeric variables, the sorting is based on magnitude. Handling missing values is also important; by default, SAS treats missing values as the smallest possible value, placing them first in an ascending sort and last in a descending sort. Careful consideration of data types and missing values ensures the final sorted output accurately reflects analytical needs.

Understanding the difference between sorting the output and physically sorting the source dataset is vital. PROC SQL's **ORDER BY** clause only affects the temporary result set displayed or saved by the query. It does not modify the physical order of the rows in the original source table. If there is a need to permanently reorder a dataset for subsequent procedures or long-term storage, the separate `PROC SORT` procedure in SAS should be used instead.

To demonstrate the versatility of the **ORDER BY** statement in **PROC SQL** in SAS, we will examine three fundamental methods used to structure query results by the values of one or more variables.

3. Defining the Three Core Sorting Methods

The application of the sorting clause can be broadly categorized into three common scenarios, ranging from simple single-column sorts to complex multi-level arrangements. These methods cover the vast majority of reporting requirements necessary for data analysis and visualization within the SAS environment.

Here are three common ways to use the **ORDER BY** statement in practice:

Method 1: Order By One Variable Ascending (Default): This is the simplest form, sorting a query result based on a single column in its natural sequence (lowest to highest). Since this is the default behavior, the keyword `ASC` can be omitted.
Method 2: Order By One Variable Descending: This method requires the explicit use of the `DESC` keyword to reverse the natural sort order, which is often useful for prioritizing highest values, such as identifying top performers or viewing recent transactions first.
Method 3: Order By Multiple Variables (Hierarchical Sorting): This advanced technique allows for nested sorting, where the result set is first sorted by the primary variable, and then, only within groups of identical values for the primary variable, the secondary variable dictates the order. This is essential for highly detailed, grouped reports.

Let's look at the basic structure for each method using illustrative code snippets before applying them to a practical dataset.

Method 1: Order By One Variable Ascending (Code Snippet)

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

Method 2: Order By One Variable Descending (Code Snippet)

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

Method 3: Order By Multiple Variables (Code Snippet)

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

4. Setting Up the Demonstration Dataset in SAS

To fully illustrate the behavior of the ORDER BY clause, we will utilize a practical dataset created directly within the SAS environment. This dataset, named `my_data`, contains information typical of statistical analysis--specifically, attributes of various basketball players, including their team assignment, position, and performance metrics such as points and assists. This table serves as the foundation for all subsequent sorting examples.

The structure of the data includes both character variables (`team` and `position`) and numeric variables (`points` and `assists`). This mix allows us to observe how PROC SQL handles sorting across different data types, which is essential for accurate data manipulation. The initial creation and viewing of this dataset serves as the baseline for comparison against the sorted outputs generated by the subsequent SQL queries.

The following SAS code block uses the `DATA` step combined with `DATALINES` to populate the `my_data` table, followed by `PROC PRINT` to show the unsorted, original arrangement of the data. Notice that the initial order of records is arbitrary, matching the physical input sequence, which is why sorting is necessary for analytical clarity.

```
/*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;
```

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

5. Example 1: Sorting by a Single Variable in Ascending Order

The first method demonstrates the most straightforward application of the sorting clause: ordering

the entire result set based on one variable using the default ascending order. When the `ASC` keyword is omitted, PROC SQL assumes the ascending direction. For character fields like `team`, this means sorting alphabetically (A to Z). For numeric fields, it would mean sorting from the smallest number to the largest.

In this example, we aim to organize the basketball player data such that all records belonging to Team A appear first, followed by Team B, Team C, and Team D. This arrangement is highly practical when generating reports segmented by organizational units or categories, ensuring that related records are displayed contiguously. The lack of an explicit `ASC` keyword showcases how SQL simplifies syntax when the default sort direction is desired.

The following code shows how to return every row in the dataset in ascending order by the values in the **team** column:

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

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

Notice that the results are clearly shown in ascending order by the values in the **team** column. All

rows are organized sequentially by team letter, demonstrating the successful application of the default sort order. This simple yet powerful command transforms raw data into easily digestible structured information.

6. Example 2: Implementing Descending Order

When analyzing performance metrics or time-sensitive data, it is frequently necessary to prioritize the highest values or the most recent observations. This requires reversing the default sort mechanism, which is achieved using the `DESC` keyword immediately following the variable name in the ORDER BY clause. Using descending order ensures that the most important or largest records appear at the beginning of the output, which is crucial for ranking or prioritizing items.

In this second example, we apply the descending sort to the `team` column. Instead of A, B, C, D, the output will sequence the teams starting with D and ending with A. This technique is often used in combination with numeric fields (e.g., `ORDER BY points DESC`) to instantly identify leaders or records of highest magnitude, but it is equally valid for character fields where a reversed alphabetical display is desired.

The code below explicitly uses the `DESC` keyword to instruct the PROC SQL engine to display the data records starting from the highest value in the `team` column down to the lowest value, thereby inverting the natural sort order established in the previous example.

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

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

Notice that the results are shown in descending order by the values in the **team** column. The teams are now sorted D, C, B, A, confirming that the `DESC` operator successfully reversed the sequence of the output table.

7. Example 3: Advanced Sorting with Multiple Variables

Often, a single sort variable is insufficient because data records may share identical values (ties) in that primary column. In such cases, a secondary, or even tertiary, sorting criterion is required to establish a precise sequence. This is achieved by listing multiple variables in the ORDER BY clause, separated by commas. This process is known as hierarchical or multi-level sorting.

The sorting operates sequentially: SAS first sorts the entire dataset by the first variable listed. For all records that share the same value for the first variable (e.g., all players on Team A), the engine then uses the second variable to sort those tied records. This continues down the list of specified variables. Furthermore, each variable can be assigned its own sort direction (`ASC` or `DESC`), allowing for highly customized arrangements within groups.

In this third and most complex example, we want to organize the data primarily by **team** in ascending order (A, B, C...). Crucially, within each team grouping, we want to rank players by their **points** scored in descending order, ensuring the top scorer for each team is listed first within their respective group, regardless of when they were entered into the original dataset.

The following code shows how to return every row in the dataset first in ascending order by **team**,

then in descending order by **points**:

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

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

Notice that the results are shown first in ascending order by **team** (A, then B, then C, then D). Then, within Team B, for instance, the points are sorted in descending order (24, 24, 23, 22, 10). This confirms the successful implementation of hierarchical sorting, providing highly structured and analytically useful output where both category and ranking are controlled simultaneously.

8. Summary and Further SAS Applications

Mastering the ORDER BY clause is a prerequisite for generating professional and clear reports using PROC SQL in SAS. Whether you are using the default ascending sort, specifying descending order, or creating complex multi-level hierarchical sorts, the consistent application of this clause ensures that your query results are presented logically and are easy for end-users or subsequent programs to interpret.

The examples demonstrated here--sorting by a single variable, reversing the sort order, and

sorting hierarchically--represent the core capabilities of the sorting function. By integrating this knowledge with other powerful SQL features such as aggregation (using `GROUP BY`) and joining (using `JOIN` statements), you can elevate your data processing efficiency and the quality of your analytical outputs within the SAS environment.

The following tutorials explain how to perform other common tasks in SAS, building upon the foundational knowledge of data manipulation provided by the PROC SQL procedure:

Tutorial on creating calculated columns within PROC SQL.
Guide to performing complex joins and merges in SAS.
Detailed explanation of using the WHERE clause for efficient data filtering.