How to use MONOTONIC function in SAS?

Authored by

stats writer

November 19, 2025

The MONOTONIC function in SAS is an extremely valuable tool for data management specialists and analysts working within the statistical computing environment. Its primary utility lies in its ability to generate a sequence of numbers that is either monotonically increasing or decreasing based on the order of the records being processed. This function is particularly powerful when used within PROC SQL, allowing users to easily assign inherent numerical order to the rows, effectively creating a persistent record index within the resulting data set.

While the function accepts two conceptual arguments--a start and an end value--when employed in standard SAS procedures like PROC SQL, it typically operates implicitly, producing a series of integers starting from one and incrementing by one for each processed row, mirroring a traditional row index. This simplicity makes it ideal for tasks requiring sequential identification, such as ensuring unique numbering for every observation or for filling down numerical sequences like dates across large-scale data structures. Understanding its proper implementation is fundamental for efficient data manipulation in complex analytical projects.

The core strength of the MONOTONIC function is its role in providing ordinality. In the absence of a unique key or explicit ordering variable, data records often lack a native sequential identifier. By applying MONOTONIC(), you impose a deterministic order derived from the processing sequence of the query. This capability is critical for quality assurance, debugging, and for facilitating subsequent operations that depend on knowing the original or relative position of a record within the processed data flow. We will explore two primary and powerful methods for utilizing this function: creating an explicit column of row numbers and filtering data sets based on row position.

Understanding Monotonic Sequences in Data Processing

The term "monotonic" originates from mathematics, where a function is described as monotonically increasing if, as the input increases, the output never decreases. Conversely, it is monotonically decreasing if the output never increases. In the context of the SAS **MONOTONIC()** function, this principle translates directly to the sequential numbering of records. Each subsequent row processed by the query is guaranteed to receive a row index value that is strictly greater than the preceding row's index, ensuring absolute sequential integrity from the start of the query execution to the end.

When dealing with huge volumes of data, having a reliable mechanism to assign temporary or permanent row IDs becomes essential for performance and accuracy. Traditional methods, sometimes involving iterative DATA steps or complex counter variables, can be cumbersome and less performant than the optimized, built-in function provided by SAS. The **MONOTONIC()** function, especially within the efficient framework of PROC SQL, provides a simple, single-line solution to this pervasive data management challenge. It fundamentally simplifies the task of identifying specific records by their relative position rather than relying solely on content-based

filtering.

It is important to differentiate the **MONOTONIC()** function from generating arbitrary sequence numbers. While it generates sequence numbers, the critical feature is the guarantee of a monotonically increasing sequence tied to the order in which the rows are retrieved from the source table. This ordered retrieval, however, is heavily dependent on the query structure; if an `ORDER BY` clause is utilized in the PROC SQL statement, the **MONOTONIC()** function will assign numbers based on that specified sort order. If no order is specified, the sequence is based on the physical retrieval order of the records, which is often not guaranteed to be consistent across different environments or query executions unless defined by the underlying storage mechanism.

Setting Up the Sample Data Set for Demonstration

To illustrate the practical application of the **MONOTONIC()** function, we will utilize a small, easily digestible sample data set. This fictional sports team data set, named `my_data`, contains information about various basketball teams, their scores (points), and assists. This consistent source data will serve as our foundation for both methods demonstrated below, ensuring clarity in how the row index is applied and used.

The following SAS code creates the necessary dataset using a standard DATA step structure, defining the variables `team` (character), `points` (numeric), and `assists` (numeric), followed by the `datalines` containing the nine distinct records, or observations. It is essential to execute this step first before attempting to run any PROC SQL statements that reference `my_data`.

The code block below outlines the creation and subsequent printing of the sample data structure. Notice that the data is naturally ordered by the sequence of entry into the `datalines` statement, which will be the basis for the **MONOTONIC()** numbering in our subsequent examples.

```
/*create dataset*/
data my_data;
input team $ points assists;
datalines;
Cavs 12 5
Cavs 14 7
Warriors 15 9
Hawks 18 9
Mavs 31 7
Mavs 32 5
Mavs 35 3
Celtics 36 9
Celtics 40 7
```

```
;
run;

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

The visual representation of this input data confirms its structure, showing nine distinct rows, each representing a statistical observation of a team's performance. The internal numbering assigned by SAS during the `proc print` step (often labeled as the OBS column) aligns perfectly with the future results of our **MONOTONIC()** usage, demonstrating the clear sequential order we aim to capture.

Obs	team	points	assists
1	Cavs	12	5
2	Cavs	14	7
3	Warriors	15	9
4	Hawks	18	9
5	Mavs	31	7
6	Mavs	32	5
7	Mavs	35	3
8	Celtics	36	9
9	Celtics	40	7

Method 1: Generating Unique Row Identifiers Using MONOTONIC()

One of the most frequent and critical uses of the **MONOTONIC()** function is the explicit creation of a unique row identifier column. Unlike systems that automatically manage row IDs, SAS data sets often require the user to define such a variable. By using MONOTONIC() within a `SELECT` statement in PROC SQL, we can effortlessly generate a new variable that ensures every single record receives a unique, sequential number, beginning at 1.

The syntax for this implementation is remarkably straightforward. By including `monotonic() AS row_ID` in the select clause, we instruct SAS to execute the function and assign its resulting value to the new column alias, `row_ID`. This approach adheres to the principle of a monotonically increasing sequence, as each row retrieved is guaranteed to increment the counter by one, providing an accurate count of the processed records up to that point. This generated column serves as an excellent primary key substitute for subsequent merging or linking operations where the relative order of the data is paramount.

For large analytical workflows, having a distinct row identifier is often non-negotiable, particularly when performing data quality checks or when trying to replicate specific subsets of the data based on their original loading order. The use of **MONOTONIC()** ensures that this assignment is handled efficiently at the database level (or within the PROC SQL engine), minimizing the need for complex procedural code. Here are the two common ways to use this function in practice:

Method 1: Use MONOTONIC() to Create Column of Row Numbers

```
/*create column called row_ID that contains row numbers*/
proc sql;
select team, monotonic() as row_ID
from my_data;
quit;
```

Method 2: Use MONOTONIC() to Filter Dataset by Row Numbers

```
/*filter where row number is less than 5*/
proc sql;
select *
from my_data
where monotonic() < 5;
quit;
```

Detailed Walkthrough of Example 1 (Creating row_ID)

This example demonstrates the exact implementation of Method 1 using the `my_data` data set. We aim to project the existing `team` variable alongside a newly generated sequence number, labeling the new column `row_ID`. This process effectively preserves the original data while annotating each record with its positional index.

The following code shows how to use the **MONOTONIC()** function to create a new column called **row_ID** that contains the row number (starting from 1) for each observation in the dataset:

```
/*create column called row_ID that contains row numbers*/
proc sql;
select team, monotonic() as row_ID
from my_data;
quit;
```

Upon execution, the resulting table clearly shows the utility of the function. A new column, **row_ID**,

has been successfully generated, containing a monotonically increasing sequence ranging from 1 to 9, corresponding exactly to the nine initial observations in the source data. This generated ID is now available for any further processing steps within the SAS environment.

team	row_ID
Cavs	1
Cavs	2
Warriors	3
Hawks	4
Mavs	5
Mavs	6
Mavs	7
Celtics	8
Celtics	9

Notice that a new column has been created called **row_ID** that contains the row number for each observation in the dataset, ranging from 1 to 9.

Method 2: Filtering Data Sets Based on Row Position

Beyond simply creating a sequence column, the **MONOTONIC()** function offers powerful capabilities for subsetting and filtering data based on its positional order. This technique is especially useful when analysts need to retrieve the "top N" or "first M" records from a table, mimicking functionality often handled by `LIMIT` clauses in other SQL dialects. By placing the **MONOTONIC()** function within the `WHERE` clause of a PROC SQL query, we can restrict the output based on the assigned sequential row number as it is generated dynamically.

This method leverages the fact that the function is evaluated sequentially for each row. When the query encounters a row, it calculates the row's ordinal position and immediately tests it against the specified condition in the `WHERE` clause. For example, setting the condition `WHERE monotonic() < 5` ensures that the query stops processing or suppresses output for all records once the sequential counter reaches five. This provides a clean and highly efficient way to extract initial segments of a data set.

The following code shows how to use the **MONOTONIC()** function to filter a dataset where the row number is less than 5:

```
/*filter where row number is less than 5*/
proc sql;
select *
from my_data
where monotonic() < 5;
quit;
```

Detailed Walkthrough of Example 2 (Filtering Rows)

To demonstrate the filtering capability, we applied the **MONOTONIC()** function within the `WHERE` clause to select only the first four observations from our `my_data` set. We achieved this by setting the condition to be strictly less than 5 (`< 5`), thereby including rows 1, 2, 3, and 4 in the output.

Executing the code block above demonstrates the dynamic filtering process. The engine begins retrieving rows, increments the implicit sequential counter using **MONOTONIC()**, and halts output generation as soon as that counter reaches the threshold of 5.

team	points	assists
Cavs	12	5
Cavs	14	7
Warriors	15	9
Hawks	18	9

Notice that only the first four rows from the dataset are shown since we used the **MONOTONIC()** function to specify that the row number must be less than 5.

Conclusion and Further Resources

The **MONOTONIC()** function remains a cornerstone utility for data manipulation in SAS, offering a simple yet robust mechanism for generating reliable, sequential row identifiers. Whether the goal is to create a primary key for joining purposes or to filter the initial portion of a large table, the function provides a high-performance solution within the PROC SQL environment. Mastery of this function significantly enhances an analyst's ability to handle ordered data tasks efficiently.

For those looking to deepen their expertise in advanced SAS data management techniques, exploring related functions and procedures is highly recommended. The principles demonstrated here regarding sequential processing are often applicable to other scenarios, such as generating

sequence numbers partitioned by group or handling iterative data transformations. Always consult the official SAS documentation for the most precise details on implementation nuances across different versions of the software.

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

Understanding the difference between monotonically increasing and non-monotonic data structures.

Using PROC SQL window functions for complex aggregation.

Techniques for large data set manipulation in the SAS environment.