How to Use _N_ in SAS (3 Examples)

Authored by

stats writer

November 19, 2025

The **_N_** automatic variable in the SAS data step serves as an indispensable tool for data manipulation and control. This system variable is automatically generated by SAS during execution and is used primarily to assign a sequential number to each observation being processed. Essentially, **_N_** counts the number of times the data step has iterated through the input data, starting at 1 for the first record.

Utilizing **_N_** allows programmers to precisely track observations within a SAS data set, facilitating complex conditional logic. Common use cases include generating unique identifiers, controlling the flow of data output, or selecting specific subsets of data based on their position in the input file. Understanding how **_N_** operates during the processing cycle is fundamental to mastering efficient SAS programming techniques.

Defining the _N_ Automatic Variable

The **_N_** automatic variable in SAS is a temporary counter that exists solely within the data step execution environment. It is not read from the input data; rather, it is created and incremented internally by the SAS system with every iteration of the data step loop. It begins counting at 1 when the first record is read and continues until the end of the input file is reached.

This variable is implicitly retained and incremented by SAS, meaning you do not need to initialize or define it explicitly. Because it reflects the current iteration number, **_N_** provides a direct and reliable way to target specific records based on their sequential position within the input data set. This is particularly valuable when the physical order of the data holds analytical significance.

Practical Applications of _N_ in Data Manipulation

The power of the **_N_** variable lies in its ability to control data flow using conditional statements, such as `IF-THEN` logic. By checking the current value of **_N_**, the programmer can decide whether to process, output, or skip the current observation. Below are the three most common and effective ways to integrate **_N_** into your SAS routines, offering solutions for filtering, subsetting, and indexing data.

Method 1: Use _N_ to Select First Observation in Dataset

This technique is essential when you only need the initial record for auditing, summary statistics, or display purposes. By checking if **_N_** equals 1, we isolate the very first iteration of the SAS data step, ensuring that only this single record is written to the output file.

```
data new_data;
set original_data;
if _N_ = 1 then output;
```

run;

Method 2: Use _N_ to Select First N Observations in Dataset

To extract a specific number of records (e.g., the top 5 or top 10), we employ a conditional check that ensures _**N**_ does not exceed the desired threshold. This is a simple and effective way to sample or limit output data, often used to create smaller testing data sets from much larger sources.

```
data new_data;
set original_data;
if _N_ <= 5 then output; /*select first 5 rows*/
run;
```

Method 3: Use _N_ to Add Sequential Row Numbers to Dataset

When an explicit record counter is required within the data itself, _**N**_ provides the perfect value. By assigning _**N**_ to a new user-defined variable (like `row_number`), we ensure that every observation receives a unique, sequential row index. This new variable is then permanently stored in the output data set.

```
data new_data;
set original_data;
row_number = _N_;
run;
```

Setup: Creating the Sample SAS Data Set

To illustrate these methods clearly and practically, we must first establish the input data set. We will be using a small sample data set named `original_data`. This data set contains 10 records of fictional sports statistics, including team name, points, and rebounds. This setup is crucial for demonstrating how the _**N**_ variable interacts with data structures sequentially during the processing cycle.

The following code block constructs the sample data using the `DATALINES` statement within the SAS data step. After creation, the PROC PRINT procedure is used to display the initial structure of the data set, allowing us to confirm the order and content before any manipulation occurs.

```
/*create dataset*/
```

```
data original_data;
input team $ points rebounds;
datalines;
Warriors 25 8
Wizards 18 12
Rockets 22 6
Celtics 24 11
Thunder 27 14
Spurs 33 19
Nets 31 20
Mavericks 34 10
Kings 22 11
Pelicans 39 23
;
run;

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

The visualization below confirms the structure of the `original_data` set, which contains 10 observations that will serve as our foundation for the subsequent examples demonstrating the utility of _**N**_.

Obs	team	points	rebounds
1	Warriors	25	8
2	Wizards	18	12
3	Rockets	22	6
4	Celtics	24	11
5	Thunder	27	14
6	Spurs	33	19
7	Nets	31	20
8	Maverick	34	10
9	Kings	22	11
10	Pelicans	39	23

Example 1: Filtering for the Initial Record (If _N_ = 1)

This method demonstrates how to efficiently filter a large data set to retain only the very first observation. We leverage the fact that _**N**_ is exactly equal to 1 during the initial iteration of the SAS data step. The use of the explicit OUTPUT statement ensures that only the record meeting the condition is written to the new data set.

The following code applies the conditional logic `if _N_ = 1 then output;`. When this condition is met, SAS processes the record for the Warriors and then moves on. As SAS continues to iterate through the remaining records, the condition `_N_ = 1` is no longer true, and because we used an explicit `OUTPUT` statement, no further records are written to `new_data`.

```
/*create new dataset that contains only the first row*/
data new_data;
set original_data;
if _N_ = 1 then output;
run;

/*view new dataset*/
proc print data=new_data;
```

As confirmed by the output below, the new data set successfully retains only the first row from the original data set, demonstrating the precise and controlled filtering capability offered by checking the _**N**_ variable.

Obs	team	points	rebounds
1	Warriors	25	8

Example 2: Extracting a Subset of Records (If _N_ <= N)

A frequent requirement in data preliminary data exploration is to quickly obtain a sample or preview of the top records in a file. By specifying an upper bound for _**N**_, we instruct SAS to process and output records only until that counter is exceeded. This technique is highly efficient for data subsetting when the desired records are known to be at the beginning of the file.

In this illustration, we aim to select the first five observations. The condition `if _N_ <= 5 then output;` ensures that the OUTPUT statement executes only for the first five iterations of the data step. Once _**N**_ reaches 6, the condition fails, and SAS stops writing records to the `new_data` file,

regardless of the size of the input data set.

```
/*create new dataset that contains first 5 rows of original dataset*/
data new_data;
set original_data;
if _N_ <= 5 then output;
run;

/*view new dataset*/
proc print data=new_data;
```

Upon reviewing the results, we confirm that `new_data` now contains only the first five records from the original data set (Warriors through Thunder), successfully demonstrating the targeted subsetting capability of the _**N**_ variable without requiring complex looping or filtering mechanisms.

Obs	team	points	rebounds
1	Warriors	25	8
2	Wizards	18	12
3	Rockets	22	6
4	Celtics	24	11
5	Thunder	27	14

Example 3: Creating a Permanent Row Number Variable

While _**N**_ is an automatic variable that exists only during the data step execution, it can be permanently captured and stored within the output data set. This is achieved by defining a new variable, `row_number`, and setting its value equal to _**N**_ in every iteration. This simple assignment statement effectively creates a valuable sequential identifier for every observation in the resulting file.

This technique is extremely useful for maintaining the original order of the data, especially after sorting or merging operations, or for creating a temporary primary key. Unlike the previous examples, this method uses _**N**_ for generation rather than filtering, thereby keeping all original records while adding new metadata.

```
/*create new dataset that contains column with row numbers*/
data new_data;
set original_data;
```

```
row_number = _N_;
run;

/*view new dataset*/
proc print data=new_data;
```

The final output confirms that the new data set, generated using the simple assignment statement, includes a column named `row_number`. This column accurately reflects the sequential order in which each record was processed during the data step, ranging from 1 to 10.

Obs	team	points	rebounds	row_number
1	Warriors	25	8	1
2	Wizards	18	12	2
3	Rockets	22	6	3
4	Celtics	24	11	4
5	Thunder	27	14	5
6	Spurs	33	19	6
7	Nets	31	20	7
8	Maverick	34	10	8
9	Kings	22	11	9
10	Pelicans	39	23	10

Conclusion and Further SAS Resources

The **_N_** automatic variable is a foundational concept in SAS programming, granting high-level control over the sequential processing of data within the data step environment. Whether you are extracting the first few records, assigning unique indices, or using it in conjunction with other flow control statements, **_N_** offers a powerful, efficient, and direct approach to data manipulation based on record position.

By effectively implementing the methods detailed above--selecting the first row, subsetting the first N rows, or adding a row counter--you can optimize your SAS code for clarity and performance. Mastery of automatic variables like **_N_** is key to writing robust and efficient SAS programs.

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