How to Easily Perform a One-to-Many Merge in SAS

Authored by

stats writer

November 19, 2025

The MERGE statement in SAS is a versatile tool used to combine data from two or more datasets based on shared identifier variables. A particularly common and powerful application is the **One-to-Many Merge**. This technique is essential when combining a dataset containing unique key identifiers (the "One" side, often master or reference data) with a second dataset where those same keys appear multiple times (the "Many" side, usually transactional or detailed data). Effectively executing this merge requires careful use of the BY statement, which aligns observations based on common values in a specified key variable. Understanding the structure of these One-to-Many Relationships is paramount, as it determines how the resulting combined dataset will be structured, ensuring that every observation from the "Many" dataset correctly inherits the corresponding attributes from the "One" dataset.

Understanding Data Relationships in SAS Merges

Before diving into the code, it is vital to firmly grasp the concept of the one-to-many relationship. This relationship exists when a single record in the primary dataset corresponds to one or more records in the secondary, or transactional, dataset. For instance, if Dataset A holds employee demographics (one row per employee ID) and Dataset B holds monthly sales figures (multiple rows per employee ID), merging them constitutes a **one-to-many operation**. The goal is to append the stable demographic data to every corresponding sales transaction row, thereby enriching the detailed data without creating redundant observations in the primary file. This process is the foundation for joining descriptive metadata to raw transaction volumes.

When performing a **one-to-many merge** using SAS, the system reads observations sequentially, matching them by the variable specified in the BY statement. If the primary dataset contains a unique key (the 'one' side), SAS retains the values from that observation until a new key value is encountered in the 'many' dataset. This mechanism ensures that the single observation's data is correctly propagated down to all matching records in the secondary file, maintaining data integrity during the combination process and allowing for accurate downstream statistical analysis.

The Core Syntax for SAS Merging

To execute a successful one-to-many merge in SAS, the process relies on the fundamental **DATA step** structure combined with the MERGE statement and the crucial BY statement. A prerequisite for this operation is that the datasets must be sorted by the key variable used in the BY statement. While modern SAS versions sometimes handle implicit sorting in simple merges, explicit use of `PROC SORT` is always recommended for robust production code, especially when dealing with large, complex data environments.

The following standard syntax demonstrates the fundamental structure for merging two datasets. Here, we assume `data_one` is the 'one' side (unique keys) and `data_many` is the 'many' side

(duplicate keys), joined via the shared identifier `ID`. This syntax is concise and highly efficient for combining files based on common index values:

```
data final_data;
merge data_one data_many;
by ID;
run;
```

This specific implementation initiates the DATA step, creating a resulting dataset named **final_data**. The process combines observations from **data_one** and **data_many**, matching them based on the common key variable, **ID**. In this merge scenario, **data_one** is required to have only one row per unique **ID** value, while **data_many** contains multiple rows for some or all of the corresponding **ID** values. The MERGE statement handles the retention and propagation of variables from the 'one' side automatically.

Example: Defining the Primary (One) Dataset

To illustrate this concept practically, let us establish two example datasets. First, we define **data_one**, which represents our primary dimension or reference file--in this case, containing information about sales personnel. Crucially, this dataset adheres to the 'one' side of the one-to-many relationship, meaning that the unique identifier (**ID**) appears exactly once for each employee. This uniqueness is what allows the merge operation to accurately propagate the employee attributes to the transactional records without ambiguity.

The following code snippet creates and displays **data_one**, which includes the employee **ID** and their **Gender**. Notice the straightforward definition using the `DATALINES` statement for in-line data entry, followed by `PROC PRINT` to verify the data structure and content:

```
/*create dataset: data_one (The 'One' side)*/
data data_one;
input ID Gender $;
datalines;
1 Male
2 Male
3 Female
4 Male
5 Female
;
run;
```

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

Obs	ID	Gender
1	1	Male
2	2	Male
3	3	Female
4	4	Male
5	5	Female

As shown in the output, the **ID** variable serves as the unique key; each value (1 through 5) occurs only once, confirming its role as the primary, non-repeating dataset in the one-to-many structure.

Example: Defining the Transactional (Many) Dataset

Next, we define **data_many**, which holds transactional information--specifically, sales data recorded by the employees listed in **data_one**. Since employees typically record multiple sales transactions over time, the **ID** variable in this dataset is expected to appear multiple times, fitting the 'many' side of the one-to-many relationship. This dataset provides granular detail, including the **Store** location and the **Sales** amount associated with each specific transaction.

The definition of **data_many** follows a similar structure to the previous dataset creation, utilizing `DATALINES`. However, the data clearly shows duplicated **ID** values. For example, ID 1 has three separate sales entries (one for Store A, one for B, and one for C), confirming that it is the detailed file we intend to enrich with the static employee data from **data_one**. It is essential for the keys in this dataset to be correctly ordered to ensure proper alignment during the subsequent merge.

```
/*create dataset: data_many (The 'Many' side)*/
data data_many;
input ID Store $ Sales;
datalines;
1 A 22
1 B 25
1 C 20
2 A 14
2 B 23
3 A 10
```

```
4 A 15
4 B 29
5 A 16
5 C 22
;
run;

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

Obs	ID	Store	Sales
1	1	A	22
2	1	B	25
3	1	C	20
4	2	A	14
5	2	B	23
6	3	A	10
7	4	A	15
8	4	B	29
9	5	A	16
10	5	C	22

Inspection of the output confirms that each unique **ID** value occurs multiple times across various transaction rows. This validation confirms its role as the transactional dataset requiring enrichment from the primary data source during our merge operation.

Executing the One-to-Many Merge

With both the primary (one) and transactional (many) datasets defined and sorted by the **ID** variable, we proceed to execute the **one-to-many merge**. The syntax is straightforward, relying entirely on the MERGE statement within a standard DATA step. It is crucial to list both datasets in the MERGE statement and specify the linking variable using the BY statement.

During execution, SAS reads one matching group of observations at a time, determined by the **ID** value. For each **ID** group, it first reads the single observation from `data_one`, holding its variables in memory. It then reads and combines that retained data with all subsequent observations

matching that **ID** from `data_many`, effectively duplicating the variables from `data_one` across the multiple transactional rows.

```
/*create new dataset using one-to-many merge*/
data final_data;
merge data_one data_many;
by ID;
run;

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

Reviewing the Merged Output

Upon execution of the DATA step above, SAS produces **final_data**. This resulting dataset successfully combines all the variables from both input files into a single, comprehensive structure. The key variables from `data_one` (specifically **Gender**) are now associated with every corresponding sales record from `data_many`, providing a rich, single source for analysis.

For example, employee ID 1, whose gender is Male (from **data_one**), now has this 'Male' attribute attached to all three of their individual sales transactions in the **final_data** output, creating three identical rows for ID 1, each differing only by the transaction details (Store and Sales). The **one-to-many merge** has effectively enriched the detailed transactional records with descriptive data, allowing analysts to perform calculations like total sales grouped by employee gender.

Obs	ID	Gender	Store	Sales
1	1	Male	A	22
2	1	Male	B	25
3	1	Male	C	20
4	2	Male	A	14
5	2	Male	B	23
6	3	Female	A	10
7	4	Male	A	15
8	4	Male	B	29
9	5	Female	A	16
10	5	Female	C	22

The output confirms that the one-to-many merge successfully produced a new dataset containing all information from both source datasets, perfectly aligned by the common **ID** variable, achieving the desired data integration goal.

Key Takeaways and Further Resources

The **One-to-Many Merge** is an indispensable technique in SAS programming when integrating stable master data with high-volume transactional data. It is crucial to remember that this operation relies fundamentally on the input datasets being ordered by the common key variable specified in the BY statement. Failure to sort the data properly can lead to inaccurate matches, where data from one ID might incorrectly flow into records belonging to a different ID, potentially corrupting the analysis and requiring careful validation of the output.

While the DATA step MERGE statement is the classic SAS method, it is worth noting that modern SAS environments also support SQL syntax via `PROC SQL`. A one-to-many merge achieved in the DATA step is functionally equivalent to an **Inner Join** or **Left Join** in SQL JOIN terms, depending on how missing keys are handled, providing alternative avenues for complex data integration tasks.

For a detailed exploration of all parameters and advanced options available, users should refer directly to the official documentation provided by SAS Institute, which covers edge cases such as dealing with unmatched records (non-matches) and using specialized options like IN= dataset options.

Note: You can find the complete documentation for the SAS MERGE statement online.

The following tutorials explain how to perform other common tasks in SAS, broadening your data management capabilities: