How to Easily Merge Datasets in SAS Using Two Variables

Authored by

stats writer

November 19, 2025

RECOMMENDED CITATION

stats writer (2025). *How to Easily Merge Datasets in SAS Using Two Variables*. PSYCHOLOGICAL SCALES. Retrieved from https://scales.arabpsychology.com/?p=97404

ARABPSYCHOLOGY.COM

The Power of Data Integration in SAS

The **Statistical Analysis System (SAS)** is recognized globally as a robust and powerful software suite designed for advanced analytics, business intelligence, and **data management**. A fundamental requirement in complex data processing is the ability to combine information from disparate sources. This process, known as **data merging**, is efficiently handled within the SAS environment, allowing users to consolidate two or more **datasets** based on common key identifiers, often referred to as **variables**.

Merging data on a single identifier is straightforward, but real-world scenarios frequently necessitate joining records using a composite key--meaning two or more variables must match simultaneously. Utilizing the **MERGE statement**, along with the crucial **BY statement**, SAS enables precise control over this complex joining process. The resulting output is a new, integrated dataset that encompasses all the fields from the original sources, structured according to the defined join criteria.

This technique is indispensable for comprehensive analysis, providing a unified view of organizational data, linking transaction records to customer demographics, or associating experimental results with specific treatment protocols. Mastering the method of merging datasets based on multiple variables ensures data integrity and facilitates accurate comparative statistical analysis.

ARABPSYCHOLOGY.COM

Understanding the SAS MERGE Statement and BY Group Logic

When performing a merge in SAS, two key statements govern the process: the **MERGE statement** itself and the corresponding **BY statement**. The **BY statement** specifies the key variables upon which the records must align. Crucially, when merging on multiple fields, all datasets involved must be sorted by the key variables specified in the BY statement, or utilize indexed datasets, although the former is the most common practice for simple merges.

To achieve a successful merge based on two matching criteria, such as a customer identifier and a store location, we place both fields within the **BY statement**. Furthermore, incorporating the optional **IN= dataset option** allows us to track which records contributed to the match, enabling conditional filtering similar to an SQL JOIN operation.

The following code provides the basic syntax necessary to merge two datasets in SAS, ensuring that records match across two specified variables:

```
data final_data;
merge data1 (in = a) data2 (in = b);
```

```
by ID Store;
if a and b;
run;
```

This specific configuration executes what is functionally equivalent to an **inner join**. It processes the input datasets, **data1** and **data2**, aligning records where both the **ID** and **Store** variables possess identical values. The inclusion of the **IN= dataset options** (`in=a` and `in=b`) creates temporary logical variables (`a` and `b`) that indicate whether a record came from **data1** or **data2**, respectively.

The conditional statement, `if a and b;`, is the filtering mechanism. It instructs SAS to write a record to the output dataset (**final_data**) only if that observation originated from both the first dataset (`a` is true) AND the second dataset (`b` is true). This ensures that only perfectly matched observations, based on the composite key of ID and Store, are retained.

Detailed Example: Preparing and Sorting Input Datasets

To illustrate this powerful merging technique, let us consider a practical scenario involving two datasets related to sales performance. We must first define the structure and content of our source data. Remember that before any SAS merge operation using a **BY statement**, both input datasets must be sorted by the key variables, or **SAS** will issue an error. While we are using the `DATALINES` statement here, in a production environment, you would typically use a `PROC SORT` step prior to merging.

Suppose we have the first **dataset**, **data1**, which contains mapping information for sales associates (identified by **ID**) across various **Store** locations:

```
/*create first dataset: Associate Store Assignments*/
data data1;
input ID Store $;
datalines;
1 A
1 B
1 C
2 A
2 C
3 A
3 B
;
```

ARABPSYCHOLOGY.COM

run;

/*view first dataset*/
title "data1";
proc print data = data1;

data1

Obs	ID	Store
1	1	A
2	1	B
3	1	C
4	2	A
5	2	C
6	3	A
7	3	B

This initial dataset establishes the potential pairings between an employee (ID) and a store (Store). Notice that employee 1 is assigned to three different stores (A, B, C), while employee 2 is assigned to two stores (A, C). This structure highlights the one-to-many relationship often encountered in relational data structures.

Defining the Secondary Dataset (data2)

Next, we introduce the second dataset, **data2**, which holds actual sales figures achieved at various stores by each associate. We aim to combine these two sources to analyze the performance (Sales) only where an official assignment (data1) exists and a sales record (data2) exists for that exact pairing:

/*create second dataset: Sales Figures by ID and Store*/
data data2;
input ID Store $ Sales;
datalines;
1 A 22
1 B 25
2 A 40
2 B 24
2 C 29
3 A 12

ARABPSYCHOLOGY.COM

```
3 B 15
;
run;

/*view second dataset*/
title "data2";
proc print data = data2;
```

data2

Obs	ID	Store	Sales
1	1	A	22
2	1	B	25
3	2	A	40
4	2	B	24
5	2	C	29
6	3	A	12
7	3	B	15

Upon inspection of **data2**, we can observe specific sales entries. For instance, employee 2 has a sales record at store B, but looking back at **data1**, employee 2 was NOT officially assigned to store B (assignments were A and C). Conversely, employee 1 was assigned to store C in **data1**, but there is no corresponding sales record in **data2**. This disparity highlights why a standard merge (full outer join) would include unmatched rows, whereas our goal is an inner join based on the composite key.

Executing the Conditional Merge (Inner Join)

Now that the source datasets are prepared, we can execute the merge operation. We utilize the **MERGE statement** combined with the **IN= dataset option** and a conditional **IF statement** within a single DATA step. This combination ensures that only records that exist simultaneously in both **data1** and **data2**--matching precisely on the composite key (ID and Store)--are written to the final output dataset, **final_data**.

This technique is critical for ensuring data quality, especially when analyzing performance metrics. By enforcing the dual-variable match, we prevent the inclusion of sales records that might be erroneously associated with an employee/store pairing that was not officially authorized or

ARABPSYCHOLOGY.COM

assigned in the administrative dataset.

We use the following **merge statement** to combine the two datasets based on matching values in the **ID** and **Store** columns, explicitly filtering for rows where a value exists in both inputs:

```
/*perform inner join merge*/
data final_data;
merge data1 (in = a) data2 (in = b);
by ID Store;
if a and b;
run;

/*view results*/
title "final_data";
proc print data=final_data;
```

final_data

Obs	ID	Store	Sales
1	1	A	22
2	1	B	25
3	2	A	40
4	2	C	29
5	3	A	12
6	3	B	15

Analyzing the Resulting Dataset (final_data)

The output dataset, **final_data**, contains only those records where the composite key--the combination of **ID** and **Store**--was present in both **data1** and **data2**. Let's compare this outcome to our initial observation of the source data:

Employee 1, Store C: This pairing existed in **data1** but lacked sales data in **data2**. Since `b` was false, this record was excluded.

Employee 2, Store B: This pairing existed in **data2** (Sales=24) but lacked an assignment in **data1**. Since `a` was false, this record was excluded.

All remaining seven records satisfy the condition `if a and b;`, meaning they had an assignment

ARABPSYCHOLOGY.COM

record AND a corresponding sales record. These are the records retained in the final output.

The resulting dataset returns precisely the intersection of the two inputs based on the two required **variables**. This demonstrates the effectiveness of the DATA step MERGE and BY statements in executing multi-variable inner joins, a technique essential for precise data linkage in **SAS** programming.

General Principles of SAS Merging

The methodology used for combining datasets in the SAS DATA step, often referred to as a match-merge, operates fundamentally differently from typical SQL joins. When **SAS** executes a **MERGE statement**, it reads observations sequentially from all listed input datasets, matching them based on the values defined in the **BY statement**. It is crucial to understand that SAS maintains a Program Data Vector (PDV) which holds the current observation being processed.

As SAS cycles through the BY groups, it collects all matching records from the input datasets into the PDV before writing the final record to the output dataset. If an observation exists in one dataset but not the other for the current BY group, the variables unique to the missing dataset are set to missing values in the PDV. The use of the **IN= dataset option** gives the programmer explicit control over this process by creating boolean flags that indicate the presence of data from each source. Without these flags and the subsequent IF filtering, the default behavior of the DATA step MERGE is a full outer join, retaining all records and filling in missing data where appropriate.

The two primary requirements for a successful match-merge are straightforward yet strictly enforced: first, the input datasets must be listed in the **MERGE statement**; second, the datasets must be sorted by the key **variables** specified in the **BY statement**. Failing to sort the data will typically result in a warning or an error, leading to incorrect or unpredictable results, as the PDV cannot properly align the BY groups if the input order is arbitrary.

Alternative Join Types Using IN= Variables

While our primary example focused on achieving an inner join (records present in both datasets), the flexibility provided by the **IN= dataset option** allows SAS programmers to replicate virtually any standard SQL join type simply by adjusting the conditional **IF statement**. This level of control is one of the distinct advantages of using the DATA step MERGE method over procedures like PROC SQL for complex data manipulation.

For instance, executing a **Left Join** (retaining all records from data1 and matching records from data2) only requires testing the presence of the first dataset. The filter would change to `if a;`. Conversely, a **Right Join** (retaining all records from data2) would use the condition `if b;`. These variations are powerful tools for ensuring that specific records are maintained, even if

ARABPSYCHOLOGY.COM

corresponding data is absent in the secondary source.

Furthermore, complex set operations, such as identifying records that are **Unique to data1** (Set Difference) or creating a **Full Outer Join** (which is the default merge behavior if no IF statement is used), can be accomplished using variations of the logical operators (AND, OR, NOT). For example, to find records unique to data1, one would use `if a and not b;`. This capability allows high precision in data integration tasks, ensuring analysts can isolate specific subsets of data for targeted statistical procedures.

Best Practices for Multi-Variable Merging in SAS

When dealing with large-scale projects involving multiple merges based on two or more key variables, adopting specific best practices ensures efficiency, maintainability, and data accuracy. The first and most critical best practice is always verifying the sort order of your input **datasets** immediately before the merge step. If the datasets are extremely large, using the `NOBS=` option during the PROC SORT step can offer performance advantages.

Pre-Sort Verification: Always explicitly use `PROC SORT` on your input datasets using the same sequence of variables listed in the **BY statement**. If you are merging data1 by ID and Store, ensure `proc sort data=data1; by ID Store; run;` is executed first, followed by the same sort for data2.

Naming Conventions: Use descriptive names for your **IN= variables** (e.g., `in=SourceA`, `in=SourceB`) rather than single letters (`a`, `b`) to improve code readability and reduce the chance of errors, especially when merging more than two datasets.

Handling Missing Keys: Carefully consider how missing values in your key variables (ID or Store in our example) should be treated. By default, SAS treats missing character values as the lowest possible value and missing numeric values as the smallest possible number, grouping them together. If missing keys should not match, filtering them out before the merge is highly recommended.

Referencing Documentation: For complex or nuanced merging needs, always refer to the official **SAS MERGE statement documentation**. This ensures that the specific behavior of data step processing is fully understood, particularly regarding how SAS handles variable retention and overwriting during the merge process.

By meticulously following these guidelines, programmers can leverage the powerful combination of the DATA step, the **MERGE statement**, and conditional logic to achieve robust and reliable data integration based on two or more **variables** within the **SAS** environment.

ARABPSYCHOLOGY.COM