SAS: Merge Datasets Based on Two Variables

Authored by

Mohammed looti

November 15, 2025

Data manipulation forms the bedrock of rigorous data analysis, and combining information from disparate sources is arguably the most frequent operation. In the world of SAS programming, the **MERGE statement** is the essential utility for integrating datasets. This detailed guide focuses on a powerful and specific application: merging two datasets that require matching values across two distinct variables. Mastering this technique is vital for analysts dealing with complex relational data structures where a single identifying column is insufficient for accurate record linkage and integrity.

The necessity for a precise merge operation arises when data integrity demands a **composite key**--a combination of two or more variables that, together, uniquely identify an observation or record. Relying on a single identifier in such scenarios can lead to ambiguous or incorrect joins, corrupting the analysis. By specifying two matching variables, the two-variable merge provides a robust and unambiguous mechanism for integrating information, ensuring that only perfectly aligned records are joined, thereby maintaining the fidelity of the resulting dataset.

The following foundational syntax illustrates the standard approach for executing an inner join in SAS using the DATA step. This structure is designed to combine records only where matching values exist in both specified key variables across the two input datasets. This method is highly effective for isolating the intersection of two data sources, yielding a clean and highly precise integrated dataset.

```
data final_data;
merge data1 (in = a) data2 (in = b);
by ID Store;
if a and b;
run;
```

Understanding the Core SAS MERGE Syntax

The syntax presented above represents the cornerstone of conditional merging within the SAS environment, specifically structured to achieve an inner join. Every command within this structure serves a distinct purpose in dictating how the datasets are processed, combined, and filtered to produce the final output. Dissecting these components is crucial for understanding the logic of composite key merging.

The process begins with the DATA step, initiated by the command `data final_data;`, which establishes a new dataset where the results of the combination will reside. The core operation is defined by the MERGE statement: `merge data1 (in = a) data2 (in = b);`. This statement names the input datasets, `data1` and `data2`, and crucially introduces the `in=` options. These options create temporary Boolean indicator variables, `a` and `b`, often called flags. The flag `a` is set to true (1) if the current observation originates from `data1`, and `b` is set to true (1) if it originates

from `data2`; otherwise, they are false (0). These flags are instrumental in controlling the type of join performed.

Immediately following the MERGE statement is the BY statement: `by ID Store;`. This command is fundamentally important as it instructs SAS which variables must be used to align observations between the input datasets. A critical prerequisite for any successful merge operation utilizing the BY statement is that both `data1` and `data2` must be sorted in ascending order by the specified key variables (`ID` and `Store`). If this pre-sorting step is neglected, SAS will either issue an error message or, worse, produce logically incorrect results. The BY statement effectively tells SAS to pair records only when the combination of values for `ID` and `Store` is identical in both sources.

The conditional statement `if a and b;` functions as the filtering mechanism that transforms a standard merge into an inner join. Since the temporary variable `a` is true only when a record is present in `data1`, and `b` is true only when a record is present in `data2`, this condition ensures that only those observations possessing a match in **both** `data1` and `data2` are written to the resulting `final_data` dataset. This exclusion process is the defining characteristic of an inner join, guaranteeing that the output contains only shared records based on the composite matching keys. The DATA step concludes with the `run;` statement, executing the program and generating the merged output dataset.

Practical Example: Preparing Your Datasets

To effectively illustrate the utility of the multi-variable MERGE statement in SAS, we will establish a realistic business scenario centered on sales tracking. We will construct two distinct datasets: the first detailing associate assignments to specific store locations, and the second containing transactional records of sales figures.

Consider the initial dataset, named `data1`, which maps individual sales associates (identified by `ID`) to their assigned store locations (`Store`). It is noteworthy that a single associate may be assigned to several stores, making the combination of `ID` and `Store` necessary to define the assignment uniquely. We use the DATALINES statement to input this assignment data directly:

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

```
2 C
3 A
3 B
;
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

After creating `data1`, we employ PROC PRINT to visualize its structure, confirming that the unique combinations of `ID` and `Store` are correctly recorded. Subsequently, we introduce the second dataset, `data2`, which contains the quantitative data--the actual `Sales` figures. This dataset records sales attributed to a specific associate (`ID`) at a particular store (`Store`). It is critical to recognize that not every assignment in `data1` may have a corresponding sale in `data2`, and vice versa. This natural data discrepancy is precisely what necessitates the use of a conditional, two-variable merge to ensure accurate data alignment based on the combined composite key.

```
/*create second dataset*/
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
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

After verifying `data2`, we confirm that both datasets share the linking columns: `ID` and `Store`. These two variables will be utilized as our composite key. The objective of the upcoming merge operation is to successfully combine the associate-store assignments from `data1` with the corresponding sales figures from `data2`, focusing exclusively on those records where the associate ID and the store location match exactly in both source files.

Executing the Two-Variable MERGE Operation

With `data1` and `data2` prepared and their composite key variables identified, the next phase involves applying the MERGE statement to integrate the information. As previously established, we are performing an inner join, meaning the resulting dataset will only contain observations where the values for both `ID` and `Store` variables align perfectly across both input datasets.

The following SAS code block executes the merge logic. It is critical to note the strict ordering imposed by the BY statement and the filtering mechanism provided by the **`IF a AND b;`** condition, which ensures the inner join behavior.

```
/*perform 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;
```

Upon execution, SAS iteratively processes `data1` and `data2`, aligning them observation by observation based on the shared `ID` and `Store` values. For any record where the `ID`/`Store` combination is found in `data1` (setting flag `a` to true) AND the exact same combination is found in `data2` (setting flag `b` to true), a new observation is written to `final_data`. This output record contains all variables present across both original sources. The deliberate use of the **`in=`** option followed by the conditional **`if`** statement is the standard and most robust method in SAS for defining and controlling specific relational join types.

Interpreting the Merged Output

The final step in the data integration process is the meticulous inspection of the generated `final_data` dataset to confirm that the merge operation performed according to the inner join logic. The output displayed by PROC PRINT for the merged dataset is critical for this verification:

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

A careful review of `final_data` confirms that only records where both `ID` and `Store` variables matched across `data1` and `data2` were retained. For instance, the record for `ID=1` at `Store=C` existed in `data1` (the assignment list), but it did not have a corresponding sales entry in `data2`. Because the **`IF a AND b;`** condition requires a match from both sources, this particular record was

correctly excluded from the `final_data` output. Similarly, any sales record in `data2` that lacked an assignment entry in `data1` would also have been excluded.

This outcome perfectly illustrates the function of an inner join: it yields the common intersection of the two datasets based on the defined composite key columns. Each row in the `final_data` is now fully integrated, containing the associate ID, the store location, and the corresponding sales amount, but only for those associate-store pairs that were actively present in both the assignment roster and the sales ledger. This results in a highly filtered, relevant, and accurate view of the transactional data.

Key Considerations for Multi-Variable Merges

While the two-variable MERGE statement is a powerful data integration tool, optimizing its use requires adherence to several best practices to avoid common errors. The single most critical requirement is ensuring that all input datasets are correctly sorted by the variables listed in the BY statement (`ID` and `Store`, in this case). SAS merges data sequentially based on the order of these key variables; therefore, improper sorting will render the results inaccurate or cause the DATA step to terminate with an error. The sequence of variables specified in the BY statement must also match the sort order of the datasets.

A second vital consideration is the consistency of data types for the matching variables. SAS mandates that all variables designated in the BY statement (e.g., `ID` and `Store`) must share the same data type--either numeric or character--across every input dataset. Any discrepancy in data type will prevent accurate matching and likely result in runtime errors. Analysts must also be mindful of missing values within BY variables; SAS treats missing values as the lowest possible value, which can unintentionally affect the matching sequence and logic, often requiring explicit handling or imputation prior to merging.

Finally, while this tutorial focused exclusively on achieving an inner join via the `IF a AND b;` condition, the SAS DATA step merge is versatile. By simply modifying the conditional statement (e.g., using `IF a;` for a left join, or omitting the `IF` condition entirely for a full outer join if the variables are unique), different integration outcomes can be achieved. For organizations managing extraordinarily large datasets, alternative high-performance joining methods, such as utilizing SQL JOINs via PROC SQL, should also be evaluated for improved efficiency and scalability.

Additional Resources for SAS Data Manipulation

To solidify your expertise in SAS data manipulation, particularly regarding the nuances of combining and merging datasets, we recommend consulting the official, authoritative documentation. These resources offer comprehensive details and advanced insights into SAS programming capabilities.

SAS MERGE Statement Documentation: For a deep dive into all functional aspects of the `MERGE` statement, including various join types, handling non-matches, and advanced options, refer to the official SAS documentation on the MERGE statement. This link provides syntax rules and detailed examples.

SAS BY Statement Documentation: The proper use of the `BY` statement is foundational to the DATA step merge. Detailed information covering its usage, sorting requirements, and interaction with other procedures can be found in the official SAS documentation for the BY statement.

SAS DATA Step Programming: Expanding your knowledge of fundamental data creation and modification in SAS is crucial. The SAS DATA Step documentation offers extensive guidance on constructing, modifying, and managing datasets, including conditional processing and iterative programming logic.

Understanding SQL JOINs: While the guide focused on the DATA step merge, SQL JOINs (accessible in SAS via **PROC SQL**) present an alternative, often more familiar approach for relational data integration. Understanding standard SQL join types (INNER, LEFT, RIGHT, FULL) provides valuable flexibility for complex data combination tasks.