How to Merge Datasets in SAS Keeping Only Records Unique to Dataset A

Authored by

stats writer

November 19, 2025

The MERGE statement in SAS is a fundamental tool for combining two or more datasets based on common key variables. While simple merging typically aims for a full or matched combination, advanced data manipulation often requires identifying differences between data sources. Specifically, the "if a not b" logical expression provides a powerful method for performing what is often referred to in database theory as an Anti-Join or a set difference operation.

This technique allows you to precisely filter the output, ensuring that only those observations found in the primary dataset (A) that lack a corresponding match in the secondary dataset (B) are retained. Understanding how to implement this specific conditional merge is crucial for tasks such as identifying missing records, tracking discrepancies between databases, or ensuring compliance where data must exist exclusively in one source. This method leverages the powerful features within the DATA Step to assign temporary variables that track the presence of observations in each input file.

This detailed guide will walk through the mechanism of the conditional MERGE statement, demonstrating the precise syntax required to execute this type of set difference operation effectively in SAS, followed by a practical, step-by-step example using sample sales data. The core principle revolves around using the special MERGE statement option, `IN=`, which acts as a flag to indicate the source of the currently processed record.

The Mechanics of the IN= Option in SAS

To execute a conditional merge based on presence or absence, we must introduce the concept of the `IN=` option, which is specified within the parentheses following each dataset name in the MERGE statement. When SAS processes the DATA Step and performs the merge, the `IN=` option creates a temporary, binary variable for each input dataset. This variable holds a value of **1 (True)** if the observation currently being built in the resulting dataset originated from that specific input source, or **0 (False)** otherwise.

For instance, if we specify `data1 (in = a)` and `data2 (in = b)`, two temporary variables, `a` and `b`, are created. During the iterative processing of the DATA Step, if the current observation contains data derived from **data1**, variable `a` will be 1. If it contains data from **data2**, variable `b` will be 1. This mechanism is critical because the merge operation itself, by default, attempts to match all records, potentially creating records that only exist in A, only exist in B, or exist in both A and B.

By leveraging these temporary variables, we can apply conditional logic using the standard **IF statement** to filter the output dataset. The goal of the "if a not b" logic is to isolate those records whose key value was present in the first dataset (A) but completely absent from the second dataset (B). This results in a highly targeted subset of the data, achieving the desired set difference operation efficiently within the SAS environment.

Syntax for the Set Difference Merge

The foundational structure for performing an anti-join or set difference merge in SAS requires four essential components: the DATA Step definition, the MERGE statement with the `IN=` options, the **BY statement** specifying the common key, and the conditional **IF statement** that defines the filtering logic.

You can use the following basic syntax to merge two datasets in SAS and only return the rows where a value exists in the first dataset (A) and not the second dataset (B). This structure ensures clarity and reproducibility in your data management workflow:

```
data final_data;
merge data1 (in = a) data2 (in = b);
by ID;
if a and not b;
run;
```

In this specific implementation, **data1** is designated as the primary source (A) via the temporary variable `a`, and **data2** is the secondary source (B) via `b`. The conditional statement `if a and not b;` instructs SAS to keep only those observations where the flag `a` is true (meaning the ID was present in **data1**) and the flag `b` is false (meaning the corresponding ID was absent from **data2**). This effectively executes the required set difference, returning only the unique rows of the first dataset relative to the second.

Detailed Example Setup: Creating Sample Datasets

To illustrate this functionality clearly, let us establish two sample datasets containing information about sales associates. The goal is to identify sales associates listed in the demographic dataset (**data1**) who do not yet have corresponding sales figures recorded in the sales dataset (**data2**).

The first dataset, **data1**, contains basic demographic information like employee **ID** and **Gender**. The second dataset, **data2**, contains the employee **ID** and their corresponding **Sales** figures. Note that the ID values overlap partially but are not identical across both sources. This overlap is precisely what the conditional merge is designed to resolve.

```
/*create first dataset*/
data data1;
input ID Gender $;
datalines;
1 Male
```

2 Male
3 Female
4 Male
5 Female
;
run;

```
title "data1";
proc print data = data1;

/*create second dataset*/
data data2;
input ID Sales;
datalines;
1 22
2 15
4 29
6 31
7 20
8 13
;
run;

title "data2";
proc print data = data2;
```

Upon reviewing the print output (represented by the image below), we can clearly see the structure of the two input files. **data1** contains IDs 1 through 5, while **data2** contains IDs 1, 2, 4, 6, 7, and 8. The objective of our "if a not b" merge is to identify IDs 3 and 5, as these exist only in **data1** and not in **data2**, indicating missing sales records for those employees.

data1

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

data2

Obs	ID	Sales
1	1	22
2	2	15
3	4	29
4	6	31
5	7	20
6	8	13

Standard Merge (Full Outer Join) vs. Conditional Merge

Before executing the conditional merge, it is beneficial to understand the default behavior of the MERGE statement without any conditional logic. When the MERGE statement is used only with a **BY statement**, SAS performs what is conceptually similar to a full outer join in SQL, although the precise handling of variables differs slightly.

In a standard merge, SAS combines observations that share common key values (**ID**). For unmatched records, variables unique to one dataset are carried over, while variables from the missing dataset are filled with missing values (e.g., `.` for numeric variables like **Sales** or a blank for character variables like **Gender**). The resulting output includes all observations present in either **data1** or **data2**, which can sometimes lead to a much larger dataset than desired if the goal is only to identify discrepancies.

Consider the following standard merge code:

```
/*perform standard merge (Full Outer Join behavior)*/
data final_data;
```

```
merge data1 data2;
by ID;
run;

/*view results*/
title "final_data: Standard Merge Output";
proc print data=final_data;
```

As illustrated below, the standard merge returns eight observations, covering all IDs from both input files. Notice that IDs 3 and 5 have missing values for **Sales**, and IDs 6, 7, and 8 have missing values for **Gender**. If the objective is simply to find those records that are unique to **data1**, this standard output requires additional filtering steps.

final_data

Obs	ID	Gender	Sales
1	1	Male	22
2	2	Male	15
3	3	Female	.
4	4	Male	29
5	5	Female	.
6	6		31
7	7		20
8	8		13

Implementing the "If A Not B" Logic for Set Difference

To move beyond the default full outer join behavior and isolate the records that exist exclusively in **data1**, we must utilize the `IN=` statements to create our conditional flags. This allows the DATA Step to precisely track the origin of the matching key variables during the iterative merge process.

We redefine the MERGE statement to include `(in = a)` for **data1** and `(in = b)` for **data2**. Crucially, we then apply the logical filter `if a and not b;`. This condition is read by SAS as: "Only output the observation if the ID key was successfully read from **data1** (a=1) AND the ID key was NOT successfully read from **data2** (b=0)."

This construction is the definitive method for executing a left anti-join in the DATA Step. It efficiently processes the merge and filters the output simultaneously, preventing the creation of unnecessary intermediate variables or the need for a subsequent `PROC SQL` step to achieve the same result. The resulting dataset will only contain the ID and demographic information for those employees who have an entry in **data1** but are missing from **data2**.

Here is the code block demonstrating the use of `IN=` variables and the filtering logic:

```
data final_data;
merge data1 (in = a) data2 (in = b);
by ID;
if a and not b;
run;

/*view results*/
title "final_data: Conditional Merge Output (If A Not B)";
proc print data=final_data;
```

Observe the resulting output shown in the image below. Only IDs 3 and 5 are returned. These are precisely the employees listed in the demographics dataset (**data1**) for whom no corresponding sales records exist in **data2**. All observations that had a match (IDs 1, 2, 4) or observations that existed only in **data2** (IDs 6, 7, 8) have been successfully excluded by the conditional logic.

final_data

Obs	ID	Gender	Sales
1	3	Female	.
2	5	Female	.

This clean and targeted output demonstrates the power of the "if a not b" construction. Notice that only the rows where a value exists in the first dataset and not the second dataset are returned, fulfilling the requirement for a set difference operation.

Advanced Applications and Related Set Operations

The "if a not b" logic is just one permutation of conditional filtering possible with the `IN=` option during a MERGE statement. Data analysts frequently require other set operations, which can be achieved through minor modifications to the conditional **IF statement**:

Intersection (Inner Join Equivalent): To keep only observations that exist in **both** datasets (A and B), you would use: `if a and b;`. This is useful for combining matched records only.
Right Anti-Join (If B Not A): To keep only observations that exist exclusively in the second dataset (B) but not in the first dataset (A), you would use: `if b and not a;`. In our example, this would return IDs 6, 7, and 8.
Symmetric Difference (Full Outer Anti-Join): To keep observations that exist in A **or** B, but not in both, you would use: `if a xor b;` or `if (a and not b) or (b and not a);`. This identifies all mismatching keys.

Mastery of these logical constructs ensures that complex data comparison and reconciliation tasks can be managed entirely within the DATA Step, optimizing performance and maintaining a high level of code readability. This is particularly valuable when working with very large datasets where efficiency is paramount.

Conclusion and Key Takeaways

The conditional use of the MERGE statement in SAS, coupled with the temporary binary variables generated by the `IN=` option, provides an elegant and powerful solution for performing set difference operations, specifically the "if a not b" requirement. This methodology is indispensable for data validation, discrepancy reporting, and ensuring data integrity across multiple sources.

By defining indicator variables (`a` and `b`) and applying simple Boolean logic (`if a and not b;`), SAS users can precisely control which subset of combined data is retained in the final output. This avoids the overhead of generating a full outer join dataset only to filter it later, streamlining the process of identifying records unique to the primary source file. Always ensure your input datasets are sorted by the common **BY statement** variable before executing the merge, as this is a prerequisite for accurate results in the DATA Step environment.

Note: You can find the complete documentation for the SAS MERGE statement and the `IN=` option on the official SAS documentation portal.