How to Use the IN= Option in a SAS Merge Statement

Authored by

stats writer

November 19, 2025

The **SAS** System is a robust statistical software package widely utilized across industries for advanced analytics, visualization, and specialized **data manipulation** tasks. One of its fundamental and most powerful features for combining information is the **Merge statement**. While a simple merge combines rows, the critical `IN=` option provides precise control over which observations are included in the resulting output.

Specifically, using the `(in=a)` syntax within a **Merge statement** designates a temporary Boolean variable (in this case, `a`) that flags whether a matching observation originated from the corresponding input **data set**. This mechanism is essential for performing conditional joins--allowing developers to compare existence across sources and accurately combine records or update existing values, mimicking different types of SQL joins (like left, right, or inner joins) within the **SAS** Data Step environment.

The Power of the IN= Option in Data Integration

When you merge two **data sets** in **SAS**, the default behavior produces a full outer join, retaining all rows from both inputs and marking missing values where matches don't exist. However, the `IN=` option allows for selective inclusion. By assigning a temporary variable to each input dataset (e.g., `in=a` or `in=b`), you create flags that are set to `1` (True) if the observation comes from that specific source, or `0` (False) if it does not.

This powerful technique is primarily utilized in conjunction with an `IF` condition following the **Merge statement**. By evaluating the Boolean flags, you dictate which subset of the merged data should be written to the final output **data set**. Mastering the `IN=` option is fundamental for efficient and precise data restructuring.

Conditional Merging: Selecting Rows Based on Dataset Existence

Below we explore the three most common applications of the `IN=` statement, corresponding to Left Joins, Right Joins, and Inner Joins, demonstrating how to precisely control which records survive the merge process. For all methods, we assume merging `data1` and `data2` based on a common key, `ID`.

Method 1: Returning Observations Exclusively from the First Dataset (in = a)

This method replicates a **Left Outer Join**. It ensures that every record found in the first specified dataset (here, `data1`) is included in the final result, regardless of whether a matching `ID` exists in `data2`. Any records exclusive to `data2` will be discarded.

data final_data;

```
merge data1 (in=a) data2;
by ID;
if a;
run;
```

In this particular example, the `(in=a)` option creates the temporary variable `a` for `data1`. The subsequent `if a;` statement filters the merged output, retaining only the rows where a corresponding value was present in **data1**.

Method 2: Selecting Observations from the Second Dataset (in = b)

This technique functions as a **Right Outer Join**. It prioritizes the observations from the second input dataset (`data2`), ensuring that all records from `data2` are preserved. Rows that exist only in `data1` will be dropped from the output.

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

Here, the `(in=b)` option is assigned to `data2`. The `if b;` condition ensures that the final dataset contains rows only if a matching observation was found in **data2**, allowing for efficient subsetting based on the second source file.

Method 3: Performing an Inner Join (Requiring Existence in Both Datasets)

To achieve an **Inner Join**, we require observations to exist in both input datasets simultaneously. This is the most restrictive type of join and is achieved by applying the `IN=` option to both data sets and then requiring both flags to be true (`a AND b`).

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

This powerful syntax merges the datasets called **data1** and **data2** and uses the combined conditional statement `if a and b;` to ensure that only rows where the joining value exists in *both* **data1** and **data2** are retained. This is critical for intersection analysis.

Setting Up the Example Datasets

To fully illustrate these methods, we will create two simple example **data sets**, `data1` and `data2`, which we will subsequently merge using the different `IN=` options. Notice that these datasets share some `ID` values (1, 2, 4) but also contain unique `ID`s, which will highlight the differences between the join types.

```
/*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;
```

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

Detailed Implementation: Example 1 - The Default Merge (Full Outer Join)

Before demonstrating the conditional joins, let's observe the standard behavior of the **Merge statement** without any `IN=` options. This process joins `data1` and `data2` based on the shared `ID` column and returns all possible rows, resulting in a **Full Outer Join**.

```
/*perform merge*/
data final_data;
merge data1 data2;
by ID;
run;

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

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

As illustrated by the output, all rows from both datasets are returned. Where a match is not found (e.g., ID 3 and 5 from `data1` lack Sales data, and ID 6, 7, and 8 from `data2` lack Gender data), the corresponding variable fields are filled with missing values. This behavior is often too inclusive for targeted **data manipulation**.

Detailed Implementation: Example 2 - Filtering by the First Dataset (in = a)

To execute a **Left Join** and ensure that all records from `data1` are present, we use the `(in=a)` option coupled with the `if a;` filter. This method is crucial when the first dataset is considered the master list and we only want to pull supplemental information from the second dataset where available, discarding any records unique to the second source.

```
/*perform merge*/
data final_data;
merge data1 (in = a) data2;
by ID;
if a;
run;

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

final_data

Obs	ID	Gender	Sales
1	1	Male	22
2	2	Male	15
3	3	Female	.
4	4	Male	29
5	5	Female	.

Observe the resulting output: IDs 1, 2, 3, 4, and 5 are retained because they all originated from the first dataset (`data1`). IDs 6, 7, and 8 from `data2` are excluded. For IDs 3 and 5, missing values are still present for the Sales column, as no match was found in `data2`, consistent with a Left Join.

Detailed Implementation: Example 3 - Filtering by the Second Dataset (in = b)

Conversely, to perform a **Right Join**, we use the `(in=b)` option assigned to `data2`, followed by the `if b;` filter. This returns only the rows where the joining key exists in `data2`. This pattern is ideal if `data2` represents transactional data and `data1` provides descriptive attributes that should only be included if the transaction exists.

```
/*perform merge*/
data final_data;
merge data1 data2 (in = b);
by ID;
if b;
run;

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

final_data

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

The resulting **data set** contains only IDs 1, 2, 4, 6, 7, and 8--the IDs present in `data2`. Notice that IDs 6, 7, and 8 now have missing values for the Gender column, as they had no corresponding record in `data1`. IDs 3 and 5, which were unique to `data1`, have been successfully filtered out.

Detailed Implementation: Example 4 - Filtering for Intersection (Inner Join)

The most precise filtering is achieved using the **Inner Join**, requiring that a matching `ID` exists in both `data1` and `data2`. This is accomplished by setting the temporary variables `a` and `b` and conditioning the output on `if a and b;`. This type of join is essential for analyses that require complete data integrity across all sources.

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

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

final_data

Obs	ID	Gender	Sales
1	1	Male	22
2	2	Male	15
3	4	Male	29

Only the common IDs (1, 2, and 4) are returned in this final, streamlined dataset. All other records, unique to either `data1` or `data2`, have been excluded. This demonstrates the effectiveness of the `IN=` option for achieving highly specific data subsets during the **Merge statement** process in **SAS**.

Conclusion and Further Resources

The ability to control merges using the `IN=` option is a fundamental skill for any **SAS** programmer performing complex **data manipulation**. Whether you need a simple Left Join or a focused Inner Join, these Boolean flags provide the necessary precision to manage data flow and integrity within the Data Step.

For more detailed technical specifications, you can find the complete documentation for the **SAS Merge statement** on the official website.

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

SAS Tip 1: How to Calculate Quartiles using PROC UNIVARIATE
SAS Tip 2: Using the PROC MEANS Statement to Summarize Data
SAS Tip 3: How to Perform a Two Sample T-Test in SAS