SAS: Use (in=a) in Merge Statement

Authored by

Mohammed looti

November 15, 2025

When performing complex data preparation or integration tasks in SAS, combining information from multiple sources is routine. The **MERGE statement** within the DATA step is the primary mechanism for this process. While a standard merge performs a full outer join by default, advanced control over observation selection is often necessary to ensure data integrity and analytical precision. This control is facilitated by the powerful, yet sometimes overlooked, **IN= option**. Understanding how to use the **IN= option** allows developers to mimic sophisticated SQL join operations, enabling precise filtering during the merging process.

The core functionality of the **IN= option** lies in its ability to create a temporary, Boolean (binary) variable for each input dataset. If you specify `data1 (in=a)`, the variable named 'a' will be generated. This variable holds the value 1 (True) if an observation for the current BY-group exists in that specific dataset, and 0 (False) otherwise. By leveraging this indicator variable in combination with IF statements, you can dictate which records are written to the output dataset, thus achieving various join types, such as inner, left, or right joins. This technique is fundamental for accurate and efficient data integration in SAS programming.

The Mechanism of the IN= Option in SAS MERGE

The **MERGE statement** is fundamentally designed to combine observations from two or more pre-sorted SAS datasets into a single resulting dataset. It matches records based on one or more variables specified in the BY statement. However, without explicit filtering, the merge operation retains all records, resulting in a full outer join where non-matching values are represented by missing system values. While this default behavior is often acceptable, it can lead to unnecessarily large or incomplete output datasets if only the intersection or records originating from a single source are required.

This is where the **IN= option** provides its critical value. When you append the **IN= option** directly to a dataset name within the **MERGE statement**--for example, `Master (IN=m)`--you instruct SAS to create a temporary variable (`m` in this case) that exists only for the duration of that DATA step. This temporary variable serves as a flag, indicating the presence (1) or absence (0) of a record from that specific input source for the current grouping defined by the BY variables.

By defining these indicator variables, the programmer gains explicit control over the output process. Instead of relying solely on the automatic outputting of the **MERGE statement**, a subsequent conditional statement (like `IF a;` or `IF a AND b;`) can be used to filter the records. This ability to selectively write observations based on their source presence allows for the powerful implementation of specific SQL join types directly within the SAS environment, streamlining analytical workflows and ensuring that only relevant data points are carried forward.

Implementing Left Outer Join (IN=a)

The left join is one of the most common data integration requirements, where the goal is to retain every record from the primary, or "left," dataset, augmenting it with matching information from the secondary, or "right," dataset. In SAS, this is achieved by attaching the **IN= option** to the primary dataset and then using a simple IF statement to filter for the presence of the record in that dataset. This technique guarantees that the final output dataset will contain a record for every observation found in the primary source.

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

In the code above, `data1` is designated as the primary source using `(in=a)`. The variable `a` is set to 1 whenever an observation exists in `data1` for the current `ID` (the BY variable). The subsequent conditional logic, `if a;`, acts as the filter, ensuring that only records where `a` is true are written to the output dataset `final_data`. This perfectly replicates a left join operation; records unique to `data1` are kept, with missing values filled for variables originating from `data2`, while records unique to `data2` are discarded entirely.

Implementing Right Outer Join (IN=b)

Conversely, a right join focuses on preserving all observations from the secondary, or "right," dataset (`data2` in a two-way merge), while integrating corresponding information from the primary dataset (`data1`). This approach is valuable when the second dataset holds the critical population or transaction records that must be retained regardless of whether a match exists in the first dataset. To execute a right join in SAS, the **IN= option** is simply assigned to the second dataset listed in the **MERGE statement**.

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

By specifying `data2 (in=b)`, the temporary indicator variable `b` is created, which flags records present in `data2`. The filter `if b;` then ensures that only records present in `data2` are outputted. This results in an outcome equivalent to a right join: all records from `data2` are preserved, and

variables originating from `data1` are set to missing if no match is found on the BY variables. This flexibility allows analysts to quickly pivot their merging strategy based on the dataset hierarchy required for their analysis.

Implementing Inner Join (IN=a and IN=b)

When the analytical requirement is to analyze only those observations that have complete data across all contributing datasets, an inner join is necessary. This operation selects only the intersection of the input datasets--records where the matching key (or BY variable) exists in every source. In SAS, achieving a precise inner join requires assigning the **IN= option** to *all* datasets involved in the merge.

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

By defining both indicator variables, `a` and `b`, the conditional logic becomes `if a and b;`. This rigorous condition ensures that an observation is written to `final_data` only if the corresponding `ID` was present in both `data1` and `data2`. This method is indispensable for ensuring the completeness and validity of merged records, particularly in auditing or reporting contexts where missing data from any source is unacceptable. This approach is the cleanest way to perform an inner join using the **MERGE statement**.

Practical Demonstration with Sample Datasets

To fully grasp the selective power offered by the **IN= option**, we will utilize two small sample SAS datasets, `data1` and `data2`. These datasets have intentionally overlapping and unique `ID` values, which will clearly illustrate how different filtering conditions based on the temporary indicator variables affect the final merged output. Observing the behavior across full, left, right, and inner joins is crucial for mastering data manipulation in SAS.

```
/*create first dataset: Employee Demographics*/
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: Sales Performance*/
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

As shown, `data1` contains records for IDs 1 through 5, while `data2` contains IDs 1, 2, 4, and 6 through 8. The common IDs are 1, 2, and 4. IDs 3 and 5 are exclusive to `data1`, and IDs 6, 7, and 8 are exclusive to `data2`. These differences provide the perfect testbed for demonstrating how filtering based on the indicator variables (a and b) allows for precise control over which records are kept in the final merged dataset.

Example 1: Full Outer Join (Default MERGE Behavior)

Before applying any filtering, it is instructive to observe the default action of the **MERGE statement** when no conditional logic or **IN= option** is utilized. This standard operation results in a full outer join, including every record from both input datasets. Where an observation is found in one dataset but not the other, the corresponding variables from the non-matching source are automatically populated with missing values, thus preserving all unique key values.

```
/*perform merge without filtering*/
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

The output confirms that the default merged dataset retains all records (IDs 1 through 8). Notice how IDs 3 and 5 have missing `Sales` values (from `data2`), and IDs 6, 7, and 8 have missing `Gender` values (from `data1`). While this provides a comprehensive view, subsequent examples will demonstrate how the **MERGE statement** can be refined using indicators to achieve more targeted results, excluding these non-matching records when appropriate.

Example 2: Left Join (Filtering with IN=a)

In this example, we apply the left join logic established earlier. We require all records from `data1` and only the matching records from `data2`. By associating the indicator variable `a` with `data1`, we explicitly check for the presence of the record in the primary source using the `if a;` statement. This is the canonical method for performing a left join in SAS data steps, ensuring that the final dataset's population mirrors that of the first input dataset.

```
/*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	.

The result clearly demonstrates the effect of the left join filter. Only IDs 1 through 5, which originated in `data1`, are present. IDs 6, 7, and 8 (unique to `data2`) have been excluded. For IDs 3 and 5, which had no corresponding match in `data2`, the `Sales` variable remains missing. This confirms the utility of `if a;` for selectively retaining records based on their presence in the first merged dataset.

Example 3: Right Join (Filtering with IN=b)

Next, we execute a right join, requiring us to focus the filter on the second dataset, `data2`. By defining `(in=b)` on `data2` and using the conditional logic `if b;`, we retain every record found in `data2`, regardless of whether a match exists in `data1`. This technique is essential when `data2` represents a complete list of required entities, such as transactions or events, that need to be augmented by static information from `data1`.

```
/*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 output shows all IDs from 1, 2, 4, 6, 7, and 8, which are the unique identifiers present in `data2`. IDs 3 and 5 (exclusive to `data1`) have been successfully filtered out. For the IDs 6, 7, and 8, which are unique to `data2`, the `Gender` variable is missing. This outcome precisely matches the behavior of a right join, showcasing the power of linking the indicator variable to the second dataset for selective record retention.

Example 4: Inner Join (Filtering with IN=a and IN=b)

Finally, we implement the inner join. This operation will produce a dataset containing only those observations where the `ID` value exists in both `data1` and `data2`. We achieve this by using **IN= options** for both datasets and applying the conditional logic `if a and b;`.

```
/*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

The resulting dataset, `final_data`, now contains only IDs 1, 2, and 4. These are the only records that satisfy the conditional logic of being present in both `data1` (where `a=1`) and `data2` (where `b=1`). All unique records have been successfully excluded, resulting in a clean, complete inner join. This demonstration underscores how the strategic use of indicator variables within the **MERGE statement** grants the SAS programmer absolute control over complex data linkage requirements.

Conclusion and Further Learning

Mastering the **MERGE statement** and its associated **IN= option** is essential for advanced data integration in SAS. By utilizing the temporary indicator variables generated by the **IN= option**, programmers can precisely replicate common SQL join operations, ensuring that only the observations meeting specific criteria are included in the final dataset. This selective merging capability leads to more efficient processing and higher data quality in analytical projects. For deeper dives into the syntax and advanced applications of data steps, consulting the official SAS documentation is highly recommended.

Additional Resources

To deepen your understanding of SAS programming and data manipulation techniques, consider exploring the following tutorials:

How to Perform Other Common Data Preparation Tasks in SAS