How to Easily Perform an Inner Join in SAS: A Step-by-Step Guide

Authored by

stats writer

December 1, 2025

An inner join in SAS is a fundamental database operation used to combine rows from two or more datasets based on common, matching values found in a designated key variable. This operation is highly selective; it exclusively returns the records that possess corresponding entries in all tables involved in the join. By focusing solely on the intersection of the data, the inner join ensures that the resulting combined table contains only complete, synchronized observations, which is crucial for maintaining data integrity during analytical processing.

While various methods exist in SAS for merging data, the most powerful, flexible, and industry-standard approach for executing an inner join is through the use of the PROC SQL procedure. This procedure provides direct implementation of standard SQL syntax, allowing users to define complex joining criteria with clarity and minimal code overhead. The use of PROC SQL eliminates the need for manual data sorting or intricate conditional logic often required by the traditional `DATA` step merge, streamlining the process of data preparation.

Understanding the Inner Join Logic in SAS

The essence of the inner join is the calculation of a set intersection. If you are combining a table of customer details (A) and a table of purchase history (B), an inner join on `customer_ID` will yield a result set containing only those customers who appear in both tables, along with their merged details. Customers without purchase history or purchases made by customers not logged in the details table are excluded entirely, providing a focused view of active, traceable transactions.

When implementing this operation in SAS, the PROC SQL method requires defining the tables to be joined and the specific condition that must be met for records to be considered a match. This condition is specified within the `ON` clause, where the linking key variable from the first dataset is equated to the corresponding key variable from the second dataset (e.g., `table_A.key = table_B.key`). Aliases are typically used to simplify the referencing of these column names.

The following syntax represents the standard template for executing an inner join on two datasets in SAS, where `data1` and `data2` are joined based on matching `ID` values:

```
proc sql;
create table final_table as
select * from data1 as x join data2 as y
on x.ID = y.ID;
quit;
```

Setting Up Sample Data for the Join Operation

To illustrate the practical application of the inner join, we will first establish two sample datasets

containing hypothetical basketball statistics. These datasets, named `data1` and `data2`, will intentionally contain both common and unique team entries to demonstrate the filtering power of the inner join. The common linking field, which acts as our key variable, is the `team` name.

The first dataset, `data1`, records team names and their respective points scored. The second dataset, `data2`, records team names and their rebounds. Notice that several teams only exist in one of the tables. For instance, 'Rockets' is in `data1` but not `data2`, and 'Knicks' is in `data2` but not `data1`. Only teams present in both lists ('Mavs', 'Spurs', 'Warriors', and 'Hawks') are expected to survive the join.

The following SAS code generates these input tables and then uses the standard `PROC PRINT` statement to visually confirm their contents and structure before the joining process begins. This verification step is vital for ensuring the integrity of the data being merged.

```
/*create datasets*/
data data1;
input team $ points;
datalines;
Mavs 99
Spurs 93
Rockets 88
Thunder 91
Warriors 104
Cavs 93
Nets 90
Hawks 91
;
run;

data data2;
input team $ rebounds;
datalines;
Mavs 21
Spurs 18
Warriors 27
Hawks 29
Knicks 40
Raptors 30
;
run;
```

```
/*view datasets*/
proc print data=data1;
proc print data=data2;
```

The resulting printout of the two initial datasets clearly highlights the disparate lengths and the non-overlapping records based on the **team** variable. This confirms that a strict intersection join is required to harmonize the data.

Obs	team	points
1	Mavs	99
2	Spurs	93
3	Rockets	88
4	Thunder	91
5	Warriors	104
6	Cavs	93
7	Nets	90
8	Hawks	91

Obs	team	rebounds
1	Mavs	21
2	Spurs	18
3	Warriors	27
4	Hawks	29
5	Knicks	40
6	Raptors	30

Executing the Inner Join using PROC SQL

With the input data verified, we now proceed to execute the inner join using the PROC SQL procedure. Our goal is to create `final_table` by merging `data1` and `data2` exclusively where the `team` variable matches. We utilize aliases, `x` and `y`, for `data1` and `data2`, respectively, which ensures that our join condition is unambiguous, especially since the key variable (`team`) exists in both sources.

The syntax employs the `INNER JOIN` keyword (or simply `JOIN` as a default) and specifies the

precise join condition in the `ON` clause: `x.team = y.team`. This condition dictates that only rows whose team names are identical across both tables will be included in the output. The `SELECT *` command requests all columns from both tables to be carried over into the final result set.

Following the join operation, another `PROC PRINT` is immediately invoked to display the results stored in `final_table`. This step confirms that the logic of the SQL join was executed successfully, resulting in the desired subset of combined data.

```
/*perform inner join*/
proc sql;
create table final_table as
select * from data1 as x join data2 as y
on x.team = y.team;
quit;

/*view results of inner join*/
proc print data=final_table;
```

Interpreting the Combined Results

The output displayed by the final `PROC PRINT` statement clearly shows that `final_table` contains only four records. These four records correspond precisely to the four teams that were successfully matched across both input datasets: Mavs, Spurs, Warriors, and Hawks. Each row in the new table now combines the `team` identifier, the `points` score, and the `rebounds` count, forming a comprehensive statistical record for the intersecting teams.

Obs	team	points	rebounds
1	Mavs	99	21
2	Spurs	93	18
3	Warriors	104	27
4	Hawks	91	29

As anticipated by the logic of the inner join, all records that had missing entries in either `data1` or `data2` (such as 'Rockets' or 'Knicks') were excluded from the final output. This filtering behavior is the core utility of this join type, ensuring that only fully populated records, where the linking key variable is present in both sources, are retained for analysis.

Best Practices for SAS SQL Joins

While the `SELECT *` statement is convenient for simple examples, professional SAS coding mandates specifying column names explicitly. This practice improves code clarity, prevents potential column duplication (especially when the join key is selected from both tables), and optimizes performance by minimizing the amount of data processed. When joining, always prefix column names with their respective table aliases to avoid ambiguity, even if the column names are unique.

Furthermore, when dealing with very large datasets, data professionals should investigate indexing the join column. Creating an index on the key variable (in this case, `team`) using PROC SQL or `PROC DATASETS` prior to the join execution can dramatically reduce execution time, as PROC SQL can leverage these indexes for quicker lookups during the matching phase. This optimization is particularly valuable in recurring ETL (Extract, Transform, Load) processes.

Conclusion: Efficiency and Precision with Inner Joins

The inner join, executed effectively through PROC SQL, provides a precise and efficient method for harmonizing data from multiple sources in SAS. It is the preferred method for generating analytical tables that require complete information across all linked datasets.

By adopting the standardized SQL syntax within the PROC SQL environment, analysts can ensure their code is readable, maintainable, and aligned with industry best practices, leading to more reliable and reproducible analytical results.

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