Learning to Combine Datasets in SAS with PROC SQL UNION

Authored by

Mohammed looti

April 30, 2026

Combining and consolidating information from disparate sources is arguably the most fundamental requirement in modern data manipulation and analysis. Within the **SAS** ecosystem, this crucial integration task is efficiently managed using the **PROC SQL** statement, which employs syntax highly consistent with industry-standard **SQL**. Among the most potent operators available for vertical data integration is **UNION**. This operator serves as an essential mechanism for merging the resulting rows generated by two or more **SELECT statements** into a single, unified result set.

The primary utility of the **UNION** operator is to append rows from multiple **datasets**, effectively stacking them vertically. This operation is indispensable when analysts need to create a master list of records collected from various segments or time periods. Crucially, the standard **UNION** operation performs an automatic, implicit de-duplication step, ensuring that any identical **duplicate rows** that exist across the combined sources are eliminated, resulting in a cleaner, consolidated output. Understanding the nuanced application of this tool is a cornerstone of efficient **PROC SQL** programming and effective data integration within the **SAS** framework.

In this comprehensive tutorial, we will thoroughly explore the practical mechanics of the **UNION** operator within **PROC SQL**. We will move beyond the basic concept to provide concrete, real-world examples, detailing the necessary syntax and illustrating its impact on the resulting **dataset** structure. Furthermore, we will highlight the critical distinction between the standard **UNION** operator and its variant, **UNION ALL**, which handles duplicate records differently. By the conclusion of this guide, you will possess a solid, working knowledge of how to efficiently and accurately combine **datasets** vertically in **SAS** using these powerful **SQL** constructs.

Core Concepts and Syntax Requirements for UNION

The **UNION** operator is a fundamental component of the **SQL** standard, specifically designed to aggregate the result sets of two or more **SELECT statements**. When implemented within **PROC SQL**, it facilitates the process of vertical concatenation, which stands in direct contrast to traditional joins that typically merge data horizontally by adding columns. Instead of expanding the width of the data, **UNION** increases the length by appending rows (or records) from one **dataset** onto the end of another, creating a single, longer **dataset** ready for downstream analysis.

A defining characteristic that sets the standard **UNION** operator apart is its inherent capability to identify and remove **duplicate rows** during the merging process. If an entire row--meaning the combination of values across all selected columns--is identical in both source **datasets**, that row will only be included once in the final output. This automatic de-duplication feature is invaluable in scenarios where the objective is to produce a definitive, unique list of entries, such as consolidating customer records or compiling a master catalog of items without redundancy. This distinct behavior ensures data integrity and prevents overcounting in analytical summaries.

To successfully execute a **UNION** operation, strict structural rules must be followed to ensure

compatibility between the merging **SELECT statements**. First and foremost, every **query** involved in the **UNION** operation must return the exact same number of columns. Failure to match the column count will inevitably lead to a runtime error in **SAS**. Secondly, the corresponding columns across all **queries** must possess compatible **data types**. For instance, the second column selected in the first statement must be compatible with the second column selected in the second statement. While column names do not strictly need to match, the column names of the final output **table** are invariably inherited from the first **SELECT statement** executed in the block.

Setting Up Illustrative Data in SAS

To provide a clear, practical demonstration of how the **UNION** operator functions in a real-world context, we must first establish two foundational sample **datasets** within the **SAS** environment. Both datasets will contain simplified hypothetical statistics relating to basketball players, specifically tracking their team affiliation and the points scored during a single game. This controlled setup is designed to effectively highlight the mechanisms of row combination and, most importantly, the precise handling of **duplicate rows** that overlap between the sources.

Our initial **dataset** is named **`data1`**, representing the performance metrics for a primary set of players. It contains several observations primarily focused on 'Team A', each row detailing a different point total. This dataset will serve as the starting point, the first operand in our vertical concatenation exercise, providing the baseline structure and content for the subsequent combination operations we will perform using **PROC SQL**.

```
/* Create the first dataset containing basketball player statistics */
data data1;
input team $ points;
datalines;
A 12
A 14
A 15
A 18
A 20
A 22
;
run;

/* Display the contents of data1 */
proc print data=data1;
```

Executing the above **SAS** **code** yields the contents of **`data1`**, demonstrating its two-column

structure (Team and Points) and its six observations:

Obs	team	points
1	A	12
2	A	14
3	A	15
4	A	18
5	A	20
6	A	22

Subsequently, we construct a second dataset, **`data2`**, which also includes basketball player data. This dataset is intentionally crafted to contain two specific rows ('A 12' and 'A 14') that are exact matches of records already present in **`data1`**, alongside new, unique entries for 'Team B'. The presence of these overlapping records is crucial, as it provides the necessary context to observe and compare the functional differences between the standard **UNION** operator and the **UNION ALL** variant.

```
/* Create the second dataset */
data data2;
input team $ points;
datalines;
A 12
A 14
B 23
B 25
B 29
B 30
;
run;

/* Display the contents of data2 */
proc print data=data2;
```

The contents of **`data2`**, following the execution of the descriptive **SAS** **code**, are displayed below, confirming the six observations and the two specific overlapping rows:

Obs	team	points
1	A	12
2	A	14
3	B	23
4	B	25
5	B	29
6	B	30

Utilizing Standard UNION for De-Duplication

The primary and most frequent application of the standard **UNION** operator is to vertically concatenate two or more **datasets** while simultaneously guaranteeing that only **unique rows** are retained in the resulting output. This behavior is incredibly valuable in data governance and reporting, especially when data analysts are required to generate a definitive, consolidated, and de-duplicated list from data sources that may contain overlapping entries due to collection or merging processes.

We will now proceed to apply the **UNION** operator to combine our preparatory **`data1`** and **`data2`** **datasets**. The **syntax** required within **PROC SQL** is remarkably intuitive: the keyword **`UNION`** is simply inserted between the two distinct **SELECT statements**. Each **SELECT statement** explicitly defines which columns to retrieve and specifies the source **dataset** using the standard **`FROM`** **clause**, ensuring that the structural requirements (same column count and compatible **data types**) are met.

```
/* Combine data1 and data2 vertically, keeping only unique rows */
proc sql;
title 'Combined Data: data1 UNION data2 (Unique Rows Only)';
select * from data1
union
select * from data2;
quit;
```

Upon the execution of this **PROC SQL** **block**, the **SAS** system processes the request and generates the following combined output, clearly illustrating the result of the **UNION** operation:

data1 UNION data2

team	points
A	12
A	14
A	15
A	18
A	20
A	22
B	23
B	25
B	29
B	30

As meticulously verified from the resulting table, the final combined **dataset** successfully integrates all unique records from both `data1` and `data2`. Most significantly, the rows that represented overlapping entries--specifically ('A', 12) and ('A', 14)--appear only once in the final output. The total count of rows (10) is less than the sum of the original rows (6 + 6 = 12), which unequivocally confirms that the powerful de-duplicating capability, inherent to the **UNION** operator, has been correctly applied.

Employing UNION ALL for Comprehensive Data Inclusion

While the standard **UNION** is the ideal mechanism for generating a distinct list of records, many analytical requirements necessitate the retention of every single row, including those that are exact **duplicates**. This is particularly true in financial reporting, auditing, or time-series analyses where the frequency or count of observations holds intrinsic meaning. For these critical scenarios, the **UNION ALL** operator is the precise and efficient tool to use within **PROC SQL**.

The crucial functional difference between **UNION** and **UNION ALL** lies entirely in their approach to handling potential **duplicate rows**. Unlike **UNION**, which implicitly executes a resource-intensive distinct operation, **UNION ALL** operates by explicitly preserving every row from all combined **datasets**, performing no de-duplication step whatsoever. If a specific row appears three times across the source **datasets**, it will appear exactly three times in the final output. This streamlined process results in superior **performance**, as the computational overhead associated with sorting and eliminating duplicates is entirely bypassed.

To demonstrate this crucial difference, we must only make a minor modification to our previous

PROC SQL statement, substituting the simple `UNION` keyword with **`UNION ALL`**. The **SELECT statements** and the structure of the source **datasets** remain identical, ensuring a direct comparison of the merging logic:

```
/* Combine data1 and data2 vertically, keeping all rows including duplicates */
proc sql;
title 'Combined Data: data1 UNION ALL data2 (All Rows)';
select * from data1
union all
select * from data2;
quit;
```

The execution of this modified **code** yields the following expanded result, which now contains 12 observations, precisely matching the sum of the rows from the two original **datasets**:

data1 UNION ALL data2

team	points
A	12
A	14
A	15
A	18
A	20
A	22
A	12
A	14
B	23
B	25
B	29
B	30

A detailed review of the output confirms that all rows from both **`data1`** and **`data2`** have been preserved without alteration. The previously identified overlapping rows, ('A', 12) and ('A', 14), now appear twice in the final result, accurately reflecting their occurrence in both source **datasets**. This behavior is crucial when analytical objectives demand the preservation of counts. Furthermore, because **UNION ALL** avoids the costly de-duplication step, it is the more efficient and recommended choice whenever you are confident that duplicates are either non-existent or must

be retained for accurate analysis, significantly improving overall **performance** within **SAS**.

Crucial Rules for Successful PROC SQL UNION Operations

When implementing the **UNION** or **UNION ALL** operators within **PROC SQL**, adherence to specific structural and data compatibility rules is paramount to ensuring accurate and error-free data combination. Neglecting these rules is a common source of runtime errors in **SAS** programming, making strict compliance essential for robust data integration workflows.

Two of the most critical requirements relate directly to the structure of the data retrieved by the **SELECT statements**. Firstly, the number of columns returned by every single **query** must be perfectly identical; a mismatch in column count will cause the operation to fail immediately. Secondly, and equally important, the corresponding columns in each statement must possess compatible **data types** (e.g., character with character, or numeric with numeric). While **SAS** is capable of attempting implicit **type conversion** in certain scenarios, relying on this functionality can lead to unexpected data truncation or loss of precision. It is always considered best practice to ensure explicit compatibility or use explicit conversion functions (like `PUT` or `INPUT`) within the **SELECT statement** if types must be reconciled.

Furthermore, analysts must be aware of how the final output metadata is determined. The column names for the resulting **table** are invariably inherited from the first **SELECT statement** in the **query** sequence. Therefore, if specific column aliases are desired in the combined dataset, they must be defined in that initial statement. Additionally, neither **UNION** nor **UNION ALL** guarantees any particular sequence for the rows in the output. If the analysis requires the data to be sorted (e.g., by team or score), an **`ORDER BY` clause** must be explicitly appended to the very end of the entire combined **query** block to enforce the desired arrangement.

Summary and Strategic Choice

The **UNION** and **UNION ALL** operators, when employed within **PROC SQL**, represent indispensable tools for vertically combining **datasets** in the **SAS** environment. They offer flexible and robust solutions for data integration, whether the analytical objective is to merge unique records from disparate sources or to aggregate every single observation from multiple origins into one comprehensive structure.

The crucial distinction to internalize is the handling of duplicates. The standard **UNION** operator automatically manages **duplicate elimination**, rendering it the ideal choice for creating consolidated, distinct lists, though this comes with a slight performance cost. Conversely, **UNION ALL** is significantly faster because it retains all rows without performing the de-duplication scan, making it the superior option in scenarios where duplicates are either desired or irrelevant, significantly boosting **performance**. Always carefully evaluate the nature of your data and the

precise analytical requirements before selecting the appropriate operator.

Mastering these fundamental **SQL** constructs and their efficient application within the **SAS** programming environment will substantially enhance your capacity to manage, integrate, and prepare complex data, leading directly to more robust and accurate data analyses.

Additional Resources

For those aspiring to further deepen their expertise in **SAS** programming and advanced **PROC SQL** techniques, the following resources and official documentation links provide valuable insights into common data manipulation tasks and complex integration methodologies:

Official SAS PROC SQL Documentation
Official SAS Software Overview Page
W3Schools SQL UNION Tutorial
SQL on Wikipedia

These reliable sources can help you solidify your understanding of core **SQL** principles and their effective execution within the **SAS** environment.