MWSUG 2017 – Paper SA06

Merge with Caution: How to Avoid Common Problems when Combining SAS® Datasets

Joshua M. Horstman, Nested Loop Consulting, Indianapolis, IN

ABSTRACT

Although merging is one of the most frequently performed operations when manipulating SAS datasets, there are many problems which can occur, some of which can be rather subtle. This paper examines several common issues, provides examples to illustrate what can go wrong and why, and discusses best practices to avoid unintended consequences when merging.

INTRODUCTION

Anyone who has spent much time programming with SAS has likely found themselves needing to combine data from multiple datasets into a single dataset. This is most commonly performed by using the MERGE statement within a DATA step. While the merge seems like a relatively simple and straightforward process, there are many traps waiting to snare the unsuspecting programmer.

In a seminal pair of papers, Foley (1997, 1998) catalogs some 28 potential traps related to merging. These range from rather mundane oversights such as omitting the BY statement to more esoteric matters relating to the inner workings of SAS. Some can be rather subtle and pernicious. In this paper, we will examine five examples that highlight five common problems: a missing BY statement, the many-to-many merge, mismatched BY variable lengths, overlapping variables, and the automatic retain.

EXAMPLE 1: MISSING BY STATEMENT

THE DATA

For our first example, we have the following two SAS datasets:

PEAKS Dataset

MOUNTAIN	HEIGHT
Aconcagua	22838
Everest	29029
Elbrus	18510
Vinson	16050
Denali	20322
Kilimanjaro	19341
Kosciuszko	7310

LOCATIONS Dataset

MOUNTAIN	CONTINENT	COUNTRY
Aconcagua	South America	Argentina
Everest	Asia	Nepal
Elbrus	Europe	Russia
Denali	North America	United States
Kilimanjaro	Africa	Tanzania
Kosciuszko	Australia	Australia

The PEAKS dataset contains information about the heights of each of the seven summits. The LOCATIONS dataset includes the continent and country in which each mountain is located. Note that, for whatever reason, there is no record in the LOCATIONS dataset corresponding to Mt. Vinson, which is in Antarctica, but not in any country.

THE MERGE

We perform a simple merge on these two datasets, but we neglect to include a BY statement.

```
data merge1;
      merge locations peaks;
run;
```

This produces the resulting MERGE1 dataset shown below. Observe that MERGE1 has seven records, but CONTINENT and COUNTRY and missing for the last one. Also, note that Mt. Vinson has now been relocated to North America, while Denali has moved to Africa and Kilimanjaro has been transferred to Australia. Clearly, this result is undesirable.

MERGE1 Dataset

MOUNTAIN	CONTINENT	COUNTRY	HEIGHT
Aconcagua	South America	Argentina	22838
Everest	Asia	Nepal	29029
Elbrus	Europe	Russia	18510
Vinson	North America	United States	16050
Denali	Africa	Tanzania	20322
Kilimanjaro	Australia	Australia	19341
Kosciuszko			7310

THE EXPLANATION

Since we did not include a BY statement, SAS performs what is sometimes called a one-to-one merge. Rather than matching up observations based on the value of one or more BY variables, observations are simply paired based on their positions within the original datasets.

This is very rarely what is wanted. The one-to-one merge should only be used very carefully and in situations where there is no need to match observations based on any sort of relationship between the two datasets. The vast majority of circumstances call for a match-merge, which requires a BY statement.

THE CORRECTION

By simply including a BY statement in our merge, we can ensure that information is matched up based on the variable MOUNTAIN. Note that we need to sort each dataset first before we can make use of any BY group processing in a DATA step.

```
proc sort data=peaks; by mountain; run;
proc sort data=locations; by mountain; run;

data merge1b;
      merge locations peaks;
      by mountain;
run;
```

This modified code produces a correctly merged dataset:

MERGE1B Dataset

MOUNTAIN	CONTINENT	COUNTRY	HEIGHT
Aconcagua	South America	Argentina	22838
Denali	North America	United States	20322
Elbrus	Europe	Russia	18510
Everest	Asia	Nepal	29029
Kilimanjaro	Africa	Tanzania	19341
Kosciuszko	Australia	Australia	7310
Vinson			16050

THE LESSON

Obviously, it's critical that we include a BY statement whenever our intention is to perform a match-merge. Note that SAS did not issue any kind of WARNING or ERROR in response to our missing BY statement. That's because, as mentioned, there are situations where one might choose to do this deliberately.

However, because this is so rare, it may be wise to consider using the MERGENOBY system option to prevent this from happening inadvertently. The MERGENOBY system option can be set to NOWARN, WARN, or ERROR. Using MERGENOBY=WARN will cause SAS to generate a warning whenever a merge is attempted without a corresponding BY statement. Similarly, MERGENOBY=ERROR will generate an error in such cases. The default is MERGENOBY=NOWARN, which will do nothing.

EXAMPLE 2: MANY-TO-MANY MERGE

THE DATA

For our second example, we will work with the following two datasets.

MENU Dataset

MEAL	FOOD
breakfast	pancakes
breakfast	waffles
breakfast	omelet
lunch	hamburger
lunch	salad
lunch	pizza
dinner	chicken
dinner	salmon
dinner	pasta

DRINKLIST Dataset

MEAL	DRINK
breakfast	milk
breakfast	juice
breakfast	coffee
lunch	tea
lunch	lemonade
lunch	water
dinner	beer
dinner	wine
dinner	soda

Each dataset contains nine records. The MENU dataset contains three foods for each of the three meals – breakfast, lunch, and dinner. Similarly, the DRINKLIST dataset contains three drinks associated with each of one of the three meals.

THE MERGE

Our goal in this exercise is to create a list of all the food and drink combinations that are available at each meal. We sort each dataset by MEAL and then perform a merge using MEAL as the BY variable.

```
proc sort data=menu;      by meal food;  run;
proc sort data=drinklist; by meal drink; run;

data merge2;
      merge menu drinklist;
      by meal;
run;
```

The resulting dataset, MERGE2, is shown to the right.

MERGE2 Dataset

MEAL	FOOD	DRINK
breakfast	omelet	coffee
breakfast	pancakes	juice
breakfast	waffles	milk
dinner	chicken	beer
dinner	pasta	soda
dinner	salmon	wine
lunch	hamburger	lemonade
lunch	pizza	tea
lunch	salad	water

Notice that the output dataset has only nine records, three for each meal. There are actually nine possible food and drink combinations for each meal, for a total of 27 combinations. The MERGE2 dataset does not contain all the results we were expecting. For example, we see “omelet” and “coffee” together in the first record, but we do not find “omelet” matched up with either of the other breakfast drinks.

THE EXPLANATION

To understand why we get this result, it is useful to keep in mind that the DATA step is really an implied loop. Each time through the loop, the MERGE statement reads in one additional record from each dataset listed, so long as there are still additional records remaining in the current BY group.

When this DATA step first executes, it begins with the BY group corresponding to MEAL=”breakfast”. It reads in the first record from MENU (FOOD=“omelet”) and the first record from DRINKLIST (DRINK=“coffee”). When the DATA step loop executes for the second time, it finds that there are still additional records remaining in each dataset for the current BY group, so it reads in the next record from MENU (FOOD=“pancakes”) and the next record from DRINKLIST (DRINK=“juice”).

In this manner, the records are paired up based solely upon their position within each BY group. In fact, this behavior is exactly what we saw in the one-to-one merge from the first example, except that here it occurs separately within each BY group. Thus, omelet and coffee are paired up even though there is no particular relationship between the two except that they both happen to come alphabetically first among breakfast items within their respective datasets.

It’s worth noting that SAS does provide a useful note in the SAS log which gives us some indication that things might not be proceeding as we had intended. This note should not be disregarded lightly:

```
NOTE: MERGE statement has more than one data set with repeats of BY values.
```

THE CORRECTION

What we are really looking for here is often referred to as a “Cartesian product” of the two datasets – that is, every possible combination consisting of one record from the first dataset and one record from the second dataset. While there are several ways to perform a Cartesian product within a DATA step, they involve the use of more advanced techniques.

The simplest and most common way of obtaining the Cartesian product of two datasets in SAS is by performing a join using the SQL procedure. In this case, we actually only want a Cartesian product of each respective BY group, which can be accomplished as follows:

```
proc sql noprint;
   create table merge2b as
      select a.meal, a.food, b.drink
         from menu as a join drinklist as b
            on menu.meal = drinklist.meal;
quit;
```

This code produces an output dataset containing 27 rows as shown to the right.

Note that our SELECT statement specifies the MEAL column from the MENU dataset (internally referenced as A), but we could select the MEAL column from the DRINKLIST dataset instead. However, if we simply refer to the MEAL column without specifying a source dataset, we will receive an error indicating an ambiguous column reference has been detected. While it is possible to avoid this error by using the NOWARN option on the PROC SQL statement, this may inadvertently suppress messages we wish to see. Accordingly, it is advisable to explicitly specify the dataset from which each column originates.

MERGE2B Dataset

MEAL	FOOD	DRINK
breakfast	omelet	milk
breakfast	omelet	coffee
breakfast	omelet	juice
breakfast	pancakes	milk
breakfast	pancakes	coffee
breakfast	pancakes	juice
breakfast	waffles	milk
breakfast	waffles	coffee
breakfast	waffles	juice
dinner	chicken	beer
dinner	chicken	soda
dinner	chicken	wine
dinner	pasta	beer
dinner	pasta	soda
dinner	pasta	wine
dinner	salmon	beer
dinner	salmon	soda
dinner	salmon	wine
lunch	hamburger	water
lunch	hamburger	lemonade
lunch	hamburger	tea
lunch	pizza	water
lunch	pizza	lemonade
lunch	pizza	tea
lunch	salad	water
lunch	salad	lemonade
lunch	salad	tea

THE LESSON

They key lesson here is that the many-to-many merge in the DATA step operates much like the one-to-one merge we saw in the first example. In both cases, observations are paired based on their positions in the datasets being merged. While there may be unique applications where this is desirable, it generally does not result in the outcome being sought.

EXAMPLE 3: MISMATCHED BY VARIABLE LENGTHS

THE DATA

For our third example, we have the following two SAS datasets:

EMPLOYEES Dataset

LASTNAME	TITLE
Brooks	Secretary
Howard	President
Slagle	Custodian

SALARIES Dataset

LASTNAME	SALARY
Brooks	50000
Brookstein	75000
Howard	100000
Slagle	25000

For the purposes of this example, it is important to note that the variable LASTNAME has different lengths in the two datasets. In the EMPLOYEES dataset, the length of LASTNAME is 6, while in the SALARIES dataset it is 10.

Notice also that the SALARIES dataset contains an extra record that is not in the EMPLOYEES dataset. Perhaps it is an old record from a terminated employee that was not properly purged from the database. Data is not always as clean in the real world as we would like it to be.

THE MERGE

We perform a simple merge of these two datasets using LASTNAME as the BY variable. Since the datasets are already sorted by LASTNAME, it is not necessary to sort them prior to the merge.

```
data merge3;
      merge employees salaries;
      by lastname;
run;
```

The resulting dataset is not what we were expecting:

MERGE3 Dataset

LASTNAME	TITLE	SALARY
Brooks	Secretary	50000
Brooks	Secretary	75000
Howard	President	100000
Slagle	Custodian	25000

We have two records with the last name of Brooks, but one of them has the salary information associated with Brookstein. What has gone wrong here? Fortunately, in this case, the SAS log provides a clue:

WARNING: Multiple lengths were specified for the BY variable lastname by input data sets. This might cause unexpected results.

Furthermore, if we inspect the properties of the MERGE3 dataset, we will find that the LASTNAME variable there has a length of 6. Thus, the value “Brookstein” was truncated to 6 characters and is now indistinguishable from “Brooks”.

THE EXPLANATION

In order to explain these strange results, we need to take a look under the hood of the DATA step and discuss the program data vector. The program data vector (PDV) is a temporary location in memory that SAS uses during the normal processing of a DATA step.

The structure of the PDV is determined during DATA step compilation by scanning the DATA step code that was submitted. In our example, since the EMPLOYEES dataset appears first in the code, the variables from the EMPLOYEES dataset and their associated attributes are added first to the PDV. Thus, the variable LASTNAME is assigned a length of 6 in the PDV.

As the scanning continues and the SALARIES dataset is encountered, the compiler recognizes that the PDV already includes a variable called LASTNAME and takes no further action with respect to that variable. The fact that the variable has a different length has no impact on the PDV at that point.

Once the compilation phase is complete and DATA step execution begins, data which are read in using our MERGE statement are placed into the appropriate locations in the PDV. If a value is too long to fit into the corresponding variable in the PDV, it is simply truncated. Thus, in our case, “Brookstein” becomes “Brooks”.

THE CORRECTION

One might think this code could be corrected by using the IN= dataset option to ensure that only records with a corresponding entry in the EMPLOYEES dataset are present in the output dataset. That code would look like this:

```
data merge3b;
      merge employees(in=a) salaries;
      by lastname;
      if a;
run;
```

However, this produces the same result as the original DATA step code. Because of the truncation, SAS matches up the Brookstein salary record with the Brooks employee record. Thus, as far as SAS is concerned, both input datasets contributed to the resulting record.

Of course, one could solve this problem by altering the input datasets to ensure that then lengths of shared BY variables match. Another simple solution is to reverse the order of the datasets on the MERGE statement so that the dataset having the longer length associated with the BY variable comes first. However, this may not always be possible in situations with multiple BY variables that have mismatched lengths.

A more proactive solution is to take control of the process by explicitly declaring the desired variable length using a LENGTH statement. It is important that the LENGTH statement appear prior to the MERGE statement in the DATA step so that it will be encountered first by the compiler during the process of constructing the PDV.

```
data merge3c;
      length lastname $10;
      merge employees salaries;
      by lastname;
run;
```

This produces the dataset one might have expected in the first place, shown at right.

If one did not wish to include observations based only on one of the input datasets, one could modify the above code using the IN= dataset option and a subsetting IF statement as shown further above.

MERGE3C Dataset

LASTNAME	TITLE	SALARY
Brooks	Secretary	50000
Brookstein		75000
Howard	President	100000
Slagle	Custodian	25000

THE LESSON

The key lesson from this example is to avoid merging datasets on BY variables having mismatched lengths. Instead, use a LENGTH statement to explicitly control the process. A second lesson is to always check the SAS log carefully and don't just ignore SAS warnings. See Virgile (2003) for additional discussion of this topic.

EXAMPLE 4: OVERLAPPING VARIABLES

THE DATA

Our fourth example is based on the following two datasets.

POSTBASE Dataset

SUBJID	VISIT	LBSTRESN	RESP
1	1	85.7	SD
1	2	94.3	SD
1	3	71.2	PD
2	1	66.6	SD
2	2	88.8	PR

BASELINE Dataset

SUBJID	LBSTRESN
1	90.0
2	75.5

The BASELINE dataset contains one observation for each subject, SUBJID, in a research study as well as a baseline value, LBSTRESN, for some unspecified laboratory test. The POSTBASE dataset contains multiple observations for each subject. Each record includes a visit number (VISIT), a lab result from that visit (LBSTRESN), as well as a response variable, RESP.

THE MERGE

Suppose we wish to perform some computation or derivation involving the response at each visit and the baseline lab result. Since it is commonly known that variables from a dataset further to the right on the MERGE statement overwrite the values of variables from datasets listed earlier, we might be tempted to merge these datasets using the following code.

```
data merge4;
      merge postbase baseline;
      by subjid;
run;
```

The result of this operation is not what was intended.

MERGE4 Dataset

SUBJID	VISIT	LBSTRESN	RESP
1	1	90.0	SD
1	2	94.3	SD
1	3	71.2	PD
2	1	75.5	SD
2	2	88.8	PR

Notice that the first and fourth rows of the resulting dataset include the value of LBSTRESN from the BASELINE dataset while the other rows still include the values from the POSTBASE dataset.

THE EXPLANATION

Once again, the explanation of these results involves the Program Data Vector (PDV). As we discussed earlier, the structure of the PDV is determined during DATA step compilation. At execution time, data which are read in using statements such as SET, MERGE, and INPUT are placed into the appropriate locations in the PDV. DATA step statements that manipulate the values of dataset variables are actually interacting with the PDV. When it is time for an output record to be written, the contents of the PDV are copied to the output dataset.

When the first record is read from the POSTBASE dataset, the value of LBSTRESN in the PDV is 85.7. Next, the first record from the BASELINE dataset is read and the value of LBSTRESN in the PDV is overwritten with 90.0. Since this DATA step contains no other executable statements, the PDV is written to the output dataset. Thus, the first record in MERGE4 contains a value of 90.0.

During the next iteration of the DATA step, the MERGE statement reads the second record from POSTBASE. This record contains a value of 94.3 for LBSTRESN, and that value is written to the PDV. Since all of the record for the current BY group (SUBJID=1) have already been read from the BASELINE dataset, the MERGE statement does not read any additional records from BASELINE. As a result, the value of 94.3 for LBSTRESN remains in the PDV, and that is what is written to the output dataset as the second record.

THE CORRECTION

If our intention was for the value of LBSTRESN from BASELINE to overwrite all of the values from POSTBASE, we will need to modify our code. One way to solve this problem is to simply drop (or rename) LBSTRESN from the POSTBASE dataset before merging. This can be accomplished as follows.

```
data merge4b;
      merge postbase(drop=lbstresn) baseline;
      by subjid;
run;
```

When the MERGE statement reads records from POSTBASE, there will be no LBSTRESN variable to read since it has already been dropped from the input dataset. Consequently, all values of LBSTRESN in the output dataset will be those read from BASELINE.

THE LESSON

When merging datasets, it is necessary that there be some variables in common on which to merge. These are the BY variables. When the datasets have additional variables in common aside from the BY variables, these are often referred to as overlapping variables. In general, it is best to avoid overlapping variables to prevent problems like the one described above. Drop (or rename) any overlapping variables so that each occurs in only one of the datasets being merged.

EXAMPLE 5: AUTOMATIC RETAIN

THE DATA

For our final example, we have two SAS datasets containing data, once again pertaining to medical research. The DEMOG dataset contains demographic information such as the patient's age and weight. This information is recorded only once at the beginning of the study, so there is only one record per patient. The VITALS dataset contains vital signs measurements such as heart rate. These measurements are recorded at each study visit, so there can be multiple records per patient.

Both datasets include a patient identification number which provides a unique key to the data. The VITALS dataset also includes a visit number. The combination of the patient identification number and the visit number uniquely identifies a particular record.

DEMOG Dataset

SUBJID	AGE	WEIGHT
1	42	185
2	55	170
3	30	160

VITALS Dataset

SUBJID	VISIT	HEART
1	1	60
1	2	58
2	1	74
2	2	72
2	3	69
3	1	71

THE MERGE

We wish to merge these two datasets. We also wish to convert the patient's weight from pounds to kilograms. We write the following SAS code:

```
data merge5;
      merge demog vitals;
      by subjid;
      weight = weight / 2.2;
run;
```

As expected, the dataset resulting from the merge contains 5 variables and 6 records.

MERGE5 Dataset

SUBJID	AGE	WEIGHT	VISIT	HEART
1	42	84.090909091	1	60
1	42	38.223140496	2	58
2	55	77.272727273	1	74
2	55	35.123966942	2	72
2	55	15.965439519	3	69
3	30	72.727272727	1	71

Unfortunately, a careful inspection of the WEIGHT variable reveals a serious error. Notice that the value of WEIGHT changes for each record, even within the same patient. This is clearly not the desired result.

THE EXPLANATION

Once again, we turn our attention to the Program Data Vector (PDV). As mentioned previously, there are two distinct phases to running SAS code: compilation and execution. To understand what has gone wrong, we'll walk step-by-step through the process of compiling and executing this code.

Compilation

As SAS compiles our example code above, the first statement that affects construction of the PDV is the MERGE statement. The first dataset listed on the MERGE statement is DEMOG, which includes three variables: SUBJID, AGE, and WEIGHT. All three are included in the PDV using the same attributes (length, format, label, etc.) present in the input dataset. The next dataset listed is VITALS, which includes three variables: SUBJID, VISIT, and HEART. Since, SUBJID is already on the PDV, only the latter two are added.

Upon the completion of DATA step compilation, the following PDV structure is in place. Note that no actual values have been written to the PDV yet. That will occur during the execution phase.

Program Data Vector for MERGE5

Variable:	SUBJID	AGE	WEIGHT	VISIT	HEART
Value:	.	.	.	.	.

Execution

As we discuss the execution of the DATA step, it is important to remember that a DATA step is essentially a loop. The statements in the DATA step are executed repeatedly until certain conditions are met that cause execution to terminate. One such condition is a SET or MERGE statement that runs out of new records to read from all of the input datasets listed within the statement. In the meantime, the contents of the PDV are written to the specified output dataset each time execution returns to the top of the DATA step (unless you override this behavior using statements such as OUTPUT).

As our example code begins, the first statement to execute is the MERGE statement. Since the DEMOG dataset is listed first, the first record from DEMOG is read into the PDV. Next, the first record from the VITALS dataset is read. Since both datasets contain the SUBJID variable, the value from VITALS overwrites what had been previously read from DEMOG. Fortunately, since SUBJID is a BY variable, it has the same value on both datasets. Once the MERGE statement has executed for the first time, the PDV looks like this:

Program Data Vector for MERGE5

Variable:	SUBJID	AGE	WEIGHT	VISIT	HEART
Value:	1	42	185	1	60

The next statement to execute is our weight conversion. This statement reads the value of WEIGHT from the PDV, divides it by 2.2, and then writes the result back to the PDV. After this statement executes, we have the following PDV:

Program Data Vector for MERGE5

Variable:	SUBJID	AGE	WEIGHT	VISIT	HEART
Value:	1	42	84.0909	1	60

We have now reached the bottom of the DATA step. Execution returns to the top and the current contents of the PDV are written to the ALLDATA dataset. So far, everything is proceeding exactly as expected.

Automatic Retain

There is a common misconception that the values in the PDV are reset to missing when execution returns to the top of the DATA step. This is only true for variables which are assigned values by an INPUT or assignment statement (unless overridden by a RETAIN statement). For variables read with a SET, MERGE, MODIFY, or UPDATE statement, the values are automatically retained from one iteration of the DATA step to the next.

In our example, all of the variables on the PDV were read with a MERGE statement, so all values are retained. When the second iteration of the DATA step begins, the PDV looks just like it did when the first iteration ended.

Next, the MERGE statement executes again. Since the DEMOG dataset does not contain any more records for the current BY group (SUBJID = 1), nothing is read from DEMOG. There is still one record for the current BY group in the VITALS dataset, so the values from that record are copied to the PDV. Since nothing was read from DEMOG, the existing values of AGE and WEIGHT survive. The PDV now has the following state:

Program Data Vector for MERGE5

Variable:	SUBJID	AGE	WEIGHT	VISIT	HEART
Value:	1	42	84.0909	2	58

Now we come once again to the weight conversion statement. The current value of WEIGHT (84.0909) is read from the PDV and divided by 2.2, and the result (38.2231) is written back to the PDV. Having reached the end of the DATA step, the contents of the PDV are written out as the second record of the output dataset.

At last we have uncovered the source of our problem. The value of WEIGHT is read only once for each BY group, while the weight conversion statement executes once for each iteration of the DATA step. The WEIGHT continues to be divided by 2.2 repeatedly until the end of the BY group is reached.

THE CORRECTION

Now that we understand what is causing this unexpected behavior, what can we do about it? The safest and most conservative option is to limit all merges to the required statements and perform additional processing in a separate DATA step.

```
data merge5b1;
      merge demog vitals;
      by subjid;
run;
data merge5b;
      set merge5b1;
      weight = weight / 2.2;
run;
```

However, it is not always necessary to take such drastic action. This merge can be made to perform as expected within a single DATA step by simply renaming one of the input variables as follows:

```
data merge5c(drop=weight_lbs);
      merge demog(rename=(weight=weight_lbs)) vitals;
      by subjid;
      weight = weight_lbs / 2.2;
run;
```

As shown below, this modified code produces the output dataset we were expecting. Since WEIGHT_LBS is retained but not modified, each record within a given BY group will have the same value of WEIGHT.

MERGE5C Dataset

SUBJID	AGE	WEIGHT	VISIT	HEART
1	42	84.090909091	1	60
1	42	84.090909091	2	58
2	55	77.272727273	1	74
2	55	77.272727273	2	72
2	55	77.272727273	3	69
3	30	72.727272727	1	71

THE LESSON

It is advisable to be very careful when adding complex logic to a DATA step that performs a merge. One should clearly understand how the PDV works and the ramifications of the automatic retain. If these concepts are unclear, or one simply wishes to play it safe, move the additional logic to a separate DATA step.

CONCLUSION

Merging datasets is one of the most basic and common functions performed in SAS. However, the underlying procedure is more complex than it might first appear

Even the most skilled programmer can sometimes overlook subtle traps. Thus, it is advisable to habitually practice certain programming techniques to defend against these errors:

1. Don't merge without a BY statement unless you know exactly what you are doing, and consider using the MERGENOBY=ERROR option to avoid doing so inadvertently.
2. Avoid performing a many-to-many merge (where multiple datasets have repeats of the same BY variable) unless you know exactly what you are doing. Use the SQL procedure when you need a Cartesian product.
3. Always set the length explicitly when merging on a BY variable with mismatched lengths, or avoid the situation in the first place.
4. Don't merge with overlapping variables unless there is a specific reason you need to do so, and then only with full knowledge of how the merge actually works.
5. Avoid adding additional statements beyond those required for the merge: the DATA statement, the MERGE statement, the BY statement, possibly a subsetting IF statement, and of course the RUN statement. If this is too cumbersome, then at the very least, refrain from modifying the values of existing variables from an input dataset in a merge.

Finally, it is imperative for an effective SAS programmer to be equipped with a thorough understanding of the internal workings of the DATA step to avoid mistakes like the ones discussed in this paper. See Johnson (2012) or Li (2013) for a comprehensive treatment of the program data vector and Virgile (2000) for additional discussion of the PDV specifically as it relates to merging.

REFERENCES

Foley, Malachy J. "Advanced MATCH-MERGING: Techniques, Tricks, and Traps." Proceedings of the Twenty-Second Annual SAS® Users Group International. Cary, NC: SAS Institute Inc., 1997. Paper 39. http://www2.sas.com/proceedings/sugi22/ADVTUTOR/PAPER39.PDF

Foley, Malachy J. "MATCH-MERGING: 20 Some Traps and How to Avoid Them." Proceedings of the Twenty-Third Annual SAS® Users Group International. Cary, NC: SAS Institute Inc., 1998. Paper 47. http://www2.sas.com/proceedings/sugi23/Advtutor/P47.pdf

Johnson, Jim. "The Use and Abuse of the Program Data Vector." Proceedings of the SAS® Global Forum 2012 Conference. Cary, NC: SAS Institute Inc., 2012. Paper 255-2012. http://support.sas.com/resources/papers/proceedings12/255-2012.pdf

Li, Arthur. "Essentials of the Program Data Vector (PDV): Directing the Aim to Understanding the DATA Step!" Proceedings of the SAS® Global Forum 2013 Conference. Cary, NC: SAS Institute Inc., 2013. Paper 125-2013. http://support.sas.com/resources/papers/proceedings13/125-2013.pdf

Virgile, Bob. "How MERGE Really Works." Proceedings of the Pharmaceutical Industry SAS® Users Group 2000 Annual Conference. Chapel Hill, NC: PharmaSUG, 2000. Paper DM12. http://www.lexjansen.com/pharmasug/2000/DMandVis/dm12.pdf

Virgile, Bob. "Danger: MERGE Ahead! Warning: BY Variable with Multiple Lengths!" Proceedings of the NorthEast SAS Users Group 2003 Conference, Washington, DC. Paper AT005. http://www.lexjansen.com/nesug/nesug03/at/at005.pdf

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Joshua M. Horstman
Nested Loop Consulting
317-721-1009
josh@nestedloopconsulting.com