Paper S111-2012

Avoiding Pitfalls when Merging Data

James Lew, Compu-Stat Consulting, Scarborough, ON, Canada
Joshua Horstman, Nested Loop Consulting, Indianapolis, IN, USA

ABSTRACT

The merge is one of the SAS programmer's most commonly used tools. However, it can be fraught with pitfalls to the unwary user. In this paper, we look under the hood of the DATA step, examine how the program data vector works, and see how this knowledge can help us avoid common missteps when performing merges. We illustrate these principles through simple examples and discuss best coding practices for merging.

1. 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. The latter are far more interesting, and this paper will focus on one such trap: the automatic retain.

2. A SIMPLE EXAMPLE

THE DATA

Consider the following example. We have two SAS datasets containing data collected in a medical research study. 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

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

VITALS Dataset

PATIENT_ID	VISIT	HEART_RATE
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 alldata;
      merge demog vitals;
      by patient_id;
      weight = weight / 2.2;
      run;
```

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

ALLDATA Dataset

PATIENT_ID	AGE	WEIGHT	VISIT	HEART_RATE
1	42	84.0909	1	60
1	42	38.2231	2	58
2	55	77.2727	1	74
2	55	35.1240	2	72
2	55	15.9654	3	69
3	30	72.7273	1	71

3. 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. 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. We'll walk through this process step-by-step using our example from above.

COMPILATION

When you submit code to SAS, it runs in two distinct phases: compilation and execution. During the compilation phase, the program data vector is constructed.

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: PATIENT_ID, 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: PATIENT_ID, VISIT, and HEART_RATE. Since, PATIENT_ID 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: ALLDATA	
Variable	***Value***
PATIENT_ID	.
AGE	.
WEIGHT	.
VISIT	.
HEART_RATE	.

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 PATIENT_ID variable, the value from VITALS overwrites what had been previously read from DEMOG. Fortunately, since PATIENT_ID 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: ALLDATA	
Variable	***Value***
PATIENT_ID	1
AGE	42
WEIGHT	185
VISIT	1
HEART_RATE	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: ALLDATA	
Variable	***Value***
PATIENT_ID	1
AGE	42
WEIGHT	84.0909
VISIT	1
HEART_RATE	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 (PATIENT_ID = 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: ALLDATA	
Variable	***Value***
PATIENT_ID	1
AGE	42
WEIGHT	84.0909
VISIT	2
HEART_RATE	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.

4. CORRECTING THE CODE

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 step1;
      merge demog vitals;
      by patient_id;
      run;

data step2;
      set step1;
      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 alldata(drop=weight_lbs);
      merge demog(rename=(weight=weight_lbs)) vitals;
      by patient_id;
      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.

ALLDATA Dataset - Corrected

PATIENT_ID	AGE	WEIGHT	VISIT	HEART_RATE
1	42	84.0909	1	60
1	42	84.0909	2	58
2	55	77.2727	1	74
2	55	77.2727	2	72
2	55	77.2727	3	69
3	30	72.7273	1	71

5. 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. It is therefore imperative for an effective SAS programmer to be equipped with a thorough understanding of the program data vector in order to avoid mistakes like the one discussed in this paper. See Johnson (2012) for a comprehensive treatment of the program data vector and Virgile (2000) for a discussion of the PDV specifically as it relates to merging.

Even the most skilled programmer can sometimes overlook subtle traps like this. Thus, it is advisable to habitually practice certain programming techniques to defend against these errors. As we have discussed, the surest way to steer clear of problems in a merge is to 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.

Some may find this practice overly cumbersome. At the very least, one should refrain from modifying the values of variables in a dataset being merged in the same DATA step unless one can be absolutely certain that no dataset will ever contain more than one record within a BY group. Creating new variables can generally be done safely since the values of new variables are not automatically retained across iterations of the DATA step.

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.

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.

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.

Virgile, Bob. "How MERGE Really Works." Proceedings of the Pharmaceutical Industry SAS® Users Group 2000 Annual Conference. Chapel Hill, NC: PharmaSUG, 2000. Paper DM12.

CONTACT INFORMATION

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

James Lew
Compu-Stat Consulting
James.Lew@Rogers.com

Joshua M. Horstman
Nested Loop Consulting
jmhorstman@gmail.com