SAS® DATA Step Merge – A Powerful Tool

Dalia C. Kahane, Westat, Rockville, MD

ABSTRACT

Through the DATA Step Merge, SAS® offers you a method by which you may join two or more datasets and output a combined product. This is a truly powerful tool. If you follow some important basic rules you will find that you may accomplish many tasks, such as: adding summary national data to individual educational data to compare individual performances to national averages, adding sales to customer data, etc.

The DATA step Merge is similar, but not identical, to a Join operation in PROC SQL. The advantages of mastering the Merge is that you are totally in charge and that you may see explicitly what happens behind the scenes. As a matter of fact, you may use the DATA step Merge to test if results achieved by other methods are accurate.

Like all powerful tools, you need to know how to use it; how to take advantage of the power without making mistakes. This paper explores the strengths inherent in the Merge and gives you suggestions on how to avoid possible pitfalls. It is a must for beginners and yet includes tips for experienced programmers too.

INTRODUCTION

The SAS® documentation describes the DATA step Merge in the following way: "The MERGE statement joins corresponding observations from two or more SAS data sets into single observations in a new SAS data set. The way the SAS system joins the observations depends on whether a BY statement accompanies the MERGE statement"[1]

Why does SAS offer this tool? Here are some "every day" examples of why you might want to combine data:

- Educational data about students appear in multiple files, one per class (e.g. Math, English, etc.); you want to combine the data to show a student's performance across all classes; you may also then get the average performance per student across all classes
- Your client wants to see educational performance data where you need to combine teachers' data with that of their students
- You have research data about physicians and you want to compare individual against national or regional averages; combining individual data with summary data
- You have two datasets: one which contains data from parents and another which contains data from children; you want to combine them; parents may have multiple children and vice versa

These examples illustrate that there are different types of merges (one-to-one, one-to-many and many-to-many). This paper explores the rules to follow in order to perform these varying types of merges and points out the importance of using an accompanying BY statement. Also, since by default, SAS keeps all the observations and all the variables that belong to all the datasets being joined, you need to carefully trim the incoming datasets, so that the combined product will reflect exactly what you want, not what automatically happens due to the SAS program data vector (PDV) rules. This paper demonstrates the use of the SAS DATA Step options such as the DROP=, KEEP=, and RENAME=, which help you take control of variables, and the WHERE= and "subsetting IF", which help you take control of the observations to be included in the merge.

All the examples used in this paper limit the merge to two source datasets, so that the rules can be more easily demonstrated. Also, in order to keep things simple, the examples are about toys and children with very few observations in each dataset/table.

DATASETS USED TO ILLUSTRATE COMBINING OF DATA

The datasets that will be used as examples for the purpose of discussing the Merge and Join actions include the following:

- Toy – dataset includes the code, description and company code for various toys

Code	Description	CompanyCode
1202	Princess	1000
0101	Baby Doll	1000
1316	Animal Set	1000
3220	Model Train	1000
3201	Electric Truck	1000
4300	Animal Cards	2000
4400	Teddy Bear	2000

- ToyGenderAge – provides associated gender and recommended-age information for each toy

Code	Gender	AgeRangeLow	AgeRangeHigh
1202	F	6	9
0101	F	4	9
1316	B	3	6
3220	M	6	9
3201	M	6	9
5500	M	2	6

- Company – provides the company code and name for toy manufacturers

CompanyCode	CompanyName
1000	Kids Toys
2000	More Toys

- Factory – provides data on all factories associated with each company

Company Code	FactoryCode	FactoryState
1000	1111	MD
1000	1112	NY
1000	1113	VT
2000	2221	AZ
2000	2222	ME
2000	2223	CA

THE BASICS OF PERFORMING THE DATA STEP MERGE

As described above, the SAS Merge allows the programmer to combine data from multiple datasets. Each observation from dataset one is combined with a corresponding observation in dataset two (and dataset three, etc.)[1] The observations and data fields from the source datasets that will be included in the resulting dataset are determined by the detailed "instructions" provided by the programmer.

DEFAULT MERGE: ONE-TO-ONE NO MATCHING

The default action taken by SAS when the code requests a merge between two datasets is to simply combine the observations one by one in the order that they appear in the datasets.

Code:

```
data Merged_ToyGA_default;
    merge Toy ToyGenderAge;
run;
```

Log Results:

```
INFO: The variable Code on data set WORK.TOY will be overwritten by data set WORK.TOYGENDERAGE.
NOTE: There were 7 observations read from the data set WORK.TOY.
NOTE: There were 6 observations read from the data set WORK.TOYGENDERAGE.
NOTE: The data set WORK.MERGED_TOYGA_DEFAULT has 7 observations and 6 variables.
```

Combined File:

Code	Description	CompanyCode	Gender	RangeAgeLow	RangeAgeHigh
1202	Princess	1000	F	6	9
0101	Baby Doll	1000	F	4	9
1316	Animal Set	1000	B	3	6
3220	Model Train	1000	M	6	9
3201	Electric Truck	1000	M	6	9
5500	Animal Cards	2000	M	2	6
4400	Teddy Bear	2000		.	.

By default the SAS Merge does the following:

- combines in order observation #1 from Toy with Observation #1 from ToyGenderAge, then Observation #2 with Observation #2, etc.; does **not** try to match on common variable(s); simply matches the observations in the random order that they appear in the datasets
- keeps all observations from both datasets
- the system option **MergeNoBy** is set to NOWARN; neither a warning nor an error message is given to alert user that a merge is being performed without matching observations through a BY statement
- in a one-to-one merge, common variables that are not part of BY statement, act as follows: the values coming in from the right dataset override those coming in from the left dataset – see SAS NOTE in Log results above

This is usually **not** what the programmer would want, since it would make much more sense to combine all data related to toy "Princess" together; all data related to "Electric Truck" together; etc.

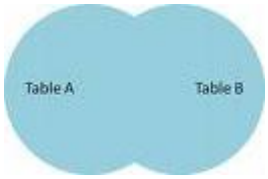
ONE-TO-ONE MATCH-MERGE KEEPING ALL OBSERVATIONS

A Match-Merge combines observations from multiple datasets into a single observation in the result dataset based on the values of one or more common variables.

It is highly recommended that you do the following:

- set the system option: **MergeNoBy** = ERROR; this ensures that if a MERGE statement is used without a corresponding BY statement the log will present an error message to that effect;
- use a BY statement in the data step to force SAS to put values into a single observation combining data from observations in the source datasets where the BY Variable has the same value;
- make sure to sort all "source" datasets by the matching variable(s) listed in the BY statement; if the datasets are not sorted properly you will get error messages in the log

Note that by default a match-merge keeps all observations from all datasets; but each "final" observation gets its data values only from data variables that are contributed by those datasets with matching Toy Code values; where there is no contributing matching observation the data values are set to missing. Here again, the product is the same as a full "outer-join" where some of the observations have variables filled from both datasets, because a match was found, but others only get values from their source-dataset.[2]



Note that by default a match-merge keeps all observations from all datasets; however, each "final" observation gets its data values only from data variables that are contributed by those datasets with matching Toy Code values; where there is no contributing matching observation the data values are set to missing.

Code:

```
options mergenoby=error;
proc sort data=Toy; by Code; run;
proc sort data=ToyGenderAge; by Code; run;
data Merged_ToyGA_ByCode;
    merge Toy (keep=Code Description)
          ToyGenderAge;
    by Code;
run;
```

Log Results:

```
NOTE: There were 7 observations read from the data set WORK.TOY.
NOTE: There were 6 observations read from the data set WORK.TOYGENDERAGE.
NOTE: The data set WORK.MERGED_TOYGA_BYCODE has 8 observations and 5 variables.
```

Combined File:

Code	Description	Gender	AgeRangeLow	AgeRangeHigh
0101	Baby Doll	F	4	9
1202	Princess	F	6	9
1316	Animal Set	B	3	6
3201	Electric Truck	M	6	9
3220	Model Train	M	6	9
4300	Animal Cards			
4400	Teddy Bear			
5500		M	2	6

Note that:

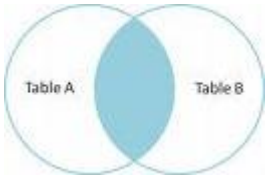
- all observations from both datasets are kept
- where identical code value was found on both datasets, all variables get filled with data coming in from the corresponding source datasets; therefore, for codes 4300 and 4400 the combined observations contain Description values from the Toy dataset but no Gender or Age Range values coming in from the ToyGenderAge dataset; for Code 5500, the combined observation contains data values from the ToyGenderAge dataset but no Description from the Toy dataset

ONE-TO-ONE MATCH-MERGE KEEPING SOME OBSERVATIONS

In many cases we may want to further control which observations (of those that match) will actually be included in the final dataset. This is done via a "subsetting if" statement combined with an "in=" data option. There are several different choices such as:

- keep only those observations where a match is found on the BY variable in all "source" datasets
- keep all those that appear in the 1st dataset whether or not a match is found in the 2nd dataset
- keep all those that appear in the 2nd dataset whether or not a match is found in the 1st dataset

In the 1st instance the product is really an "inner join"



Code:

```
data Merged_ToyGA_KeepOnlyMatched;
    merge Toy (in=a keep=Code Description)
          ToyGenderAge (in=b)
    ;
    by Code;
    if a and b;
run;
```

Log Results:

```
NOTE: There were 7 observations read from the data set WORK.TOY.
NOTE: There were 6 observations read from the data set WORK.TOYGENDERAGE.
NOTE: The data set WORK.MERGED_TOYGA_KEEPONLYMATCHED has 5 observations and 5 variables.
```

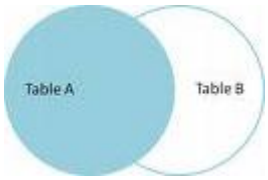
Combined File:

Code	Description	Gender	AgeRangeLow	AgeRangeHigh
0101	Baby Doll	F	4	9
1202	Princess	F	6	9
1316	Animal Set	B	3	6
3201	Electric Truck	M	6	9
3220	Model Train	M	6	9

There are several things to note here:

- the “in=” option which uniquely identifies each “source” dataset;
- the “by Code” statement which ensures a one-to-one matching;
- the “if a and b” statement, which instructs SAS to keep only those observations with matching code values in both datasets; those observations with code values that appear in only one or the other dataset get excluded from the final “combined” dataset;

In the 2nd instance, what we get is



a statement of “if a” ensures that all observations and only observations from dataset Toy will be kept in the “final” dataset; the data values coming from the ToyGenderAge will be set to missing when no matching observation with that Code value is found there; result in log

Code:

```
data Merged_ToyGA_KeepOnlyLeft;
   merge Toy (in=a keep=Code Description)
         ToyGenderAge (in=b)
   ;
   by Code;
   if a;
run;
```

Log Results:

NOTE: The data set WORK.MERGED_TOYGA_KEEPONLYLEFT has 7 observations and 5 variables.

Combined File:

Code	Description	Gender	AgeRangeLow	AgeRangeHigh
0101	Baby Doll	F	4	9
1202	Princess	F	6	9
1316	Animal Set	B	3	6
3201	Electric Truck	M	6	9
3220	Train Thomas	M	6	9
4300	Animal Cards		.	.
4400	Teddy Bear		.	.

In the 3rd instance

a statement of “if b” ensures that all observations and only observations from dataset ToyGenderAge will be kept in the “final” dataset; the data values coming from Toy will be set to missing when no matching observation with that Code value is found in the Toy dataset; result in log

Code:

```
data Merged_ToyGA_KeepOnlyRight;
   merge Toy (in=a keep=Code Description)
         ToyGenderAge (in=b)
   ;
   by Code;
   if b;
run;
```

Log Results:

NOTE: The data set WORK.MERGED_TOYGA_KEEPONLYRIGHT has 6 observations and 5 variables.

Combined File:

Code	Description	Gender	AgeRangeLow	AgeRangeHigh
0101	Baby Doll	F	4	9
1202	Princess	F	6	9
1316	Animal Set	B	3	6
3201	Electric Truck	M	6	9
3220	Train Thomas	M	6	9
5500		M	2	6

ONE-TO-MANY MERGE

One of the most common purposes of the one-to-many merge is to join individual observations with summary-statistics for purposes of evaluations and comparisons; another purpose is to combine “detailed” data with “higher level” data, as in our example datasets, combining the Toys with their Company name.

There are many toys per company.

Code:

```
proc sort data=Toy; by CompanyCode; run;
proc sort data=Company; by CompanyCode; run;
data Merged_ToyCompany;
   merge Toy Company;
   by CompanyCode;
run;
```

Log Results:

NOTE: There were 7 observations read from the data set WORK.TOY.
NOTE: There were 2 observations read from the data set WORK.COMPANY.
NOTE: The data set WORK.MERGED_TOYCOMPANY has 7 observations and 4 variables.

Combined File:

Code	Description	Company Code	Company Name
0101	Baby Doll	1000	Kids Toys
1202	Princess	1000	Kids Toys
1316	Animal Set	1000	Kids Toys
3201	Electric Truck	1000	Kids Toys
3220	Model Train	1000	Kids Toys
4300	Animal Cards	2000	More Toys
4400	Teddy Bear	2000	More Toys

Important issues to note for the one-to-many merge are:

- In DATA Step MERGE the one-to-many and many-to-one merges are the same! The order of the dataset names within the MERGE statement has no significance; the **actual merge action** still combines one observation from a dataset to one observation from another;
- The data from each observation coming from the "one" side is retained through all merges with the "many" side (until a new observation comes in from the "one" side); see later discussion of the Program data Vector
- Common variables that are not included in the BY statement should be renamed since bringing them in may lead to errors. In a one-to-one merge a common variable's value coming from a "later" dataset simply overwrites the value from an "earlier" dataset (in the order that the datasets appear in the MERGE statement). This is **not always** true in the case of a one-to-many merge. It is **good practice** to **always** rename common variables as they come in from the source files and calculate a "final" value in the result dataset. Use the RENAME= option per dataset in the MERGE statement; see later discussion of the Program data Vector

MANY-TO-MANY MERGE

The many-to-many merge refers to the instance where at least two source datasets contain multiple repeats of values in the BY variables used for match-merging. An example of a many-to-many merge using our datasets is to present all possible factories from where all toys of a given company may be shipped. Another way to describe this: show any factory from which any toy may be shipped.

Code:

```
proc sort data=Toy; by CompanyCode Code; run;
proc sort data=factory; by CompanyCode FactoryCode; run;
/* create cross walk dataset to tell us how many factories per company and assigns them numbers */
data Factory
     Company_Xwalk(keep=CompanyCode MaxFactory);
     retain FactoryNumber 0;
     set Factory;
     by CompanyCode;
     if first.CompanyCode then cnt=0;
     FactoryNumber +1;
     MaxFactory = FactoryNumber;
     output factory;
     if last.CompanyCode then output Company_Xwalk;
 run;
```

```
/* prepare the toys dataset by merging with the company Xwalk */
data ToyWithFN (drop=i);
    merge Toy(in=a)
          Company_Xwalk(in=b)
    ;
    by CompanyCode;
    if a;
    if not b then put "No Company record found for Toy: " _all_;
    /* per toy - output multiple records - one per factory */
    do i =1 to MaxFactory;
        FactoryNumber=i;
        output;
    end;
run;

/* we now finaly merge Toys with Factories */
proc sort data=ToyWithFN; by CompanyCode FactoryNumber Code; run;
proc sort data=factory; by CompanyCode FactoryNumber; run;
data Merged_ToyFactory (drop=MaxFactory FactoryNumber);
     merge ToyWithFN(in=a)
           Factory(in=b)
     ;
     by CompanyCode FactoryNumber;
run;
```

Log Results:

```
NOTE: The data set WORK.COMPANY_XWALK has 2 observations and 2 variables.
NOTE: The data set WORK.TOYWITHFN has 21 observations and 5 variables.
NOTE: There were 21 observations read from the data set WORK.TOYWITHFN.
NOTE: There were 6 observations read from the data set WORK.FACTORY.
NOTE: The data set WORK.MERGED_TOYFACTORY has 21 observations and 5 variables.
```

Combined File:

Code	Description	CompanyCode	FactoryCode	FactoryState
0101	Baby Doll	1000	1111	MD
1202	Princess	1000	1111	MD
1316	Animal Set	1000	1111	MD
3201	Electric Truck	1000	1111	MD
3220	Model Train	1000	1111	MD
0101	Baby Doll	1000	1112	NY
1202	Princess	1000	1112	NY
1316	Animal Set	1000	1112	NY
3201	Electric Truck	1000	1112	NY
3220	Model Train	1000	1112	NY
0101	Baby Doll	1000	1113	VT

1202	Princess	1000	1113	VT
1316	Animal Set	1000	1113	VT
3201	Electric Truck	1000	1113	VT
3220	Model Train	1000	1113	VT
4300	Animal Cards	2000	2221	AZ
4400	Teddy Bear	2000	2221	AZ
4300	Animal Cards	2000	2222	ME
4400	Teddy Bear	2000	2222	ME
4300	Animal Cards	2000	2223	CA
4400	Teddy Bear	2000	2223	CA

Important notes for the many-to-many merge:

- As shown in the SAS code, the goal here is to combine many toys with many factories. The MERGE statement however still ends up combining observations from the 1st dataset with observations of the other datasets, one by one
- Before performing the merge, several steps are needed to prepare the source datasets: we use PROC SORT for each source dataset and we create a crosswalk dataset; etc.
- Much less work is needed to accomplish the task of combining many-to-many via the PROC SQL JOIN[3]

MERGE AND THE PROGRAM DATA VECTOR

In order to illustrate what happens in the Program Data Vector (PDV) when a MERGE is done, we'll use two new and very simple datasets that contain student grades. The 1st dataset contains grades for English class and the 2nd contains grades for a Math class. Each of these datasets contains three variables per record: StudentID, Grade and TeacherID.

The **English** dataset contains 1 record each for StudentIDs 101,102 and 104 but two records for 103

StudentI	Grade	TeacherID
101	80	E02
102	70	E02
103	60	E01
103	80	E01
104	90	E02

The **Math** dataset contains one record each for StudentIDs 101, 103 and 104

StudentID	Grade	TeacherID
101	70	M11
103	60	M11
104	80	M12

In order to get the students grades and averages into a single dataset we'll use the following code:

```
proc sort data=English; by StudentID; run;
proc sort data=Math;    by StudentID; run;
data Grades;
   merge English (in=a rename=(Grade=English_Grade))
         Math (in=b rename=(Grade=Math_Grade))
   ;
   by StudentID;
run;
```

When data is "mapped" into the PDV, any variable with identical name across the two datasets, and which is not used as a BY variable, may become corrupt. Therefore, in this example the PDV contains the following:

StudentID	English Grade	Math_ Grade	TeacherID	First. StudentID	Last. StudentID	_N_	_ERROR_
101	80	70	M11	1	1	1	0
102	70	.	E02	1	1	2	0
103	60	60	M11	1	0	3	0
103	80	60	E01	0	1	4	0
104	90	80	M12	1	1	5	0

When the first record comes in from the English dataset, the variables StudentID, English_Grade, and TeacherID are properly stored in the appropriate storage units. When the Math record for that StudentID comes in, the Math_Grade is correctly added. But, the TeacherID value coming in from the Math dataset replaces the TeacherID which was already there from the English dataset. If no record exists for this StudentID in the Math dataset the Math_Grade remains missing and the TeacherID is not replaced. The variable of TeacherID just became meaningless because some of the time it contains the English TeacherID and at other times it contains the Math TeacherID. The code should have included a RENAME= for the TeacherID variable similar to that done for the Grade variables, or the TeacherID variable should have been dropped.

The following principles should always be remembered[4]:

1. Only one value can be held under one name in the PDV at any one time
2. No common variable between two or more data sets listed in the merge with the exception of the BY variables that must be common to all the data sets
3. No more than one data set with multiple records for any combination of BY values

Remember that it is a mistake to think that the PDV always gets filled with the value of the variable which comes from the last dataset mentioned on the MERGE. It is important to note that a variable coming from any dataset is RETAINED by the PDV only if there does not exist a variable by the same name already coming in from a previous dataset! In cases where there are multiple records per BY variable in the 1st dataset, only the 1st of these gets a value from the last dataset to replace the value from the previous dataset. All the other records do not receive this new value!

BEWARE OF THE FOLLOWING NOTES IN THE LOG

Some notes that appear in the LOG, even though they are not marked as errors or warnings are very dangerous to ignore. It is much better to add the code that is necessary to make such notes disappear. These include:

(1) NOTE: More than one dataset have repeats of the BY variables

(2) NOTE: the variable xxx in data A is replaced by variable xxx in data B

Note (1) may appear under the following conditions: merging the English dataset with the Math dataset by StudentID, where there are multiple grade records per student in each (due to multiple sessions). Possible solutions: keep only the latest or highest grade per student; or, add the sessionID to the BY statement and join the datasets by both student ID and SessionID.

Note (2) may appear in the case shown above where TeacherID appeared in both datasets. Either rename the variable or drop it, as appropriate.

CONCLUSION

The DATA step Merge is a powerful tool but you need to know all rules governing its functionality and use them to advantage. Here is a list of the salient features of the merge:

- One-to-one oriented (consequently limited but offers very tight control)
- Observations are read once (SAS data retained)
- Default is a one-to-one "outer-join" using the given order of observations in the source datasets
- BY variables must have same name and type and preferably same length too in all source tables
- The order of the data sets should not matter; therefore no variables (other than the BY variables) should be in common; variables should be renamed (if needed)
- IN= variables are not reset by the system unless new data is read
- Match-merge of two datasets (merge with "BY Common variable(s)" but without a subsetting "IF" still results in a full outer-join but combines observations that have “something in common”
- "Outer-joins" may be done on multiple source datasets
- Need to sort the source datasets before the data step match-merge
- Merge of datasets with "BY common variables(s)” and "If a and b" results in an “inner join” and therefore, a limited product
- Merge of datasets with "BY common variables(s)” and “If a” or “If b” is used, results in another type of limited product where all records from either source are kept; same rule holds for any dataset “selected” from however many datasets that participate in the merge
- In a one-to-one match-merge, common variables that are not included in the match condition (i.e. are not part of the BY statement): the value from the latter dataset sometimes overwrites the value coming from the left-more dataset
- Impossible to do a many-to-many merge due to one-to-one behavior of MERGE

NOTES

[1] Refer to the “SAS Language: Reference”, chapter 9, description of the Merge statement

[2] all Venn diagram images were taken from www.codinghorror.com by Jeff Atwood

[3] comparison of MERGE to PROC SQL offered in Kahane paper referenced below

[4] Refer to Ian Whitlock’s paper shown below

REFERENCES

Kahane, Dalia C. (2009), *“*Using DATA Step MERGE and PROC SQL JOIN to Combine SAS® Datasets*”*, Proceedings of the 2009 North East SAS Users Group Conference

Schreier, Howard (2005), *“*Let Your Data Power Your DATA Step: Making Effective Use of the SET, MERGE, UPDATE, and MODIFY Statements*”*, Proceedings of the 30th Annual SAS Users Group International Conference

SAS Institute Inc. (1990), SAS Language: Reference, Version 6, 1st Edition, Cary, NC: SAS Institute Inc.

Whitlock, Ian (2006), *“How to Think Through the SAS DATA Step”*, Proceedings of the 31st Annual SAS Users Group International Conference

DISCLAIMER

The contents of this paper are the work of the author and do not necessarily represent the opinions, recommendations, or practices of Westat.

ACKNOWLEDGEMENTS

I would like to thank Ian Whitlock and Michael Raithel who reviewed my paper, provided comments, and made this a true learning experience for me.

CONTACT INFORMATION

If you have any questions or comments about the paper, you can reach the author at:

Dalia Kahane, Ph.D.
Westat
1600 Research Blvd.
Rockville, MD 20850
Kahaned1@Westat.com