Paper 89-27

Many-to-Many Merging Using the SAS® Macro Facility

Jonathan D. Mahnken, University of Texas Medical Branch, Galveston, TX

ABSTRACT

One-to-many merging by an identifying (ID) variable is a useful feature for database management. However, in certain situations, a many-to-many merge may be needed; in other words, there are repeated observations for some of the ID variables in both data sets being merged. The many-to-many merge can be done through a series of DATA steps after first dividing one of the data sets into multiple sub-sets such that all of the ID variables are unique within these new sub-sets. The problem with this method is that the programmer must count (PROC FREQ) the maximum number of times any one ID value may appear in a data set, and this information must then be written into the program doing the merge. This introduces more possibility for errors, especially when the original data sets are updated or changed. With the SAS macro facility, a many-to-many merge can be done within the program, without checking the number of times an observation appears in the data and without needing to change the code with each new data set revision. Programmers with a basic understanding of data manipulation and merging will be able to use these macros.

INTRODUCTION

People involved in database management may encounter data that needs to be merged together by an ID variable. When there are multiple observations with the same ID variable in both of the data sets being merged, the macro facility can be employed to handle the merge without having to look at the data and update the program. This method is particularly useful when the program must be submitted frequently with changing data. It is also notably efficient when the program needs to be submitted only once but involves many observations with the same ID variable in either data set.

AN EXAMPLE OF WHEN TO USE A MANY-TO-MANY MERGE

Some researchers use the Medicare database which is divided into several parts. One of these parts contains the inpatient hospital records and another contains outpatient records. If we are interested in whether patients are receiving the appropriate outpatient follow-up care after an inpatient surgery and the patients can have this inpatient surgery more than once, then we need to do a many-to-many merge to create the analysis data file. In other words, any patient can have several surgeries; so, the inpatient data will contain multiple records for some of the patients. Similarly, there can be multiple follow-up visits to the outpatient clinics; hence, the outpatient data will also contain multiple records for some of the patients. Once merged, each patient's surgery record will have one observation for each separate outpatient visit. For example, if patient #232 had three hospital surgeries and eight outpatient follow-up visits, patient #232 will have twenty-four records in the final merged data set. There will be one record containing patient #232's first surgery data and first outpatient visit data, one record containing patient #232's first surgery data and second outpatient visit data, etc., one record containing patient #232's second surgery data and first outpatient visit data, and so on. As seen in this brief example, the size of a many-to-many merged data set can be considerably larger than even the two original data sets combined!

METHODS

There are three types of many-to-many merges. The first is used when all of the observations in one of the data sets are kept and the only observations from the second data set that are kept are those containing an ID that matches an ID in the first data set. The macro in the appendix that corresponds to this is %KEEP_1ST. The second type occurs when all of the observations in both data sets are kept, regardless of whether the particular ID is present in the other data set or not. The macro that corresponds to this is %KEEP_EIT. The third type of many-to-many merge is used when only those observations whose ID values appear in both data sets are kept. The corresponding macro for this type is %KEEP_BTH.

Each of these macros requires four local macro variables to be input. The first two macro variables are the names of the two SAS data sets to be merged. The third macro variable is the name of the ID variable (which is the same in both data sets). The fourth macro variable is the name of the newly merged SAS data set that the macro creates.

Here is a basic description of what happens in these macros.

- Each of the original data sets are sorted by the ID variable.
- The number of times each ID value occurs in each data set is counted and placed in a new variable in a count data set. A count data set consists of the ID variable and a variable that counts the number of times that each unique ID occurs. There is one observation for each unique ID value.
- The two count data sets are merged together (the criteria by which they are merged depends on the type of many-to-many merge). This is a one-to-one merge handled by the MERGE statement in the DATA step.
- The newly merged count data set (which contains the ID variable and two count variables – each representing the number of times that unique ID occurred in one of the given original data sets) is separately merged with each of the original files. These are one-to-many merges handled by the MERGE statement in the DATA step.
- Each observation in these data sets is replicated to the number of times that its corresponding ID value occurs in the other original data set.
- The observations within one data set are then ready to be merged. They are sorted by the ID variable. The observations within the other data set are sorted so that each unique record of information from the first data set will be aligned with each unique record in the second data set (with the same ID value).

EXAMPLE USING THESE MACROS

Here is an example of how to use these macros to perform a many-to-many merge.

```
data data_1;
  input id x;
  lines;
1 11
1 12
1 13
1 14
3 31
4 41
4 42
4 43
7 71
9 91
9 92
9 93
;;;;

data data_2;
  input id y;
```

```
  lines;
1 11
1 12
2 21
2 22
2 23
2 24
4 41
7 71
7 72
7 73
7 74
8 81
8 82
9 91
;;;;
```

The two data sets (DATA_1 and DATA_2) contain the same ID variable (ID), but also contain unique information about the observation. For data set DATA_1, the unique information is contained in variable X; and for data set DATA_2, the unique information is contained in the variable Y.

Once the macros in the appendix have been submitted into the program (either by typing the code into the text or by storing the code in a given location and referencing it using the %INCLUDE macro statement), the following code can be used to generate the three different many-to-many merged data sets. To create a data set containing all of those observations from DATA_1, and only those from DATA_2 with an ID value that appears in DATA_1, use the following code:

```
%keep_1st (data_1, data_2, id, data_1_2);
```

Here is the resulting data set (named DATA_1_2).

Obs	id	x	y
1	1	11	11
2	1	11	12
3	1	12	11
4	1	12	12
5	1	13	11
6	1	13	12
7	1	14	11
8	1	14	12
9	3	31	.
10	4	41	41
11	4	42	41
12	4	43	41
13	7	71	71
14	7	71	72
15	7	71	73
16	7	71	74
17	9	91	91
18	9	92	91
19	9	93	91

Notice that observation 9 (ID = 3) contains a value for the X variable but not for the Y variable. This is because this observation was in the first data set (DATA_1) but not in the second (DATA_2). Also, note that there are no observations with ID values of 2 or 8. This is because these IDs only appeared in the second data set (DATA_2).

To generate the data set that contains all of the observations, regardless of whether the observations' ID values are present in the other data set or not, use the following code:

```
%keep_eit (data_1, data_2, id, data_1_2);
```

Here is the resulting data set (named DATA_1_2).

Obs	id	x	y
1	1	11	11
2	1	11	12
3	1	12	11
4	1	12	12
5	1	13	11
6	1	13	12
7	1	14	11
8	1	14	12
9	2	.	21
10	2	.	22
11	2	.	23
12	2	.	24
13	3	31	.
14	4	41	41
15	4	42	41
16	4	43	41
17	7	71	71
18	7	71	72
19	7	71	73
20	7	71	74
21	8	.	81
22	8	.	82
23	9	91	91
24	9	92	91
25	9	93	91

Because there were no observations in DATA_1 with IDs equal to 2 or 8, the values of X for these observations are missing. Similarly, because there were no observations with ID values of 3 in DATA_2, the value of Y for that observation is also missing.

To create a data set that contains only those observations whose ID values are present in both data sets, use the following code:

```
%keep_bth (data_1, data_2, id, data_1_2);
```

Here is the resulting data set (named DATA_1_2).

Obs	id	x	y
1	1	11	11
2	1	11	12
3	1	12	11
4	1	12	12
5	1	13	11
6	1	13	12
7	1	14	11
8	1	14	12
9	4	41	41
10	4	42	41

11	4	43	41
12	7	71	71
13	7	71	72
14	7	71	73
15	7	71	74
16	9	91	91
17	9	92	91
18	9	93	91

In this data set, observations with the ID values 2, 3, and 8 were not included because they only appeared in one of the two data sets being merged.

LIMITATIONS

At times, a data set may be indexed by more than one ID variable. For example, a company may use both an employee's last name and the year he/she was hired as the two index variables. This would help differentiate between employees that have the same last name. However, these macros will not work when there is more than one variable indexing the data set (due to the LAG statements in the macros). Two simple fixes for this are to combine all of the indices into one variable, or to re-index the data sets altogether. Caution should be used when either of these methods are employed so that observations in the two different data sets with matching index variables remain matches under the new index scheme.

New data sets are created in these macros, as are new variables within these data sets. If the user has temporary data sets being used during the SAS session named _____a (seven underscores, then an 'a'), _____b (seven underscores, then a 'b'), ___a_cnt (three underscores, then 'a_cnt'), ___b_cnt (three underscores, then 'b_cnt'), or __ab_cnt (two underscores, then 'ab_cnt'), then these macros will overwrite and delete the data set at the end of the macro. These data set names with the leading underscores were selected because it seemed unlikely that you would chose these names for other temporary SAS data sets; however, to use these macros without losing other data sets, you should rename all temporary data sets to something other than these five names. Also, if the two data sets being merged contain the variables ___cnt_a (three underscores, then 'cnt_a'), ___cnt_b (three underscores, then 'cnt_b'), __cnt_a2 (two underscores, then 'cnt_a2'), or __cnt_b2 (two underscores, then 'cnt_b2'), then these macros may overwrite the values of these variables. As with data set names that match those used within these macros, pre-existing variable names that matched those mentioned should be changed prior to running the macros to prevent a loss of information, an improper many-to-many merging of the data sets, or both.

These macros were written for use with SAS version 8.2 on a Windows® 98 operating system. Conflicts with this code on other versions of SAS or operating systems may exist.

CONCLUSION

In certain situations, one-to-one and one-to-many merging are not adequate methods for merging two data sets together. These many-to-many merge macros can be used to appropriately join information from the two data sets in a way that is both simple and smooth. Use of these macros is significantly more efficient than counting the number of occurrences of each unique ID and then incorporating that information in the SAS code to merge data sets. This counting method inevitably increases the possibility of errors because an incorrect count could be entered into the program yielding incorrect results and no warning in the program log. This is of particular concern when the program is run repeatedly with changing data. Ideally, once the program is written, you simply need to submit the code to run the entire program, regardless of updates or changes to the data.

ONE FINAL NOTE

Lastly, I would like to mention that these macros can also be used to perform a many-to-many merge on more than two data sets. To merge several data sets together, simply merge two at a time (using the appropriate macro for the particular job), and continue to merge a new data set to the new many-to-many merged data set until all of the data has been merged.

REFERENCES

SAS Institute Inc. (1999), *SAS Language Reference, Version 8*, Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1997), *SAS Macro Language: Reference, First Edition*, Cary, NC: SAS Institute Inc.

ACKNOWLEDGMENTS

This project was funded in part by a grant from the National Cancer Institute (CA72076).

TRADEMARK CITATION

CONTACT INFORMATION

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

Jonathan D. Mahnken
University of Texas Medical Branch
700 Harborside Drive
Galveston, Texas 77555-1148
(409) 772-1992
jdmahnke@utmb.edu

APPENDIX

```
*********************************************
Programmer:    Jon Mahnken
Email:         jdmahnke@utmb.edu (for questions
               or electronic copy of code)
Last Update:   06AUG2001
*********************************************;


*-------------------------------------------*
 This macro performs a many-to-many merge.  The
 first macro variable to be input contains the
 name of the 1st data set.  The final data set
 will contain all of the observations in this
 first data set, and only those observations
 from the second data set that were also in the
 first data set.  Each observation in the first
 data set will contain every observation from
 the second data set with the same ID variable.
 The second macro variable is the name of the
 second  data set.  The third macro variable is
 the name of the ID variable which the data sets
 will be merged by.  The last macro variable is
 the name of the final many-to-many merged data
 set.
*-------------------------------------------*;
%macro keep_1st (data_a, data_b, id, data_ab);

 *--- Renames the first original data set ---*;
 *--- so it will not be changed ---*;
 data _______a;
   set &data_a;
 run;
```

```
*--- Sort the first original data set by ---*;
*--- the ID variable ---*;
proc sort  data= _______a;
  by &id;
run;

*--- Creates a data set that counts the ---*;
*--- number of obs w/ each different ID ---*;
*--- ONLY one obs w/ each unique ID is ---*;
*--- kept ---*;
data ___a_cnt;
  set _______a (keep=&id);
  by &id;
  retain ___cnt_a;
  if (&id ne lag1(&id))
    then ___cnt_a= 1;
  else
    ___cnt_a= ___cnt_a + 1;
  if (last.&id);
run;

*--- Renames the second original data set ---*;
*--- so it will not be changed ---*;
data _______b;
  set &data_b;
run;

*--- Sort the second original data set by ---*;
*--- the ID variable ---*;
proc sort  data= _______b;
  by &id;
run;

*--- Creates a data set that counts the ---*;
*--- number of obs w/ each different ID ---*;
*--- ONLY one obs w/ each unique ID is ---*;
*--- kept ---;
data ___b_cnt;
  set _______b (keep=&id);
  by &id;
  retain ___cnt_b;
  if (&id ne lag1(&id))
    then ___cnt_b= 1;
  else
    ___cnt_b= ___cnt_b + 1;
  if (last.&id);
run;

*--- Merges (one-to-one) the count data ---*;
*--- sets ---*;
data __ab_cnt;
  merge ___a_cnt (in=in_a) ___b_cnt;
  by &id;
  if (in_a); *keeps only obs in the first data
              set;
  if (___cnt_b = .)
    then ___cnt_b= 1; *for obs w/ an ID in A
                       but not in B;
run;

*--- Expands each obs in A by the number ---*;
*--- of obs w/ that ID in B ---*;
data _______a;
  merge _______a __ab_cnt (drop=___cnt_a);
  by &id;
  do __cnt_b2= 1 to ___cnt_b;
    output;
  end;
  drop
    ___cnt_b
    __cnt_b2;
run;

*--- Expands each obs in B by the number ---*;
*--- of obs w/ that ID in A ---*;
data _______b;
  merge _______b __ab_cnt
        (in=in_ab drop=___cnt_b);
  by &id;
  if (in_ab);
  do __cnt_a2= 1 to ___cnt_a;
    output;
  end;
  drop
    ___cnt_a;
run;

*--- Sorts B so that each element in B is ---*;
*--- lined up w/ each element in A ---*;
proc sort  data= _______b;
  by &id __cnt_a2;
run;

*--- Merges the two data sets (which are ---*;
*--- lined up properly) ---*;
data &data_ab;
  merge _______a _______b (drop=__cnt_a2);
run;

*--- Delete the data sets used that are ---*;
*--- no longer needed ---*;
proc datasets;
  delete _______a _______b
         ___a_cnt ___b_cnt __ab_cnt;
run;
quit;

%mend;
*-------------------------------------------*;


*-------------------------------------------*
 This macro performs a many-to-many merge.  The
 first two macro variables to be input contain
 the names of the original data sets.  The final
 data set will contain all of the observations
 in both data sets.  Each observation in the
 first data set will contain every observation
 from the second data set with the same ID
 variable.  Similarly, each observation in the
 second data set will contain every observation
 from the first.  The third macro variable is
 the name of the ID variable which the data sets
 will be merged by.  The last macro variable is
 the name of the final many-to-many merged data
 set.
*-------------------------------------------*;
%macro keep_eit (data_a, data_b, id, data_ab);

 *--- Renames the first original data set ---*;
 *--- so it will not be changed ---*;
 data _______a;
   set &data_a;
 run;

 *--- Sort the first original data set by ---*;
 *--- the ID variable ---*;
 proc sort  data= _______a;
   by &id;
 run;

 *--- Creates a data set that counts the ---*;
 *--- number of obs w/ each different ID ---*;
 *--- ONLY one obs w/ each unique ID is ---*;
 *--- kept ---*;
 data ___a_cnt;
   set _______a (keep=&id);
   by &id;
   retain ___cnt_a;
   if (&id ne lag1(&id))
     then ___cnt_a= 1;
   else
     ___cnt_a= ___cnt_a + 1;
   if (last.&id);
 run;

 *--- Renames the second original data set ---*;
 *--- so it will not be changed ---*;
 data _______b;
```

```
  set &data_b;
run;

*--- Sort the second original data set by ---*;
*--- the ID variable ---*;
proc sort  data= _______b;
  by &id;
run;

*--- Creates a data set that counts the ---*;
*--- number of obs w/ each different ID ---*;
*--- ONLY one obs w/ each unique ID is ---*;
*--- kept ---*;
data ___b_cnt;
  set _______b (keep=&id);
  by &id;
  retain ___cnt_b;
  if (&id ne lag1(&id))
    then ___cnt_b= 1;
  else
    ___cnt_b= ___cnt_b + 1;
  if (last.&id);
run;

*--- Merges (one-to-one) the count data ---*;
*--- sets ---*;
data __ab_cnt;
  merge ___a_cnt ___b_cnt;
  by &id;
  if (___cnt_a = .)
    then ___cnt_a= 1; *for obs w/ an ID in B
                       but not in A;
  if (___cnt_b = .)
    then ___cnt_b= 1; *for obs w/ an ID in A
                       but not in B;
run;

*--- Expands each obs in A by the number ---*;
*--- of obs w/ that ID in B ---*;
data _______a;
  merge _______a __ab_cnt (drop=___cnt_a);
  by &id;
  do __cnt_b2= 1 to ___cnt_b;
    output;
  end;
  drop
    ___cnt_b
    __cnt_b2;
run;

*--- Expands each obs in B by the number ---*;
*--- of obs w/ that ID in A ---*;
data _______b;
  merge _______b __ab_cnt (drop=___cnt_b);
  by &id;
  do __cnt_a2= 1 to ___cnt_a;
    output;
  end;
  drop
    ___cnt_a;
run;

*--- Sorts B so that each element in B is ---*;
*--- lined up w/ each element in A ---*;
proc sort  data= _______b;
  by &id __cnt_a2;
run;

*--- Merges the two data sets (which are ---*;
*--- lined up properly) ---*;
data &data_ab;
  merge _______a _______b (drop=__cnt_a2);
run;

*--- Delete the data sets used that are ---*;
*--- no longer needed ---*;
proc datasets;
  delete _______a _______b
         ___a_cnt ___b_cnt __ab_cnt;
run;
 quit;

%mend;
*-------------------------------------------*;


*-------------------------------------------*
 This macro performs a many-to-many merge.  The
 first two macro variables to be input contain
 the names of the original data sets.  The final
 data set will contain ONLY the observations
 with matching ID variables in both data sets.
 Each observation in the first data set that has
 a corresponding ID variable in the second will
 be included in the final data set.  Similarly,
 each observation in the second data set that
 has a corresponding ID variable in the first
 will be included in the final data set.  The
 third macro variable is the name of the ID
 variable which the data sets will be merged
 by.  The last macro variable is the name of
 the final many-to-many  merged data set.
*-------------------------------------------*;
%macro keep_bth (data_a, data_b, id, data_ab);

 *--- Renames the first original data set ---*;
 *--- so it will not be changed ---*;
 data _______a;
   set &data_a;
 run;

 *--- Sort the first original data set by ---*;
 *--- the ID variable ---*;
 proc sort  data= _______a;
   by &id;
 run;

 *--- Creates a data set that counts the ---*;
 *--- number of obs w/ each different ID ---*;
 *--- ONLY one obs w/ each unique ID is ---*;
 *--- kept ---*;
 data ___a_cnt;
   set _______a (keep=&id);
   by &id;
   retain ___cnt_a;
   if (&id ne lag1(&id))
     then ___cnt_a= 1;
   else
     ___cnt_a= ___cnt_a + 1;
   if (last.&id);
 run;

 *--- Renames the second original data set ---*;
 *--- so it will not be changed ---*;
 data _______b;
   set &data_b;
 run;

 *--- Sort the second original data set by ---*;
 *--- the ID variable ---*;
 proc sort  data= _______b;
   by &id;
 run;

 *--- Creates a data set that counts the ---*;
 *--- number of obs w/ each different ID ---*;
 *--- ONLY one obs w/ each unique ID is ---*;
 *--- kept ---*;
 data ___b_cnt;
   set _______b (keep=&id);
   by &id;
   retain ___cnt_b;
   if (&id ne lag1(&id))
     then ___cnt_b= 1;
   else
     ___cnt_b= ___cnt_b + 1;
   if (last.&id);
 run;

 *--- Merges (one-to-one) the count data ---*;
```

```
 *--- sets ---*;
 data __ab_cnt;
   merge ___a_cnt (in=in_a) ___b_cnt (in=in_b);
   by &id;
   if (in_a and in_b); *keeps only obs in the
both data sets;
 run;

 *--- Expands each obs in A by the number ---*;
 *--- of obs w/ that ID in B ---*;
 data _______a;
   merge _______a __ab_cnt
         (in=in_ab drop=___cnt_a);
   by &id;
   if (in_ab);
   do __cnt_b2= 1 to ___cnt_b;
     output;
   end;
   drop
     ___cnt_b
     __cnt_b2;
 run;

 *--- Expands each obs in B by the number ---*;
 *--- of obs w/ that ID in A ---*;
 data _______b;
   merge _______b __ab_cnt
         (in=in_ab drop=___cnt_b);
   by &id;
   if (in_ab);
   do __cnt_a2= 1 to ___cnt_a;
     output;
   end;
   drop
     ___cnt_a;
 run;

 *--- Sorts B so that each element in B is ---*;
 *--- lined up w/ each element in A ---*;
 proc sort  data= _______b;
   by &id __cnt_a2;
 run;

 *--- Merges the two data sets (which are ---*;
 *--- lined up properly) ---*;
 data &data_ab;
   merge _______a _______b (drop=__cnt_a2);
 run;

 *--- Delete the data sets used that are ---*;
 *--- no longer needed ---*;
 proc datasets;
   delete _______a _______b
          ___a_cnt ___b_cnt __ab_cnt;
 run;
 quit;

%mend;
*--------------------------------------------*;
```