Countdown of the Top 10 Ways to Merge Data

David Franklin, TheProgrammersCabin.com, Litchfield, NH

ABSTRACT

JJoining or merging data is one of the fundamental actions carried out when manipulating data to bring it into a form for either storage or analysis. The use of the MERGE statement inside a datastep is the most common way data is merged within the SAS® language but there are others.

Based on an unscientific poll of SAS programmers and others who use SAS, this paper gives a countdown of the top 10 ways to merge two datasets, introducing the SAS code needed to join the datasets into one. Although discussion is focused on a one-to-one or one-to-many merge, some of these methods can be, and will be indicated as such, adapted to a many-to-many merge.

INTRODUCTION

Merging variables from one dataset into another is one of the basic data manipulation tasks that a SAS programmer has to do. The most common way to merge on data is using the MERGE statement in the DATA step but there are six other ways that can help. First though, some data:

```
Dataset: PATDATA
SUBJECT  TRT_CODE
124263      A
124264      A
124265      B
124266      B

Dataset: ADVERSE
SUBJECT  EVENT
124263   HEADACHE
124266   FEVER
124266   NAUSEA
124267   FRACTURE
```

This data will be used throughout the paper for each method described.

MERGING THE DATA

10. PEEK(C) AND POKE

Starting the countdown at number ten is a rather obscure and rarely heard from duo who have been around since data was first put on electronic computer, PEEK and POKE. It must be noted that the ARRAY statement sets aside the memory necessary to contain the data and uses CALL POKE to load it. When it comes to checking the comparison and getting the treatment code, the PEEKC (PEEK function for characters) is used.

```
DATA alldata0;
   ARRAY f{&xpatdata.} $6 _TEMPORARY_;
      /*Store SUBJECT values*/
   ARRAY g{&xpatdata.} $1 _TEMPORARY_;
      /*Store TRT_CODE values*/
   LENGTH trt_code $1;
   DO i=1 TO &xpatdata.;
      SET patdata (RENAME=(trt_code=trt_dict));
      CALL POKE(CATS(subject),
                ADDR(f[1])+((i-1)*6),6);
      CALL POKE(CATS(trt_dict),
                ADDR(g[1])+((i-1)*1),1);
   END;
   DO i=1 TO &xadverse.;
      SET adverse;
      trt_code='';
      DO j=1 TO &xpatdata.;
         IF subject=PEEKC(ADDR(f[1])+((j-1)*6),6) THEN DO;
            trt_code=PEEKC(ADDR(g[1])+((j-1)*1),1);
            OUTPUT;
         END;
         IF ^MISSING(trt_code) THEN LEAVE;
      END;
      IF MISSING(trt_code) THEN OUTPUT;
   END;
   DROP i j trt_code_dict;
RUN;
```

There is a bit of arithmetic is involved in calculating the actual memory address, but the first physical address for array f is gathered from the statement addr(f[1]), and similarly for array g using the statement addr(g[1]). During actual production, using the PEEK and CALL POKE functions have resulted in merges between 30 and 40 times faster that using the ARRAY method above. This method may be used for a many-to-many merge.

9. MERGE WITH UPDATE

At number nine we have the UPDATE statement, something that sounds like it should be easy to use but has a little trouble with handling multiple observations in the master dataset - it is well documented that for the master dataset to be updated correctly there must be a unique key in the master dataset.

One way to get around this is to use the RETAIN statement where a temporary variable carries the TRT_CODE value across multiple records, as the following example demonstrates:

```
DATA alldata0 (DROP=_trt_code);
   LENGTH _trt_code $1; /*Temporary variable containing TRT_CODE*/
   RETAIN _trt_code '';
   UPDATE adverse (in=a) patdata;
   BY subject;
   IF a;
   IF FIRST.subject THEN _trt_code=trt_code;
   ELSE trt_code=_trt_code;
RUN;
```

A WARNING message indicating that "The MASTER data set contains more than one observation for a BY group" will appear but will be redundant since the value is carried from one observation to another by the variable _TRT_CODE that is defined using the RETAIN statement - this message cannot be suppressed since there is no option in SAS to stop WARNING messages from appearing in the SAS LOG.

8. MERGE WITH ARRAY

Loading the dataset with unique records into an array and then do the match is at position number eight on our countdown. The following code will do the merge needed:

```
DATA _null_;
   * Get the number of observations in PATDATA and ADVERSE;
   SET sashelp.vtable;
   WHERE libname='WORK' and memname in('PATDATA','ADVERSE');
   CALL SYMPUT('X'||memname,put(nobs,8.));
DATA alldata0;
   LENGTH trt_code $1;
   * Create a two dimensional array with the subject numbers
     and treatment codes, then load;
   ARRAY f{&xpatdata.,2} $6 _TEMPORARY_;
   DO i=1 TO &xpatdata.;
      SET patdata (RENAME=(trt_code=trt_code_dict));
      f{i,1}=PUT(subject,6.);
      f{i,2}=trt_code_dict;
   END;
   * Go though each record of the dataset ADVERSE and find a
     match -- if one exists, output;
   DO i=1 TO &xadverse.;
      SET adverse;
      trt_code='';
      DO j=1 TO &xpatdata.;
         IF subject=INPUT(f(j,1),best.) THEN DO;
           trt_code=f{j,2};
           OUTPUT;
         END;
         IF ^MISSING(trt_code) THEN LEAVE;
      END;
```

```
         IF MISSING(trt_code) THEN OUTPUT;
      END;
      DROP i j trt_code_dict;
   RUN;
```

The first datastep finds the number of records within PATDATA and ADVERSE so that the correct number of elements can be set for the array and the correct number of iterations is used when calling the ADVERSE dataset. The method above does have one surprising feature - the line:

```
   IF subject=INPUT(f(j,1),best.) THEN DO;
```

where the actual compare is done, can be changed to use any comparison, whether it be an INDEX function or greater than/less than operators.

This technique will work with a many-to-many merge with a bit of tinkering of the code.

7. MERGE WITH MODIFY

The MODIFY statement technique is interesting as it is necessary to do a loop within a loop due to the ADVERSE dataset having multiple records per subject. This technique is at number seven on the chart. The code below does the merge required:

```
DATA adverse alldata0;
   DO p=1 TO totobs;
      _iorc_=0;
      SET patdata point=p nobs=totobs;
      DO WHILE(_iorc_=%sysrc(_sok));
         MODIFY adverse KEY=subject;
         SELECT (_iorc_);
            WHEN (%sysrc(_sok)) DO; /*Match Found*/
               SET patdata POINT=p; OUTPUT alldata0; END;
            WHEN (%sysrc(_dsenom)) _error_=0; /*No Match*/
            OTHERWISE DO; /*A major problem somewhere*/
               PUT 'ERR' 'OR: _iorc_ = ' _iorc_ /
                   'program halted.'; _error_ = 0;
               STOP;
            END;
         END;
      END;
   END;
   STOP;
RUN;
```

It is necessary to note that this method creates the ALLDATA0 dataset but it is necessary to put that as the second dataset in the DATA statement. Note also that the dataset ADVERSE has an index called SUBJECT applied before the datastep is run.

6. MERGE WITH HASH TABLE

Coming in at number six is the Hash Table. Trumpeted by SAS since version 9.1, and used by database programmers in other languages, this is considered one of the fastest ways to merge data in two datasets. Many papers have been written about this recent feature, how it works, and their use within SAS - references to some notable papers are the Reference section below. The code below does the merge required:

```
DATA alldata0;
   IF _n_=0 THEN SET patdata;
   IF _n_=1 THEN DO;
      DECLARE HASH _h1 (dataset: "PATDATA");
      rc=_h1.definekey("SUBJECT");
      rc=_h1.definedata("TRT_CODE");
      rc=_h1.definedone();
      call missing(SUBJECT,TRT_CODE);
   END;
   SET adverse;
   rc=_h1.find();
   IF rc^=0 THEN trt_code=" ";
   DROP rc;;
RUN;
```

In the example above, the dataset PATDATA gets loaded into a hash table, then the ADVERSE dataset is loaded into the datastep and the match is made using the FIND() method.

A LITTLE BREAK IN THE COUNTDOWN ...

Now that we are half way though our countdown, lets just take a moment to view a rather obscure method for merging our two datasets, the use of the PROC EXECUTE:

```
DATA _null_;
   SET patdata;
   CALL EXECUTE("DATA alldat;"||
                " SET adverse;"||
                " WHERE subject='"||STRIP(subject)||"';"||
                " trt_code='"||STRIP(trt_code)||"';"||
                "PROC APPEND BASE=alldata0 DATA=dat0 FORCE;"||
                "RUN;");;
RUN;
```

This method uses CALL EXECUTE to add TRT_CODE from PATDATA to ADVERSE by SUBJECT, appending the result each time to the dataset ALLDATA0. Unfortunately this method will only produce a dataset with the intersection of data from PATDATA and ADVERSE, but is something to occasionally use, particularly if you have only a few observations in dataset PATDATA.

5. POINT OPTION IN THE SET STATEMENT

Continuing with the countdown, we are now at number five. The POINT option in the SET statement gives the most control of all the techniques here on out, with one major benefit being that it will do a many-to-many merge:

```
DATA alldata0;
   SET ae;
   DROP _: match; * Drop temporary variables;
   match=0;
   DO i=1 TO xnobs;
      SET patdata (rename=(subject=_subject)) NOBS=xnobs POINT=i;
      IF subject=_subject THEN DO;
         match=1; OUTPUT; END;
   END;
   IF match=0 THEN DO; * Output AE record if no match in CM;
      CALL MISSING(trt_code); OUTPUT; END;
 RUN;
```

The key to this method is that the dataset takes each observation in AE and then tries to match this observations with an observation in PATDATA -- if there is a match, then the observation will be output, and if there is no match the AE record will still be output with no value for the variable TRT_CODE. Because all observations in AE are read and matched will all observations in PATDATA, it is possible to do a many to many merge -- effectively, it is loop within a loop.

4. MERGE WITH FORMAT

Coming in at number four on our countdown is a method that creates a format from the PATDATA dataset and and sets the treatment from the created format:

```
DATA fmt;
   RETAIN fmtname 'TRT_FMT' type 'C';
   SET patdata;
   RENAME subject=start trt_code=label;
PROC FORMAT CNTLIN=fmt;
DATA alldata0;
   SET adverse;
   ATTRIB trt_code LENGTH=$1 LABEL='Treatment Code';
   trt_code=PUT(subject,$trt_fmt.);
RUN;
```

In the example a character format TRT_FMT is created from the PATDATA dataset, and then this format is used to set the TRT_CODE variable within the ADVERSE dataset. This method is useful as the data does not have to be sorted or indexed beforehand.

3. MERGE WITH SET-KEY

Over the years many options have been added to the SET statement which brings our number three on the chart, using the KEY= option as shown in the following example:

```
DATA alldata0;
   SET adverse;
   SET patdata KEY=subject /UNIQUE;
   DO;
      IF _IORC_ THEN DO;
         _ERROR_=0;
         trt_code='';
      END;
   END;
RUN;
```

Before this type of merge can work the dataset PATDATA must have an index created inside it, using either the INDEX statement inside a DATASETS or SQL procedure, or INDEX option inside a DATA step. It is important to have the DO loop is if no match is found then TRT_CODE will be set to missing - if this is not done then unexpected results may occur.

2. MERGE WITH SQL

At number two of the countdown is something that SAS introduced in SAS version 6.07, SQL -- this gave the ability to merge data using the SQL language. To merge our two datasets and get the expected results, the SQL code would look something similar to that below:

```
PROC SQL;
   CREATE TABLE alldata0 AS
      SELECT a.*, b.trt_code
      FROM adverse a
         LEFT JOIN
           patdata b
      ON a.subject=b.subject;
   QUIT;
RUN;
```

SQL is a well known language that is very good at working with databases and is liked by many who deal with large datasets.

The SQL technique is the most common way a many-to-many merge of data is completed.

1. MERGE IN A DATA STEP

After starting at number ten, we have now come to number one of the countdown, and the winner is of no real surprise. The winner by far, and is therefore the most common way to merge the two datasets is the MERGE statement used inside a datastep, an example of which is given below:

```
DATA alldata0;
   MERGE adverse (in=a)
         patdata (in=b);
   BY subject;
   IF a;
RUN;
```

This method is the most common way of merging data as it gives most control, with the exception of the POINT= option discussed above, in the the way data is to be merged. Most commonly the datastep is preceded by a call to the SORT procedure making sure that both ADVERSE and PATDATA are in the same sort order, but it is possible to use indexed datasets instead but the indexes must be defined before the datastep that does the merging of the data.

CONCLUSION

So our countdown of the Top 10 Ways to Merge Data is now complete. As can been seen there are a number of methods which can be used to merge data, beyond the MERGE statement within a DATA step. No one method is better than another, and the methods shown here are by no means exhaustive. It is only though trying these different methods at your installation that you will see resource efficiencies between the methods.

REFERENCES

Secosky, Jason and Bloom, Janice. 2007. “Getting Started with the DATA Step Hash Object”. SAS Global Forum 2007. Cary, NC : SAS Institute Inc.

Loren, Judy. 2006. “How Do I Love Hash Tables? Let Me Count The Ways!”. NESUG 2006. Portland, ME

CONTACT INFORMATION

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

Name: David Franklin
Enterprise: TheProgrammersCabin.com
Work Phone: 603-275-6809
E-mail: dfranklin@TheProgrammersCabin.com
Web: http://www.TheProgrammersCabin.com