9 Ways to Join Two Datasets

David Franklin, Independent Consultant, New Hampshire, USA

ABSTRACT

Joining 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. This paper looks at nine methods, including the use of the MERGE statement, for a one to one, or one to many merge, introducing the SAS code needed to join to datasets into one.

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

MERGE IN A DATA STEP

The most commonly used statement used when merging data within SAS 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 control the the way data is to be merged. In the example, the ADVERSE and PATDATA records are merged by SUBJECT and those records that come from the ADVERSE dataset are output - this results in subject 124267 not having a TRT_CODE value in the ALLDATA0 dataset, and subjects 124264 and 124265 not being represented in the same. 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.

MERGE WITH SQL

SAS version 6.07 introduced SQL into SAS which gave the ability to merge data using the SQL language. To merge our two datasets and get the same result as in the code above, 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.

MERGE WITH SET-KEY

Over the years many options have been added to the SET statement which brings the third method for merging data, 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 the third example is run 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.

MERGE WITH FORMAT

The fourth method that is useful 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.

MERGE WITH HASH TABLE

Since version 9.1 another possibility that has been available is the use of hash tables. Many papers have been written about this recent feature, how it works, and their use within SAS - references to some notable papers are in 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.

MERGE WITH MODIFY

The MODIFY statement – this is an interesting technique as it is necessary to do a loop within a loop due to the ADVERSE dataset having multiple records per subject:

```
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.

MERGE WITH ARRAY

A variation on the hash table is to load the dataset with unique records into an array and then do the match, as the following example demonstrates:

```
DATA _null_;
   SET sashelp.vtable;
   WHERE libname='WORK' and memname in('PATDATA','ADVERSE');
   CALL SYMPUT('X'||memname,put(nobs,8.));
DATA alldata0;
   LENGTH trt_code $1;
   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;
   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.

MERGE WITH UPDATE

The use of the UPDATE statement is useful but it is necessary to get around the issue of handling multiple observations in the master dataset – it is well documented that for the master dataset to be updated correctly their 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.

CALL EXECUTE

The last merge that is presented in this paper is something that I have seen and at best is only good if you are dealing with cases where you are dealing with small subsets of very large datasets:

```
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.

CONCLUSION

As shown in the paper 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 site that you will see resource efficiencies between the methods.

REFERENCES

Getting Started with the DATA Step Hash Object - Jason Secosky and Janice Bloom, SAS Institute Inc., Cary, NC (SAS Global Forum 2007)

How Do I Love Hash Tables? Let Me Count The Ways! - Judy Loren, Independent Consultant, Portland, ME (NESUG 2006)

CONTACT INFORMATION

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

David Franklin
16 Roberts Road
Litchfield, NH 03052
Cell: 603-275-6809
Email: 100316.3451@compuserve.com
Web: http://www.TheProgrammersCabin.com