ABSTRACT

Often two data sets have data from several dates for numerous subjects. A need to compare related data across the data sets arises. This can be accomplished by a simple merge if the observations in both data sets come from the same dates for each subject. This task becomes more complicated when the dates are close, but not exact. In these instances, one data set is used as the base and the closest observation is selected from the other data set. The DATA step method to do this requires multiple SORTs, a PROC TRANSPOSE, multiple MERGEs, and a complicated DO loop. PROC SQL greatly facilitates this task; two data sets can be merged on the closest match in a single step.

INTRODUCTION

Merging two data sets can be very simple, if there are key variables in each data set on which to merge. Merging becomes more complicated when key variables do not match.

For example, one dataset may have lab data drawn every week, while the other dataset may have quality of life data collected once every three weeks. If the data sets are merged, none of the dates will match. This makes the merge useless for comparing information from one data set to the other. The goal in any such situation is to find and merge on the closest match.

The SAS/BASE MERGE function only allows for exact matches. Code must first be developed that will identify the dates for the closest matches in each data set. Then a merge can be accomplished using these dates. DATA step code to achieve this involves multiple SORTs, a PROC TRANSPOSE, multiple DATA steps, and a complicated DO loop. Utilizing the PROC SQL function, a merge on the closest match can be done in a one step.

For simplicity, this paper will focus on merging two data sets with observations on different dates. This basic method can be generalized to fit other key variables.

DATA STEP METHOD

Merging on the closest match requires that one data set, which will be called dsnA, be used as the base data set. The base dataset is generally the one with fewer observations; that is, the one with data collected at greater intervals. From our example, this would be the quality of life dataset. Then, the closest match from the other data set, which will be called dsnB, must be identified.

Each observation from dsnB must be made available as a possible match for each observation in dsnA. This is accomplished by transposing dsnB and merging onto dsnA. Then, a complicated DO loop must be incremented and used to identify which of the matches is, in fact, the closest. For our example, the code to accomplish this part of the process would be similar to the following:

```
** Find max. # of obs in dsnB, for DO loop **;
    data _null_;
      set dsnB2 end=eof;
      by subject;
      retain count max 0;
      if first.subject then count=1;
      else count+1;
      if count gt max then max=count;
      if eof then do;
        call symput('maxct', trim(left(put(max,2.))));
      end;
    run;

** Merge and find visitdt of closest match **;
    data nextdsn;
      merge dsnA
        transdsnB;
      by subject;
      array labdt date1 - date&maxct;
      array store store1 - store&maxct;
      do i=1 to &maxct;
        if visitdt ne . then do;
          store(i)=labdt(i)-visitdt;
          if store(i) ne .
            and meas=. then do;
              test=store(i);
              meas=i;
            end;
        else if store(i) ne .
          and abs(store(i)) lt abs(test)
            then do;
              test=store(i);
              meas=i;
            end;
        else if test lt store(i) then do;
          test=test;
          meas=meas;
        end;
      end;
      mergedt=labdt(meas);
    run;
```

After the closest match has been identified and stored in the variable mergedt, and after an additional SORT, the data sets can finally be successfully merged. For our example, the code to accomplish the second part of the process would be:

```
** Sort the new dsnA by closest match **;
    proc sort data=nextdsn;
      by subject mergedt;
    run;

** Merge using the closest match **;
    data finaldsn;
      merge nextdsn(in=a) 
        dsnB(rename=(visitdt=mergedt)); 
      by subject mergedt; 
      if a;
    run;
```
SQL METHOD
The steps required to merge on the closest match can be significantly reduced using PROC SQL. The FULL JOIN function of PROC SQL, allows every observation in dsnB to be matched to each observation in dsnA, replacing the first three steps in the DATA step method. PROC SQL’s MIN and GROUP BY functions identify the closest match for each observation in dsnA as the minimum difference between dates, replacing the complicated DO loop. Once the closest match is identified, PROC SQL’s HAVING function keeps only the closest match in the final data set, replacing the final SORT and MERGE.

All of the previous data step coding can be simplified to the following PROC SQL code:
proc sql;
create table finaldsn as
select a.*,
b.*
a.visitdt-b.labdt as days
, abs(calculated days) as absdays
, min(calculated absdays) as close
from dsnA as a
full join
dsnB(rename=(visitdt=labdt subject=id)) as b
on a.subject=b.id
where a.visitdt < b.labdt and b.labdt < a.subject
and b.id in (select distinct subject
: from dsnA
)
group by a.subject, a.visitdt
having calculated absdays=calculated close
;
quit;

Note that selecting only the necessary variables from dsnA and/or dsnB, instead of selecting a.* and b.*, can reduce run-time.

There are a few other issues to consider in order to achieve the best results with the PROC SQL method. They include the following: (1) Observations with missing visitdt in dsnA or dsnB will NOT be retained in the final data set; (2) Multiple closest matches will ALL be retained in the final data set; and (3) The closest match will be found and made even if there is a very large difference. Solutions for each of these issues are provided as variations on the basic method presented above.

VARIATION 1 – MISSING OBSERVATIONS
If either dsnA or dsnB has observations with missing visitdt, these observations will not be included in the final data set. In general, this is not a problem because the match is being made on the closest date. If the date is missing, there is no way to determine which is closest. If, however, these observations are needed in the final data set, they can be retained by subsetting the observations with missing visitdt from dsnA and/or dsnB before finding the closest match. Then, dsnA and dsnB can be passed through the PROC SQL step as usual. Once the data sets have been merged, the missing observations can be added back into the final data set with a simple SET statement.

VARIATION 2 – MULTIPLE MATCHES
If the final data set has more observations than the original dsnA, multiple matches have been identified for one or more observations. Multiple matches can be caused by two problems in dsnB: (1) Some subjects in dsnB have duplicate visitdt’s; or (2) Two observations in dsnB are equidistant from an observation in dsnA, one before and one after. If both matches are not desired, additional steps must be taken to remove them.

For duplicate observations in dsnB, unduplicating the data set BEFORE merging on the closest match is usually preferable. Observations can be selected after the merge is made, but selecting before the merge can reduce the PROC SQL run-time. For equidistant observations, a simple PROC SORT and a FIRST. (to keep the match on the date in dsnB AFTER the date in dsnA) or LAST. (to keep the match on the date in dsnB BEFORE the date in dsnA) can be used to keep only one match for each observation in dsnA. For our example, to select unique matches, keeping the match on the date in dsnB after the date in dsnA for equidistant matches, the code would be the following:
proc sort data=finaldsn;
by subject visitdt;
run;
data finaldsnB;
set finaldsn;
by subject visitdt;
if first.visitdt;
run;

VARIATION 3 – DEFINING “CLOSE” FOR MATCHES
For every observation in dsnA, a “closest” match will be identified from dsnB. Depending on the data, this “closest” match could be too far to be accurate. To prevent large differences in the matched variables, a window can be established. For our example, it can be specified that all matches must be within 2 weeks. To add this requirement, simply make an addition to the HAVING statement. Instead of
having calculated absdays=calculated close
use
having calculated absdays=calculated close and calculated absdays le 14

When employing a window to select matches, it is important to note that if no match can be found within the window for an observation in dsnA, that observation will not be included in the final data set. To recover observations lost from dsnA, merge the finaldsn back with the original dsnA. In some situations, it is quicker and easier to allow all matches to be made and then subset the final data set before analyzing any variables. For example,
proc freq data=finaldsn
(where=(absdays le 14));
tables b_var2*a_var2/out=freq2;
run;

CONCLUSION
PROC SQL provides an alternate method for merging data on the closest match. Although the various issues may seem to be a drawback to the PROC SQL method, duplicates, missing values, and other data issues become immediately apparent. This is unlike the DATA step method where problems in the data can be easily overlooked because no flags are raised.

Though the runtime for both methods is similar, using PROC SQL reduces the amount of code that must be written. The PROC SQL method also provides an excellent tool for validating merges achieved with the DATA step method.

CONTACT INFORMATION
Your comments and questions are valued and encouraged. Contact the author at:
Erin Christen
STATPROBE, Inc.
10052 Mesa Ridge Court
Suite 200
San Diego, CA 92121
Work Phone: (865) 522-0993
Email: erin.christen@sd.statprobe.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.