Paper: 98-2010

Set, Match, Merge … Don't You Love SAS®

Peter Eberhardt, Fernwood Consulting Group Inc, Toronto ON, Canada
Ying Liu, Toronto ON, Canada

ABSTRACT

One table has accounts, another has new transactions. One table has patient data, another has treatments. No matter what your business or application it is rare that a single data set or table has everything you need to answer the questions you are expected to answer. SAS® has a simple and powerful mechanism to bring your tables together: the DATA Step Merge. But this simple operation can easily run awry. In this paper we will explain the basics of the DATA step merge and the issues and problems you will encounter if you do not understand its workings. This paper is geared towards beginning SAS programmers, but it can be a useful refresher for anyone.

INTRODUCTION

The SAS DATA Step provides a wealth of opportunity to bring together or combine data from multiple input datasets; datasets can be concatenated, interleaved or merged. Although the mechanics of these operations is relatively straightforward, there are many "gotcha's" that can creep in when the unwary programmer does not pay attention to the messages in the log. This paper will outline some of the basic DATA Step methods for combining datasets and highlight some of the common problems that can occur; although the paper will address combining and interleaving datasets, its main focus will be on merging datasets. After looking at the DATA Step merge, the paper will also address some of the alternatives to using the DATA Step merge.

TERMINOLOGY

SAS DATA Step programmers are used to certain terms when discussing data. In this paper I will be using some DATA Step and some SQL terminology interchangeably. The table below shows the common DATA Step terms and the SQL equivalent.

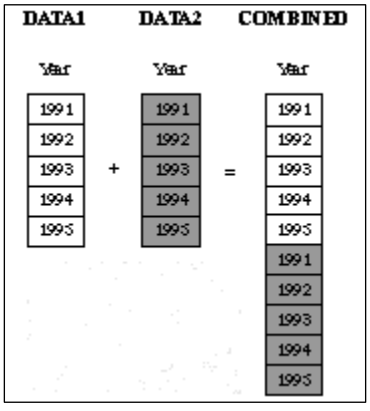
DATA Step Term	SQL Term
dataset	Table
field, variable	Column
observation, record	Row
merge	Join

DATA

Appendix A has the layout and contents of the datasets used in the paper. These are small datasets designed to make it easy to see good and bad results. The purpose of the paper is to show techniques so the small datasets will facilitate that.

CONCATENATING DATASETS

Concatenating datasets essentially means stacking one dataset on top of the other; that is, given two datasets, all of the records from the second dataset will be added to the end of the first as depicted in the following diagram:



In general, when concatenating datasets we would expect the datasets to have identical structure but different contents; by structure we mean the tables would have the same columns names and the columns would have the same type (numeric or character). For example, we might have monthly sales records in tables JAN, FEB, MAR etc. and want to concatenate them into a year-to-date (YTD) table. Code to do this would look something like:

```
DATA YTD;
     set JAN FEB MAR;
run;
```

When processing this data step essentially the following steps are taken by SAS:

- SAS opens dataset JAN and adds its columns to the program data vector (PDV)
- SAS opens the dataset FEB, any columns not in JAN are added to the PDV
- SAS opens the dataset MAR, any columns not in JAN or FEB are added to the PDV
- All of the records from the JAN dataset are read and output
- All of the records from the FEB dataset are read and output
- All of the records from the MAR dataset are read and output

The resulting dataset, YTD, has all of the records and all of the columns from JAN, FEB and MAR with the JAN records first, followed by the FEB, then the MAR records; that is, the records are added in the order of the dataset list. Moreover, if a column exists in one or more of the datasets but not in another, that column is included in all of the output records but with a missing value for all of the records in the table(s) that did not have that column, For example, suppose that in the MAR table a new column called DISCOUNT was added. Since DISCOUNT did not exist in the JAN and FEB tables all of the records in YTD that came from these tables would have a missing value for DISCOUNT. The log below shows the results of this DATA Step code.

```
104  data YTD;
105        set JAN FEB MAR;
106  run;

NOTE: There were 10 observations read from the data set WORK.JAN.
NOTE: There were 10 observations read from the data set WORK.FEB.
NOTE: There were 10 observations read from the data set WORK.MAR.
NOTE: The data set WORK.YTD has 30 observations and 5 variables.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

In the example code, notice there is only ***one*** SET statement, with the list of datasets to be concatenated . What would happen if the following code was submitted? That is, three **`set`** statements instead of one.

```
DATA YTD;
     set JAN;
     set FEB;
     set MAR;
run;
```

SAS would read the first record from the JAN dataset (set JAN;), then read the first record from the FEB dataset (set FEB;). If we assume the datasets have the same fields, that is SALEDATE, REP, PRODUCT and QUANTITY, then the first time through after the **`set JAN`** command is executed, these fields would have the values from the JAN data. After the **`set FEB`** command is executed the values in these fields would be overwritten with the values from

the FEB data. Similarly, when the **set MAR** command is executed, these fields would be over written with the values from the MAR data. At the end of the DATA Step, a record is written to YTD; this record would have the values of the first MAR record. In this case, three records were read from three different tables, but only the last record was written out. This behaviour would continue until one of the datasets ran out of records; at the end of that DATA Step cycle the DATA Step would stop executing and the YTD table would be complete. The log below shows the results of this DATA Step code.

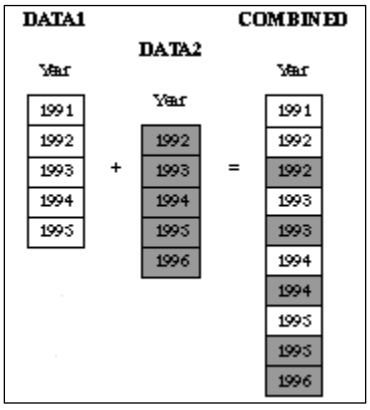
```
107  data YTD;
108       set JAN;
109       set FEB;
110       set MAR;
111  run;

NOTE: There were 10 observations read from the data set WORK.JAN.
NOTE: There were 10 observations read from the data set WORK.FEB.
NOTE: There were 10 observations read from the data set WORK.MAR.
NOTE: The data set WORK.YTD has 10 observations and 5 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

Notice there are no errors generated since the code is syntactically correct. However, the final dataset has only 10 records, not the 30 records we would have if we concatenated the datasets. The moral here is to carefully check your log to be sure the numbers add up.

If you goal is just to concatenate the tables and no other processing is to be done in the DATA Step, then you should consider using PROC APPEND, or PROC DATASETS with append; these alternatives are slightly more efficient.

INTERLEAVING DATASETS

When we concatenated datasets, records were added to the output in the order in which they were found – the JAN records followed by the FEB records etc...What if we wanted to do something similar, stack the tables, but the tables we want to combine are ordered by *saledate* and we want the results ordered by *saledate*? For example, each table represents the data from a different store, each store sends the data ordered by *saledate*, and we want to combine them by *saledate*. That is, we want the output in *saledate* order, not in the order of the datasets in the SET statement that we had when we concatenated the tables. Once again, all of the tables will have the same structure.



First, we could concatenate them, and then sort the result. When working with small datasets this might be a viable alternative. However, when working with large datasets we can incur a significant processing cost with the final sort. The better alternative is to interleave them. Interleaving requires we have the input tables sorted by a common key column. The code to interleave looks something like:

```
DATA YTDstores;
     set store1 store1 store3;
     by saledate;
run;
```

Once again, as can be seen from the log below we have 30 records in our output dataset.

```
281  data ytdStores;
```

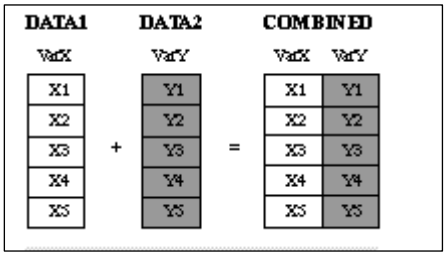
```
282       set store1 store2 store3;
283       by saledate;
284  run;

NOTE: There were 10 observations read from the data set WORK.STORE1.
NOTE: There were 10 observations read from the data set WORK.STORE2.
NOTE: There were 10 observations read from the data set WORK.STORE3.
NOTE: The data set WORK.YTDSTORES has 30 observations and 6 variables.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

CONCATENATION AND INTERLEAVING SUMMARY

Concatenating and Interleaving are two method of stacking datasets; that is, the output dataset has all of the records of each of the input datasets. Moreover, the input datasets usually have the same structure (i.e. same columns), although we saw that if a column exists in one of the input datasets but not in any of the others, it is included in the output dataset with missing values in the records that did not have it originally. The difference between concatenating and interleaving is that with concatenating the output dataset has all of the records of the first dataset followed by all of the records of the second dataset and so on, whereas with interleaving, the output dataset is in the order of the "*BY*" (or key) field.

MERGING DATASETS

We saw that concatenating and Interleaving datasets allowed us to combine multiple input datasets and create an output dataset that was the sum of the input records, and that the input datasets had the same basic structure. Another way of combing datasets is to merge them; in this case the input datasets will have different structures, that is different columns, and the output will contain all of the columns from input datasets. The following figure demonstrates this.



Here we see the two input tables have different structures, and the output table has the columns from each of the tables. In simplified terms, concatenating and interleaving resulted in output that was the same 'width' as the input but was much 'taller', merge results in output that is much 'wider' than the input, but is no 'taller'. Let's look at some of the variety we have when it comes to merging tables.

MERGING DATASETS – ONE TO ONE MERGING

As the name suggests, One to One Merging is the process of matching the first record on the first dataset with the first record of the second dataset, and so on. A DATA Step to do this would look something like:

```
data merged;
     merge class
           reward;
run;

ods rtf body='c:\temp\Merge.rtf';
title1 "Results of Merge ";
```

```
title2 "Not Sorted BY values";
proc print data= Merged;
run;
ods rtf close;
```

SAS will match the first record in ***class*** with the first record in ***reward*** and write out a record. This process is repeated until one of the datasets runs out of records. The results from the PROC PRINT looks like:

Results of Interleave
Not Sorted BY values

Obs	id	student	reward
1	a	Andrew	Agate
2	b	Bob	Bauble
3	c	Charlie	Coaster
4	d	Doug	Duster
5	e	Edward	Eagle
6	f	Fred	Firetruck

Normally, when doing a One to One matching the input datasets will have the same order, as the example showed. But problems arise when the order in the two datasets gets misaligned as shown in the following example:

```
data Merge_oops;
      merge class
            reward_oops;
run;

ods rtf body='c:\temp\ Merge_oops.rtf';
title1 "Results of Merge";
title2 "Not Sorted BY values and Records Misaligned";
proc print data= Merge_oops;
run;
ods rtf close;
```

In this case our ***reward*** table has two records that had their order switched; records for ***id***'s ***c*** and ***d*** are switched. The SAS log gives us no error since the program is correct. However, as we can see from the output of PROC PRINT, the output is not correct. Note that Charlie has a value of ***d*** (not ***c***) for his ***id*** and that Doug has a value of ***c*** (not ***d***) for his ***id***.

Results of Merge
Not Sorted BY values and Records Misaligned

Obs	id	student	Reward
1	a	Andrew	Agate

2	b	Bob	Bauble
3	d	Charlie	Duster
4	c	Doug	Coaster
5	e	Edward	Eagle
6	f	Fred	Firetruck

MERGING DATASETS – MATCH MERGING

Fortunately there is a simple solution to this problem. Since there is a common key value, id, we can sort the input datasets by the key value before the merge, then merge the datasets on this key value. In this way the records for each ***id*** will match properly. The following code shows the sort, merge, and print:

```
proc sort data=class;
     by id;
run;

proc sort data=reward;
     by id;
run;

data Merge_sort;
      merge class
            reward;
     by id;
run;

ods rtf body='c:\temp\Merge_sort.rtf';
title1 "Results of Merge";
title2 "Sorted BY values";
proc print data=Merge_sort;
run;
ods rtf close;
```

The following table shows the result of PROC PRINT. Notice that it looks just like our original merge, but now we can be assured that the ***student*** and ***reward*** for each ***id*** is properly aligned.

Results of Merge
Sorted BY values

Obs	id	student	reward
1	a	Andrew	Agate
2	b	Bob	Bauble
3	c	Charlie	Coaster
4	d	Doug	Duster
5	e	Edward	Eagle
6	f	Fred	Firetruck

So far we have had data where we have a match for the common key. Let's examine what happens when we have values for our key field in one of the tables but not on the other. The following example shows merging tables where we have some common by values in both table and some in one table and not the other.

```
data merge_miss;
      merge class_miss
            reward_miss;
     by id;
run;

ods rtf body='c:\temp\merge_miss.rtf';
title1 "Results of Merge";
title2 "With Different BY values";
proc print data=merge_miss;
run;
ods rtf close;
```

Results of Merge
With Different BY values

Obs	id	student	reward
1	a	Andrew	Agate
2	b		Bauble
3	c	Charlie	Coaster
4	d	Doug	
5	e	Edward	Eagle
6	f		Firetruck

In this case the results are not unsurprising. For those records that had a matching ***id*** value (eg the first observation), we get the appropriate value for ***student*** and ***reward***. In the case where the record came from one of

the tables but not the other then we get a missing value for the field from the table that did not have a matching ***id***. For example the ***id*** value ***b*** (see observation 2) was in the reward table but not the class table; the value of the ***student*** column is missing since there was no matching ***id*** in the class table. Similarly for observation 4, there is a value for the ***student*** column but the ***reward*** column has a missing value because there was an ***id*** in the class table but no matching ***id*** in the reward table.

We have seen some simple examples of the DATA Step merge highlighting its ease and some of the possible problems that can arise with its use. The next section will introduce some alternatives to the DATA Step merge. It will be left as an exercise for you to track down more information on these alternatives.

MERGING DATASETS – SOME ALTERNATIVES

The first alternative to a DATA Step merge you will probably come across is a SQL join. The general syntax of a SQL join that is similar to our sorted merge above would be:

```
proc sql;
create table merge_sql as
select c.id,
       c.student,
       r.reward
from class as c inner join reward as r
on c.id = r.id
order by c.id
;
quit;

ods rtf body='c:\temp\merge_sql.rtf';
title1 "Results of Merge";
title2 "Using SQL";
proc print data=merge_sql;
run;
ods rtf close;
```

Results of Merge
Using SQL

Obs	id	student	reward
1	a	Andrew	Agate
2	b	Bob	Bauble
3	c	Charlie	Coaster
4	d	Doug	Duster
5	e	Edward	Eagle
6	f	Fred	Firetruck

This program uses an ***inner-join*** that is a join where only records that are in both tables are kept. PROC SQL has a variety of other types of joins; refer to the SAS documentation for more information on using PROC SQL;

Another common approach to joining tables is to not join them at all! In many cases the purpose of the join is to do a look-up. For example an employee record would have a department code and we would merge with the department table to get the department name. A simple alternative to this type of merge is to use a SAS format. SAS formats can be easily built from existing tables; using the format, and the SAS ***put()*** function, the lookup value can be easily and efficiently computed. There are numerous papers available on using formats as lookups.

CONCLUSION

This paper was an introduction to combining tables in SAS. It showed that the mechanics of combining tables is very straightforward, but that possible errors can occur in the results, even if there were no errors in the program. It also showed that there are some alternatives to the DATA Step merge that can offer some efficiencies at the expense of learning new techniques. As with almost all projects in SAS, there can be multiple solutions to a 'data combine' problem.

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

Peter Eberhardt
Fernwood Consulting Group Inc
Toronto, ON
peter@fernwod.ca

www.fernwood.ca

Ying Liu
HSBC Financial
Toronto, ON
ying.x.liu@hsbc.ca

APPENDIX A – DATA TABLES AND PROGRAMS

```
data jan;
    attrib saledate format=yymmdd10.
                   label='Date of Sale'
                                ;
    attrib rep length=$2
                   format=$2.
                     label='Sales Rep'
                     ;
    attrib product length=$5.
                    format=$5.
                      label="Product"
                      ;
    attrib quantity format=4.
                        label="Quantity Sold"
                                ;
   keep saledate rep product quantity;
```

```
    saledate = '05jan2007'd;
     do i = 1 to 10;
       saledate = saledate + 1;
       r = int(ranuni(0)*100);
          if      r < 20 then rep = '20';
          else if r < 40 then rep = '40';
          else if r < 70 then rep = '50';
          else                rep = '60';
       p = int(ranuni(0)*1000);
          if      p < 200 then product = 'PROD5';
          else if p < 400 then product = 'PROD4';
          else if p < 700 then product = 'PROD3';
          else                 product = 'PROD2';
       quantity = int(ranuni(0)*10) + 1;
          output;

    end;
run;
data feb;
     attrib saledate format=yymmdd10.
                    label='Date of Sale'
                                ;
     attrib rep length=$2
                    format=$2.
                       label='Sales Rep'
                       ;
     attrib product length=$5.
                     format=$5.
                        label="Product"
                        ;
     attrib quantity format=4.
                         label="Quantity Sold"
                               ;
    keep saledate rep product quantity;
    saledate = '02feb2007'd;
     do i = 1 to 10;
       saledate = saledate + 1;
       r = int(ranuni(0)*100);
          if      r < 20 then rep = '20';
          else if r < 40 then rep = '40';
          else if r < 70 then rep = '50';
          else                rep = '60';
       p = int(ranuni(0)*1000);
          if      p < 200 then product = 'PROD5';
          else if p < 400 then product = 'PROD4';
          else if p < 700 then product = 'PROD3';
          else                 product = 'PROD2';
```

```
       quantity = int(ranuni(0)*10) + 1;
         output;

    end;
run;
data mar;
     attrib saledate format=yymmdd10.
                    label='Date of Sale'
                              ;
     attrib rep length=$2
                    format=$2.
                       label='Sales Rep'
                       ;
     attrib product length=$5.
                     format=$5.
                        label="Product"
                        ;
     attrib quantity format=4.
                          label="Quantity Sold"
                              ;
    keep saledate rep product quantity discount;
    retain discount 0.1;
    saledate = '02mar2007'd;
     do i = 1 to 10;
       r = int(ranuni(0)*100);
         if      r < 20 then rep = '20';
         else if r < 40 then rep = '40';
         else if r < 70 then rep = '50';
         else                rep = '60';
       p = int(ranuni(0)*1000);
         if      p < 200 then product = 'PROD5';
         else if p < 400 then product = 'PROD4';
         else if p < 700 then product = 'PROD3';
         else                 product = 'PROD2';
       quantity = int(ranuni(0)*10) + 1;
         output;

    end;
run;

data YTD;
      set JAN FEB MAR;
run;

data YTD;
      set JAN;
      set FEB;
```

```
      set MAR;
run;

data store1;
     attrib store length=$3.
                     format=$3.
                        label="Store"
                        ;
     attrib saledate format=yymmdd10.
                    label='Date of Sale'
                              ;
     attrib rep length=$2
                    format=$2.
                      label='Sales Rep'
                      ;
     attrib product length=$5.
                     format=$5.
                        label="Product"
                        ;
     attrib quantity format=4.
                         label="Quantity Sold"
                              ;
    keep store saledate rep product quantity discount ;
    store = "001";
    retain discount 0.1;
    basedate = '01Jan2007'd;
    seed = 100;
     do i = 1 to 10;
       call ranuni(seed, r);
          if i > 7
          then
            do
           r = ranuni(-1);;
            end;
          r = int(r * 90);
          saledate = basedate + r;
       r = int(ranuni(0)*100);
          if      r < 20 then rep = '20';
          else if r < 40 then rep = '40';
          else if r < 70 then rep = '50';
          else                rep = '60';
       p = int(ranuni(0)*1000);
          if      p < 200 then product = 'PROD5';
          else if p < 400 then product = 'PROD4';
          else if p < 700 then product = 'PROD3';
          else                 product = 'PROD2';
       quantity = int(ranuni(0)*10) + 1;
```

```
          output;
    end;
run;
data store2;
     attrib store length=$3.
                    format=$3.
                       label="Store"
                       ;
     attrib saledate format=yymmdd10.
                    label='Date of Sale'
                              ;
     attrib rep length=$2
                    format=$2.
                      label='Sales Rep'
                      ;
     attrib product length=$5.
                     format=$5.
                       label="Product"
                       ;
     attrib quantity format=4.
                        label="Quantity Sold"
                              ;
    keep store saledate rep product quantity discount ;
    store = "002";
    retain discount 0.1;
    basedate = '01Jan2007'd;
    seed = 100;
     do i = 1 to 10;
       call ranuni(seed, r);
          if i > 7
          then
            do
           r = ranuni(-1);;
            end;
          r = int(r * 90);
          saledate = basedate + r;
       r = int(ranuni(0)*100);
          if      r < 20 then rep = '20';
          else if r < 40 then rep = '40';
          else if r < 70 then rep = '50';
          else                rep = '60';
       p = int(ranuni(0)*1000);
          if      p < 200 then product = 'PROD5';
          else if p < 400 then product = 'PROD4';
          else if p < 700 then product = 'PROD3';
          else                 product = 'PROD2';
       quantity = int(ranuni(0)*10) + 1;
```

```
          output;
     end;
run;
data store3;
     attrib store length=$3.
                    format=$3.
                       label="Store"
                       ;
     attrib saledate format=yymmdd10.
                    label='Date of Sale'
                              ;
     attrib rep length=$2
                    format=$2.
                      label='Sales Rep'
                      ;
     attrib product length=$5.
                     format=$5.
                       label="Product"
                       ;
     attrib quantity format=4.
                         label="Quantity Sold"
                              ;
    keep store saledate rep product quantity discount ;
    store = "001";
    retain discount 0.1;
    basedate = '01Jan2007'd;
    seed = 100;
     do i = 1 to 10;
       call ranuni(seed, r);
          if i > 7
          then
            do
           r = ranuni(-1);;
            end;
          r = int(r * 90);
          saledate = basedate + r;
       r = int(ranuni(0)*100);
          if      r < 20 then rep = '20';
          else if r < 40 then rep = '40';
          else if r < 70 then rep = '50';
          else                rep = '60';
       p = int(ranuni(0)*1000);
          if      p < 200 then product = 'PROD5';
          else if p < 400 then product = 'PROD4';
          else if p < 700 then product = 'PROD3';
          else                 product = 'PROD2';
       quantity = int(ranuni(0)*10) + 1;
```

```
          output;
     end;
run;
proc sort data=store1;
      by saledate;
run;
proc sort data=store2;
      by saledate;
run;
proc sort data=store3;
      by saledate;
run;

DATA YTDstores;
     set store1 store1 store3;
     by saledate;
run;

data class;
     length id       $1.
            student $12.
            ;
     input id $ student;
datalines;
a     Andrew
b     Bob
c     Charlie
d     Doug
e     Edward
f     Fred
;;
run;

data reward;
     length id       $1.
            reward  $12.
            ;
     input id $ reward $;
datalines;
a     Agate
b     Bauble
c     Coaster
d     Duster
e     Eagle
f     Firetruck
;;
run;
```

```
data merged;
      merge class
           reward;
run;
ods rtf body='c:\temp\merge.rtf';
title1 "Results of Merge";
title2 "Not Sorted BY values";
proc print data=merged;
run;
ods rtf close;

data reward_oops;
     length id       $1.
            reward  $12.
            ;
     input id $ reward $;
datalines;
a     Agate
b     Bauble
d     Duster
c     Coaster
e     Eagle
f     Firetruck
;;
run;
data merge_oops;
      merge class
           reward_oops;
run;
ods rtf body='c:\temp\merge_oops.rtf';
title1 "Results of Merge";
title2 "Not Sorted BY values and Records Misaligned";
proc print data=merge_oops;
run;
ods rtf close;

proc sort data=class;
     by id;
run;
proc sort data=reward;
     by id;
run;
data merge_sort;
      merge class
           reward;
     by id;
```

```
run;
ods rtf body='c:\temp\merge_sort.rtf';
title1 "Results of Merge";
title2 "Sorted BY values";
proc print data=merge_sort;
run;
ods rtf close;


* ------------------------------------------;
* duplicate BY;
* ------------------------------------------;

data class_dup;
     length id       $1.
            student $12.
            ;
     input id $ student;
datalines;
a      Andrew
a      Axel
b      Bob
c      Charlie
d      Doug
e      Edward
;;
run;
proc sort data=class_dup;
     by id;
run;

data reward_dup;
     length id       $1.
            reward  $12.
            ;
     input id $ reward $;
datalines;
a     Agate
b     Bauble
c     Coaster
c     Celery
d     Duster
e     Eagle
f     Firetruck
;;
run;
```

```
proc sort data=reward_dup;
     by id;
run;

data merge_dup;
      merge class_dup
            reward_dup;
     by id;
run;

ods rtf body='c:\temp\merge_DUP.rtf';
title1 "Results of Merge";
title2 "With Duplicate BY values";
proc print data=merge_dup;
run;
ods rtf close;

data class_miss;
     length id       $1.
            student $12.
            ;
     input id $ student;
datalines;
a      Andrew
c      Charlie
d      Doug
e      Edward
;;
run;
proc sort data=class_miss;
     by id;
run;

data reward_miss;
     length id       $1.
            reward  $12.
            ;
     input id $ reward $;
datalines;
a     Agate
b     Bauble
c     Coaster
e     Eagle
f     Firetruck
;;
run;
```

```
proc sort data=reward_miss;
     by id;
run;

data merge_miss;
      merge class_miss
            reward_miss;
     by id;
run;

ods rtf body='c:\temp\merge_miss.rtf';
title1 "Results of Merge";
title2 "With Different BY values";
proc print data=merge_miss;
run;
ods rtf close;

proc sql;
create table merge_sql as
select c.id,
       c.student,
      r.reward
from class as c inner join reward as r
on c.id = r.id
order by c.id
;
quit;

ods rtf body='c:\temp\merge_sql.rtf';
title1 "Results of Merge";
title2 "Using SQL";
proc print data=merge_sql;
run;
ods rtf close;
```