(11) COMBINING SAS DATA SETS

There are a number of ways to combine SAS data sets:

- **concatenate** - stack data sets (place one after another)
- **interleave** - stack data sets, but form the result in order by one or more variables present in the data sets
  
  NOTE: both concatenation and interleaving result in a data set that has as many observations as the sum of all the observations in the data sets being combined

- **one-to-one merge** - combine observation one in data set one with observation one in data set two, observation two-with-two, three-with-three, etc.
  
  NOTE: the data set resulting from one-to-one merge will have as many observations as the number of observations in the largest data set involved in the merge

- **matched-merge** - combine observations in data sets based upon the value of one or more variables
  
  NOTE: the number of observations in the resulting data set depends on the content of the data sets being combined and on the matching options being used

- **update** - change the values of the variables in certain observations in a data set based upon the values of variables in another data set
  
  NOTE: as with matched-merge, the number of observations in the resulting data set depends upon the content of the data sets being combined and on the matching options being used

---

...**CONCATENATION**

There are a number of methods that can be used to concatenate data sets. The easiest, but probably the least efficient way, is to use the SET statement.

---

*Example 11.1...*

```sas
data jan;
retain month 'JAN';
input ssn weight; 1
datalines;
  123456789 150
  987654321 120
  001001234 180
; run;

* data for subjects in study in February;*
data feb;
retain month 'FEB';
input ssn weight; 2
datalines;
  123456789 148
  001001234 115
  987654321 122
  888888888 190
; run;

* make a new data set out of the two old data sets;*
data jan_feb;
set jan feb; 3
run;
```

---
Introduction to SAS®
Mike Zdeb (402-6479, msz03@albany.edu) #122

proc print data=jan_feb noobs;
runk;

<table>
<thead>
<tr>
<th>month</th>
<th>ssn</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>JAN</td>
<td>123456789</td>
<td>150</td>
</tr>
<tr>
<td>JAN</td>
<td>987654321</td>
<td>120</td>
</tr>
<tr>
<td>JAN</td>
<td>001001234</td>
<td>180</td>
</tr>
<tr>
<td>FEB</td>
<td>123456789</td>
<td>148</td>
</tr>
<tr>
<td>FEB</td>
<td>001001234</td>
<td>115</td>
</tr>
<tr>
<td>FEB</td>
<td>987654321</td>
<td>122</td>
</tr>
<tr>
<td>FEB</td>
<td>888888888</td>
<td>190</td>
</tr>
</tbody>
</table>

1 Data set JAN is created with three variables: month, ssn, weight.
2 Data set FEB is created with the same three variables.
3 The two data sets are combined with a SET statement.
4 The output from PROC PRINT shows the two data sets combined 'tot-to-bottom'.

You can concatenate as many data sets as you want with a SET statement. This method forces SAS to read every observation in each of the data sets being combined. Concatenation using the SET statement is very robust, i.e. there are not many conditions that will cause it to fail. However, concatenation using SET is sensitive to the TYPE of variables in the data sets. You can't have a variable with the same name across data sets defined as NUMERIC in one data set and as CHARACTER in another. If SSN had been NUMERIC in data set JAN and CHARACTER in data set FEB, the data sets could not be combined and the following error would appear in the LOG when the data sets were used in the same SET statement...

ERROR: Variable SSN has been defined as both character and numeric.

Each data set need not have the same number of variables. Variables not present in one of the data sets will have their values set to missing in the combined data set. If data set FEB contained a new variable not present in data set JAN, e.g. ZIP, all values of ZIP in observations that are contributed to data set JAN_FEB by data set JAN would be set to missing. If variables have different LENGTHS in the data sets being combined, the LENGTHS of variables in the resulting data set will be those in the first data set in the SET statement in which the variables are present.

...Example 11.2...
data jan;
  input (name weight) ($8. 3.);
datalines;
  ZDEB 180
  SMITH 200
;
run;

data feb;
  input (name weight zip) ($10. 3. $5.);
datalines;
  SMITH 21012202
  ZDEB 18012203
  WASHINGTON19000001
;
run;

data jan_feb;
  set jan (in=j) feb;
  if j then month='JAN';
  else month='FEB';
runk;

proc contents data=jan_feb;
run;

proc print data=jan_feb noobs;
run;

<table>
<thead>
<tr>
<th>#</th>
<th>Variable</th>
<th>Type</th>
<th>Len</th>
<th>Pos</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>month</td>
<td>Char</td>
<td>3</td>
<td>21</td>
</tr>
<tr>
<td>1</td>
<td>name</td>
<td>Char</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>weight</td>
<td>Num</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>zip</td>
<td>Char</td>
<td>5</td>
<td>16</td>
</tr>
</tbody>
</table>

month   name         weight     zip
JAN     ZDEB          180
JAN     SMITH         200
FEB     SMITH         210     12202
FEB     ZDEB          180     12203
FEB     WASHINGT      190     00001

1 Data set JAN is created with two variables: name, weight. Name has a length of 8.
2 Data set FEB is created with an additional variable, zip, and a name has a length of 10.
3 The data sets are combined using SET and an IN= data set option.
4 The IN= variable is used to assign a month to each observation.
5 PROC CONTENTS run using the new data set shows that the variable name has a length of 8.
6 Values for the variable zip are missing in observations contributed by data set JAN. One name in an
   observation from data set FEB has been truncated.

The variable NAME had a length of 8 in data set JAN. Since data set JAN occurs first in the SET
statement, the length of variable NAME in the combined data set JAN_FEB is also 8 and the names from
data set FEB are truncated. A LENGTH statement in the data step that combines JAN and FEB would
override this problem, or just change the order of the data sets in the SET statement...

length name $10;
-or-
set feb jan;

The values for variable ZIP in observations contributed to JAN_FEB by data set JAN are missing and the
length of ZIP is 5, taken from information contributed by data set FEB.

Rather than creating data sets with a month variable as in example 11.1, neither data set JAN nor FEB
had a month variable. When the data sets were combined, a IN= data set option was used to determine
if an observation had been provided to the new data set by data set JAN or FEB. The IN= option creates
a new variable (in this case J) that only exists for the duration of the data step and that can have only two
values, 0 or 1. In this example, if an observation is contributed by data set JAN, then the variable J will
have a value of 1, otherwise it will be 0. The statement beginning with 'IF J' will only be true when J=1, or
when an observation from data set JAN is read.

PROC APPEND is a more efficient way to concatenate data sets than SET. However, only two data sets
can be combined at one time, a BASE data set and a DATA data set. Observations from the DATA data
set are added to the end of the BASE data set. Concatenation via PROC APPEND is less robust than
concatenation via SET. PROC APPEND is sensitive not only to the TYPE of variables in the two data
sets, but also to what variables are present in each data set, and their lengths. The two data sets from
example 11.1 are used again since we want to have the variable month in the combined data set and
there is no way to add it to the combined data set as was done in example 11.2.
...Example 11.3...

data jan;
retain month 'JAN';
input (name weight) ($8. 3.);
datalines;
ZDEB 180
SMITH 200
;
run;

data feb;
retain month 'FEB';
input (name weight zip) ($10. 3. $5.);
datalines;
SMITH 21012202
ZDEB 18012203
WASHINGTON 19000001
;
run;

proc append base=jan data=feb; 1
run;

the LOG from PROC APPEND...
NOTE: Appending WORK.FEB to WORK.JAN.
WARNING: Variable zip was not found on BASE file.
WARNING: Variable name has different lengths on BASE and DATA files (BASE 8 DATA 10).
ERROR: No appending done because of anomalies listed above.
   Use FORCE option to append these files.
   NOTE: 0 observations added.
NOTE: The data set WORK.JAN has 2 observations and 3 variables.
NOTE: Statements not processed because of errors noted above.
NOTE: PROCEDURE APPEND used:
       real time     0.05 seconds
NOTE: The SAS System stopped processing this step because of errors.

1 The two data sets from example 11.1 are combined using PROC APPEND.
2 The LOG file from PROC APPEND shows an error and warnings. The data sets are not combined.

Due to the differences noted in the LOG, PROC APPEND will not combine the data sets. If you want to
override these limitations, you can use the FORCE option (but be careful).

...Example 11.4...

proc append base=jan data=feb force; 1
run;

proc contents data=jan;
run;

the LOG file from PROC APPEND...
NOTE: Appending WORK.FEB to WORK.JAN.
WARNING: Variable zip was not found on BASE file.
WARNING: Variable name has different lengths on BASE and DATA files (BASE 8 DATA 10).
NOTE: FORCE is specified, so dropping/truncating will occur.
NOTE: There were 3 observations read from the data set WORK.FEB.
NOTE: 3 observations added.
NOTE: The data set WORK.JAN has 5 observations and 3 variables.
NOTE: PROCEDURE APPEND used:
       real time     0.32 seconds
The FORCE option is used with PROC APPEND.
There are warnings but no errors in the log file. The data sets are combined.
The variable zip is not in the combined data set. The variable name has the same length as name in the base data set JAN.

The combined data set ends up with the same name as the BASE data set. Unlike combining data sets with a SET statement, variables in the DATA data set not present in the BASE data set are dropped from the combined data set. The lengths of variables are controlled by those in the BASE data set.

With all these additional limitations, PROC APPEND might seem less useful than combining data sets with a SET statement. The advantage of PROC APPEND is EFFICIENCY. Combining two data sets with a SET statement results in SAS reading all the observations in both data sets, and writing all the observations to the combined data set. PROC APPEND merely adds the observations in the two data sets together. If you have data sets with the same number of variables and with variables having the same lengths, PROC APPEND can save a lot of computer time. Look at the following LOG file that resulted from combining a large BASE data set (10,000,000 observations) with a small DATA data set (100 observations) first using SET, then using PROC APPEND.

Example 11.5...
data one;
do j=1 to 10e6;
   output;
end;
run;
data two;
do j=1 to 100;
   output;
end;
run;
data one_two;
set one two;
run;
proc append base=one data=two;
run;
the log from combining data sets with SET...
NOTE: There were 10000000 observations read from the data set WORK.ONE.
NOTE: There were 100 observations read from the data set WORK.TWO.
NOTE: The data set WORK.ONE_TWO has 10000100 observations and 1 variables.
NOTE: DATA statement used:
   real time       14.23 seconds
the log from combining data sets with PROC APPEND...
NOTE: Appending WORK.TWO to WORK.ONE.
NOTE: There were 100 observations read from the data set WORK.TWO.
NOTE: 100 observations added.
NOTE: The data set WORK.ONE has 10000100 observations and 1 variables.
NOTE: PROCEDURE APPEND used:
   real time       0.32 seconds
1 A data set is created with one variable (J) and 10 million observations.
2 Another data set is created, also with only one variable (J), but with only 100 observations.
3 The data sets are combined using SET.
4 The data sets are combined using PROC APPEND.
5 The LOG files show that PROC APPEND is much faster than SET.

In each instance, the resulting combined data set has 10,000,100 observations. However, the SET statement took almost 15 seconds seconds while PROC APPEND took less than a second. If you want the combined data set to have a new name, you will have to use PROC DATASETS to rename the data set produced by PROC APPEND.

...INTERLEAVING
Interleaving is 'sorted concatenation' and is done using a SET statement and a BY variable(s).

...Example 11.6...

data jan;
retain month 'JAN';
input (ssn weight) ($9. 3.);
datalines;
123456789150
987654321120
001001234180
;
runc;

data feb;
retain month 'FEB';
input (ssn weight) ($9. 3.);
datalines;
123456789148
001001234115
987654321122
888888888190
;
runc;

proc sort data=jan; by ssn;
runc;
proc sort data=feb; by ssn;
runc;

data jan_feb;
set jan feb;
by ssn;
runc;
proc print data=jan_feb noobs;
runc;

month       ssn       weight
JAN     001001234      180
FEB     001001234      115
JAN     123456789      150
FEB     123456789      148
FEB     888888888      190
JAN     987654321      120
FEB     987654321      122
The data sets from example 11.1 are used again. Both data sets are sorted by variable SSN. The data sets are combined with a SET statement, plus a BY statement. The results from PROC PRINT show the data set in order by SSN.

You can accomplish the same task by using PROC APPEND, then sorting the resulting data set according to the BY variable(s).

```
proc append base=jan data=feb;
run;
```

```
proc sort data=jan;
by ssn;
run;
```

There is one consideration about sorting the new, larger combined data set in lieu of sorting the two smaller data sets then interleaving. The time to sort a data set is not always linearly related to the size of the data set, i.e. it might not take just twice as long to sort a data set that has twice as many observations than another. If the combined data set is very large, it may be more EFFICIENT to first sort the smaller data sets then use SET and BY to interleave.

**...ONE-TO-ONE MERGE**

Concatenation and interleaving combine data sets 'top-to-bottom' - observations in one data set are added to observations in another data set. Merging combines data sets 'side-to-side' - variables in one data set are added to variables in another data set. A one-to-one merge combines observation #1 in data set one with observation #1 in data set two, #2 with #2, etc. SAS 'sees' the data set mentioned on the right LAST. If you have variables with the same names in data sets that are being merged, SAS will keep the values of the variables from the right-most named data set in the MERGE statement. A one-to-one merge is only useful when you are sure that the order of the observations in the data sets being combined is appropriate, i.e. observation #1 in data set one should be combined with #1 in data set two, etc.

```
...Example 11.7...
data jan;
input ssn weight zip : $5.;
datalines;
001001234 180 12203
123456789 150 13502
987654321 120 12345
;
run;

data feb;
input ssn weight;
datalines;
001001234 115
123456789 148
987654321 122
;
run;
data jan_feb;
merge jan feb;
run;
```

```
proc print data=jan_feb noobs;
format ssn ssn.;
run;
```
Introduction to SAS®
Mike Zdeb (402-6479, msz03@albany.edu)

Data set JAN is created with three variables: ssn, weight, zip.
Data set FEB is created with only two variables: ssn, weight.
The data sets are combined using MERGE.
The resulting data set has three observations and three variables.

The WEIGHT in data set JAN_FEB is from data set FEB, the right-most data set in the merge statement (the one SAS 'sees' last). The SSN is also from data set FEB, but since it is the same as in data set JAN, that is not as obvious. The variable ZIP is contributed by data set JAN (there is no variable ZIP in data set FEB to overwrite the values of ZIP).

If there are variables with the same name in the data sets, e.g. WEIGHT in JAN and FEB, you can use the RENAME data set option to avoid the replacement of values of variables.

Example 11.8...

```sas
data jan_feb;
merge jan (rename=(weight=wt1)) feb (rename=(weight=wt2));
diff_wt = wt2-wt1;
run;
proc print data=jan_feb noobs;
var ssn zip wt1 wt2 diff_wt;
format ssn ssn.;
run;
```

Example 11.9...

```sas
data jan;
input ssn weight zip :$5.;
datalines;
001-00-1234 115 12203
123-45-6789 148 13502
987-65-4321 122 12345
987654321 222 14001
888868888 222 14001
987654321 120 12345;
run;
```
data feb;
input ssn weight;
datalines;
001001234 115
123456789 148
987654321 122
;
run;

data jan_feb;
merge jan (rename=(weight=wt1)) feb (rename=(weight=wt2));
  avg_wt = mean(of wt1-wt2);
run;

proc print data=jan_feb;
var ssn zip wt1 wt2 avg_wt;
run;

1 A new observation is added to data set JAN. There is no matching observation (same SSN) in FEB.
2 The two data sets are combined with MERGE.
3 One-to-one merging was done. The third observation in data set JAN was paired with the third
   observation in data set FEB.
4 There were only three observations in data set FEB. All the values for variables in the fourth
   observation of JAN_FEB are taken from data set JAN.

The first two observations in JAN_FEB are identical to those found in example 11.8. However,
observations three and four in JAN_FEB are not correct. Observation three from JAN is combined with
observation three from FEB. Since the variable SSN is in both data sets, the SSN from FEB (the right-
most data set in the merge statement) ends up in data set JAN_FEB. Though observations having two
different SSNs are combined, SAS still computes AVG_WT. Observation four in data set JAN_FEB is
observation four from data set JAN. It was not combined with any data from FEB since FEB was all out
of observations for one-to-one matching. The variable WT2 has a value of missing, i.e. there is no weight
for February. Since a function was used to compute the average weight, the missing value for WT1 did
not result in missing value for AVG_WT.

The moral to this example is that a one-to-one merge should not be used unless you know all the details
about the data sets being combined. If you can be sure that both data sets are in the proper order and
that there are not any extra observations in any of the data sets being combined, one-to-one merge is
safe.

...MATCHED MERGE
Matched merging is 'side-to-side' combination of data sets in which observations are combined on the
basis of the value(s) of one or more variables, i.e. BY variable(s). Assume that the data sets JAN and
FEB from example 11.9 are used.

...Example 11.10...
data jan_feb;
merge jan (rename=(weight=wt1)) feb (rename=(weight=wt2));
  by ssn;
  avg_wt = mean(of wt1-wt2);
run;

proc print data=jan_feb;
var ssn zip wt1 wt2 avg_wt;
run;
  ssn   zip   wt1   wt2   avg_wt
001-00-1234  12203  180   115   147.5
123-45-6789  13502  150   148   149.0
888-88-8888  14001  222   .     222.0
987-65-4321  12345  120   122   121.0
1  A BY statement is added to the data step merge. All data set involved in the merge must be sorted
   according to values of the variable(s) in the by statement.
2  Observation three from data set JAN is not matched to an observation in data set FEB.
3  The correct match is made for the observation having SSN 987654321.

Since a by-variable is used, all data sets in the MERGE statement must be sorted in order of that by-
variable. In this example, both JAN and FEB are in SSN-order so no sorts are done. Data set JAN_FEB
is now 'correct' in that it contains matched records, plus the extra unmatched observation from data set
JAN.

Match-merging can be applied to a problem posed earlier in these notes. Example 8.9 contains data on
patients, with multiple observations for most patients. One question to answer with these data is how the
value of cholesterol changed from first visit to last visit.

...Example 11.11...
  data manyids;
  input id : $2. visit : mmddyy8. chol;
  format visit mmddyy8. ;
  datalines;
  01  01/05/89   400
  01  05/18/90   350
  01  11/11/90   305
  01  02/18/91   260
  02  12/25/91   200
  03  01/01/90   387
  03  02/02/91   380
  04  05/15/91   380
  04  08/20/91   370
  04  03/23/92   355
  04  07/05/92   261
  ;
  run;
  proc sort data=manyids;
  by id visit;
  run;
  data first_visit last_visit;
  set manyids;
  by id;
  if not (first.id and last.id);
  if first.id then output first_visit;
  else
    if last.id the output last_visit;
  run;
  data first_last;
  merge
  first_visit (rename=(visit=visit1 chol=chol1))
  last_visit (rename=(visit=visit2 chol=chol2));
  by id;
  diff_days = visit2 - visit1;
  diff_chol = chol2 - chol1;
  run;
  proc print data=first_last noobs;
  var id visit1 visit2 diff_days chol1 chol2 diff_chol;
run;

<table>
<thead>
<tr>
<th>id</th>
<th>visit1</th>
<th>visit2</th>
<th>days</th>
<th>chol1</th>
<th>chol2</th>
<th>diff chol</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>01/05/89</td>
<td>02/18/91</td>
<td>774</td>
<td>400</td>
<td>260</td>
<td>-140</td>
</tr>
<tr>
<td>03</td>
<td>01/01/90</td>
<td>02/02/91</td>
<td>397</td>
<td>387</td>
<td>380</td>
<td>-7</td>
</tr>
<tr>
<td>04</td>
<td>05/15/91</td>
<td>07/05/92</td>
<td>417</td>
<td>380</td>
<td>261</td>
<td>-119</td>
</tr>
</tbody>
</table>

1. This data step is the same as example 8.9, creating a data set with multiple observations for all patients except for ID=02.
2. The data set is sorted in ID order and in date order within each patient.
3. Two data sets are created, one with data from the first visit for each patient, another with data from the last visit.
4. The original data are read with a SET plus BY, allowing the use of a FIRST. and LAST. variable, ID.
5. The observations are screened to see if there are any patients who only had one visit or an observation that is both the FIRST and LAST observation in a group of IDs. All single visit patients are excluded.
6. FIRST and LAST variables are used to write observations to either of two data sets.
7. The new data sets are merge by ID. Variables with names common to both data sets are renamed. The number of days in the study and the change in cholesterol are computed.
8. The final, matched data set is the same as that produced in example 8.9, with ID=02 excluded.

The results of a matched merge are predictable when there is only one observation with a specific value of a BY variable in one or all of the data sets that are being combined. In example 11.10, there were no repeated values of the variable SSN in either data set JAN or FEB. Another name for this combination of data sets is one-to-one match merge. If there are multiple observations with the same value of the BY-variable in one of the data sets being merged, the term one-to-many match merge (or many-to-one match merge) is used.

...Example 11.12...
data demographic;
input name : $5. age zip : $5.;
datalines;
  ADAMS 20 12203
  BROWN 21 10001
  SMITH 50 12005
  SMITH 33 12012;
run;

data medical;
input name : $5. age hr chol ;
label
  hr = 'heart rate'
  chol = 'cholesterol' ;
datalines;
  ADAMS 20 89 200
  BROWN 21 60 140
  SMITH 34 71 150;
run;

data both;
merge demographic medical;
by name;
run;
proc print data=both noobs;
run;
name   age   zip    hr    chol
ADAMS  20    12203  89    200
BROWN  21    10001  60    140
SMITH  34    12005  71    150
SMITH  33    12012  71    150

1  The first data set contains demographic data.
2  The second data set contains medical data.
3  Demographic and medical data are combined using merge with a by variable.
4  Each SMITH (MANY) in data set demographic was combined with SMITH (ONE) in data set medical.

In a matched merge, the BY statement causes SAS to look for possible matches based on the value of the by-variable(s). Each SMITH in DEMOGRAPHIC 'found' SMITH in MEDICAL. There is one extra bit of information that will allow us to determine which of the matched SMITH observations is correct. What if you changed the matching criteria to: names must match; age must be within one year.

...Example 11.13...

data both;
merge
demographic (rename=(age=age1))
medical (rename=(age=age2))
by name;
if abs(age1 - age2) le 1;
run;

proc print data=both noobs;
run;

1  The data sets are merged and the age is renamed in both data sets.
2  The ages in the two data sets are compared. ABS(...) is the absolute value function. This statement is the same as writing... if -1 <= (age1 - age2) <= 1;
3  Data set BOTH now only has the observation for SMITH in which the values for the variable ae were within one year in the two data sets.

Use of match-merge is more complicated when there are observations with repeated values of the by-variable(s) used to match records in both data sets. This type of merge is referred to as a many-to-many match merge. Whether you get the correct result whenever you perform this type of merge depends on the both the data sets being used, the order of the data sets, and the matching criteria. You may or may not get the correct result.
...Example 11.14...

data dataset1;
  input name $ age;
  datalines;
  ADAMS  20
  BROWN  21
  BROWN  30
  JONES  45
  JONES  46
  JONES  47
  LAWRENCE 10
  LAWRENCE 14
  LAWRENCE 16
  SMITH  50
  WALTERS 29
; run;

data dataset2;
  input name $ age hr;
  label hr = 'heart rate';
  datalines;
  ADAMS     21  89
  BROWN     15  60
  BROWN     21  75
  BROWN     40  80
  JONES     48  60
  KELLY     57  90
  LAWRENCE  16  60
  LAWRENCE  20  84
  SMITH     30  71
  SMITH     50  55
; run;

data both;
merge dataset1 (rename=(age=age1)) dataset2 (rename=(age=age2));
  by name;
  if abs(age1 - age2) le 1;
run;

proc print data=both noobs label;
var name age1 age2 hr;
run;

from the LOG file for the merge data step....

NOTE: MERGE statement has more than one data set with repeats of BY values.

<table>
<thead>
<tr>
<th>name</th>
<th>age1</th>
<th>age2</th>
<th>rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADAMS</td>
<td>20</td>
<td>21</td>
<td>89</td>
</tr>
<tr>
<td>JONES</td>
<td>47</td>
<td>48</td>
<td>60</td>
</tr>
<tr>
<td>KELLY</td>
<td>.</td>
<td>57</td>
<td>90</td>
</tr>
<tr>
<td>SMITH</td>
<td>50</td>
<td>50</td>
<td>55</td>
</tr>
<tr>
<td>WALTERS</td>
<td>29</td>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>

1  The first data set contains only the variables name and age and there are repeated names.
2  The second data set adds the variable heart rate, and once again names are repeated.
3  The data sets are merged by name.
4  Only those observations in which the age is within one year are output.
5  A NOTE in the LOG file gives you a warning about the structure of the data sets.
6  Data set both has missed some observations that met the matching criteria.
Three observations in data set BOTH meet the matching criteria, same NAME in both data sets and ages within one year of each other. Two other observations are present since no observations with corresponding names are found for either. The DEFAULT behavior of a matched merge is to put matched records in the resulting data set, PLUS all unmatched records from all the data sets being merged.

How did the match-merge form data set BOTH? When there are repeated values of BY-variables in both data sets, SAS will combine the first observation in a BY-group in one data set with the first observation in a by-group in the other data set, the second with the second, etc. This is what occurred during the merge of the observations in DATASET1 and DATASET2 in example 11.14.

Here is a look at how merge created data set BOTH...

<table>
<thead>
<tr>
<th>OBS</th>
<th>DATASET1</th>
<th>DATASET2</th>
<th>AGE WITHIN ONE YEAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ADAMS/20</td>
<td>ADAMS/21</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>BROWN/21</td>
<td>BROWN/15</td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td>BROWN/30</td>
<td>BROWN/21</td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BROWN/40</td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td>JONES/45</td>
<td>JONES/48</td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td>JONES/46</td>
<td></td>
<td>NO</td>
</tr>
<tr>
<td>2</td>
<td>JONES/47</td>
<td></td>
<td>YES</td>
</tr>
<tr>
<td>3</td>
<td>NO MATCH</td>
<td>KELLY/57</td>
<td>UNKNOWN</td>
</tr>
<tr>
<td></td>
<td>LAWRENCE/10</td>
<td>LAWRENCE/16</td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td>LAWRENCE/14</td>
<td>LAWRENCE/20</td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td>LAWRENCE/16</td>
<td></td>
<td>NO</td>
</tr>
<tr>
<td></td>
<td>SMITH/50</td>
<td>SMITH/30</td>
<td>NO</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>SMITH/50</td>
<td>YES</td>
</tr>
<tr>
<td>5</td>
<td>WALTERS/29</td>
<td>NO MATCH</td>
<td>UNKNOWN</td>
</tr>
</tbody>
</table>

This illustration shows that one match-merge may be a combination of all three matching types: one-to-one; one-to-many / many-to-one; many-to-many. Within the larger merge, those observations involved in either a one-to-one or a one-to-many / many-to-one match presented no problems. The matching criteria produced the correct result in data set BOTH for ADAMS, JONES, and SMITH. However, there were two many-to-many matches within the larger merge, one for BROWN and another for LAWRENCE.

Were any possible matches missed for these names? The match-merge missed combining the observations in both data sets with NAME=BROWN and AGE=21, and with NAME=LAWRENCE and AGE=16. These observations that do meet the matching criteria never 'saw' each other. During the merge process, the observations that would have met the matching criteria were never combined.

The moral to this example is that you never know if you will get the 'correct' result in a many-to-many match merge. In this example, with a small number of observations you can see that you got the wrong answer. If you are working with large numbers of observations, you will not have the opportunity to look at every possible mistake.

...PROC SQL
The only way to be sure you get the correct result in a situation with observations with repeated BY values in both data sets (or a many-to-many match) is to have each observation in data set one paired with every observation in data set two that share a common BY-variable. There are a few ways in which you can get the correct answer using this 'all possible pairs' matching. One way is with elaborate data
step programming. Another is to use a SAS procedure, PROC SQL (Structured Query Language). One of the tasks that can be accomplished with PROC SQL is the merging of data sets. The SAS code needed to properly match the two data sets used earlier by name and age range is as follows.

...Example 11.15...

```sas
proc sql;
create table both as
select dataset1.name, dataset1.age as age1, dataset2.age as age2, hr
from dataset1, dataset2
where dataset1.name eq dataset2.name and
  abs(age1-age2) le 1;
quit;
```

```sas
proc print data=both noobs;
run;
```

<table>
<thead>
<tr>
<th>name</th>
<th>age1</th>
<th>age2</th>
<th>hr</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADAMS</td>
<td>20</td>
<td>21</td>
<td>89</td>
</tr>
<tr>
<td>BROWN</td>
<td>21</td>
<td>21</td>
<td>75</td>
</tr>
<tr>
<td>JONES</td>
<td>47</td>
<td>48</td>
<td>60</td>
</tr>
<tr>
<td>LAWRENCE</td>
<td>16</td>
<td>16</td>
<td>60</td>
</tr>
<tr>
<td>SMITH</td>
<td>50</td>
<td>50</td>
<td>55</td>
</tr>
</tbody>
</table>

1 PROC SQL is used to combine two data sets.
2 Data sets in PROC SQL are referred to as TABLES. The procedure creates data set BOTH.
3 Variables to be present in the data set being created must be selected from the data sets being combined with a SELECT statement. Those variables with common names must be referred to with a combination of the data set name and the variable name, e.g. dataset1.name. Variables can be renamed by specifying AS and a new name, sometimes called an alias. Since HR (heart rate) only occurs in one data set, no data set name is needed. Since we only need one name in the data set BOTH, the name is selected from only one data set, i.e. dataset1.
4 The two data sets being combined are specified in a FROM statement.
5 The matching criteria are listed in a WHERE statement.
6 Data set both now has all the records that meet the matching criteria and no unmatched records.

Why does PROC SQL work when a match-merge did not? In theory, PROC SQL produces a cartesian product of all the observations in the two data sets. That is a shortcut way of saying that each observation in dataset1 is at some point in the matching process paired with each observation in dataset2. Once the observations are paired, the matching criteria are checked. Another way to think about this is that there are 11 observations in dataset1 and 10 observations in dataset2, resulting in 110 possible pairs of observations (the product of multiplying 11 by 10). Each of these 110 pairs are examined to see if they meet the matching criteria.

Notice that there are ONLY MATCHED records in the combined data set. The match-merge put both matched and unmatched records in data set BOTH. PROC SQL as used in example 11.15 will only put matched records in data set BOTH. With either method, match-merge or PROC SQL, there are ways to control the types of observations that are placed in the combined data set.

PROC SQL has its own syntax, and yes, its own manual. It is very powerful and worth learning. Combining data sets is very small part of the tasks performed with PROC SQL. The bit of knowledge to remember for now about PROC SQL is that is one of several ways to successfully accomplish many-to-many match-merging with SAS.
..MORE ABOUT MATCHED MERGE

There is still more to learn about combining data sets using match-merge. In the match-merge examples, there were instances when observations in one data set did not match any observations in the other data set. What if you only wanted to keep paired observations in the data set that results from a merge, i.e. observations with contributions from both data sets. The data from example 11.9 will be used again.

...Example 11.16..

data jan;
  input ssn weight zip : $5.;
datalines;
  001001234 180 12203
  123456789 150 13502
  888888888 222 14001
  987654321 120 12345
; run;

data feb;
  input ssn weight;
datalines;
  001001234 115
  123456789 148
  987654321 122
; run;

data jan_feb;
  merge jan (rename=(weight=wt1) in=d1) 1
  feb (rename=(weight=wt2) in=d2);
  by ssn;
  if d1 and d2; 2
  avg_wt = mean(of wt1-wt2);
run;

proc print data=jan_feb noobs;
  var ssn zip wt1 wt2 avg_wt;
run;

  ssn         zip     wt1    wt2    avg_wt
  001-00-1234    12203    180    115     147.5
  123-45-6789    13502    150    148     149.0
  987-65-4321    14001    222    122     172.0

1 A new option is added to both data sets in the merge statement, IN=. Two new variables are created with the IN= option, D1 and D2.

2 Variables D1 and D2 are checked. If an observation contributes variables to a matched observation, the value of a variable created with IN= will be 1, otherwise it will have a value of 0.

3 Data set JAN_FEB has only matched observations.

An IN option is used on each data set involved in the merge. The IN option variables, in this case D1 and D2, can be assigned any valid SAS names and these variables only exist for the duration of the data step (similar to what you have already learned about first. and last. variables). In this example, they will not appear in data set JAN_FEB. The values of D1 and D2 are contingent upon whether the data set with which they are associated contributes variables to an observation formed by the MERGE statement.

<table>
<thead>
<tr>
<th>D1</th>
<th>D2</th>
<th>CONDITION</th>
<th>VARIABLES</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>a matched observation</td>
<td>variables from both JAN and FEB</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>an unmatched observation</td>
<td>variables from JAN ONLY</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>an unmatched observation</td>
<td>variables from FEB ONLY</td>
</tr>
</tbody>
</table>
When SAS evaluates the result of the statement...

```
if d1 and d2;
```

it is only true when D1=1 and D2=1. Only matched observations are kept. If we want to keep all
observations from JAN whether or not they are matched to an observation in FEB, we can rewrite the
data step as...

```
data jan_feb;
merge
jan (rename=(weight=wt1) in=jan)
feb (rename=(weight=wt2));
if jan;
  avg_wt = mean(of wt1-wt2);
run;
```

The only IN= variable added to the data step is associated with data set JAN. The subsetting if statement
only checks to see if an observation contains information from an observation in the JAN data set.

If there were extra observations in data set FEB, the changes necessary to keep all observations from
FEB whether or not they match an observation in JAN should be obvious, but just in case...

```
data jan_feb;
merge
jan (rename=(weight=wt1))
feb (rename=(weight=wt2) in=feb);
if feb;
  avg_wt = mean(of wt1-wt2);
run;
```

The IN= variable(s) can also be used to create multiple data sets during a data step merge. What if all
the matched observations were to be put in one data set, and unmatched observations put in another.

...Example 11.17...

```
data jan;
input ssn weight zip : $5.;
datalines;
001001234 180 12203 1 123456789 150 13502
888888888 222 14001 987654321 120 12345 
;
run;
```

```
data feb;
input ssn weight;
datalines;
777777777 101 001001234 115 123456789 148
987654321 122 000111222 300 987654321 122
000111222 300 
;
run;
```

```
proc sort data=jan;
by ssn;
run;
```

```
proc sort data=feb;
by ssn;
run;
```
Introduction to SAS®
Mike Zdeb (402-6479, msz03@albany.edu)

data jan_feb
only_jan (keep=ssn wt1 zip)
only_feb (keep=ssn wt2);
format ssn ssn.;
merge
jan (rename=(weight=wt1) in=jan)
feb (rename=(weight=wt2) in=feb);
by ssn;
avg_wt = mean(of wt1-wt2);
if jan and feb then output jan_feb;
else if jan         then output only_jan;
else                output only_feb;
run;

title 'JAN_FEB';
proc print data=jan_feb noobs;
var ssn zip wt1 wt2 avg_wt;
run;

title 'ONLY_JAN';
proc print data=only_jan noobs;
run;

title 'ONLY_FEB';
proc print data=only_feb noobs;
run;

1 Data set JAN has one observation that does not match an observation in data set FEB.
2 Two observations have been added to data set FEB that do not match any observations in data set JAN.
3 Neither data set is sorted by SSN, so they are both sorted now prior to the MERGE BY SSN.
4 Three data sets are created: one with matched observations; one with unmatched observations from data set JAN; one with unmatched observations from data set FEB. Notice that KEEP data set options are used for the data sets that contain unmatched observations.
5 An IN= variable is created for each data set.
6 The IN= variables are used to determine the destination for each observation.
7 The data sets are all correct.

The default action of a data step match-merge is to create a data set with both matched and unmatched observations. You can limit the observations placed in a data set and/or create multiple data sets using a combination of IN= variables and subsetting if statements.

from dset1 inner join dset2

If the type of join is specified, the word WHERE must be replaced with the word ON when specifying the conditions for combining observations. Example 100 would be rewritten as follows if the type of join was added to PROC SQL rather than using a FROM with the data set names separated by commas...
proc sql;
  create table both as
  select dset1.name, age1, age2, hrtrate
  from dset1 inner join dset2
  on dset1.name eq dset2.name and
  abs(age1-age2) le 1;
quit;

The type of join used in PROC SQL allows you to control the type of observations that are placed into the combined data set, performing the same function as the IN= variables in a data step merge. If two data sets are being combined with PROC SQL, different joins result in the following combined data sets.

<table>
<thead>
<tr>
<th>TYPE OF JOIN</th>
<th>OBSERVATIONS IN COMBINED DATA SET</th>
</tr>
</thead>
<tbody>
<tr>
<td>inner</td>
<td>only matched</td>
</tr>
<tr>
<td>full</td>
<td>all matched plus all unmatched from both data sets</td>
</tr>
<tr>
<td>left</td>
<td>all matched plus all unmatched from data set mentioned first</td>
</tr>
<tr>
<td>right</td>
<td>all matched plus all unmatched from data set mentioned last</td>
</tr>
</tbody>
</table>

The FULL, LEFT, and RIGHT joins are known as OUTER joins and can only be used when combining two data sets. The type of expression needed to create multiple data sets when using PROC SQL to combine data sets is not as straightforward as the process of using IN= variables with a data set merge. If you want to put all the matched observations into one data set and unmatched observations into another, you will have to learn a little more about PROC SQL.

...UPDATE
An UPDATE statement within a data step combines information from observations in much the same way as a MERGE statement. However, rather than creating a new data set that contains information from two or more combined data sets, UPDATE is used to replace the values of variables in one data set (referred to as a MASTER data set) with the values of variables in another (referred to as a TRANSACTION data set). An UPDATE statement is always used with a BY statement, specifying the variable(s) to be used to determine the observations to be updated with information from the TRANSACTION data set.

...Example 11.18...

data oldinfo;
  format ssn ssn.;
  input ssn weight zip : $5.;
  datalines;
  001001234 180 12203
  123456789 150 13502
  888888888 222 14001
  987654321 120 12345
; run;

data newinfo;
  input ssn weight age;
  datalines;
  001001234 148 56
  777777777 100 30
  987654321 40
; run;

data updated;
update oldinfo newinfo;
by ssn;
r
proc print data=updated;
rn;
In this example, the result of the UPDATE statement is seen to be very similar to that of a MERGE statement. The new data set contains both matched and unmatched observations. The variables present in only one data set (ZIP in OLDINFO and AGE in NEWINFO) are present in the combined data set. The difference between MERGE and UPDATE is seen in the values of the variable WEIGHT. The values from the dataset named last (NEWINFO) in the UPDATE statement have replaced the values from data set named first (OLDINFO) for all observations except for the observations in NEWINFO that had a missing value for WEIGHT. The value for WEIGHT in the combined data set for these observations (SSN 001-00-1234 and 987-65-4321) is still the value from data set OLDINFO. If values for variables in the TRANSACTION (second in the UPDATE statement) data set are missing, values for variables in the MASTER data set are left unchanged. Missing data in the TRANSACTION data set can be handled differently, but that requires a look at the language reference manual. Finally, just as with MERGE, the IN= dataset option can be used with either data set involved in the UPDATE to control the observations that are written to the combined data sets.

...A LITTLE MORE ABOUT MATCHING DATASETS
Sometimes the matching (i.e. matched merge) of data sets involves the use of a unique by variable or a unique group of by variables. For example, if you are combing data sets using social security number and will only accept exact matches of social security number, matching is straight forward. Observations match or they do not, case closed. However, matching of observations often involves situations where a combination of information provided by different variables is evaluated in deciding whether to accept or reject matched observations. In examples 11.14 and 11.15, there were multiple records in the data sets being combined where the name was the same. In addition to the name, the age in both data sets was used to accept or reject observations that matched by name.

This type of inexact matching is sometimes referred to as ‘fuzzy’ matching. It often involves the use of a BLOCKING variable. In examples 11.14 and 11.15, the BLOCKING variable was NAME. All observations with the same name in both data sets were combined and then more information (the AGE) was used to evaluate matching. Matching was done in BLOCKS, the minimum criterion for evaluating being a match by NAME. All matched observations had to meet this minimum before any more information (AGE) was used. The choice of the BLOCKING variable determines the observations that will ‘see’ each other and be subjected to additional matching criteria.

When using a name as a blocking variable, there is always the chance that a misspelling or variation on spelling will prevent the matching of observations that really should be combined. There is a SAS function that allows you to correct for some of this ‘fuzziness’ in names, i.e. SOUNDEX. The soundex of a name is a combination of the first letter of the name, plus a string of numbers that is created by applying a set of rules to the remaining consonants in the name.

...Example 11.19...

```sas
data names;
  input lastname : $15. @@;
  sndx=soundex(lastname); 1
  datalines;
  SMITH SMYTHE O’HENRY OHENRY ZDEB SOO CLINTON WASHINGTON
; run;
```

```sas
proc contents data=names;
run;
```
Introduction to SAS®
Mike Zdeb (402-6479, msz03@albany.edu) #141

1 The SOUNDEX function is used to create a new variable named SNDX.
2 Variable SNDX is the same length as the variable LASTNAME.
3 The number of characters in the variable SNDX vary based on the number of consonants in LASTNAME.

The SOUNDEX returns the same value for names that differ only by a slight degree in spelling. If a name contains only vowels after the first letter, the soundex will only be the first letter of the name. The original rules for constructing a soundex limited its value to one letter plus three numbers, replacing the blanks returned by the SAS function with zeroes, e.g. the soundex for SOO would be S000, for ZDEB would be Z310, and for WASHINGTON would be W252. The data step in example 11.19 could be modified as follows to adhere to this rule.

...Example 11.20

data names;
    length soundex $4; 1
    input lastname : $15. @@;
    sndx = soundex(lastname);
    sndx = translate(sndx,'0',' '); 2
    datalines;
    SMITH SMYTHE O’HENRY OHENRY ZDEB SOO CLINTON WASHINGTON ;
    run;

data names;
    length soundex $4; 1
    input lastname : $15. @@;
    sndx = soundex(lastname);
    sndx = translate(sndx,'0',' '); 2
    datalines;
    SMITH SMYTHE O’HENRY OHENRY ZDEB SOO CLINTON WASHINGTON ;
    run;
A LENGTH statement limits the variable SNDX to four characters.

The TRANSLATE function changes all spaces to zeroes.

PROC CONTENTS shows that SNDX is only four characters.

PROC PRINT shows the SNDX padded with zeroes, not spaces.

In addition to the SOUNDEX function, there is a SAS operator that compares the soundex of names (for that matter, any character variable) rather than the exact name. If you had matched observations by social security number, but then also wanted to compare the soundexes of the last names in both observations, you do not have to create new variables that contain the soundex. You can use the 'sounds like' operator, =*, to compare soundexes.

...Example 11.21

data one;
  input name1 : $15. ssn @@;
datalines;
  SMITH 123456789 JONES 987654321 BROWN 999368888
;
run;

data two;
  input name2 : $15. ssn @@;
datalines;
  SMYTH 123456789 JONES 987654321 ADAMS 999368888
;
run;

proc sql;
  create table both as
  select one.ssn, name1, name2
  from one, two
  where one.ssn = two.ssn and name1 =* name2;
run;

proc print data=both;
run;

Obs   ssn   name1   name2
1 123456789 SMITH  SMYTH
2 987654321 JONES  JONES

PROC SQL is used to join observations from data sets one and two based on social security number and soundex of names.

The 'sounds like' operator is used to compare the soundex of NAME1 to the soundex of NAME2.

Observations

The 'sounds like' operator can only be used with a WHERE statement. Notice that observations with names SMITH and SMYTH were joined and that no new variables were created.