Paper 1526-2017

So You Think You Can Combine Data Sets?

Christopher J. Bost, Independent SAS Consultant

ABSTRACT

The syntax to combine SAS® data sets is simple: use the SET statement to concatenate, and use the MERGE statement to merge. The data sets themselves, however, might be complex. Combining the data sets might not result in what is needed. This paper reviews techniques to perform before combining data sets, including checking for: common variables; common variables with different attributes; duplicate identifiers; duplicate observations; and acceptable match rates.

INTRODUCTION

The code to combine data sets is easy. The way data sets are combined is more complicated. It wastes time and resources when large data sets are combined and problems are discovered only after the fact.

Some issues can be identified with PROC COMPARE, but it can only compare two data sets at a time. This paper describes quick, easy ways to identify issues with any number of data sets to be combined.

The first section covers checks to perform before concatenating data sets. The second section covers checks to perform before merging data sets. Sample data, syntax, code descriptions, and results are supplied for each section.

BEFORE CONCATENATING DATA SETS

Check for: 1) variables that are not in all data sets; 2) common variables with different attributes; 3) duplicate identifiers; and 4) duplicate observations before concatenating data sets.

1. CHECK FOR VARIABLES THAT ARE NOT IN ALL DATA SETS

Data sets that are concatenated usually have all variables in common. If a variable is not in all data sets, observations from data sets that do not contain that variable will have missing values.

Consider the following three data sets:

```
CONTENTS of data set DS1

#    Variable     Type    Len    Format     Label

1    id           Char      2               Unique ID
2    q1           Num       8    2.
3    q2           Num       8               Question 2
4    q3           Num       8
```

```
CONTENTS of data set DS2

#    Variable     Type    Len    Format     Label

1    ID           Char      2               Subject's ID
2    Q1           Num       8    Z2.        Question 1
3    Q2           Num       8
4    Q3           Num       8
5    Q4           Num       8               Question 4
```

Output 1a. PROC CONTENTS of data sets DS1 and DS2.

```
CONTENTS of data set DS3

#    Variable    Type    Len    Format    Label

1    id          Char      3              Identifier
2    q2          Num       8              2nd Question
3    q3          Num       8    3.        3rd Question
4    q4          Num       8              4th question
5    q5          Num       8
```

Output 1b. PROC CONTENTS of data set DS3.

Missing values will be generated when these three data sets are concatenated. Specifically, observations from data set DS1 will have missing values for variables Q4 and Q5; observations from data set DS2 will have missing values for the variable Q5; and observations from data set DS3 will have missing values for the variable Q1. It is useful to know this before concatenating data sets but tedious to inspect all variables.

Use PROC SQL to identify variables that are not in all data sets being concatenated. The DICTIONARY table DICTIONARY.COLUMNS has information about variables in all SAS data sets in currently defined libraries. Use DESCRIBE TABLE to display its structure:

```
proc sql;
describe table dictionary.columns;
quit;
```

Results are printed in the log:

```
40         describe table dictionary.columns;
NOTE: SQL table DICTIONARY.COLUMNS was created like:

create table DICTIONARY.COLUMNS
  (
   libname char(8) label='Library Name',
   memname char(32) label='Member Name',
   memtype char(8) label='Member Type',
   name char(32) label='Column Name',
   type char(4) label='Column Type',
   length num label='Column Length',
   npos num label='Column Position',
   varnum num label='Column Number in Table',
   label char(256) label='Column Label',
   format char(49) label='Column Format',
   informat char(49) label='Column Informat',
   idxusage char(9) label='Column Index Type',
   sortedby num label='Order in Key Sequence',
   xtype char(12) label='Extended Type',
   notnull char(3) label='Not NULL?',
   precision num label='Precision',
   scale num label='Scale',
   transcode char(3) label='Transcoded?'
  );
```

Output 2. DESCRIBE TABLE output for DICTIONARY.COLUMNS.

Run a query on DICTIONARY.COLUMNS and select columns and rows for inspection:

```
proc sql;
select memname,name
from dictionary.columns
where libname='WORK' and memname in ('DS1','DS2','DS3');
quit;
```

The SELECT clause selects columns MEMNAME (data set name) and NAME (variable).

The FROM clause reads rows from DICTIONARY.COLUMNS.

The WHERE clause limits rows to those for columns in tables DS1, DS2, and DS3 in the WORK library. Note that values of LIBNAME and MEMNAME are stored in uppercase. The results are:

```
Member    Column
Name      Name
------------------
DS1       id
DS1       q1
DS1       q2
DS1       q3
DS2       ID
DS2       Q1
DS2       Q2
DS2       Q3
DS2       Q4
DS3       id
DS3       q2
DS3       q3
DS3       q4
DS3       q5
```

Output 3. Select columns and rows in DICTIONARY.COLUMNS.

Note that there is one row per column name (variable) in each member name (data set) and that column names can be in lowercase, uppercase, or mixed case.

Run a query that selects MEMNAME (data set name) and NAME (variable) when NAME does not occur three times (i.e., the variable is not in all three data sets being concatenated):

```
proc sql;
select memname,upcase(name) as name
from dictionary.columns
where libname='WORK' and memname in ('DS1','DS2','DS3')
group by upcase(name)
having count(*) < 3
order by name,memname;
quit;
```

The SELECT clause converts values of NAME to uppercase.

The GROUP BY clause groups rows by uppercase values of NAME.

The HAVING clause evaluates each group and includes it (i.e., all rows in that group) in results when there are fewer than three rows.

The ORDER BY clause sorts results by NAME and MEMNAME.

The results are:

```
Member
Name      name
------------------
DS1       Q1
DS2       Q1
DS2       Q4
DS3       Q4
DS3       Q5
```

Output 4. Variables that do not occur in all three data sets.

Variable Q1 is only in data sets DS1 and DS2. Variable Q4 is only in data sets DS2 and DS3. Variable Q5 is only in data set DS3. This output is easy to review, but it would be more difficult with a larger number of data sets and/or variables. A table with variables in the rows and data sets in the columns would be useful.

Run the previous query, save the results to a SAS data set, and run PROC FREQ to produce a crosstab:

```
proc sql;
create table notinall as
select memname,upcase(name) as name
from dictionary.columns
where libname='WORK' and memname in ('DS1','DS2','DS3')
group by upcase(name)
having count(*) < 3;
quit;

proc freq data=notinall;
tables name*memname/nopercent norow nocol;
run;
```

The CREATE TABLE clause saves query results in data set NOTINALL. (Results are not printed.)

The PROC FREQ step processes data set NOTINALL.

The TABLES statement requests a crosstab of NAME (variable name) and MEMNAME (data set name). The NOPERCENT, NOROW, and NOCOL options after the slash suppress printing any percentages.

The results are:

```
Table of name by memname

name      memname(Member Name)

Frequency|DS1     |DS2     |DS3     |  Total
---------+--------+--------+--------+
Q1       |      1 |      1 |      0 |      2
---------+--------+--------+--------+
Q4       |      0 |      1 |      1 |      2
---------+--------+--------+--------+
Q5       |      0 |      0 |      1 |      1
---------+--------+--------+--------+
Total           1        2        2        5
```

Output 5. PROC FREQ crosstab of NAME and MEMNAME (variables not in all data sets).

This crosstab documents the names and sources of any variables that are not in all three data sets. A frequency of 1 indicates a variable is present; a frequency of 0 indicates a variable is not present.

Summary

Use PROC SQL to identify and PROC FREQ to summarize variables that are not in all data sets being concatenated.

This technique can be used with any number of variables and any number of data sets. It runs quickly, as only metadata (descriptor information) is processed. The SQL syntax might be unfamiliar to some.

If a variable is not in all data sets being concatenated, there will be missing values. This might or might not be a problem depending on your data. For example, you might have multiple data sets from multiple times a survey was administered. If new questions were added over time, observations from the earlier data sets will (appropriately) have missing values for the new variables.

If any variables contain the same information but have different names, however, it is an inconsistency. Rename variables as needed before concatenating data sets.

2. CHECK FOR COMMON VARIABLES WITH DIFFERENT ATTRIBUTES

Data sets that are concatenated usually have variables with the same attributes (i.e., the same name, type, length, format, and label). If variable attributes are not the same across data sets, the variable attributes in the concatenated data set might not be what are expected.

Consider the three data sets used in the previous section:

```
CONTENTS of data set DS1

#    Variable     Type    Len    Format    Label

1    id           Char      2              Unique ID
2    q1           Num       8    2.
3    q2           Num       8              Question 2
4    q3           Num       8
```

```
CONTENTS of data set DS2

#    Variable     Type    Len    Format    Label

1    ID           Char      2              Subject's ID
2    Q1           Num       8    Z2.       Question 1
3    Q2           Num       8
4    Q3           Num       8
5    Q4           Num       8              Question 4
```

```
CONTENTS of data set DS3

#    Variable     Type    Len    Format    Label

1    id           Char      3              Identifier
2    q2           Num       8              2nd Question
3    q3           Num       8    3.        3rd Question
4    q4           Num       8              4th question
5    q5           Num       8
```

Output 6. PROC CONTENTS of data sets DS1, DS2, and DS3.

When these three data sets are concatenated:

```
data ds123;
set ds1 ds2 ds3;
run;

proc contents data=ds123 varnum;
title 'CONTENTS of data set DS123';
run;
```

The results are:

```
CONTENTS of data set DS123

#    Variable     Type    Len    Format    Label

1    id           Char      2              Unique ID
2    q1           Num       8    2.        Question 1
3    q2           Num       8              Question 2
4    q3           Num       8    3.        3rd Question
5    Q4           Num       8              Question 4
6    q5           Num       8
```

Output 7. PROC CONTENTS of data set DS123.

When the DATA step is compiled, the name, type, and length are determined the first time a variable is encountered. Optional variable attributes like a format or label, however, are determined the first time a user-assigned value is encountered.

The variable attributes in data set DS123 are compiled as follows (see highlighted text in Output 6):

id The name, type, length, and label are from DS1. (No format is ever associated with ID.) The length of 3 bytes in DS3, and the labels in DS2 and DS3, are not used.

q1 The name, type, length, and format are from DS1; the label is from DS2. (There is no label for Q1 in DS1.) The format Z2. in DS2 is not used.

q2 The name, type, length, and label are from DS1. (No format is ever associated with Q2.) The label from DS3 is not used.

q3 The name, type, and length are from DS1; the format and label are from DS3.

Q4 The name (note the uppercase spelling), type, length, and label are from DS2. (No format is ever associated with Q4.) The label from DS3 is not used.

q5 The name, type, and length are from DS3. (No format or label is ever associated with Q5.)

In other words, variable attributes in the concatenated data set are determined by the variable attributes in the source data sets and the order in which the source data sets are concatenated.

Metadata that differs between data sets might be problematic. Use PROC SQL to identify variables that occur in more than one data set being concatenated and that have different attributes. For example:

```
proc sql;
title 'Check for variables with different types';
select memname,upcase(name) as name,type
from dictionary.columns
where libname='WORK' and memname in ('DS1','DS2','DS3')
group by upcase(name)
having count(distinct type) > 1
order by name,memname;
quit;
```

The GROUP BY clause groups rows by uppercase values of NAME as in previous queries.

The HAVING clause evaluates each group and includes it (i.e., all rows in that group) in results when there is more than one distinct value of TYPE (variable type) for a variable.

No results are returned in this example (i.e., there are no variables with different types across data sets). Note, however, that having a variable with more than one type will cause the DATA step to fail during the compilation phase. SAS prints an ERROR in the log:

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

Use a similar query to check for variables with different lengths:

```
proc sql;
title 'Check for variables with different lengths';
select memname,upcase(name) as name,length
from dictionary.columns
where libname='WORK' and memname in ('DS1','DS2','DS3')
group by upcase(name)
having count(distinct length) > 1
order by name,memname;
quit;
```

The results are:

```
Check for variables with different lengths

Member          Column
Name     name   Length
---------------------
DS1      ID          2
DS2      ID          2
DS3      ID          3
```

Output 8. Variables that have different lengths.

The variable ID is in data sets DS1, DS2, and DS3. It has a length of 2 bytes in data sets DS1 and DS2 but a length of 3 bytes in data set DS3. This produces a warning in the log:

```
WARNING: Multiple lengths were specified for the variable id by input data
set(s). This can cause truncation of data.
```

Note that the length of ID in data set DS123 is 2 bytes, the value first encountered when concatenating. This means values of ID from DS3 (with a length of 3 bytes) might be truncated during concatenation.

Use a similar query to check for variables with different formats:

```
proc sql;
title 'Check for variables with different formats';
select memname,upcase(name) as name,format
from dictionary.columns
where libname='WORK' and memname in ('DS1','DS2','DS3')
group by upcase(name)
having count(distinct format) > 1
order by name,memname;
quit;
```

The results are:

```
Check for variables with different formats

Member          Column
Name     name   Format
-------------------------
DS1      Q1     2.
DS2      Q1     Z2.
```

Output 9. Variables that have different formats.

The format 2. is associated with variable Q1 in data set DS1. The format Z2. is associated with variable Q1 in data set DS2.

The format for Q1 in data set DS123 is 2., the value first encountered. This means values of ID will be formatted with the 2. format, although values from data set DS2 were originally formatted with the Z2. format (i.e., leading zeroes).

Finally, use a similar query to check for variables with different labels:

```
proc sql;
title 'Check for variables with different labels';
select memname,upcase(name) as name,label
from dictionary.columns
where libname='WORK' and memname in ('DS1','DS2','DS3')
group by upcase(name)
having count(distinct label) > 1
order by name,memname;
quit;
```

The results are:

```
Check for variables with different labels

Member
Name    name   Column Label
-----------------------------------
DS1     ID     Unique ID
DS2     ID     Subject's ID
DS3     ID     Identifier
DS1     Q2     Question 2
DS2     Q2
DS3     Q2     2nd Question
DS2     Q4     Question 4
DS3     Q4     4th question
```

Output 10. Variables that have different labels.

The variable ID has three different labels in data sets DS1, DS2, and DS3. The label for ID in data set DS123 is 'Unique ID', the value first encountered.

The variable Q2 has two different labels in data sets DS1 and DS3 (and one missing label in data set DS2). The label for Q2 in data set DS123 is 'Question 2', the first value encountered.

(The COUNT function ignores missing values, but there is more than one label for Q2. All rows in the group for Q2 are returned, including the missing label in data set DS2.)

The variable Q4 has two different labels in data sets DS2 and DS3. The label for Q4 in data set DS123 is 'Question 4', the first value encountered.

Summary

Use PROC SQL to identify any variables with different attributes across data sets being concatenated.

This technique can be used with any variable attribute(s) of interest. It can be used with any number of variables and any number of data sets. The results, however, must be inspected "by hand." This could be tedious when there are multiple differences.

Inconsistent attributes might or might not be a problem:

- Variables defined as both character and numeric will cause an error and must always be addressed.

- Variables defined with different lengths will cause a warning and, because it might result in truncation, should always be addressed.

- Variables defined with different formats and labels might be acceptable. The programmer must decide if differences across data sets are significant.

Inspect the query results carefully and code as necessary to explicitly assign desired variable attributes.

If a common variable has different:

Types Convert the variable type (e.g., using the INPUT function or PUT function) to either all numeric or all character before concatenating.

Lengths Use a LENGTH statement *before* the SET statement to specify the longest variable length. Subsequent variables with shorter lengths will not produce a warning.

Formats Use a FORMAT statement *after* the SET statement to specify the preferred format. This format will replace any format that was previously compiled.

Labels Use a LABEL statement *after* the SET statement to specify the preferred label. This label will replace any label that was previously compiled.

3. CHECK FOR DUPLICATE IDENTIFIERS

Data sets that are concatenated usually have a variable with values that uniquely identify each observation. If an identifier is duplicated, the concatenated data set might have incorrect observations.

Consider the following two data sets:

```
PRINT of data set DS1

Obs     id     x

 1       1     1
 2       2     1
 3       2     2
 4       3     1
```

```
PRINT of data set DS2

Obs     id     x

 1       3     1
 2       4     1
 3       5     1
```

Output 11. PROC PRINT of data sets DS1 and DS2.

Data set DS1 has a duplicate identifier. (ID=2 occurs twice.) Data set DS2 has no duplicate identifiers. Note that ID=3 occurs in both DS1 and DS2 and would be a duplicate in the concatenated data set.

Use PROC SQL to check for duplicate identifiers in data set DS1:

```
proc sql;
title 'Check for duplicate IDs in data set DS1';
select count(distinct id) as Ndistinct,
       count(*) as N
from ds1;
quit;
```

The SELECT clause counts the number of distinct values of ID and names the column (variable) NDISTINCT.

The SELECT clause also counts the number of rows (observations) in the table (data set) and names it N.

The results are:

```
Check for duplicate IDs in data set DS1

Ndistinct         N
-------------------
        3         4
```

Output 12. Duplicates in data set DS1.

There are 3 distinct values of ID among the 4 observations in the data set (i.e., there is a duplicate).

The same query can be run on data set DS2:

```
proc sql;
title 'Check for duplicate IDs in data set DS2';
select count(distinct id) as Ndistinct,
       count(*) as N
from ds2;
quit;
```

The results are:

```
Check for duplicate IDs in data set DS2

Ndistinct         N
-------------------
        3         3
```

Output 13. No duplicates in data set DS2.

There are 3 distinct values of ID among the 3 observations in the data set (i.e., there are no duplicates).

Duplicate identifiers can also occur across data sets. A similar query can be run on ID values in data sets DS1 and DS2 combined:

```
proc sql;
title 'Check for duplicate IDs in data sets DS1 and DS2 combined';
select count(distinct id) as Ndistinct,
       count(*) as N
from (select id
      from ds1
      outer union corresponding
      select id
      from ds2);
quit;
```

The SELECT clause again counts the number of distinct values of ID and the number of observations. It reads values of ID from both DS1 and DS2 using an in-line view, or subquery on the FROM clause.

In this example, the in-line view contains two queries. The first SELECT clause selects all values of ID from DS1. The second SELECT clause selects all values of ID from DS2. The results of both queries are combined using the set operators OUTER UNION. The CORRESPONDING operator links query results by column name.

In other words, the in-line view creates a “virtual table” that is used by the outer query.

The results are:

```
Check for duplicate IDs in data sets DS1 and DS2 combined

Ndistinct         N
-------------------
        5         7
```

Output 14. Duplicates in data sets DS1 and DS2 combined.

There are 5 distinct values of ID among the 7 observations in data sets DS1 and DS2 combined (i.e., there are duplicates).

It can be useful to summarize duplicates within and across data sets with a single query:

```
proc sql;
title 'Check for duplicate IDs within and across data sets DS1 and DS2';
select 'DS1' as DataSet,
       count(distinct id) as Ndistinct,
       count(*) as N
from ds1
outer union corresponding
select 'DS2' as DataSet,
       count(distinct id) as Ndistinct,
       count(*) as N
from ds2
outer union corresponding
```

```
select 'DS1+DS2' as DataSet,
       count(distinct id) as Ndistinct,
       count(*) as N
from (select id
      from ds1
      outer union corresponding
      select id
      from ds2);
quit;
```

The three queries described above are combined into one. (Note the single semicolon at the end.) Results are combined using the set operators OUTER UNION CORRESPONDING.

The only difference is that each query includes a column named DATASET for the name of the data set being queried. Include any desired text in quotes.

The results are:

```
Check for duplicate IDs within and across data sets DS1 and DS2

DataSet     Ndistinct         N
-------------------------------
DS1                 3         4
DS2                 3         3
DS1+DS2             5         7
```

Output 15. Duplicates within and across data sets DS1 and DS2.

Note that there are 3 distinct values of ID in both data sets DS1 and DS2 but only 5 distinct values of ID (not 3+3=6) in both data sets combined (i.e., duplicate values of ID across data sets DS1 and DS2).

When duplicate identifiers occur within and/or across data sets, print those observations for inspection (i.e., where practical). This can be done with the following two queries:

```
proc sql;
create table dupids as
select id
from (select id
      from ds1
      outer union corresponding
      select id
      from ds2)
group by id
having count(id) > 1;
```

The FROM clause contains an in-line view with two queries as in previous examples. The first SELECT clause selects all values of ID from DS1. The second SELECT clause selects all values of ID from DS2. The results of both queries are combined using OUTER UNION CORRESPONDING.

The GROUP BY clause groups rows by ID.

The HAVING clause counts the number of occurrences of each ID. The ID is kept if the count is greater than 1.

The CREATE TABLE clause saves query results in data set DUPIDS. It is a “lookup table” of ID values that are duplicated within and/or across data sets DS1 and DS2. Results are not printed but would be:

```
      id
--------
       2
       3
```

Output 16. Duplicate ID values within and across data sets DS1 and DS2 stored in data set DUPIDS.

```
title 'Observations with duplicate IDs within and across data sets DS1 and DS2';
select 'DS1' as DataSet,*
from ds1
where id in (select id
             from dupids)
outer union corresponding
select 'DS2' as DataSet,*
from ds2
where id in (select id
             from dupids)
order by id,DataSet;
quit;
```

The first SELECT clause selects all columns (*) from data set DS1.

The WHERE clause selects rows where the ID is in the lookup table. (The SELECT clause in parentheses is a subquery. It returns values of ID from data set DUPIDS that are used by the outer query.)

The second SELECT clause and WHERE clause select rows from data set DS2 using the same method.

Both queries include a column named DATASET for the name of the data set from which the observation comes. Specify the data set name in quotes.

The results of both queries are combined using the set operators OUTER UNION CORRESPONDING.

The ORDER BY clause sorts the results by ID and DATASET.

The results are:

```
Observations with duplicate IDs within and across data sets DS1 and DS2

DataSet          id         x
----------------------------
DS1               2         1
DS1               2         2
DS1               3         1
DS2               3         1
```

Output 17. Observations with duplicate ID values and source data sets.

ID=2 occurs twice in data set DS1 (i.e., duplicate within a data set). ID=3 occurs once in data set DS1 and once in data set DS2 (i.e., duplicate across data sets).

Summary

Use PROC SQL to check for duplicate identifiers within and across data sets being concatenated.

These techniques can be used to summarize the number of duplicate identifiers and to print observations with duplicate identifiers and their sources for review. The results must be inspected to determine why there are duplicate identifiers and which observation to keep. This could be time-consuming when there are multiple duplicate identifiers.

It might be preferable to correct duplicate identifiers in each source data set before concatenating them. It is also possible to use the WHERE= data set option to keep desired observations during concatenation. For example:

```
data ds12;
set ds1(where=(not(id=2 and x=2)))
    ds2(where=(not(id=3)));
run;
```

Duplicate identifiers are a data quality problem. Delete observations with duplicate identifiers on a case-by-case basis.

4. CHECK FOR DUPLICATE OBSERVATIONS

Data sets that are concatenated usually contain unique observations. Duplicate observations (i.e., where the value of every variable is repeated) are a problem.

Consider the following two data sets:

```
PRINT of data set DS1

Obs    id    a    b    c

 1      1    1    2    3
 2      2    4    5    6
 3      2    4    5    6
 4      3    7    8    9
```

```
PRINT of data set DS2

Obs    id     a     b     c

 1      3     7     8     9
 2      4    10    11    12
 3      5    13    14    15
```

Output 18. PROC PRINT of data sets DS1 and DS2.

Data set DS1 has a duplicate observation. (Both observations with ID=2 have all variables in common.)

Data set DS2 has no duplicate observations. Note that the observations with ID=3 in both data sets DS1 and DS2 have all variables in common and would be a duplicate observation in the concatenated data set.

Start with the technique described in “Check for Duplicates Identifiers.” If duplicates are detected and the results show entirely duplicate observations (i.e., all variables in common and all values repeated), delete them “manually” during the concatenation as described. If there is a large number of duplicate observations, however, consider using one of the following techniques.

Use PROC SQL to eliminate duplicate observations within and across data sets DS1 and DS2:

```
proc sql;
create table ds12union as
select *
from ds1
union
select *
from ds2;
quit;

proc print data=ds12union;
title 'PRINT of data set DS12UNION';
run;
```

The CREATE table clause saves query results in data set DS12UNION.

The first SELECT clause selects all columns (variables) from table (data set) DS1.

The second SELECT clause selects all columns (variables) from table (data set) DS2.

The UNION set operator combines the results of both queries and keeps all unique rows (observations).

The results are:

```
PRINT of data set DS12UNION

Obs    id     a     b     c

 1      1     1     2     3
 2      2     4     5     6
 3      3     7     8     9
 4      4    10    11    12
 5      5    13    14    15
```

Output 19. PROC PRINT of data set DS12UNION with no duplicate observations.

There are 5 unique observations in data set DS12UNION.

Note that UNION produces all unique rows from both queries. This requires two passes through the data, one to combine results and one to eliminate duplicates.

Alternately, use the DATA step and PROC SORT to delete duplicate observations:

```
data ds12;
set ds1 ds2;
run;

proc sort data=ds12 nodupkey out=ds12nodupkey;
by _all_;
run;

proc print data=ds12nodupkey;
title 'PRINT of data set DS12NODUPKEY';
run;
```

The DATA statement creates data set DS12.

The SET statement concatenates data sets DS1 and DS2.

The PROC SORT statement starts the procedure. The NODUPKEY option deletes any observations with duplicate occurrences of variable(s) on the BY statement. The OUT= option saves sorted results to data set DS12NODUPKEY.

The BY statement specifies to sort based on all variables in the data set.

The results are:

```
PRINT of data set DS12NODUPKEY

Obs    id     a     b     c

 1      1     1     2     3
 2      2     4     5     6
 3      3     7     8     9
 4      4    10    11    12
 5      5    13    14    15
```

Output 20. PROC PRINT of data set DS12NODUPKEY with no duplicate observations.

There are 5 unique observations in data set DS12NODUPKEY.

Summary

Use PROC SQL or a DATA step and PROC SORT to eliminate duplicate observations.

Both of these approaches require two passes through the data.

Eliminating duplicate observations can be a very resource-intensive task.

BEFORE MERGING DATA SETS

Check for: 1) variables in more than one data set; 2) duplicate identifiers in more than one data set; and 3) acceptable match rates before merging data sets.

1. CHECK FOR VARIABLES IN MORE THAN ONE DATA SET

Data sets being merged usually have different variables. If the data sets have common variables (i.e., other than the BY variable[s]), values will be overwritten. Only the last value of a variable that occurs more than once is stored in the merged data set.

Consider three data sets: DS1, DS2, and DS3. DS1 contains variables id, a, b, and c. DS2 contains variables ID, B, C, and D. DS3 contains variables id, d, e, and f. The three data sets are sorted by ID and merged.

Use PROC SQL and PROC FREQ to identify common variables:

```
proc sql;
create table common as
select memname,upcase(name) as name
from dictionary.columns
where libname='WORK' and memname in ('DS1','DS2','DS3')
group by upcase(name)
having count(*) > 1;
quit;

proc freq data=common;
tables name*memname/nopercent norow nocol;
title 'Check for variables in more than one data set';
run;
```

This technique is similar to one used previously. (See “Check for Variables That Are Not In All Data Sets.”)

The HAVING clause evaluates each group and includes it (i.e., all rows in that group) in results when there is more than one row (i.e., the variable occurs more than once).

The results are:

```
Check for variables in more than one data set

The FREQ Procedure

Table of name by memname

name      memname(Member Name)

Frequency|DS1     |DS2     |DS3     |  Total
---------+--------+--------+--------+
B        |      1 |      1 |      0 |      2
---------+--------+--------+--------+
C        |      1 |      1 |      0 |      2
---------+--------+--------+--------+
D        |      0 |      1 |      1 |      2
---------+--------+--------+--------+
ID       |      1 |      1 |      1 |      3
---------+--------+--------+--------+
Total           3        4        2        9
```

Output 21. PROC FREQ crosstab of NAME and MEMNAME (common variables only).

Variable B occurs in data sets DS1 and DS2. Variable C also occurs in data sets DS1 and DS2. Variable D occurs in data sets DS2 and DS3. Variable ID occurs in data sets DS1, DS2, and DS3.

When data sets are merged, variables B and C in data set DS1 will be overwritten by variables B and C in data set DS2. Variable D in data set DS2 will be overwritten by variable D in data set DS3.

Summary

Use PROC SQL to identify variables that occur in more than one data set and PROC FREQ to summarize the results.

When data sets are merged and there is more than one variable with the same name, values will be overwritten. The last value encountered will be saved. This might be acceptable in some situations (e.g., when a subsequent data set contains more up-to-date information), but it is usually not ideal.

Rename common variables as needed to prevent overwriting.

2. CHECK FOR DUPLICATE IDENTIFIERS IN MORE THAN ONE DATA SET

Data sets being merged can have duplicate identifiers in one of the data sets. For example, a data set with credit card transactions could have multiple observations, one for each purchase. The credit card number is on each observation and, therefore, duplicated. A companion data set might could have one observation for each credit card holder with her name, address, and phone number. The credit card number on each observation is unique to the individual and never duplicated. These data sets could be merged "one-to-many" to add the credit card holder variables to each credit card transaction.

Data sets being merged can also have duplicate identifiers in more than one data set. The resulting data set is rarely useful.

Consider the following two data sets:

```
PRINT of data set DS1

Obs    id    x

 1      1    1
 2      2    1
 3      2    2
 4      3    1
 5      3    2
 6      3    3
 7      4    1
 8      5    1
 9      5    2
```

```
PRINT of data set DS2

Obs    id    y

  1     1    1
  2     1    2
  3     2    1
  4     3    1
  5     3    2
  6     3    3
  7     3    4
  8     4    1
  9     5    1
 10     5    2
```

Output 22. PROC PRINT of data sets DS1 and DS2.

Data set DS1 has duplicate identifiers: ID=2 occurs twice; ID=3 occurs three times; and ID=5 occurs twice.

Data set DS2 has duplicate identifiers: ID=1 occurs twice; ID=3 occurs four times; and ID=5 occurs twice.

Data sets can be sorted and merged:

```
proc sort data=ds1 out=ds1S;
by id;
run;

proc sort data=ds2 out=ds2S;
by id;
run;

data ds12merge;
merge ds1S ds2S;
by id;
run;

proc print data=ds12merge;
title 'PRINT of data set DS12MERGE';
run;
```

PROC SORT is used to sort each data set by values of ID. OUT= saves sorted observations to the data sets DS1S and DS2S, respectively.

The DATA step merges data sets DS1S and DS2S by ID.

The results are:

```
PRINT of data set DS12MERGE

Obs    id    x    y

  1     1    1    1
  2     1    1    2
  3     2    1    1
  4     2    2    1
  5     3    1    1
  6     3    2    2
  7     3    3    3
  8     3    3    4
  9     4    1    1
 10     5    1    1
 11     5    2    2
```

Output 23. PROC PRINT of data set DS12MERGE.

Because ID=3 and ID=5 are duplicated in both data sets, SAS performed a "many-to-many" merge.

SAS merges observations one-to-one within the same BY group (i.e., observations that have the same value of the BY variable; ID in this case). For example, the first occurrence of ID=5 in data set DS1 is merged with the first occurrence of ID=5 in data set DS2. The second occurrence of ID=5 in data set DS1 is merged with the second occurrence of ID=5 in data set DS2. This might or might not be correct.

When SAS "runs out of observations" in the current BY group from one data set, it retains the last values of variables unique to that data set and merges them with any additional observations in the current BY group from the other data set.

In this example, the first three occurrences of ID=3 in data set DS1 are merged one-to-one with the first three occurrences of ID=3 in data set DS2. When SAS encounters the fourth occurrence of ID=3 in data set DS2, it retains the value of X from the final observation in that BY group in data set DS1. SAS prints a note in the log:

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

Use PROC SQL to detect more than one data set with repeats of BY values before merging data sets.

PROC SQL can count the number of occurrences of each ID in the respective data sets. For example:

```
proc sql;

title 'Count occurrences of ID in data set DS1';
select id,count(*) as count1
from ds1
group by id;

title 'Count occurrences of ID in data set DS2';
select id,count(*) as count2
from ds2
group by id;

quit;
```

The first query uses the GROUP BY clause to group rows from DS1 by ID. The SELECT clause selects ID, counts the number of rows in each group (i.e., the number of occurrences of each ID), and saves the values as COUNT1.

The second query performs the same operations on data set DS2 and saves the values as COUNT2.

The results are:

```
Count occurrences of ID in data set DS1

      id    count1
------------------
       1         1
       2         2
       3         3
       4         1
       5         2

Count occurrences of ID in data set DS2

      id    count2
------------------
       1         2
       2         1
       3         4
       4         1
       5         2
```

Output 24. ID values and number of occurrences in data sets DS1 and DS2.

In data set DS1, ID=1 occurs 1 time; ID=2 occurs 2 times; ID=3 occurs 3 times; and so on.

In data set DS2, ID=1 occurs 2 times; ID=2 occurs 1 time; ID=3 occurs 4 times; and so on.

(Results are shown for illustrative purposes. Running these queries on most data sets would produce a prohibitive amount of output.)

If an ID occurs more than once in data set DS1, it has a COUNT1 value greater than 1. Similarly, if an ID occurs more than once in data set DS2, it has a COUNT2 value greater than 1. Query results could be joined (merged) by ID.

If an ID occurs more than once in both data sets DS1 and DS2, the values of COUNT1 and COUNT2 would both be greater than 1. It is useful to know this before sorting and merging as the merged data set might not be useful.

Use PROC SQL to identify IDs that occur more than once in both data sets:

```
proc sql;
title 'Check for duplicate identifiers in both data sets DS1 and DS2';
select ds1.id,count1,count2
from (select id,count(*) as count1
       from ds1
       group by id),
      (select id,count(*) as count2
       from ds2
       group by id)
where ds1.id=ds2.id and (count1 > 1 and count2 > 1);
quit;
```

This query uses two in-line views on the FROM clause. The first in-line view groups observations in data set DS1 by ID. It counts how many times each occurs and stores the result in COUNT1. The second in-line view does the same for data set DS2 and stores the result in COUNT2.

The two in-line views are separated by a comma. This tells PROC SQL to join (merge) them. Technically, it requests a Cartesian product (i.e., each row in the first in-line view is merged with every row in the second in-line view).

The WHERE clause subsets the Cartesian product to include only rows where the ID from data set DS1 equals the ID from data set DS2 and both COUNT1 and COUNT2 are greater than 1. Note that because both variables are named ID, they must be prefixed with their data set name to avoid ambiguity.

The outer SELECT clause selects ID from data set DS1. (It does not matter if ID is selected from DS1 or DS2: only rows with matching IDs were subset on the WHERE clause.) It also selects both COUNT1 and COUNT2.

The results are:

```
Check for duplicate identifiers in both data sets DS1 and DS2

      id    count1    count2
----------------------------
       3         3         4
       5         2         2
```

Output 25. ID values with duplicate occurrences in both data sets DS1 and DS2.

ID=3 occurred 3 times in data set DS1 and 4 times in data set DS2.

ID=5 occurred 2 times in data set DS1 and 2 times in data set DS2.

Summary

Use PROC SQL to check for duplicate identifiers in more than one data set being merged.

When there is more than one data set with repeats of BY values in a merge, the resulting data set is probably not useful. Observations are combined one-to-one within the same BY group. If one data set has more observations in a BY group than another, the data set with fewer observations has variables unique to it retained and merged with any remaining observations from the other data set.

This technique can be used to count occurrences of identifiers in each data set, merge them, and keep only observations where identifiers are duplicated in more than one data set. Results could be saved to a data set with the CREATE TABLE clause.

The observations must be inspected to determine why there are duplicate identifiers in both data sets and which observation(s) to keep. It is recommended that there be duplicate identifiers in no more than one of the data sets being merged.

3. CHECK FOR ACCEPTABLE MATCH RATES

Data sets being merged usually have some, if not all, observations that match on values of BY variables. A low match rate might make the merged data set less useful or even unusable.

Consider the following two data sets:

```
PRINT of data set DS1

Obs    id     x

 1      1    11
 2      2    12
 3      3    13
 4      5    15
```

```
PRINT of data set DS2

Obs    id     y

 1      1    11
 2      2    22
 3      4    44
 4      5    55
 5      6    66
```

Output 26. PROC PRINT of data sets DS1 and DS2.

Data set DS1 has 4 observations and 2 variables.

Data set DS2 has 5 observations and 2 variables.

Three values of ID occur in both data sets: ID=1, ID=2, and ID=5.

Data sets can be merged and the match rate assessed:

```
proc sort data=ds1 out=ds1S;
by id;
run;

proc sort data=ds2 out=ds2S;
by id;
run;

data ds12allmerge;
merge ds1S(in=inds1SX)
      ds2S(in=inds2SX);
by id;
inds1S=inds1SX;
inds2S=inds2SX;
run;

proc freq data=ds12allmerge;
title 'Check IN= variables for match rate';
tables inds1S*inds2S;
run;
```

Both data sets are sorted with PROC SORT.

The sorted data sets are then merged by ID.

The MERGE uses IN= data set options to create variables that equal 1 when the data set contributed to the current observation and 0 when it did not. The IN= variables are temporary, so values are assigned to new variables.

PROC FREQ is used to run a crosstab of the new variables and summarize matches.

The results are:

```
Check IN= variables for match rate

The FREQ Procedure

Table of inds1S by inds2S

inds1S     inds2S

Frequency|
Percent  |
Row Pct  |
Col Pct  |       0|       1|  Total
---------+--------+--------+
       0 |      0 |      2 |      2
         |   0.00 |  33.33 |  33.33
         |   0.00 | 100.00 |
         |   0.00 |  40.00 |
---------+--------+--------+
       1 |      1 |      3 |      4
         |  16.67 |  50.00 |  66.67
         |  25.00 |  75.00 |
         | 100.00 |  60.00 |
---------+--------+--------+
Total           1        5        6
            16.67    83.33   100.00
```

Output 27. PROC FREQ output summarizing the match rate between data sets DS1 and DS2.

There was only a 50% match rate between data sets DS1 and DS2 when observations were merged by ID (i.e., when INDS1S=1 and INDS2S=1).

This might well be lower than what was expected. The merged data set might or might not be useful.

It wastes computer time and resources to sort and merge data sets with an unacceptable match rate.

Use a similar approach, but keep only ID, to confirm that the match rate is acceptable before merging:

```
proc sort data=ds1(keep=id) out=ds1S2;
by id;
run;

proc sort data=ds2(keep=id) out=ds2S2;
by id;
run;

data ds12idmerge;
merge ds1S2(in=inds1S2X)
      ds2S2(in=inds2S2X);
by id;
inds1S2=inds1S2X;
inds2S2=inds2S2X;
run;
```

```
proc freq data=ds12idmerge;
title 'Check IN= variables for match rate (ID only)';
tables inds1S2*inds2S2;
run;
```

Both data sets are again sorted with PROC SORT and merged by ID.

In both PROC SORT steps, the KEEP= data set option is specified in parentheses after the input data set to be sorted. This means only values of ID are sorted and output to the respective OUT= data set.

The MERGE again uses data set options to create IN= variables that are assigned to new variables.

PROC FREQ is used to run a crosstab of the new variables and summarize matches.

The results are:

```
Check IN= variables for match rate (ID only)

The FREQ Procedure

Table of inds1S2 by inds2S2

inds1S2     inds2S2

Frequency|
Percent  |
Row Pct  |
Col Pct  |       0|       1|  Total
---------+--------+--------+
       0 |      0 |      2 |      2
         |   0.00 |  33.33 |  33.33
         |   0.00 | 100.00 |
         |   0.00 |  40.00 |
---------+--------+--------+
       1 |      1 |      3 |      4
         |  16.67 |  50.00 |  66.67
         |  25.00 |  75.00 |
         | 100.00 |  60.00 |
---------+--------+--------+
Total           1        5        6
            16.67    83.33   100.00
```

Output 28. PROC FREQ output summarizing the match rate between data sets DS1 and DS2.

The results in Output 28 are identical to Output 27. There was only a 50% match rate between data sets DS1 and DS2 when observations were merged by ID.

This time, however, only values of ID were sorted, merged, and summarized with PROC FREQ.

The low match rate might preclude merging the full data sets, depending on prior expectations.

Summary

Use PROC SORT, keeping only the identifier variable(s), to create small output data sets. Merge these data sets using IN= variables and assign results to new variables.

Summarize the match rate with PROC FREQ and a crosstab of the IN= variables.

Review the results to determine if the match rate is acceptable (i.e., warrants sorting and merging the full data sets).

CONCLUSION

It is best to identify any issues before combining data sets. This is especially important when working with very large data sets that require significant computer time and resources to combine.

Check for variables that are not in all data sets; common variables with different attributes; duplicate identifiers; and duplicate observations before concatenating data sets.

Check for variables in more than one data set; duplicate identifiers in more than one data set; and acceptable match rates before merging data sets.

Resolve issues with the source data sets before combining them or during the concatenation or merge.

RECOMMENDED READING

- "Uncommon Techniques for Common Variables"
- "Dealing with Duplicates"
- "Automatically Renaming Common Variables Before Merging"
- "Subsetting Observations from Large SAS® Data Sets"

CONTACT INFORMATION

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

chrisbost@gmail.com