

Matching SAS® Data Sets with PROC SQL: If at First You Don't Succeed, Match, Match Again

Imelda C. Go, South Carolina Department of Education, Columbia, SC

ABSTRACT

Two data sets are often matched by using the MERGE and BY statements in a DATA step. If records between the two data sets did not match successfully, there may be a need to continue matching the remaining records using a different set of BY variables. If the goal is to match as many records as possible between the two original data sets, then this process may have to be repeated several times using many sets of BY variables. This paper provides a PROC SQL solution to matching data using several sets of matching criteria. This is the PROC SQL version of a 2004 SouthEast SAS® Users Group (SESUG) paper that exclusively used PROC SORT, the DATA step, and the MERGE and BY statements to match data using several sets of criteria.

Acknowledgments: I would like to thank Stephanie Thompson, Todd Weiss, and SAS Technical Support for helping me convert the 2004 SAS code. I am now a PROC SQL fan.

Although SAS programmers in the education field are used to producing student longitudinal data involving test scores for various evaluation and accountability purposes, *No Child Left Behind* (NCLB) legislation has raised the premium for longitudinal data. Because of the high stakes involved in determining *Adequate Yearly Progress* (AYP) for public schools, the programmer is charged with the task of producing as many matches as possible.

Critical to any high-stakes programming situation is an understanding of data quality issues, sources of error, and how SAS processes data. Knowing how things can go wrong during the matching process is essential to anticipating problems that need to be addressed during longitudinal data collection. Being unaware of how problems can arise could potentially allow problems to occur undetected, and thus endanger the validity of the data.

Some of the things that need to be considered when matching data are:

- **Data quality**
Data quality affects the data processing and the programming required to produce the longitudinal data.
- **Matching process**
By what variables can we match the data? Can the data be matched on a single value, such as the social security number (SSN)? What other sets of variables can the data be matched on? Scrutinizing the data can lead to insights on what else can be done to match records. For example, if two data sets produced much fewer than expected matches, a visual inspection can reveal that one data set's last name field is only 10 characters long instead of the 14 characters in the other data set.
- **Tolerance for error**
The tolerance for error is usually determined by the consequences of an incorrect match. The SSN may have incorrect digits and may accidentally match another SSN. If the SSN is expected to be problematic, consider matching with the SSN and other variables. If the tolerance for error is high, less stringent (e.g., fuzzy matching methods) can be used.

BRIEF REVIEW OF MATCHING IN SAS

There are different tools for matching data in SAS. It is important to know how each tool behaves and what its limitations are.

Do Not Use the DATA Step for Many-to-Many Matching

The DATA step can handle one-to-one, one-to-many, and many-to-one matching but not many-to-many matching. For true many-to-many matches, the result should be a cross product. For example, if there are two records that match from one contributing data set to two records from the other, the resulting data set should have $2 \times 2 = 4$ records.

Consider two data sets, data1 and data2.

data1 data set		data2 data set	
GENDER	NAME1	GENDER	NAME2
Female	Linda	Female	May
Female	Marcy	Female	Gloria

When the data1 records are merged by gender with the data2 records, the ideal result would be $2 \times 2 = 4$ records.

GENDER	NAME1	NAME2
Female	Linda	May
Female	Linda	Gloria
Female	Marcy	May
Female	Marcy	Gloria

Using the following statements, only two records will result and all possible combinations are **not** given. The DATA step requires that both data sets be sorted by gender first before matching. To merge the data by gender using the DATA step, the gender variable must have the same variable name on both data sets so that it can be used in the DATA step's BY statement.

```
proc sort data=data1; by gender;
proc sort data=data2; by gender;
data combo; merge data1 data2;
               by gender;
```

Obs	gender	name1	name2
1	Female	Linda	May
2	Female	Marcy	Gloria

PROC SQL for Many-to-Many Matches

PROC SQL can handle all types of matching, even many-to-many matching. PROC SQL can be used to join records without sorting and without requiring that variables from different sources have the same names. The SQL statement replaces two PROC SORT procedures and one DATA step. When large data sets are being matched, the resulting cross product could be a very large data set.

```
proc sql;
  select data1.gender, name1, name2
  from data1, data2
  where data1.gender=data2.gender;
```

gender	name1	name2
Female	Linda	May
Female	Linda	Gloria
Female	Marcy	May
Female	Marcy	Gloria

MATCHING ON KEY VARIABLES

Matching can be done on one or more key variables. When only one variable is used, a nonmatch implies the values on that one variable were not the same. If two variables are used for the match, a nonmatch implies the values differed on one or both variables. As more variables are used, the matching criteria become more stringent. Matches must be equal on more variables and differences between those variables are more likely to occur than when fewer variables are involved.

When a unique identifier, such as the SSN, is used to match, make sure the SSN data type is the same. Can the SSN be a character type? If one or more digits of the SSN are missing and a space is used to indicate the missing digit(s), the character type would preserve the information. For example, if the SSN of '251112222' is gridded on a machine-scannable form and some of the grid marks were not scanned properly, the data file might have '25?12222' where ? is a digit that did not scan properly. If the data type is numeric, '251 12 22' (4th and 7th digits are missing) and '25?12222' are invalid numeric values and are set to missing.

When data are matched on character variables (e.g., last name, first name), make sure the character values are as consistent as possible. Reducing inconsistencies in character values can help maximize the number of successful matches.

Comparing strings is case-sensitive. That is, 'Edward' is not the same as 'EDWARD'. Typing is subject to human error. For example, 'EDWARD' may have been typed as 'EDWard'. One solution is to use the UPCASE or LOWCASE functions.

sample statement	value of test
test=upcase('Edward');	'EDWARD'
test=lowcase('EDWard');	'edward'

'Mary Ann' might be entered as 'Mary Ann' where there are two spaces between Mary and Ann. Use the COMPBL function to convert two or more consecutive blanks into one blank.

test=compbl(name);	
value of name	value of test
'Mary Ann'	'Mary Ann'
'Mary Ann'	'Mary Ann'

'Mary Ann' might be entered as ' Mary Ann' where there is a leading space before Mary. Use the LEFT function to left align a character expression.

test=left(name);	
value of name	value of test
' Mary Ann'	'Mary Ann'
' Mary Ann'	'Mary Ann'

SAS ignores trailing spaces when character expressions are compared (e.g., 'Mary Ann ' is equivalent to 'Mary Ann'). If there are any unwanted spaces in the first name, use the COMPRESS function to remove specified characters from a string. If no characters for removal are specified, the function removes spaces by default.

test=compress(name);	
value of name	value of test
'Edw ard'	'Edward'
' Edward'	'Edward'

There are first names that consist of two words? If all the spaces are removed from the names used to match, then the only issue is whether the original name value needs to be retained. If there is any such concern, keep two name fields. The first one would be the original name, the second one would be the edited name used for matching.

Other characters may also need to be removed. List the characters to be removed in single quotes.

```
test=compress(name, '?-');
```

value of name	value of test
'Kidman-Cruise?'	'KidmanCruise'

There may be an attempt to type accent marks into the value of a name. For example, *André* might be typed in as Andre' and this is really a data entry standard issue. When the typist encounters an accent mark, should they attempt to type the accent mark? That particular accent mark lends itself well to an apostrophe. There are of course other foreign accent marks (e.g., ö) that would not lend themselves well to plain text characters.

What if the apostrophe itself needs to be eliminated? Use the following syntax.

```
test=compress(name, ''');
```

value of name	value of test
Andre'	Andre

Should nonletter characters be eliminated? An actual example encountered is the accidental or intentional typing of nonletter characters in names on student databases. Student names were sent to a testing company that creates barcoded labels for test documents. In the process of creating the labels, the testing company eliminated all nonletter characters from the student names. Students with no barcoded labels during testing indicate their names by manually gridding a machine-scannable form. The forms often only provided grids for letter characters in name fields. If data on the student database needed to be matched to data received from the testing company later, the first set (original data from databases) of names had nonletter characters while the second set (data for barcoding) of names has no such characters. The number of matches would not be maximized if the nonletter characters were not taken out of the original names prior to matching.

PROGRAMMING EXAMPLE

The programming example uses the following data sets. The variable names were assigned to facilitate the discussion of the examples. The variables in data set amaster1 all begin with a and those in data set bmaster1 all begin with b.

Description	Data Set amaster1 Variables	Data Set bmaster1 Variables
School	aschool	bschool
Student ID	astudentid	bstudentid
Name	aname	bname
Grade	agrade	bgrade
Date of Birth	adob	bdob

The goal is to match as many records as possible between the two data sets. Another constraint is the matching that needs to be done can only involve one-to-one matches. That is, one-to-many, many-to-one, and many-to-many matches are invalid. Hence, additional SAS statements were written to exclude records that will produce a many-to-many situation from the data sets being merged. However, the excluded stage1 records are later considered for stage 2 matching. (If one-to-many, many-to-one, or many-to-many matches are not an issue, the master data sets do not need to be divided.)

In the example, two stages were used. The first one matched on student ID and name while the second one matched on school, name, and date of birth. In the next pages, the process is outlined first before showing the corresponding SAS programming statements. The process can be easily extended to more stages.

OUTLINE OF THE PROCESS

This section provides an outline of the process with data set examples.

Stage 1 matched two master data sets, amaster1 and bmaster1, on studentid and name.

amaster1						bmaster1					
Obs	aschool	astudentid	aname	agrade	adob	Obs	bschool	bstudentid	bname	bgrade	bdob
1	ABC School	12	Jane	2	19950101	1	ABC School	12	Jane	2	19950101
2	ABC School	13	Jim	2	19950101	2	ABC School	13	Jim	2	19950101
3	ABC School	14	Jane	2	19950101	3	ABC School	16	Jane	2	19950101
4	ABC School	15	Jim	2	19950101	4	ABC School	17	Jim	2	19950101
5	ABC School	19	Jane	2	19950101	5	ABC School	19	Jane	2	19950101
6	ABC School	19	Jane	2	19950101	6	ABC School	19	Jane	2	19950101
7	ABC School	20	Jenn	2	19950102	7	ABC School	20	Jen	2	19950102

Step 1.1 The amaster1 data set was divided into data sets aunique1 (records with unique astudentid and aname values) and adup1 (records with duplicate astudentid and aname values).

aunique1					adup1				
aschool	astudentid	aname	agrade	adob	aschool	astudentid	aname	agrade	adob
ABC School	12	Jane	2	19950101	ABC School	19	Jane	2	19950101
ABC School	13	Jim	2	19950101	ABC School	19	Jane	2	19950101
ABC School	14	Jane	2	19950101					
ABC School	15	Jim	2	19950101					
ABC School	20	Jenn	2	19950102					

Step 1.2 The bmaster1 data set was divided into data sets bunique1 (records with unique bstudentid and bname values) and bdup1 (records with duplicate bstudentid and bname values).

bunique1					bdup1				
bschool	bstudentid	bname	bgrade	bdob	bschool	bstudentid	bname	bgrade	bdob
ABC School	12	Jane	2	19950101	ABC School	19	Jane	2	19950101
ABC School	13	Jim	2	19950101	ABC School	19	Jane	2	19950101
ABC School	16	Jane	2	19950101					
ABC School	17	Jim	2	19950101					
ABC School	20	Jen	2	19950102					

Step 1.3 Data sets aunique1 and bunique1 were matched on studentid and name. These variables did not have the same name on both master data sets. The successful matches were placed in data set matches1. The recordsource variable was added to the data set to indicate that the match was a result of stage 1.

matches1										
aschool	astudentid	aname	agrade	adob	bschool	bstudentid	bname	bgrade	bdob	recordsource
ABC School	12	Jane	2	19950101	ABC School	12	Jane	2	19950101	1
ABC School	13	Jim	2	19950101	ABC School	13	Jim	2	19950101	1

Step 1.4 The aunique1 records with no matches were placed in the aremainder1 data set.

aremainder1				
aschool	astudentid	aname	agrade	adob
ABC School	14	Jane	2	19950101
ABC School	15	Jim	2	19950101
ABC School	20	Jenn	2	19950102

Step 1.5 The bunique1 records with no matches were placed in the bremainder1 data set.

bremainder1				
bschool	bstudentid	bname	bgrade	bdob
ABC School	16	Jane	2	19950101
ABC School	17	Jim	2	19950101
ABC School	20	Jen	2	19950102

Stage 2 matched two master data sets, amaster2 and bmaster2, on school, name, and dob values.

Step 2.1 The amaster2 data set consisted of aremainder1 and adup1 records.

amaster2				
aschool	astudentid	aname	agrade	adob
ABC School	14	Jane	2	19950101
ABC School	15	Jim	2	19950101
ABC School	20	Jenn	2	19950102
ABC School	19	Jane	2	19950101
ABC School	19	Jane	2	19950101

Step 2.2 The bmaster2 data set consisted of bremainder1 and bdup1 records.

bmaster2				
bschool	bstudentid	bname	bgrade	bdob
ABC School	16	Jane	2	19950101
ABC School	17	Jim	2	19950101
ABC School	20	Jen	2	19950102
ABC School	19	Jane	2	19950101
ABC School	19	Jane	2	19950101

Step 2.3 The amaster2 data set was divided into data sets aunique2 (records with unique aschool, aname, and adob values) and adup2 (records with duplicate aschool, aname, and adob values).

aunique2				
aschool	astudentid	aname	agrade	adob
ABC School	20	Jenn	2	19950102
ABC School	15	Jim	2	19950101

adup2				
aschool	astudentid	aname	agrade	adob
ABC School	19	Jane	2	19950101
ABC School	14	Jane	2	19950101
ABC School	19	Jane	2	19950101

Step 2.4 The bmaster2 data set was divided into data sets bunique2 (records with unique bschool, bname, and bdob values) and bdup2 (records with duplicate bschool, bname, and bdob values).

bunique2				
bschool	bstudentid	bname	bgrade	bdob
ABC School	20	Jen	2	19950102
ABC School	17	Jim	2	19950101

bdup2				
bschool	bstudentid	bname	bgrade	bdob
ABC School	19	Jane	2	19950101
ABC School	16	Jane	2	19950101
ABC School	19	Jane	2	19950101

Step 2.5 Data sets aunique2 and bunique2 were matched on school, name, and dob values. These variables did not have the same name on both master data sets. The successful matches were placed in data set matches2. The recordsource variable was added to the data set to indicate that the match was a result of stage 2.

matches2										
aschool	astudent id	aname	agrade	adob	bschool	bstudent id	bname	bgrade	bdob	recordsource
ABC School	15	Jim	2	19950101	ABC School	17	Jim	2	19950101	2

Step 2.6 The aunique2 records with no matches were placed in the aremainder2 data set.

aremainder2				
aschool	astudentid	aname	agrade	adob
ABC School	20	Jenn	2	19950102

Step 2.7 The bunique2 records with no matches were placed in the bremainder2 data set.

bremainder2				
bschool	bstudentid	bname	bgrade	bdob
ABC School	20	Jen	2	19950102

THE SAS CODE

This section provides the SAS statements that produced each data set.

Stage 1 matched two master data sets, amaster1 and bmaster1, on studentid and name.	
Step 1.1 The amaster1 data set was divided into data sets aunique1 (records with unique astudentid and aname values) and adup1 (records with duplicate astudentid and aname values).	<pre> title 'aunique1'; proc sql; create table aunique1 as select * from amaster1 group by astudentid, aname having count(*)=1; quit; title 'adup1'; proc sql; create table adup1 as select * from amaster1 group by astudentid, aname having count(*)>1; quit; </pre>
Step 1.2 The bmaster1 data set was divided into data sets bunique1 (records with unique bstudentid and bname values) and bdup1 (records with duplicate bstudentid and bname values).	<pre> title 'bunique1'; proc sql; create table bunique1 as select * from bmaster1 group by bstudentid, bname having count(*)=1; quit; title 'bdup1'; proc sql; create table bdup1 as select * from bmaster1 group by bstudentid, bname having count(*)>1; quit; </pre>
Step 1.3 Data sets aunique1 and bunique1 were matched on studentid and name. These variables did not have the same name on both master data sets. The successful matches were placed in data set matches1. The recordsource variable was added to the data set to indicate that the match was a result of stage 1.	<pre> title 'matches1'; proc sql; create table matches1 as select *, "1" as recordsource from aunique1, bunique1 where aunique1.astudentid=bunique1.bstudentid and aunique1.aname=bunique1.bname; select * from matches1; quit; </pre>
Step 1.4 The aunique1 records with no matches were placed in the aremainder1 data set.	<pre> title 'aremainder1'; proc sql; create table aremainder1 as select aschool, astudentid, aname, agrade, adob from aunique1 left join bunique1 on aunique1.astudentid=bunique1.bstudentid and aunique1.aname=bunique1.bname where bunique1.bstudentid is null and bunique1.bname is null; quit; </pre>
Step 1.5 The bunique1 records with no matches were placed in the bremainder1 data set.	<pre> title 'bremainder1'; proc sql; create table bremainder1 as select bschool, bstudentid, bname, bgrade, bdob from bunique1 left join aunique1 on aunique1.astudentid=bunique1.bstudentid and aunique1.aname=bunique1.bname where aunique1.astudentid is null and aunique1.aname is null; quit; </pre>
Stage 2 matched two master data sets, amaster2 and bmaster2, on school, name, and dob values.	
Step 2.1 The amaster2 data set consisted of aremainder1 and adup1 records.	<pre> title 'amaster2'; proc sql; create table amaster2 as select * from aremainder1 union all select * from adup1; quit; </pre>

Step 2.2 The bmaster2 data set consisted of bremainder1 and bdup1 records.	<pre> title 'bmaster2'; proc sql; create table bmaster2 as select * from bremainder1 union all select * from bdup1; quit; </pre>
Step 2.3 The amaster2 data set was divided into data sets aunique2 (records with unique aschool, aname, and adob values) and adup2 (records with duplicate aschool, aname, and adob values).	<pre> **step 2.3; title 'aunique2'; proc sql; create table aunique2 as select * from amaster2 group by aschool, aname, adob having count(*)=1; quit; title 'adup2'; PROC SQL; create table adup2 as select * from amaster2 group by aschool, aname, adob having count(*)>1; quit; </pre>
Step 2.4 The bmaster2 data set was divided into data sets bunique2 (records with unique bschool, bname, and bdob values) and bdup2 (records with duplicate bschool, bname, and bdob values).	<pre> title 'bunique2'; proc sql; create table bunique2 as select * from bmaster2 group by bschool, bname, bdob having count(*)=1; quit; title 'bdup2'; PROC SQL; create table bdup2 as select * from bmaster2 group by bschool, bname, bdob having count(*)>1; quit; </pre>
Step 2.5 Data sets aunique2 and bunique2 were matched on school, name, and dob values. These variables did not have the same name on both master data sets. The successful matches were placed in data set matches2. The recordsource variable was added to the data set to indicate that the match was a result of stage 2.	<pre> title 'matches2'; proc sql; create table matches2 as select *, "2" as recordsource from aunique2, bunique2 where aunique2.aschool=buunique2.bschool and aunique2.aname=buunique2.bname and aunique2.adob=buunique2.bdob; select * from matches2; quit; </pre>
Step 2.6 The aunique2 records with no matches were placed in the aremainder2 data set.	<pre> title 'aremainder2'; proc sql; create table aremainder2 as select aschool, astudentid, aname, agrade, adob from aunique2 left join bunique2 on aunique2.aschool=buunique2.bschool and aunique2.aname=buunique2.bname and aunique2.adob=buunique2.bdob where buunique2.bschool is null and buunique2.bname is null and buunique2.bdob is null; quit; </pre>
Step 2.7 The bunique2 records with no matches were placed in the bremainder2 data set.	<pre> title 'bremainder2'; proc sql; create table bremainder2 as select bschool, bstudentid, bname, bgrade, bdob from bunique2 left join aunique2 on aunique2.aschool=buunique2.bschool and aunique2.aname=buunique2.bname and aunique2.adob=buunique2.bdob where aunique2.aschool is null and aunique2.aname is null and aunique2.adob is null; quit; </pre>

USING MACROS

The steps in stages 1 and 2 have a noticeable pattern and can be generalized using macros. SAS macros can be used to facilitate coding for several stages of matching. The following statements can be used to replace the previous code. Even more aspects of the program can be rewritten using macros. However, it is good to maintain a balance between the use of macros and the readability of programs. Sometimes the time saved by using macros is negated by a program having little or no documentation, that is difficult to read, or that cannot be easily modified even by the original programmer.

```
%macro divide (prefix,stage,groupvars);
    title "&prefix.unique&stage";
    proc sql;
    create table &prefix.unique&stage as
    select *
    from &prefix.master&stage
    group by &groupvars
    having count(*)=1; quit;
    title "&prefix.dup&stage";
    proc sql;
    create table &prefix.dup&stage as
    select *
    from &prefix.master&stage
    group by &groupvars
    having count(*)>1; quit;
%mend divide;

%macro match(stage,wherecondition);
    title "matches&stage";
    proc sql;
    create table matches&stage as
    select *, "&stage" as recordsource
    from aunique&stage, bunique&stage
    where &wherecondition; quit;
%mend match;

%macro aremainder(stage,avars,oncondition,wherecondition);
    title "aremainder&stage";
    proc sql;
    create table aremainder&stage as
    select &avars
    from aunique&stage left join bunique&stage
    on &oncondition
    where &wherecondition; quit;
%mend aremainder;

%macro bremainder(stage,bvars,oncondition,wherecondition);
    title "bremainder&stage";
    proc sql;
    create table bremainder&stage as
    select &bvars
    from bunique&stage left join aunique&stage
    on &oncondition
    where &wherecondition; quit;
%mend bremainder;

%macro nextmasters(stage);
    title "amaster&stage";
    proc sql;
    create table amaster&stage as
    select * from aremainder%eval(&stage-1)
    union all
    select * from adup%eval(&stage-1); quit;
    title "bmaster&stage";
    proc sql;
    create table bmaster&stage as
    select * from bremainder%eval(&stage-1)
    union all
    select * from bdup%eval(&stage-1); quit;
%mend nextmasters;
```

```

%divide(prefix=a, stage=1, groupvars=%str(astudentid, aname));
%divide(prefix=b, stage=1, groupvars=%str(bstudentid, bname));

%match(stage=1, wherecondition=%str(aunique1.astudentid=bunique1.bstudentid and
aunique1.aname=bunique1.bname));

%aremainder(stage=1, avars=%str(aschool, astudentid, aname, agrade, adob),
oncondition=%str(aunique1.astudentid=bunique1.bstudentid and aunique1.aname=bunique1.bname),
wherecondition=%str(bunique1.bstudentid is null and bunique1.bname is null))

%bremainder(stage=1, bvars=%str(bschool, bstudentid, bname, bgrade, bdob),
oncondition=%str(aunique1.astudentid=bunique1.bstudentid and aunique1.aname=bunique1.bname),
wherecondition=%str(aunique1.astudentid is null and aunique1.aname is null))

%nextmasters(stage=2);

%divide(prefix=a, stage=2, groupvars=%str(aschool, aname, adob));
%divide(prefix=b, stage=2, groupvars=%str(bschool, bname, bdob));

%match(stage=2, wherecondition=%str(aunique2.aschool=bunique2.bschool and
aunique2.aname=bunique2.bname
and aunique2.adob=bunique2.bdob));

%aremainder(stage=2, avars=%str(aschool, astudentid, aname, agrade, adob),
oncondition=%str(aunique2.aschool=bunique2.bschool and aunique2.aname=bunique2.bname
and aunique2.adob=bunique2.bdob),
wherecondition=%str(bunique2.bschool is null and bunique2.bname is null
and bunique2.bdob is null))

%bremainder(stage=2, bvars=%str(bschool, bstudentid, bname, bgrade, bdob),
oncondition=%str(aunique2.aschool=bunique2.bschool and aunique2.aname=bunique2.bname
and aunique2.adob=bunique2.bdob),
wherecondition=%str(aunique2.aschool is null and aunique2.aname is null
and aunique2.adob is null))
run;

```

CONCLUSION

An understanding of data quality issues, sources of error, how SAS processes data, and how things can go wrong during the matching process are essential to anticipating problems that need to be addressed when matching data. Being unaware of how problems can occur could potentially allow such problems to occur undetected, and thus endanger the validity of the data.

REFERENCES

Go, Imelda. 2004. "Matching SAS® Data Sets: If at First You Don't Succeed, Match, Match Again." Proceedings CD of the Twelfth Annual Conference of the SouthEast SAS Users Group, Nashville, TN.

SAS Institute Inc. 2002. *SAS®9 Language Reference: Concepts*. Cary, NC: SAS Institute Inc. SAS Institute Inc. 2002.

SAS Institute Inc. 2002. *SAS®9 Language Reference: Dictionary, Volumes 1 and 2*. Cary, NC: SAS Institute Inc.

TRADEMARK NOTICE

SAS is a registered trademark or trademark of the SAS Institute Inc. in the USA and other countries. ® indicates USA registration.