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`.

<table>
<thead>
<tr>
<th><code>data1</code> data set</th>
<th><code>data2</code> data set</th>
</tr>
</thead>
<tbody>
<tr>
<td>GENDER NAME1</td>
<td>GENDER NAME2</td>
</tr>
<tr>
<td>Female Linda</td>
<td>Female May</td>
</tr>
<tr>
<td>Female Marcy</td>
<td>Female Gloria</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>GENDER NAME1 NAME2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Female Linda May</td>
</tr>
<tr>
<td>Female Linda Gloria</td>
</tr>
<tr>
<td>Female Marcy May</td>
</tr>
<tr>
<td>Female Marcy Gloria</td>
</tr>
</tbody>
</table>

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;
```

<table>
<thead>
<tr>
<th>Obs</th>
<th>gender</th>
<th>name1</th>
<th>name2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Female</td>
<td>Linda</td>
<td>May</td>
</tr>
<tr>
<td>2</td>
<td>Female</td>
<td>Marcy</td>
<td>Gloria</td>
</tr>
</tbody>
</table>

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;
```

<table>
<thead>
<tr>
<th>gender</th>
<th>name1</th>
<th>name2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Female</td>
<td>Linda</td>
<td>May</td>
</tr>
<tr>
<td>Female</td>
<td>Linda</td>
<td>Gloria</td>
</tr>
<tr>
<td>Female</td>
<td>Marcy</td>
<td>May</td>
</tr>
<tr>
<td>Female</td>
<td>Marcy</td>
<td>Gloria</td>
</tr>
</tbody>
</table>
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 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 ‘25112222’ is grided 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
---------- | ----------
upcase('Edward'); | 'EDWARD'
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.

```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.

```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.

```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.

\[
\text{test=compress(name,'?-');}
\]

<table>
<thead>
<tr>
<th>value of name</th>
<th>value of test</th>
</tr>
</thead>
<tbody>
<tr>
<td>'Kidman-Cruise?'</td>
<td>'KidmanCruise'</td>
</tr>
</tbody>
</table>

There may be an attempt to type accent marks into the value of a name. For example, Andrê 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.

\[
\text{test=compress(name,'''');}
\]

<table>
<thead>
<tr>
<th>value of name</th>
<th>value of test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Andre'</td>
<td>Andre</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Description</th>
<th>Data Set amaster1 Variables</th>
<th>Data Set bmaster1 Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>School</td>
<td>aschool</td>
<td>bschool</td>
</tr>
<tr>
<td>Student ID</td>
<td>astudentid</td>
<td>bstudentid</td>
</tr>
<tr>
<td>Name</td>
<td>aname</td>
<td>bname</td>
</tr>
<tr>
<td>Grade</td>
<td>agrade</td>
<td>bgrade</td>
</tr>
<tr>
<td>Date of Birth</td>
<td>adob</td>
<td>bdob</td>
</tr>
</tbody>
</table>

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, a\textit{master1} and b\textit{master1}, on studentid and name.

<table>
<thead>
<tr>
<th>a\textit{master1}</th>
<th>b\textit{master1}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Obs</td>
<td>aschool</td>
</tr>
<tr>
<td>1</td>
<td>ABC School</td>
</tr>
<tr>
<td>2</td>
<td>ABC School</td>
</tr>
<tr>
<td>3</td>
<td>ABC School</td>
</tr>
<tr>
<td>4</td>
<td>ABC School</td>
</tr>
<tr>
<td>5</td>
<td>ABC School</td>
</tr>
<tr>
<td>6</td>
<td>ABC School</td>
</tr>
<tr>
<td>7</td>
<td>ABC School</td>
</tr>
</tbody>
</table>

Step 1.1 The a\textit{master1} data set was divided into data sets a\textit{unique1} (records with unique astudentid and aname values) and a\textit{dup1} (records with duplicate astudentid and aname values).

<table>
<thead>
<tr>
<th>a\textit{unique1}</th>
<th>a\textit{dup1}</th>
</tr>
</thead>
<tbody>
<tr>
<td>aschool</td>
<td>astudentid</td>
</tr>
<tr>
<td>ABC School</td>
<td>12</td>
</tr>
<tr>
<td>ABC School</td>
<td>13</td>
</tr>
<tr>
<td>ABC School</td>
<td>14</td>
</tr>
<tr>
<td>ABC School</td>
<td>15</td>
</tr>
<tr>
<td>ABC School</td>
<td>19</td>
</tr>
<tr>
<td>ABC School</td>
<td>19</td>
</tr>
<tr>
<td>ABC School</td>
<td>20</td>
</tr>
</tbody>
</table>

Step 1.2 The b\textit{master1} data set was be divided into data sets b\textit{unique1} (records with unique bstudentid and bname values) and b\textit{dup1} (records with duplicate bstudentid and bname values).

<table>
<thead>
<tr>
<th>b\textit{unique1}</th>
<th>b\textit{dup1}</th>
</tr>
</thead>
<tbody>
<tr>
<td>bschool</td>
<td>bstudentid</td>
</tr>
<tr>
<td>ABC School</td>
<td>12</td>
</tr>
<tr>
<td>ABC School</td>
<td>13</td>
</tr>
<tr>
<td>ABC School</td>
<td>16</td>
</tr>
<tr>
<td>ABC School</td>
<td>17</td>
</tr>
<tr>
<td>ABC School</td>
<td>19</td>
</tr>
<tr>
<td>ABC School</td>
<td>19</td>
</tr>
<tr>
<td>ABC School</td>
<td>20</td>
</tr>
</tbody>
</table>

Step 1.3 Data sets a\textit{unique1} and b\textit{unique1} 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 a\textit{matches1}. The recordsource variable was added to the data set to indicate that the match was a result of stage 1.

<table>
<thead>
<tr>
<th>a\textit{matches1}</th>
<th>a\textit{matches1}</th>
</tr>
</thead>
<tbody>
<tr>
<td>aschool</td>
<td>astudentid</td>
</tr>
<tr>
<td>ABC School</td>
<td>12</td>
</tr>
<tr>
<td>ABC School</td>
<td>13</td>
</tr>
</tbody>
</table>

Step 1.4 The a\textit{unique1} records with no matches were placed in the a\textit{remainder1} data set.

<table>
<thead>
<tr>
<th>a\textit{remainder1}</th>
</tr>
</thead>
<tbody>
<tr>
<td>aschool</td>
</tr>
<tr>
<td>ABC School</td>
</tr>
<tr>
<td>ABC School</td>
</tr>
<tr>
<td>ABC School</td>
</tr>
</tbody>
</table>

Step 1.5 The b\textit{unique1} records with no matches were placed in the b\textit{remainder1} data set.

<table>
<thead>
<tr>
<th>b\textit{remainder1}</th>
</tr>
</thead>
<tbody>
<tr>
<td>bschool</td>
</tr>
<tr>
<td>ABC School</td>
</tr>
<tr>
<td>ABC School</td>
</tr>
<tr>
<td>ABC School</td>
</tr>
</tbody>
</table>
Stage 2 matched two master data sets, \textit{amaster2} and \textit{bmaster2}, on school, name, and dob values.

Step 2.1 The \textit{amaster2} data set consisted of \textit{aremainder1} and \textit{adup1} records.

\begin{tabular}{|c|c|c|}
\hline
\textit{amaster2} & \\
\hline
school & studentid & name \\
\hline
ABC School & 14 & Jane \\
ABC School & 15 & Jim \\
ABC School & 20 & Jane \\
ABC School & 19 & Jane \\
ABC School & 18 & Jane \\
\hline
\end{tabular}

Step 2.2 The \textit{bmaster2} data set consisted of \textit{bremain1} and \textit{bdup1} records.

\begin{tabular}{|c|c|c|}
\hline
\textit{bmaster2} & \\
\hline
bschool & bstudentid & bname \\
\hline
ABC School & 16 & Jane \\
ABC School & 17 & Jim \\
ABC School & 20 & Jane \\
ABC School & 19 & Jane \\
ABC School & 18 & Jane \\
\hline
\end{tabular}

Step 2.3 The \textit{amaster2} data set was divided into data sets \textit{aunique2} (records with unique school, name, and dob values) and \textit{adup2} (records with duplicate school, name, and dob values).

\begin{tabular}{|c|c|c|c|c|}
\hline
\textit{aunique2} & \\
\hline
school & studentid & name & grade & dob \\
\hline
ABC School & 20 & Jane & 2 & 19550101 \\
ABC School & 15 & Jim & 2 & 19550101 \\
\hline
\end{tabular}

\begin{tabular}{|c|c|c|c|c|}
\hline
\textit{adup2} & \\
\hline
school & studentid & name & grade & dob \\
\hline
ABC School & 19 & Jane & 2 & 19550101 \\
ABC School & 14 & Jane & 2 & 19550101 \\
ABC School & 18 & Jane & 2 & 19550101 \\
\hline
\end{tabular}

Step 2.4 The \textit{bmaster2} data set was divided into data sets \textit{bunique2} (records with unique school, name, and dob values) and \textit{bdup2} (records with duplicate school, name, and dob values).

\begin{tabular}{|c|c|c|c|c|}
\hline
\textit{bunique2} & \\
\hline
bschool & bstudentid & bname & grade & dob \\
\hline
ABC School & 20 & Jane & 2 & 19550101 \\
ABC School & 17 & Jim & 2 & 19550101 \\
\hline
\end{tabular}

\begin{tabular}{|c|c|c|c|c|}
\hline
\textit{bdup2} & \\
\hline
bschool & bstudentid & bname & grade & dob \\
\hline
ABC School & 19 & Jane & 2 & 19550101 \\
ABC School & 15 & Jane & 2 & 19550101 \\
ABC School & 18 & Jane & 2 & 19550101 \\
\hline
\end{tabular}

Step 2.5 Data sets \textit{aunique2} and \textit{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 \textit{matches2}. The \textit{recordsource} variable was added to the data set to indicate that the match was a result of stage 2.

\begin{tabular}{|c|c|c|c|c|c|c|c|c|}
\hline
\textit{matches2} & \\
\hline
aschool & astudentid &aname & agrade & adob & bschool & bstudentid & bname & bgrade & bdob & recordsource \\
\hline
ABC School & 15 & Jim & 2 & 19550101 & ABC School & 17 & Jim & 2 & 19550101 & 2 \\
\hline
\end{tabular}

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

\begin{tabular}{|c|c|c|c|}
\hline
\textit{aremainder2} & \\
\hline
school & astudentid & aname & agrade & adob \\
\hline
ABC School & 20 & Jane & 2 & 19550102 \\
\hline
\end{tabular}

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

\begin{tabular}{|c|c|c|c|}
\hline
\textit{bremainder2} & \\
\hline
bschool & bstudentid & bname & bgrade & bdob \\
\hline
ABC School & 20 & Jane & 2 & 19550102 \\
\hline
\end{tabular}
THE SAS CODE

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

<table>
<thead>
<tr>
<th>Stage 1 matched two master data sets, amaster1 and bmaster1, on studentid and name.</th>
<th></th>
</tr>
</thead>
</table>
| **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). | 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; |
| **Step 1.2** The bmaster1 data set was be divided into data sets bunique1 (records with unique bstudentid and bname values) and bdup1 (records with duplicate bstudentid and bname values). | 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; |
| **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. | 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; |
| **Step 1.4** The aunique1 records with no matches were placed in the aremainder1 data set. | 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; |
| **Step 1.5** The bunique1 records with no matches were placed in the bremainder1 data set. | title 'bremainder1';
|   | proc sql;
|   | create table bremainder1 as
|   | select bschool,bstudentid,bname,bgrade,bdob
|   | from bunique1 left join aunique1
|   | on bunique1.bstudentid=aunique1.astudentid and
|   | bunique1.bname=aunique1.aname
|   | where aunique1.astudentid is null and aunique1.aname is null; quit; |

<table>
<thead>
<tr>
<th>Stage 2 matched two master data sets, amaster2 and bmaster2, on school, name, and dob values.</th>
<th></th>
</tr>
</thead>
</table>
| **Step 2.1** The amaster2 data set consisted of aremainder1 and adup1 records. | title 'amaster2';
|   | proc sql;
|   | create table amaster2 as
|   | select * from aremainder1
|   | union all
|   | select * from adup1; quit;
Step 2.2 The `bmaster2` data set consisted of `bremainder1` and `bdup1` records.

```sql
title 'bmaster2';
proc sql;
create table bmaster2 as
    select * from bremainder1
union all
    select * from bdup1;
quit;
```

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).

```sql
**step 2.3;**
title 'aunique2';
proc sql;
create table aunique2 as
    select * from amaster2
    group by aschool, aname, adob
    having count(*)=1;
quit;

PROC SQL;
create table adup2 as
    select * from amaster2
    group by aschool, aname, adob
    having count(*)>1;
quit;
```

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).

```sql
step 2.4;
title 'bunique2';
proc sql;
create table bunique2 as
    select * from bmaster2
    group by bschool, bname, bdob
    having count(*)=1;
quit;

PROC SQL;
create table bdup2 as
    select * from bmaster2
    group by bschool, bname, bdob
    having count(*)>1;
quit;
```

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.

```sql
step 2.5;
title 'matches2';
proc sql;
create table matches2 as
    select *, "2" as recordsource
    from aunique2, bunique2
    where aunique2.aschool=bunique2.bschool and
    aunique2.aname=bunique2.bname and aunique2.adob=bunique2.bdob;
select * from matches2;
quit;
```

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

```sql
step 2.6;
title 'aremainder2';
proc sql;
create table aremainder2 as
    select aschool,astudentid,aname,agrade,adob
    from aunique2 left join bunique2
    on aunique2.aschool=bunique2.bschool
    where bunique2.bschool is null and bunique2.bname is null
    and bunique2.bdob is null;
quit;
```

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

```sql
step 2.7;
title 'bremainder2';
proc sql;
create table bremainder2 as
    select bschool,bstudentid,bname,bgrade,bdob
    from bunique2 left join aunique2
    on aunique2.aschool=bunique2.bschool
    where aunique2.aschool=bunique2.bschool
    and aunique2.aname=bunique2.bname and aunique2.adob=bunique2.bdob
    where aunique2.aschool is null and aunique2.bname is null
    and aunique2.bdob is null;
quit;
```
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;
```
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


TRADEMARK NOTICE

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