Paper 98-28

Danger: MERGE Ahead!
Warning: BY Variable with Multiple Lengths!

Bob Virgile
Robert Virgile Associates, Inc.

Overview

Normally, when a `data` step merges two data sets, any common variables will have the same length in both sources of data. When a variable has different lengths in the incoming data sets, and when that variable is also a `by` variable, the merge can produce truly bizarre results. For example, change the order of the data sets in the `merge` statement and the `data` step generates a different number of observations. Or `merge` two data sets that have a unique sorted order and the resulting data set still contains multiple observations for some `by` values. Or `merge` two data sets that are in sorted order and the `data` step issues an error message claiming that the data sets are not in sorted order. The results are so unusual, and difficult to decipher, that current releases of the SAS® software issue a warning message:

```
WARNING:  Multiple lengths were
specified for the BY variable [variable
name] by input data sets.  This may
cause unexpected results.
```

This paper examines the situation, some of its manifestations, and solutions to the problem.

The Nature of the Problem

Consider a simple `merge`, such as:

```
data combined;
merge males females;
by lastname;
run;
```

Assume that:

- `Lastname` is sufficient to identify the observations, yet
- `Lastname` has a length of 5 in `males`, but a length of 10 in `females`.

During the compilation phase of the `data` step, the software assigns `lastname` a length of 5. The reason is that the first time the `data` step encounters `lastname` is when it compiles the word `males`. At that point, it uses the length of `lastname` in `males` to define the length of `lastname` in `combined`. Subsequent references to `lastname` (such as compiling the word `females`) cannot alter the length of `lastname`. At a minimum, the results are incorrect because values of `lastname` from `females` get truncated to a length of 5. However, the ways in which this problem manifests itself are far trickier than a truncation problem. The next section examines some of the possible outcomes.

Some Bizarre Manifestations

Truncation problems can cause bizarre results. Compare these `data` steps, for example:

```
data combined1;
merge females males;
by lastname;
run;

data combined2;
merge males females;
by lastname;
run;
```

The new data sets would normally contain the same number of observations. If there were common variables besides `lastname`, the values of those common variables might change. But the number of observations would remain the same ... unless there is a truncation problem. Consider the case when `males` defines `lastname` as five characters long:

```
James
Smith
```

At the same time, females defines lastname as ten characters long:

```
Jameson
Smithfield
```

In that case, combined1 contains the expected four observations. However, combined2 defines lastname as five characters long. As a result, the merge takes place on the basis of only five characters and the data set contains only two observations.

Next, consider two data sets with a unique sorted order:

```
proc sort data=males nodupkey;
by lastname;
run;

proc sort data=females nodupkey;
by lastname;
run;
```

How many observations would you expect this combination to contain?

```
data combined;
merge males females;
by lastname;
if first.lastname=0
or last.lastname=0;
run;
```

Logically there can't be any observations in the combined data set ... unless there is a truncation problem. Consider this version of males which, as before, defines lastname as five characters long:

```
James
Smith
```

Once again, females defines lastname as ten characters long:

```
James
Jameson
Smith
Smithfield
```

When the final data step assigns lastname a length of 5, duplicates suddenly appear where none existed before.

Both examples up to this point generated impossible results without any error or warning, except for the typical warning:

```
WARNING:  Multiple lengths were
specified for the BY variable lastname
by input data sets.  This may cause
unexpected results.
```

In this last example, however, an impossible error appears. This time, the program sorts by two variables:

```
proc sort data=males;
by lastname firstname;
run;

proc sort data=females;
by lastname firstname;
run;



data combined;
merge males females;
by lastname firstname;
run;
```

Perhaps the last thing you would expect is an error message, complaining that the data are not in order ... unless there is a truncation problem. Consider this incoming version of males:

```
Evans  Robert
Smith  Francis
Smith  Leslie
```

The matching females data set contains:

```
Jameson     Francis
Smith       Paula
Smithfield  Ashley
Smithfield  Leslie
```

Both data sets are in order. Yet if a truncation problem occurs, the incoming values from females appear to be out of order:

```
James       Amy
Smith       Paula
Smith       Ashley
Smith       Leslie
```

This truncated version of the data illustrates why the program generates an error message, complaining that the data are out of order.

The Actual Warning

The warning message builds in some intelligent features. First, it displays only when warranted. For example, only one of these `data` steps can generate a warning:

```
data combined1;
merge males females;
by lastname;
run;

data combined2;
merge females males;
by lastname;
run;
```

If `lastname` has the same length in both data sets, no warning appears. But if it has different lengths, only one `data` step truncates the values of `lastname`. The warning appears for that `data` step only.

A second clever feature is that the warning message automatically generates more information when another statement is causing the problem. Consider this program:

```
data combined3;
length lastname $ 8;
merge males females;
by lastname;
run;
```

The `length` statement assigns `lastname` a length of 8. If the incoming data sets both use a length of 8 or less, no warning appears. But if one of the data sets were to use a length longer than 8, the warning message would become:

```
WARNING:  Multiple lengths were
specified for the BY variable lastname
by input data sets and LENGTH, FORMAT,
INFORMAT, or ATTRIB statements.  This
may cause unexpected results.
```

Notice that the warning mentions the input data sets, but does not mention the `merge` statement. Technically, the `merge` statement does not trigger these warning messages. Rather, the `by` statement triggers them. The following program would never generate a warning, regardless of the length of `lastname` in the two data sets:

```
data combined;
merge males females;
run;
```

On the other hand, if the earlier `data` step for `combined3` produces a warning, then the same warning would be issued if the `merge` statement were changed to either a `set` statement or an `update` statement.

An earlier assignment statement can also truncate a `by` variable:

```
data combined;
lastname='Smith';
merge females males;
by lastname;
run;
```

If the assignment statement truncates `lastname` (because the length in `females` or `males` is longer than five characters), the shorter warning message will appear. In practice, it is rare to encounter an assignment statement before a `merge` statement, especially when the assignment statement changes a `by` variable's value. It would take a much more complex `data` step to realistically illustrate such a `data` step structure. Other unusual structures follow the same general rules:

```
data combined;
if _n_=1 then set males;
set females;
by lastname;
run;
```

If there is a truncation problem, the warning appears. However, if the `by` statement were removed, the warning would never appear.

Simple Solutions

Plenty of solutions exist, to work around the problem. As usual, one of the best pieces of advice is to read the SAS log, and pay attention to any warning messages. Above and beyond that, there are two simple ways to deal with this situation.

First, change the order of the data sets in the `merge` statement, so that the first data set has the longest length for `lastname`:

```
merge females males;
```

In rare cases, this method may not be possible. It is conceivable that many `by` variables have multiple lengths. In that case, changing the order of the data sets might correct the length for one `by` variable while truncating another. In addition, there could be other

common variables which don't appear in the by statement. Changing the order of the data sets in the merge statement would affect the values of those other variables.

Fortunately, another simple solution exists. Add a length statement before the merge statement:

```
length lastname $ 10;
merge males females;
```

Notice that the order of the data sets in the merge statement no longer matters, since lastname has already been defined by the prior length statement. Also, this approach works well when there are many by variables with multiple lengths. However, all of the simple solutions require knowledge of the lengths of the by variables. What if those lengths are not known? Certainly it is easy to run a proc contents on each data set and note the lengths of by variables. But how could this process be automated? That question leads to some more complex solutions.

More Complex Solutions

All complex solutions share a common theme: have the program examine the structure of each incoming data set, determine the maximum length for each character by variable, and use macro language to issue a length statement. In the examples using the males and females data sets, the macro invocation might look like this:

```
%combine (outdata=combined3,
          indata=males females,
          byvars=lastname)
```

That macro invocation should generate the beginning of a data step:

```
data combined3;
length lastname $ 10;
merge males females;
by lastname;
```

The generated code omits the run statement, since the programmer would usually intend to add statements before concluding the data step. Much of the generated code involves text substitution, which is an easy task for macro language. But generating the length statement requires delving into the structure of the incoming data sets.

The SAS software supports a variety of tools that examine the structure of SAS data sets:

- proc contents out=
- SCL functions
- dictionary.columns
- sashelp.vcolumn

This paper will work with just one of the available tools, sashelp.vcolumn. It is an automatic SAS data set, readable only, containing information about all variables in other SAS data sets. In the program below, it supplies the maximum length of lastname from either data set:

```
proc means data=sashelp.vcolumn max;
var length;
where libname='WORK' and
memname in ("MALES", "FEMALES")
and upcase(name)="LASTNAME";
run;
```

This proc means would generate a simple report, such as the following:

```
      Maximum
------------
  10.0000000
------------
```

On the plus side, this program demonstrates that the software provides the needed information in readily retrievable form. While helpful, however, this program falls short of the mark in a few ways. First, it prints the maximum length instead of generating a length statement in a data step. Second, it hard codes the data set names and variable name. Third, the example works with just one variable instead of multiple by variables. In addition, more information would be needed if some of the by variables could be numeric instead of character.

A different approach would overcome one of the obstacles, copying the maximum length into a macro variable:

```
proc sql noprint;
select max(length) into :maxlen
  from sashelp.vcolumn
 where libname='WORK'
   and memname in
       ("MALES", "FEMALES")
   and upcase(name)="LASTNAME";
quit;
```

Once the program has captured the maximum length of lastname into a macro variable, generating the proper

data step code is relatively simple:

```
data &outdata;
length &byvars $ &maxlen;
merge &indata;
by &byvars;
```

The more difficult task is generalizing the code so that it works for multiple data sets, with multiple by variables that includes a mix of numeric and character variables. The remainder of this paper will illustrate techniques that overcome most of these problems. Any remaining issues, as well as the assembly of the techniques into a working program, is left to the reader.

Issue #1: Handling multiple data sets

The assumptions about the incoming data sets include: (a) all data sets are stored in the WORK library, (b) the user will pass the names of at least two data sets using the indata= parameter, and (c) macro language must use these data set names to create a portion of the SQL query. For example, the macro call might specify:

```
indata=males females androids
```

The matching portion of the SQL query would be:

```
memname in
("MALES", "FEMALES", "ANDROIDS")
```

Within a macro definition, the related code could be:

```
memname in (
%local i nextdsn;
%let i=0;
%let indata=%upcase(&indata);
%do %until (%length(&nextdsn)=0);
   %let i = %eval(&i + 1);
   %let nextdsn = %scan(&indata, &i);
   %if %length(&nextdsn) > 0 %then %do;
       %if &i > 1 %then ,;
    "&nextdsn"
   %end;
%end;
)
```

Issue #2: Multiple by Variables

Each by variable needs its maximum length assigned to a separate maro variable by an SQL select statement. Here is a macro %do loop that can loop through all the names in &byvars:

```
%local j;
let j=0;
%do %until (%scan(&byvars, &j+1)=);
  %let j = %eval(&j + 1);
  %* SQL select statement here;
%end;
```

Inside the loop, the program must generate a matching SQL select statement for one variable within &byvars. The interior code might look like this:

```
select max(length) into : maxlen&j
  from sashelp.vcolumn
 where libname='WORK'
   and memname in
  ("MALES", "FEMALES", "ANDROIDS")
   and upcase(name)=
  "%upcase(%scan(&byvars,&j))";
```

In practice, the code related to Issue #1 would be embedded within this code.

Issue #3: Are They Character?

Clearly, some by variables may be numeric. In that case, the easiest approach would be to assign them a length as well, but to omit the $ in the length statement. One way to extract the necessary information would be a set of similar SQL select statements:

```
select type into : vartype&j
  from sashelp.vcolumn
 where libname='WORK'
   and memname = "MALES"
   and upcase(name)=
  "%upcase(%scan(&byvars,&j))"
   ;
```

These new macro variables will take on values of either char or num, thereby supplying the information needed to add dollar signs to a length statement. It is sufficient to reference one data set (in this case, MALES), because by variables must have the same type in all the incoming data sets.

Clearly, there are many gaps. It would take more work to assemble all of these pieces into a working macro. However, the intended scope of this paper is to cover the issues related to merge. The macro code illustrates that an automated approach should be possible. However, the macro is left unfinished, perhaps to form the basis for another paper.

Comments, questions, and suggestions are always welcome:

Bob Virgile
Robert Virgile Associates, Inc.
23 Independence Drive
Woburn, MA 01801
(781) 938-0307
virgile@rcn.com