Interleaving a Dataset with Itself: How and Why

Howard Schreier, U.S. Dept. of Commerce, Washington DC

ABSTRACT

When two or more SAS datasets are combined by means of a SET statement and an accompanying BY statement, they are said to be interleaved. It is also possible tto interleave a dataset with itself, and that technique is useful in solving a fairly common type of problem.

INTRODUCTION

Here is a simple and compact SAS dataset, named DEMO, which will be used in all of the examples in this paper:

Obs	id	value
1	a	2
2	a	1
3	a	3
4	b	3
5	b	5

Here is another dataset, named MORE, with the same variables:

Obs	id	value
1	a	0
2	b	0

We will begin by looking at the simplest form of the SET statement. It names a single SAS dataset, with no options. The observations in that dataset are brought into the program data vector one at a time. So, if you run this code

```
data _null_;
set demo;
put _all_;
run;
```

you should see

```
id=a value=2 _N_=1
id=a value=1 _N_=2
id=a value=3 _N_=3
id=b value=3 _N_=4
id=b value=5 _N_=5
NOTE: There were 5 observations read from the data set WORK.DEMO.
```

It is also possible to name multiple datasets on the SET statement. In that case, the two series of observations will be concatenated. That is, they will be sequenced end to end. Coding the IN= option for at least one of the datasets helps to track which observations came from which dataset. So, if you run this code

```
data _null_;
set demo more(in=from_more);
put _all_;
run;
```

you should see

```
from_more=0 id=a value=2 _N_=1
from_more=0 id=a value=1 _N_=2
from_more=0 id=a value=3 _N_=3
from_more=0 id=b value=3 _N_=4
from_more=0 id=b value=5 _N_=5
from_more=1 id=a value=0 _N_=6
from_more=1 id=b value=0 _N_=7

NOTE: There were 5 observations read from the data set WORK.DEMO.

NOTE: There were 2 observations read from the data set WORK.MORE.
```

If a BY statement is applied, the datasets will be interleaved. That is, BY groups will be assembled from the multiple datasets. Within each BY group, observations will be sequenced in the order in which datasets are named in the SET statement. Thus, this code

```
data _null_;
set demo more(in=from_more);
by id;
put _all_;
run;
```

should generate

```
from_more=0 id=a value=2 FIRST.id=1 LAST.id=0 _N_=1
from_more=0 id=a value=1 FIRST.id=0 LAST.id=0 _N_=2
from_more=0 id=a value=3 FIRST.id=0 LAST.id=0 _N_=3
from_more=1 id=a value=0 FIRST.id=0 LAST.id=1 _N_=4
from_more=0 id=b value=3 FIRST.id=1 LAST.id=0 _N_=5
from_more=0 id=b value=5 FIRST.id=0 LAST.id=0 _N_=6
from_more=1 id=b value=0 FIRST.id=0 LAST.id=1 _N_=7

NOTE: There were 5 observations read from the data set WORK.DEMO.

NOTE: There were 2 observations read from the data set WORK.MORE.
```

Those are the basic techniques. Now we can move on to the topic of this paper.

HOW

The datasets named in the SET statement do not have to be different. One dataset can be named multiple times. When this is done, SAS behaves as if there were multiple copies of the dataset in question and DATA step processing proceeds accordingly. You can concatenate; for example, this code

```
data _null_;
set demo(in=firstpass) demo;
put _all_;
run;
```

yields

```
firstpass=1 id=a value=2 _N_=1
firstpass=1 id=a value=1 _N_=2
firstpass=1 id=a value=3 _N_=3
firstpass=1 id=b value=3 _N_=4
firstpass=1 id=b value=5 _N_=5
firstpass=0 id=a value=2 _N_=6
firstpass=0 id=a value=1 _N_=7
firstpass=0 id=a value=3 _N_=8
firstpass=0 id=b value=3 _N_=9
firstpass=0 id=b value=5 _N_=10

NOTE: There were 5 observations read from the data set WORK.DEMO.

NOTE: There were 5 observations read from the data set WORK.DEMO.
```

Or, you can interleave; so, this code

```
data _null_;
set demo(in=firstpass) demo;
by id;
put _all_;
run;
```

produces

```
firstpass=1 id=a value=2 FIRST.id=1 LAST.id=0 _N_=1
firstpass=1 id=a value=1 FIRST.id=0 LAST.id=0 _N_=2
firstpass=1 id=a value=3 FIRST.id=0 LAST.id=0 _N_=3
firstpass=0 id=a value=2 FIRST.id=0 LAST.id=0 _N_=4
firstpass=0 id=a value=1 FIRST.id=0 LAST.id=0 _N_=5
firstpass=0 id=a value=3 FIRST.id=0 LAST.id=1 _N_=6
firstpass=1 id=b value=3 FIRST.id=1 LAST.id=0 _N_=7
firstpass=1 id=b value=5 FIRST.id=0 LAST.id=0 _N_=8
firstpass=0 id=b value=3 FIRST.id=0 LAST.id=0 _N_=9
firstpass=0 id=b value=5 FIRST.id=0 LAST.id=1 _N_=10

NOTE: There were 5 observations read from the data set WORK.DEMO.

NOTE: There were 5 observations read from the data set WORK.DEMO.
```

In a sense, concatenating is a special case of interleaving. It is like interleaving with no BY variables, or with BY variables whose values form but a single BY group. So for the rest of this paper we will look at interleaving, keeping in mind that the conclusions generally apply even if there are not multiple BY groups.

WHY

At first glance, interleaving a dataset with itself seems like a pointless stunt. But in fact there are classes of problems for which the technique is applicable and possibly convenient.

PROCESSING BASED ON GROUP SUMMARIZATION OR ANALYSIS

It's not unusual to encounter a programming task in which the output is to preserve the level of detail provided in the input, yet incorporate in some way summary information, or depend on some evaluation spanning groups of observations. For example, suppose that we want to process DEMO and create a new variable (SCALE) which transforms VALUE to a zero-one scale for each level of ID; that is, within each

BY group, the smallest value will correspond to 0 (zero), the largest to 1 (one), and others to proportionate intermediate fractions.

This task is not difficult. We could run PROC SUMMARY to get the MIN and MAX statistics, merge these back with DEMO, and calculate SCALE. Or, we could do it all with a single SQL statement. But suppose we would like to accomplish the task in a single DATA step. That presents a difficulty, because ordinarily group summary statistics are not known until the last observation in the group is processed, and at that point the earlier observations have already been output.

However, the self-interleaving technique can solve the problem by bringing the detail through a second time. Here is the code:

```
data zero_one_scale;
set demo(in=firstpass)
    demo;
by id;
retain min max;
drop   min max;
if first.id then do;
   min = .;
   max = .;
   end;
if firstpass then do;
   min = min(min,value);
   max = max(max,value);
   end;
else do;
   scale = (value-min)/(max-min);
   output;
   end;
run;
```

Notice that variables MIN and MAX are initialized to missing at the beginning of each BY group. During the first pass through the data, theses two variables are updated in memory using the corresponding (MIN and MAX) functions. Nothing else happens; in particular, no observations are added to the new dataset. In the second pass (under control of the ELSE DO block), the transformation calculation is done and the explicit OUTPUT statement causes the new dataset to be built.

Here are the results:

Obs	id	value	scale
1	a	2	0.5
2	a	1	0.0
3	a	3	1.0
4	b	3	0.0
5	b	5	1.0

The program structure in this example is typical of that used in many applications of this technique. The first pass is sort of a preview, during which the data comprising the BY group are summarized or checked for some property or properties. The second pass performs processing at the detail level and triggers an OUTPUT statement.

It's also typical that there are alternatives for this class of problem.

- The traditional SAS approach uses multiple steps. Usually the first step creates the summary information, which is then connected back to the detail by means of a MERGE.
- The most concise solution will usually be provided by SQL.
- When efficiency is a concern, one is driven toward DATA step techniques which accumulate all of the data from a BY group in memory, for processing when the end of the group is encountered. This can become an intricate exercise, though SAS Version 9 promises to make it easier through the use of associative arrays.

ORDERING WITHOUT SORTING

Here's a different application of self-interleaving. Suppose that we want to re-order the observations from the dataset DEMO; in particular suppose that within each BY group we want observations in which the variable VALUE has odd values to appear first, followed by observations in which VALUE is even.

We could do it by constructing a sort key and then running PROC SORT. Here's a single-step alternative based on self-interleaving:

```
data odd_then_even;
set demo(where=(mod(value,2)=1))
    demo(where=(mod(value,2)=0));
by id;
run;
```

Notice that the expressions in the two WHERE= filters are mutually exclusive. Thus, even though DEMO is referenced twice in the SET statement and will in fact be interleaved with itself, no single observation will be passed into the output dataset more than once. The result:

Obs	id	value
1	a	1
2	a	3
3	a	2
4	b	3
5	b	5

ADDITIONAL TOPICS

DETECTING THE BOUNDARY

Sometimes it is necessary or convenient to perform some processing upon encountering the boundary between groups of observations. If the groups in question are BY groups, then the FIRST.[byvariable] And LAST.[byvariable] constructs are available. For interleave boundaries within a BY group, the information provided by the IN= dataset option can be used. However, this information *per se* does not indicate boundaries. So it is necessary to do something a bit more elaborate. Here is an illustration:

```
data _null_;
set demo(in=p1) demo(in=p2);
by id;
boundary = p2 and lag(p1);
put _all_;
run;
```

When SAS is processing any observation from the second pass through DEMO for a particular BY group, P2 has a value of TRUE (1). When and only when it is processing the first such observation, the **lagged** value of P1 also has a value of TRUE (1). These two conditions in combination are more or less the definition of the boundary. Here are the results:

```
p1=1 p2=0 id=a value=2 FIRST.id=1 LAST.id=0 boundary=0 _N_=1
p1=1 p2=0 id=a value=1 FIRST.id=0 LAST.id=0 boundary=0 _N_=2
p1=1 p2=0 id=a value=3 FIRST.id=0 LAST.id=0 boundary=0 _N_=3
p1=0 p2=1 id=a value=2 FIRST.id=0 LAST.id=0 boundary=1 _N_=4
p1=0 p2=1 id=a value=1 FIRST.id=0 LAST.id=0 boundary=0 _N_=5
p1=0 p2=1 id=a value=3 FIRST.id=0 LAST.id=1 boundary=0 _N_=6
p1=1 p2=0 id=b value=3 FIRST.id=1 LAST.id=0 boundary=0 _N_=7
p1=1 p2=0 id=b value=5 FIRST.id=0 LAST.id=0 boundary=0 _N_=8
p1=0 p2=1 id=b value=3 FIRST.id=0 LAST.id=0 boundary=1 _N_=9
p1=0 p2=1 id=b value=5 FIRST.id=0 LAST.id=1 boundary=0 _N_=10

NOTE: There were 5 observations read from the data set WORK.DEMO.

NOTE: There were 5 observations read from the data set WORK.DEMO
```

Notice in particular that for each level of ID, BOUNDARY has a value of 1 when the DATA step is processing the first observation of the second pass. This can be useful to trigger processing which transforms summary statistics gathered during the first pass, for use during the second pass.

Because there is no symmetric "look ahead" counterpart to the LAG function, I know of no similar construct to detect the observation immediately preceding the interleave boundary.

WHERE PROCESSING

An earlier example demonstrated the use of self-interleaving with the WHERE= dataset option. A caution: if it is possible that this technique will filter out all observations from one pass or another, program logic should allow for that situation.

REFERENCES

SAS Institute Inc., *SAS Language Reference: Concepts*, Version 8

CONTACT INFORMATION

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

Howard Schreier
U.S. Department of Commerce
Stop H-2815
Washington DC 20230

(202) 482-4180

Howard.Schreier@mail.doc.gov