Paper BB63

A Quick View of SAS® Views

Elizabeth Axelrod, Abt Associates Inc.

ABSTRACT

Looking for a handy technique to have in your toolkit? Consider SAS® Views, especially if you work with large data sets. After a brief introduction to Views, I'll show you several cool ways to use them that will streamline your code and save workspace.

INTRODUCTION

SAS Views are a handy technique to have in your toolkit, especially if you work with large data sets. While there are three kinds of SAS Views (DATA Step Views, PROC SQL Views, and SAS/ACCESS Views) this paper will focus on only of them: DATA Step Views.

WHAT IS A DATA STEP VIEW?

A SAS DATA Step View is the set of instructions needed to create a data set; it is not the data set itself. A View is useful because it gives you access to the data without creating an actual file. It's sort of like a template for a data set (or a virtual data set) – it holds everything you need to know about a data set – except the data. Here are a few important points to keep in mind about Views:

- During program execution the View is compiled, but input records are processed only when the View is referenced by another DATA or PROC Step.
- When referenced, the View passes data, one record at a time, to the 'calling' DATA or PROC step. In a manner similar to a MACRO, if you don't reference the View, nothing happens – no records are processed.
- Every time the View is referenced, the records will be processed. *(Caution!! More on this later)*
- You may create a maximum of ONE View per DATA step.

WHY USE A VIEW?

DATA step Views are really cool, but I suspect they're underutilized. The literature about them can be a bit technical (sometimes daunting). But I hope to illustrate how easy they are to code and use. Views can provide an elegant, simple solution to a lot of what I call 'oh darnit' situations. I'm going to give you some examples of what I mean by this, but the essential benefit is that you can use a View to preprocess your data without creating a temporary, intermediate data set that might clutter-up (or exceed) your SAS WORK area. The following examples illustrate simple uses of a DATA Step View.

Example 1

You have two data sets that you want to merge, so you can append Region-level data from FileA onto FileB. *Oh Darnit!* You realize that FileB doesn't have REGION. But it does have STATE, and you can create REGION from STATE.

FileA		
REGION	**VAR1**	**VAR2**
1	XX	123
2	YY	234
3	ZZ	345
4	AA	456

FileB	
STATE	**VAR3**
MA	15
NH	15
VT	30
SC	35

Table 1. Sample data for Example 1

Without using a View, you would have to create an intermediate data set that contains the newly-created REGION:

```
data PREP;
    set FileB;
    region = put(state,$region.);
run;
data out.FileC;
    merge PREP
          FileA;
    by REGION;
run;
```

But you realize that you don't need to keep the intermediate data set PREP. This is a perfect situation for a View! Here's all you have to do to modify the code:

```
data PREP / view = PREP;
    set FileB;
    region = put(state,$region.);
run;
data out.FileC;
    merge PREP
           FileA;
    By REGION;
run;
```

That's how easy it is to replace a temporary, intermediate SAS data set with a SAS View!

The code "`/ view = PREP`" informs SAS that you want to create a View called PREP, not a data set called PREP. When this code is executed, the first data step is compiled, but no data are processed… until the second data step is run, and the View *PREP* is referenced. At this point, records from FileB are processed, one record at a time, and passed to the 'calling' data step. *No intermediate file is created.*

Since the sample data sets are so teeny, this may seem like a trivial gain, but imagine you're processing millions of records. Now we start to see the benefit of creating a View, rather than an intermediate file that you have no use for after the code is run.

At this point I'd like to mention a convention I use. I name my Views using a consistent prefix (or suffix) so I can readily distinguish them from data sets. In my subsequent examples I will use *V_* as a prefix to all my View names. This is strictly a user convention, and not required by SAS.

Example 2

We want to merge two data sets, keep records that are in both files, then sort the result and output a permanent data set. The following code (which does not take advantage of Views) achieves this:

```
data foo;
    merge one (in=in1)
          two (in=in2);
    by keyvar;
    if in1 & in2;
run;

proc sort data=foo
    out=whatever;
    by othervar;
run;
```

But we realize that we don't need the file *foo* for any further processing, and it's so large that it will quickly consume our workspace. This is a perfect candidate for using a View. All we have to do is to inform SAS that we're making a View, and not a data set. Here's what the code it looks like:

```
data v_foo / view=v_foo;
    merge one (in=in1)
          two (in=in2);
    by keyvar;
    if in1 & in2;
run;

proc sort data=v_foo
    out=whatever;
    by othervar;
run;
```

Here is the log that results from running this code.

```
50   data v_foo / view=v_foo;
51       merge one (in=in1)
52             two (in=in2)
53             ;
54       by keyvar;
55       if in1 & in2;
56   run;

NOTE: DATA STEP view saved on file WORK.V_FOO.
NOTE: A stored DATA STEP view cannot run under a different operating
system.
NOTE: DATA statement used (Total process time):
      real time           0.04 seconds
      cpu time            0.00 seconds

57   proc sort data=v_foo
58       out=whatever;
59       by othervar;
60   run;
NOTE: There were 10 observations read from the data set WORK.V_FOO.
NOTE: View WORK.V_FOO.VIEW used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds

NOTE: There were 19 observations read from the data set WORK.ONE.
NOTE: There were 10 observations read from the data set WORK.TWO.
NOTE: The data set WORK.WHATEVER has 10 observations and 5
variables.
NOTE: PROCEDURE SORT used (Total process time):
      real time           0.04 seconds
```

Notice that when the first data step is run, the view is defined, but no records are processed.

It's only when the next step (in this case, a PROC Sort) references the View, that the data step above is executed and records are processed and made available, one-by-one, to the calling PROC.

Example 3 (finally… the cool stuff!)

In Example 2 we enhanced our code by using a View, but we can make it even better. In the example above, we're only keeping records that are in both files (one and two). What about the mismatched records? Ideally, we might want to create a separate data set that captures these mismatches. But a data step can only create one View. So how can we capture the mismatches? We can create a View … AND one or more data sets in a single data step!

We start with our example above, and add some code to create a data set (not a View) of mismatches. We'll also we add a step (in this case a PROC FREQ) to take a look at the mismatches.

```
data mismatch                    /* ← SAS data set */
     v_foo / view=v_foo;         /* ← SAS View */
    merge  one (in=in1)
           two (in=in2);
    by keyvar;
    if in1 & in2 then
           output v_foo;
    else /* not in both */
           output mismatch;
run;
proc sort data=v_foo
    out=whatever;
    by othervar;
run;
proc freq data=mismatch;
    tables /* some revealing vars */ ;
run;
```

Sequence of processing steps is important!

If you are creating a data set and a View in the same data step, it is critical that in your subsequent processing, you reference the View before you reference the data set – otherwise, you'll get an error. Referencing the View in a subsequent step is what causes the code to execute and process the data. Take a look at the following log to see how this works:

```
445  data mismatch               /* ←  SAS data set */
446         v_foo / view=v_foo;   /* ←  SAS View     */
447         ;
448         merge one (in=in1)
449               two (in=in2)
450               ;
451         by name;
452         if in1 & in2 then
453               output v_foo;
454         else /* not in both */
455               output mismatch;
456  run;

NOTE: DATA STEP view saved on file WORK.V_FOO.
NOTE: A stored DATA STEP view cannot run under a different
operating system.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

Even though we're creating a data set (along with a View), notice that when this code is run, only the View is defined.

The data set *mismatch* is not created until the View is referenced in the next step…

```
458  proc sort data=v_foo
459      out= whatever;
460      by weight;
461  run;

NOTE: There were 10 observations read from the data set WORK.V_FOO.
NOTE: View WORK.V_FOO.VIEW used (Total process time):
      real time           0.01 seconds
      cpu time            0.03 seconds

NOTE: There were 19 observations read from the data set WORK.ONE.
NOTE: There were 10 observations read from the data set WORK.TWO.
NOTE: The data set WORK.MISMATCH has 9 observations and 5 variables.
NOTE: The data set WORK.WHATEVER has 10 observations and 5 variables.
NOTE: PROCEDURE SORT used (Total process time):
      real time           0.01 seconds
      cpu time            0.03 seconds
462
463  proc freq data=mismatch;
464      tables var1 var2 /list missing;
465  run;

NOTE: There were 9 observations read from the data set WORK.MISMATCH.
NOTE: PROCEDURE FREQ used (Total process time):
      real time           0.00 seconds
```

Now, when the PROC Sort references the View (and these records are processed), our data set *mismatch* is also created, and becomes available for subsequent processing.

Ta-da! The data set *mismatch* is available for us to use.

The log below shows what would happen if we try to use the data set mismatch before we reference the View.

```
530  data mismatch
531         v_foo / view=v_foo;
532         ;
533         merge one (in=in1)
534               two (in=in2)
535               ;
536         by name;
537         if in1 & in2 then
538               output v_foo;
539         else /* not in both */
540               output mismatch;
541  run;

NOTE: DATA STEP view saved on file WORK.V_FOO.
NOTE: A stored DATA STEP view cannot run under a different operating
system.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
542
543  proc freq data=mismatch;
ERROR: File WORK.MISMATCH.DATA does not exist.
544      tables var1 var2 /list missing;
ERROR: No data set open to look up variables.
545  run;

NOTE: The SAS System stopped processing this step because of errors.
```

Once again, we define the View. This code is compiled, but won't execute until the View is referenced.

Ouch! All we did was change the order of the steps. Since we haven't referenced the View, the first data step hasn't executed, and the data set *mismatch* was not created.

Example 4 (more cool stuff!)

As long as you're careful and follow the basic guidelines, you can even nest your Views!

```
data v_one / view=v_one;
   set one;
   /* do something here */
run;

data v_two / view=v_two;
   set v_one;
   /* do something else here */
run;

proc sort data=v_two
   out=three;
   by keyvars;
run;
```

WATCH OUT!

Keep in mind that *every* time a View is referenced, the data step that defines the View is *executed,* and records will be processed, one record at a time, and made available to the calling step. Consider this code:

```
data v_prep / view=v_prep;    /* Define the View */
   set one;
   region = put(state,$region.);
run;
proc means data=v_prep; /* execute the data step - process the records */
run;
data freq data=v_prep; /* execute the data step AGAIN - process the records */
   tables var1 var2;
run;
proc sort data=v_prep  /* execute the data step YET AGAIN - process the records */
   out=two;
   by var1 var2;
run;
```

If you're going to reference a View multiple times in your program stream, you might want to consider a different approach. Do you *really* want to *execute* that first data step more than once? It might be more efficient to actually create a data set, which you can reference more than once.

SUMMARY / GUIDELINES

Here are tips to keep in mind as you work with Views:

- You may create a maximum of ONE View per DATA Step. But…
- You can also create one or more data sets when you create a View.
- You may NOT create a View from a PROC step.
- You may create nested Views.
- EVERY time you reference a View, your data will be processed again – so be careful!

CONCLUSION

SAS data step Views are extremely useful when working with large data sets, streamlining your code and saving workspace. They're also incredibly easy to code. Once you start using them, you'll wonder how you managed without them!

RECOMMENDED READING

- *Stokes, James C., "SAS® Data Views Simply Stated", SUGI 29 Proceedings, http://www2.sas.com/proceedings/sugi29/067-29.pdf*
- *Boling, John C., "SAS Data Views: A Virtual View of Data", SUGI 22 Proceedings http://www2.sas.com/proceedings/sugi22/ADVTUTOR/PAPER36.PDF*
- *SAS Language Reference: Concepts*

CONTACT INFORMATION

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

Elizabeth Axelrod
Abt Associates Inc.
55 Wheeler Street
Cambridge, MA 02138
elizabeth_axelrod@abtassoc.com