Paper 026-2010

SAS Views – The Best of Both Worlds

As seasoned SAS programmers, we have written and reviewed many SAS programs in our careers. What I have noticed is that more often than not, people who are developing these programs do so in a hurry to meet a deadline. Because of these tight timelines, very little attention is paid to the efficiency of these programs. We spend a great deal of time bending our brains around the logic needed to solve a problem. Once the problem is solved, we go onto the next problem. Enhancing efficiency is like documentation, most of us just don't seem to like it or get around to it.

Unfortunately, if we don't pay attention to the efficiency of our programs, they become a nuisance to ourselves or our coworkers who have to run them – not to mention those who share the cpu they run on. This is especially true if you are passing millions of lines of data through these inefficient programs.

Fortunately, there is a solution that you can employ that will yield significant time savings and you won't need to completely rethink your logic. It comes in the form of SAS views. Employed properly, SAS views can allow you to move onto the next logic puzzle that keeps us all in this profession while reducing the resources necessary to continue running the tried and true gem you just created.

Focus of this paper

SAS views can help a programmer in many ways:

- They assure processing of current data
- They allow source code to be hidden from users
- They allow processing of proprietary file structures
- They minimize data replication

This paper will focus on the minimization of data replication.

Background of SAS views:

SAS views have been around since version 6 was introduced. However, they are rarely at the forefront of our thinking when we sit down to write code. The idea behind a SAS view is that a multi-step process can be coded against a large amount of data while only reading/writing the data to disk once. Most multi-step processes read/write the data once for each step. Since reading and writing to disk (I/O) is typically the most time-consuming part of any process, it really pays to minimize this activity. After demonstrating the syntax to create a SAS view, I will illustrate the potential time savings of employing a SAS view using three approaches to solving the same problem. My charge is to summarize claim data for a health insurance company. Granted, this is a very simple illustration from a logic perspective. However, it requires that I work with a great deal of data. I will explain each approach with text, a picture and the log of the resulting code execution.

Syntax for creating a PROC SQL View:

PROC SQL;
Create view *view_name* as
  1 as X,
  1 as Y;
Quit;

Syntax for creating a DATA Step View:

DATA *view_name / view=view_name;*
   X=1;
   Y=1;
Run;

When creating a data step view, the name and the view name must be the same. Also if you use a data step to create a view, it is possible to view the code used to create a SAS view:

DATA view=*view_name*;
  describe;
run;

The log will show this:

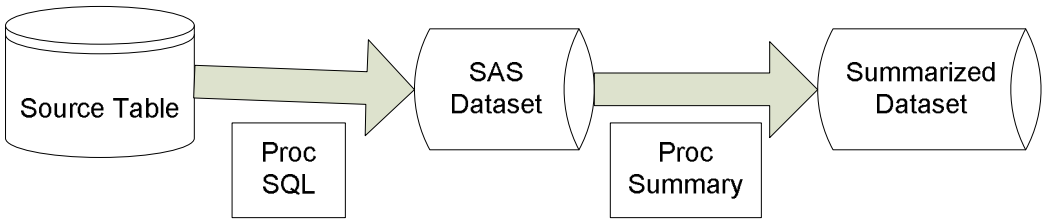
```
NOTE: DATA step view view_name is defined as:

DATA view_name / view = view_name;
   x=1;
   y=2;
run;
```

I am not aware of a way to view the code used to create a SQL view.

Approach 1: Pull data with SQL then summarize:

I created a SAS dataset with net pay, cover amount and process month from all claims that were processed between July 2009 and December 2009. I then applied a PROC SUMMARY to that dataset to summarize net pay and cover amount fields. This activity took approximately 46.5 minutes (adding the SQL and Summary steps together) to process 46,620,217 claim lines.

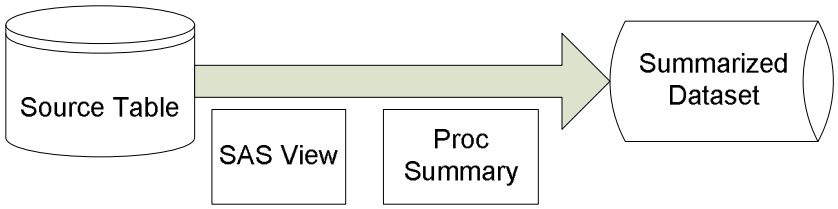
SAS Code/Log:




```
/*  Pull data with SQL then summarize */
        PROC SQL;
        create table test as
        select netpayln, coverln, prcsmnth
        from phidp.clm
        where prcsmnth between 200907 and 200912;
      quit;
NOTE: PROCEDURE SQL used (Total process time):
    real time          45:54.17
    cpu time           2:10.66

      PROC SUMMARY DATA=test nway missing;
        class prcsmnth;
        var netpayln coverln;
        output out=summ  (drop=_type_ _freq_) sum=;
      run;

NOTE: PROCEDURE SUMMARY used (Total process time):
    real time          33.37 seconds
    cpu time           30.33 seconds

NOTE: There were 46620217 observations read from the data set TEST.
NOTE: The data set WORK.SUMM has 6 observations and 3 variables.
```

Approach 2: Summarize using a SAS VIEW:

I applied a PROC SUMMARY to a SAS view that was created in the same way as the dataset in the step above. To process the same number of rows and achieve the same PROC SUMMARY results took 17.5 minutes. That is a 62% time savings.



SAS Code/Log:

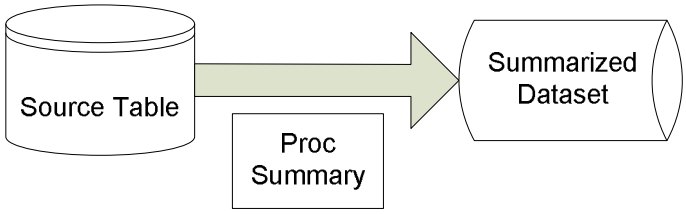
```
/*  Summarize directly from Oracle table   */
      PROC SQL;
        create view test_view as
        select netpayln, coverln, prcsmnth
        from phidp.clm
        where prcsmnth between 200907 and 200912;

NOTE: SQL view WORK.TEST_VIEW has been defined.
28      quit;
NOTE: PROCEDURE SQL used (Total process time):
    real time          0.01 seconds
    cpu time           0.00 seconds


      PROC SUMMARY DATA=test_view nway missing;
        class prcsmnth;
        var netpayln coverln;
        output out=summ (drop=_type_ _freq_) sum=;
      run;

NOTE: There were 46620217 observations read from the data set PHIDP.CLM_VIEW.
    WHERE (PRCSMNTH>=200907 and PRCSMNTH<=200912);
NOTE: There were 46620217 observations read from the data set WORK.TEST_VIEW.
NOTE: The data set WORK.SUMM has 6 observations and 3 variables.
NOTE: PROCEDURE SUMMARY used (Total process time):
    real time          17:30.45
    cpu time           2:32.69
```

CAVEAT: A few weeks after running this test, I discovered that at the time of the test, our SAN storage system was experiencing some issues that were impeding I/O. When I repeated this test after the SAN was fixed, I could not reproduce the same time savings. I believe this actually reinforces the results, however. The advantage SAS views offer increases as the number of desired rows increases or the system becomes busier or more distressed.

Approach 3: Summarize directly from Oracle table:

After seeing this incredible time savings, I wondered how long it would take to perform a PROC SUMMARY directly against The Oracle Table. My thought is that this method would require only a single read and write of the data similar to how a view would process the data. The time it took for this was just over 31 minutes. It was faster than using a SAS dataset (Approach 1) but not as fast as the view (Approach 2). The only explanation I could come up with was perhaps the SAS to SQL conversion (Oracle can only process SQL so SAS must

convert the PROC SUMMARY to SQL) isn't as efficient as coding native SQL. This is only a theory on my part, however.



SAS Code/Log:

```
libname claims oracle path="claim_path" schema="schema_name" user=claim_user
password=password;

/*   Sum directly from Oracle table    */
       PROC SUMMARY DATA=claims.clm nway missing;
         class prcsmnth;
         var netpayln coverln;
         where prcsmnth between 200907 and 200912;
         output out=summ (drop=_type_ _freq_) sum=;
       run;

NOTE: There were 46620217 observations read from the data set PHIDP.CLM_VIEW.
     WHERE (prcsmnth>=200907 and prcsmnth<=200912);
NOTE: The data set WORK.SUMM has 6 observations and 3 variables.
NOTE: PROCEDURE SUMMARY used (Total process time):
     real time          31:11.65
     cpu time           2:01.18
```

Cautions:

If you create a view with a name that has previously been given to a view, the new view will overwrite the old one. However, if you try to create a view with the same name as a SAS dataset that already exists in the same library, you will get an error. Similarly, you cannot sort a view to itself because that creates a table with the same name as the view. You must use an out= clause when applying PROC SORT to a view.

Summary:

SAS views offer you the ability to significantly reduce the runtime of some of your tried and true SAS programs without forcing you to significantly alter those programs. Don't neglect the fellow users of your SAS servers or the poor soul who has to run your program down the road. Views allow you to write tighter compiled code while allowing you to maintain your preferred source coding style. Essentially they give you the best of both worlds.

Author Contact Information:

David Thul
Blue Cross Blue Shield of Minnesota
david_a_thul@bluecrossmn.com

Reference:

SAS Data Views: A Virtual View of Data
John C. Boling, SAS Institute Inc., Cary, NC