Paper 19-27

Save Time Today Using SAS® Views

Robert Ray, SAS Institute Inc., Cary NC

ABSTRACT

In this time of exploding data volumes, your old friend the SAS® data view can be more important than ever as a tool to reduce total job I/O and, thereby, total time. Views can be created in SAS with either SQL or the DATA Step and used singly or in combination. SAS data views can be used as one might use Unix 'pipes' to chain multiple data manipulation stages together without intervening I/O. Judicious use of this established technology can cut execution real-time significantly and bring your critical jobs back into their time windows. Additionally, this paper will briefly discuss the new "pipes" feature of the multi-process (MP) SAS/CONNECT® product.

.

LOTS MORE DATA, NO MORE TIME:

With the explosion of information flowing from new sources such as the Internet, it's no surprise that IT departments are feeling the time-to-solution squeeze for their large business-critical applications. Those batch time windows seem to be getting smaller every year. If your are feeling the pinch, it may be time to take a careful look at your SAS applications to see if you can use some old tools so shave significant time from your large-data jobs. SAS data views, both DATA Step and SQL, can be used to circumvent unnecessary reading and writing of temporary SAS data files by allowing the user to add functionality to SAS procedures.

How much time can be saved using views? That answer is obviously "that all depends". Views do not reduce the CPU cycles required to complete the task. If anything, they increase the total CPU time a bit. What they eliminate is writing and reading back temporary data sets. The purpose of using views is to reduce the *real time* required to complete a job by eliminating one or more I/O bound segments. If a forty-minute DATA Step that takes only ten minutes of CPU time can be converted into a DATA Step view, the potential real time savings for the entire job could be as much as thirty minutes. So when searching for good view candidates, look for DATA Steps and PROC SQL steps that create intermediate temporary data files and have real execution times that greatly exceed the associated CPU times.

In addition to time savings, SAS data views can reduce the peak disk space requirements for a given job by reducing the redundant copies of data required to be held on disk at any given instant. This could mean the difference between a job succeeding or failing when disk resources are tight.

VIEW BASICS:

Views can be created using PROC SQL or the DATA Step. SQL views were available first, but both view flavors have been available since SAS version 6.07. Both view types store the intermediate results of partial compilation in special file types. When referenced by a client procedure, the view metadata is retrieved, compilation is completed, data sources are opened and output buffers are produced and fed to the client, just as if a SAS data file was being read from disk. Most data clients are blissfully unaware that their data is being created on the fly one buffer at a time.

Both view types divide the compilation process across the definition and execution phases, but to a different degree. DATA Step views are more complete when they are stored even to the point of opening all referenced data sources to validate variable references (more on this later). This means that a DATA Step view is less likely to fail at execution time due to syntactic errors but also is a bit less flexible.

SQL views, on the other hand, go through only a basic syntactic check before the query tree representation is stored in the view file. This allows the SQL view to complete its optimization at execution time when table metadata is available. One of the optimizations that an SQL view can perform is *view collapsing*. If multiple SQL views are nested without intervening DATA Step views, the top level SQL view will pull nested views into its context to allow better global optimization of the execution plan. Views execute in a separate SAS "task" from the client. When views are nested, each nested view exists in its own task. Crossing these task boundaries involves some processing overhead. By collapsing multiple views into one, SQL eliminates task boundaries and thereby reduces overhead.

DATA Step Example:

Here's a simple example that answers the question of "how do I sort multiple (similar) data sets at once?" The original code that fueled this question looked something like this:

```
data ALL;
  set A B C…;
run;

proc sort data=ALL; by <keys>; run;
```

This looks harmless until you consider the I/O flow.

Paper 19-27

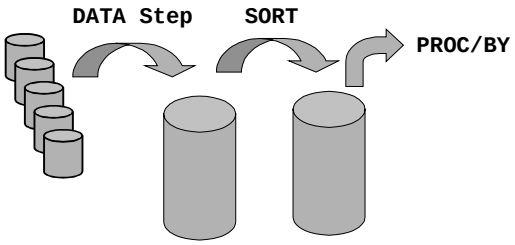

Figure 1

Now, with the addition of "/view=ALL" in the code below, we can avoid one complete write/read cycle.

```
data ALL / view=ALL;
  set A B C…;
run;

proc sort data=ALL; by <keys>; run;
```

Our new flow diagram looks like this…

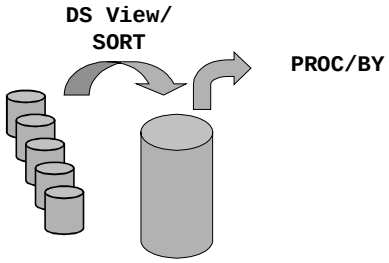

Figure 2

To the Next Level with SQL:

For this example, the SQL view can be used to great advantage due to it's ORDER BY syntax. Using ORDER BY allows us to not only concatenate multiple data sets together but also to sort the data and "pipe" the results of that sort directly into the client procedure. The sort is performed with either SAS SORT or a host sort depending on the setting of the global option SORTPGM. This eliminates writing an intermediate sorted data file to disk just so that BY–processing can be performed.

```
proc sql;
  create view concat(var1, var2,… varN) as
    select var1, var2,… varN
      from
        (select * from A
           union all
         select * from B
           union all
         select * from C)
    order by <keys>; quit;
```

The data flow when this view is applied can be seen in Figure 3 below.

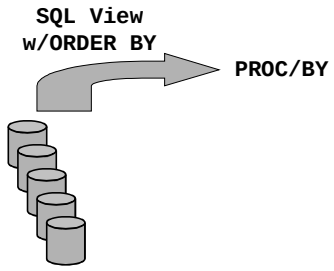

Figure 3

To go one step further, if the client procedure in Figure 3 is a simple PROC MEANS/BY, it is possible that the summarization could also be performed in the SQL view, eliminating the PROC MEANS altogether. The example below is modified slightly to include summary statistics by adding a GROUP BY statement.

```
proc sql;
  create view concat(var1, var2,… varN) as
    select mean(var1), mean(var2),…
      from
        (select * from A
           union all
         select * from B
           union all
         select * from C)
    group by <keys>
    order by <keys>; quit;
```

The data flow for this view eliminates the procedure altogether and can pipe data directly into the subsequent step as show in Figure 4.

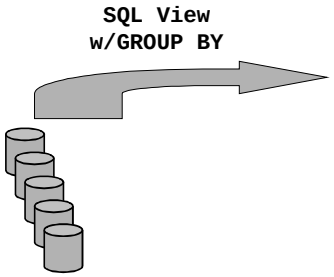

Figure 4

Keep in mind that any sort, whether called directly or by SQL using ORDER BY/GROUP BY, may require utility files to be opened to complete the sort. Having the sort as part of a view does not affect whether or not a sorting utility file is required. The data volume and the value of SORTSIZE govern this decision.

Paper 19-27

COMBINING VIEWS:

From the previous example, it might seem as if SQL views could be a replacement for DATA Step views. One important advantage that DATA Step views maintain is their ability to read data from flat files, which SQL views cannot do. If in the previous example, one or more of the original data sources had been flat input files, the best solution would have been to combine both DATA Step and SQL views together. The combined data flow could look something like Figure 5 below.

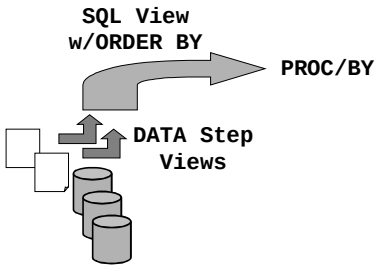

Figure 5

WHEN THE MAGIC DOESN'T WORK:

The motivation for using views as a performance-enhancing tool is the potential to avoid unnecessary writes and reads of temporary data files. Given this, using views where possible should result in decreased I/O and real time, right? Unfortunately, there is a daemon-in-the-works known as the *spill file*. When a view client opens a view in a non-sequential mode, both DATA Step and SQL views will open a utility file to spool the results of the first pass. Subsequent passes will be read back from this spill file. From an I/O space and time perspective, this becomes the equivalent of the intermediate data file that the view was supposed to avoid in the first place. The good news is that most of the BASE procedures, PROC SUMMARY, MEANS, TABULATE, REPORT, etc. open their data sources in single-pass sequential mode. The bad news is that many of the statistical procedures open their data sources in multi-pass mode. Even some of the BASE procedures can cause spill files such as when PROC PRINT is used with the UNIFORM option.

Prior to Version 7, DATA Step views opened for non-sequential or multi-pass access would create a complete spill file before returning the first record to the client. This could result in long delays before the first record was received. Starting with SAS Version 7, DATA Step view spill files are written while records are being supplied to the client thus allowing concurrent processing of the view. For SAS Version 9, refinements to view spill file logic are being investigated that will give the users the option of avoiding spill files at the cost of re-computing the view information for each pass. That is, for multi-pass applications, the view engine would rewind the *original* data source (if possible) and re-compute the observations. Although this approach can add to the total CPU time required, it does prevent duplicating the input data in the SASWORK directory.

OPEN=DEFER AND %SETALL():

Another significant Version 7 improvement for DATA Step views was in introduction of the OPEN=DEFER option for the SET statements. Prior to Version 7, a DATA Step would open *all* data sources referenced by the SET statement at once to read-in variable metadata. This can be problematic with large numbers of data sets or tape based data sets that need to be concatenated with a view. The OPEN=DEFER option instructs the DATA Step to open only the first data set referenced and use it's variable layout as the pattern for the concatenated set. This allows the set statement to operate with only one data set open at a time but does cause any variables not encountered in the first data set to be dropped from the concatenated set.

One way to get around the OPEN=DEFER limitation is to use the %SETALL() macro, published as a SAS Technical Tip. It demonstrates the use of a DATA Step view and an SQL view, showing the advantages of each. %SETALL() handles OPEN=DEFER by creating an empty dataset which contains all of the variables in all of the datasets to be concatenated. This data set is referenced under OPEN=DEFER first, so that it creates all of the variables that are needed, but with no values. Then the original datasets are concatenated, and each one's variables finds a home.

%SETALL() begins with a DATA Step view to read the variable information from each dataset. It opens them in succession using the OPEN() function, and obtains the names and types of each variable as the view's output. It also has the flexibility to support two different ways to specify the data set list: using either an explicit list or a "prefix start count suffix" to implicitly list many datasets.

%SETALL() feeds the variable information into an SQL view. The SQL view eliminates duplicate names and sets the length of the character variables to the longest length encountered for that particular name. A real DATA Step then opens the SQL view, and converts the results into macro variables that can be expanded to create the empty data set containing all of the variables. Using these views, all of the data is generated and consumed in this step. The final DATA Step is then generated, with the help of macro variables, that uses OPEN=DEFER to concatenate all of the data sets without data loss.

Note that a simple DATA Step without OPEN=DEFER can be used to concatenate a few data sets in the same manner, but it will not work when there are a large

number of datasets to concatenate. %SETALL() was developed specifically to concatenate ODS output datasets, of which there can be thousands.

SAS/CONNECT (MP) PIPES in V9:

As was stated earlier, SAS data views are similar to Unix "pipes" which connect the output of one process directly to the input of another. The significant difference is that SAS views operate in the same thread as their client, thus providing no opportunity for parallelism of tasks on SMP computers. For SAS Version 8, the SAS/CONNECT Multiprocess (MP) feature was introduced. It provides convenient syntax to spawn multiple SAS sessions to work on a single task in concert. These sessions can reside on the same SMP machine or on multiple machines across a network. Currently, MP CONNECT does not provide a true piping facility but instead relies on complete data sets for process communication.

For Version 9 of the SAS system, MP CONNECT can be used in conjunction with the new pipe engine that will allow separate SAS sessions to transfer their results to the next stage of the process one buffer at a time. This is similar to the SAS data view concept except that it uses TCP/IP connections via a special libname syntax. This technology takes the view concept to the next level. With MP CONNECT and the pipe engine, it should be possible not only to avoid data duplication on disk but also to distribute the CPU overhead of view execution across multiple CPUs.

CONCLUSION:

Long available SAS data view technology can be a powerful tool in the battle against the clock. If you process vast volumes of data, using SQL and DATA Step views may cut significant percentages off the real time for your large SAS jobs. To best leverage existing view technology, both SQL and DATA Step views should be considered, and used in combination when needed as illustrated in the %SETALL() macro. In the future, combining (MP) CONNECT technology with existing SAS views should provide new levels of scalability for existing SAS programs.

REFERENCES:

Doninger, C. (2002), "Up and Out: Where We're Going with Scalability in SAS® Version 9," *Proceedings of the Twenty-seventh Annual SAS Users Group International Conference.*

Heffner, W. F. (1998), "DATA Step in Version 7: What's New?," *Proceedings of the Twenty-third Annual SAS Users Group International Conference.*

Boling, J. C. (1997), "SAS Data Views: A Virtual View of Data," *Proceedings of the Twenty-second Annual SAS Users Group International Conference.*

First, S. (1997), "Faster SAS Jobs and Fewer Passes via DATA Step Views," *Proceedings of the Twenty-second Annual SAS Users Group International Conference.*

Polzin, J. A. (1993), "Introduction to DATA Step Views," *Proceedings of the Eighteenth Annual SAS Users Group International Conference.*

CREDITS:

The SETALL macro was jointly developed by Gordon Keener, Warren Kuhfeld and Jason Secosky. For details see http://www.sas.com/service/techtips/ts_qa/setall.html.

CONTACT INFORMATION:

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

Robert Ray
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
Robert.Ray@sas.com