Faster SAS® Jobs and Fewer Passes Via DATA Step Views

Steven First
Systems Seminar Consultants
Madison, WI 53716 (608) 222-7081

Abstract

DATA step views have been around for a few years, but very few jobs take advantage of them. There are many situations where DATA step views require less passes, less intermediate work space, less CPU, and less clock time than traditional DATA steps do. The improvement in certain cases can be significant, and in fact, using views may be the only way to get some jobs to run at all. This paper will examine the pros and cons of using DATA step views and benchmark several different scenarios taken from actual production applications

Introduction

In release 6.07 DATA step views were introduced. "Introducton to DATA Step views", POLZIN SUGI 18 along with Tech report P-222 describe this feature. The intention here, as it is with all views is to make data appear *logically* as a SASdataset, even though *physically* the data may be in flat files, SAS data files, DBMS systems, or other sources. Though similar to other views, DATA step views have a much broader scope and can utilize all of the power of the SAS data step including calculations.

Some features of DATA step views are:

1. Simplification of code. Very complex programs can be stored once and when referenced observations are returned as with any typical SAS dataset. Later DATA or PROC steps that request an observation need not be concerned with physical structures, but instead they only need to deal with SAS variables and observations.

2. Multiple passes may be eliminated saving CPU, wall time and intermediate work space. These resources vary of course, depending on the access, operating system, and SAS software release. There are some jobs that require more work space than is available and will just not run in the traditional way. Using views at least would allow such jobs to run.

3. WHERE processing in some cases can be pushed earlier in the job stream resulting in less data moved and thus less resources.

4. Once created, DATA step views can be used to migrate data to other SAS datasets or to external DBMS systems, or to other supported file structures. All of the physical structures can be transparent to the consumer.

5. DATA step views can be combined with other data sources, again appearing as a single SAS dataset. Examples would be virtually any merging, concatenation, interleaving, or other joining of datasets that a SAS job might do.

Types of DATA Step Views

1. Input DATA Step Views

DATA step views that function as input SAS datasets are called *input DATA step views*. This is the most commonly used of the two types of DATA step views and is the easiest to understand. This view can be virtually any DATA step that eventually outputs an observation. When a consumer of an observation (a PROC or DATA step) requests an observation, the code in the view is executed and an observation is returned. Note that the data is generally only passed once in contrast to the many passes that traditional SAS datafiles require.

2. Output DATA Step Views

DATA step views that function as output SAS datasets are called *output DATA step views*. This type of DATA step view is a little harder to conceptualize. When some DATA or PROC step outputs an observation to a view, that view is then executed for the observation. An example might be a SORT step that passes observations to a DATA step for further processing. Examples and explanations below will hopefully help the understanding of output DATA step views.

Creating DATA Step Views

Input DATA Step View Syntax

DATA *SASds1 SASds2 . . .* / **VIEW** =*data-view- name*;
DATA step statements . . .
RUN;

Where:

SASds1 SASds2 etc. are valid one or two level SAS dataset names.

data-view-name is a valid one or two level SAS dataset name and must match one of the SASdatasets specified before the slash.

Examples

1. A temporary INPUT DATA step view reading a raw file.

```
data a / view=a;
 infile dd1;
 input . . .;
run;
proc print data=a;
run;
```

2. A permanent INPUT DATA step view reading a raw file.

```
data perm.a / view=perm.a;
 infile dd1;
 input . . .;
run;
proc print data=perm.a;
run;
```

3. A permanent INPUT DATA step view reading a raw file that also creates a SAS data file.

```
data b
     perm.a / view=perm.a;
 infile dd1;
 input . . .;
run;
proc print data=perm.a;
proc print data=b;run;
```

OUTPUT DATA Step Views

Output DATA Step View Syntax

DATA . . . / **VIEW** =*data-view name*;
data step statements . . .
<SET> | <MERGE> |
<UPDATE> | <MODIFY> *input-sasdata-set-names ;*
more data step statements . . .
RUN;

Where:

data-view-name is a valid one or two level SAS dataset name and must match one and only one of the SASdatasets in the SET, MERGE, UPDATE, or MODIFY statement. Other datasets can appear on the reading statements. Since this view is replacing the output side of a DATA or PROC step, it is important that the lengths and types of variables used in the step match those from the input dataset. If there is any question, coding LENGTH or ATTRIBUTE statements may be necessary.

Examples

1. A temporary OUTPUT DATA step view that takes the output from PROC SORT and produces a report.

```
data _null_/view=file3;
  length field1 $ 10 field2 $10
         acct $20 balance 8 . . .;
  set file3;
  file print;
  put etc. . . .;
run;

proc sort data=file1 out=file3;
  by balance;
run;
```

Applications for DATA Step Views

Applications that make sense for DATA step views can be looked at from various perspectives.

If trying to make thinks simple, very complex processing, external file access etc. can be hidden from the final user.

For transparency and portability, even though the underlying physical data changes, views can be updated correspondingly without final users even knowing or caring in most cases.

An example of achieving both simplicity and transparency is the following:

A large local company has defined DATA step views for hundreds of user raw files. To most casual SAS users at that company, there is no such thing as a raw file. To those users all data that they use is thought of as a SAS dataset, and somehow all dates are SAS dates, and other local logic is done before their programs ever see the data. This approach has greatly simplified the SAS user's processing and has worked out very well.

From an efficiency standpoint, the best applications for DATA step views will be those applications that pass the data once sequentially. Applications that may *not* lend themselves to DATA Step views might be:

1. Programs that read the dataset into several different steps. Especially if the input to the view is a flat file, because of data conversion, using views can cost more than building a SAS data file once, and then reading it as necessary.

2. PROCs or other DATA steps that require multiple passes of the data such as PROC PRINT with the UNIFORM option. In this case underlying datasets may be created anyway, and as such the performance gains are lost. This is done transparently to the user, but the application would certainly be less efficient than if there were no spill dataset.

3. DATA steps that access data randomly may also require the extra spill datasets as above, and thus should probably avoid views.

4. It would appear that indexing and views together

would not work well if at all, though I have not investigated this.

Differences Between DATA Step Views And Stored Programs

DATA step views and SAS programs created using the Stored Program Facility, while similar, differ in the following ways:

1. An Input DATA step view, when referenced, provides data one observation to the requesting DATA or PROC step

2. A Stored Program, when executed, creates SAS data files.

Processing With DATA Step Views

When a DATA step view is submitted, the DATA step view engine processes the DATA step through the compilation phase, and stores an intermediate code version of the program in a SAS library as member type VIEW. This can be stored in either a temporary or permanent SAS library.

For Input DATA step views:

The intermediate code is further resolved during the compilation of a later DATA or PROC step that references the view. The DATA step view will be *executed* when that later DATA or PROC step executes and requests an observation. The DATA step view will execute until an observation is output from the view, at which time the consuming step gets control again. Obviously the actual hand-offs and task switching are much more complex than that described especially if more than one file is being processed, but in effect, the consumer asks for an observation, and the DATA step view runs until one is provided.

For OUTPUT DATA step views:

The compilation process is similar to that mentioned above with the execution phase working somewhat as follows:

This time some DATA or PROC step that outputs *to* a DATA step view will execute first. When an observation is written out, the DATA step view is loaded and executed for that one observation, does its processing, and then gives control back to the first step. Another way of stating this, is that the output DATA step view is replacing the output side of some PROC or DATA step.

Restrictions and Requirements

1. Only one view can be created in a single DATA step. Other SAS data files can be created along with the view however.

2. A DATA step view and a SAS data file with the same name cannot exist in the same SAS library.

3. The DATA step view can contain any SAS DATA step statement except:
   a. SAS Global statements (ignored if coded).
   b. Engine-specific or host-specific options or parameters.

4. Like other compiled members in SAS software, source is not saved. Make sure source, passwords etc. are saved and secure.

5. DATA step views will not transfer to another SAS software release or to a different platform. In this case the VIEW will need to be recreated from the source.

6. Prior to Release 6.08 TS 407, use of the STIMER option causes significant CPU overhead and higher costs when using DATA step views.
   In release 6.07 this CPU penalty was approximately a 10% increase, but buffering and other performance enhancements in the 6.08 TS 407 has reduced this penalty significantly, and in some cases CPU overhead will be less with views.

7. Like other views, updating is generally not allowed. For example FSEDIT will be opened in browse mode if used against a view, and again a spill file may be generated.

8. Some options such as the SET NOBS= option may be unavailable or undependable since there is not dataset header value with the observation count.

9. In some releases Output DATA step views are considered experimental and should not be used for production or important applications.

Performance Considerations

It is always difficult to come up with general efficiency statements regarding SAS software. What works well in one release, may not work well in another release, and each platform is unique. In addition there are certainly efficiency issues that only the Institute's developers would be able to answer and as such is beyond the scope of this paper..

In general it would be reasonable to assume for sequential one-pass applications that:

1. Elapsed time, I/O and EXCPS should decrease.
2. CPU time should be lower or at least not increase significantly.
3. Much less intermediate storage should be required, in some cases allowing jobs that would not otherwise run to run.

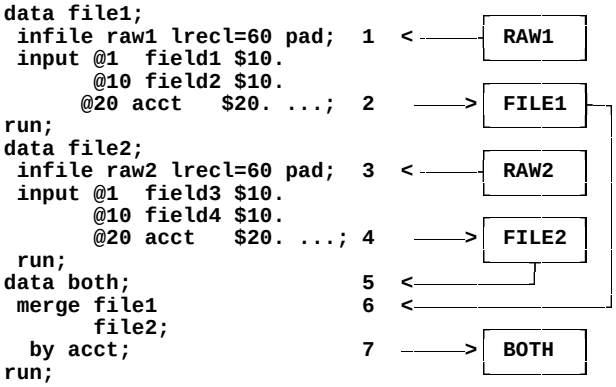
Examples and Benchmarks

Some basic benchmarks were run under:
Windows 3.1
SAS Release 6.11
Pentium 200
Each raw file contains 68081 lines of data.

The benchmarks were not extensive and should not be taken as absolute recommendations. Again platforms and releases as well as actual dataset sizes and characteristics could affect results significantly.
Bench marking here is something that is well worth doing for large runs, and we plan on doing much more in the future.

Example 1: Two large flat files that are already sorted by account number need to be merged.

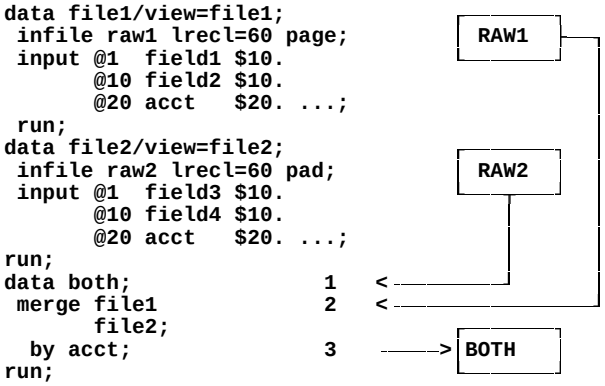
Solution 1: The traditional method is to convert each flat file to a SASdataset, then in a third step merge them. The data from file one is passed three times, that from file two is passed three times, and finally the final dataset is created.



Elapsed Times (seconds) from the SAS log:

Step 1	14.60
Step 2	14.44
Step 3	16.37
Total	45.41

Solution 2: By using DATA step views, the flat files are not passed until the consumer (the merge step) requests an observation, and thus the data from the two input files are only passed one time. Depending on file sizes, some significant time savings as well as significant intermediate space savings should be realized.



Elapsed Times (seconds) from the SAS log:

Step 1	00.48
Step 2	00.48
Step 3	30.87
Total	31.83

In this very simple example the elapsed time decreased by approximately 30%.

Running a similar test under MVS also reduced elapsed times with slight savings in CPU costs as well. This was a very small run however and may not be representative. Where the most gains would appear may be with very large datasets. Clearly much more MVS bench marking is worth it and is necessary before drawing conclusions for that platform.

Note that the remainder of the examples are run twice; once using SAS data file(s) as input then rerun with DATA step view(s) as input. The programs are shown only once with a table of the two program's resources.

Example 2: Using the same two input DATA steps and altering the third step to concatenate the two datasets.

```
data both;
 set file1
     file2;
run;
```

Total Elapsed Times (seconds) from the SAS log:

data file	data step view

48.09	31.83

Example 3: Alter the third step to interleave the data.

```
data both;
 set file1
     file2;
 by acct;
run;

;
```

Total Elapsed Times (seconds) from the SAS log:

data file	data step view
49.14	27.15

Example 4: Concatenate two files with a WHERE statement.

```
data both;
  set file1 file2;
  where balance lt 32000;
run;
```

Total Elapsed Times (seconds) from the SAS log:

data file	data step view
44.40	25.51

Example 5: PROC PRINT some of the observations.

```
proc print data=file1(obs=10000);
run;
```

Total Elapsed Times (seconds) from the SAS log:

data file	data step view
19.64	7.37

Example 6: Repeat the above using the UNIFORM option. Note that this requires passing the data twice and as such elapsed time increases when using the data step view.

```
proc print data=file1(obs=10000) uniform;
run;
```

Total Elapsed Times (seconds) from the SAS log:

data file	data step view
20.80	30.62

Example 7: Read the dataset directly. Again note that a spill dataset may be created and using DATA step views results in longer run time.

```
data random;
  do I=1 to 30000;
   set file1 point=I;
   output;
  end;
  stop;
run;
```

Total Elapsed Times (seconds) from the SAS log:

data file	data step view
21.02	16.58

Example 9: First use SAS data files, then an OUTPUT DATA step view to sort data, then print in a data step.

First the traditional SORT and DATA step.

```
proc sort data=file1 out=file3;
  by balance;
run;

data _null_;
  set file3;
  file print;
  put etc. . . . ;
run;
```

Secondly use an OUTPUT DATA step view.

```
data _null_/view=file3;
  length field1 $ 10 field2 $10
         acct $20 balance 8 . . .;
  set file3;
  file print;
```

```
  put etc. . . . ;
run;

proc sort data=file1 out=file3;
 by balance;run;
```

Total Elapsed Times (seconds) from the SAS log:

data file	data step view
28.39	16.58

Summary and Conclusions

In summary I think it is fair to say that DATA step views present unique features that are not readily apparent.

1. For most single pass sequential applications, elapsed times consistently decrease significantly with very little effort.
2. Jobs that are impossible to run otherwise, now run.
3. For those jobs requiring random access or multiple passes, DATA step views may actually run longer.
4. Recent releases have significant performance enhancements.
5. More bench marking is needed and worthwhile.

Acknowledgements

I would like to thank the following for input and support.

Iain Robertson Systems Seminar Consultants
Jeff Polzin SAS Institute
Bill Heffner SAS Institute

References

Proceedings of the 18th Annual SUGI Proceedings
SAS Tech. Report P-222
Changes and Enhancements to BASE SAS Software

The author will be glad to answer questions and accept suggestions at the following address:

Steven First
Systems Seminar Consultants
Suite 206
1220 Femrite Drive
Madison, WI 53716

Voice: (608) 222-7081
Fax: (608) 222-7310
Internet: train@sys-seminar.com