

The View Review: PROC SQL Views, SAS/ACCESS® Views, and DATA-step Views

Andrew A. Norton, Trilogy Consulting Corporation

INTRODUCTION

Views are fundamental. They may not appear to be, because we have gone so many years without them. But we have also gone without the applications that views make possible, and the clarity that they bring to programming.

Views are also complex. Their properties are poorly understood, their features rudimentary, and their documentation limited. But this was once true of DATA steps as well.

Views not only add new capabilities to the SAS® System, they also can improve the efficiency and maintainability of traditional SAS applications.

The objectives of this paper are to help programmers use views more effectively and to encourage the future development of views by SAS Institute. Advice is offered regarding how to choose the type of view best suited to your applications.

WHAT IS A VIEW?

In version 6 of the SAS System, the term "SAS data set" incorporates both "SAS data files" (called tables in SQL terminology) and "SAS data views" (SAS Institute, 1990, page 200).

Views can be thought of as "virtual data sets." Views are easy to incorporate into the SAS system because they mimic an existing concept: SAS data files.

SAS version 6 reads SAS data sets by sending messages to interchangeable data-reading modules called "engines" (SAS Institute, 1990, pages 207-209). Library engines respond to messages by reading data from a file. View engines respond to messages by executing a program. As far as the SAS core is concerned, there is no difference: each message receives an appropriate response. As a result, SAS data files and SAS data views can be interchanged without modifications to existing code.

Views can also be thought of as "programs run on demand." Rather than producing output and storing it for future use, views materialize output as it is needed, using data from files termed "base tables."

This paper only discusses "input" views, but in the future, the SAS System will also support output DATA-step views (Polzin, 1993), which are experimental in releases 6.07 and 6.08.

WHY TO USE VIEWS

When I write a program, my first choice is to create a view. I only create a table when I have a good reason not to use a view. My reasons for preferring views include:

Transparency

Views let you modify data structures without having to modify existing code.

Views provide physical independence between data sources: SAS data files, external data files, DBMS files, and (in the future) files located on different machines.

Although a data file must be physically stored in a single SAS library, views of that file may appear in many different libraries.

Up-to-date results

Data updates completed before the view is opened will be reflected in the view results. Updates after the view has been opened may also be reflected in the output, depending upon the procedure using the view and whether that observation has already been materialized.

Efficiency of CPU

Views are processed on demand, so they are ideal for derived data sets which may become obsolete without ever being used.

Views that support the "push-down WHERE clause" feature (discussed below) only need to materialize observations of immediate interest rather than the entire data set.

For debugging purposes, you can examine a few sample observations without having to construct the entire data set.

Efficiency of input/output

Traditional SAS programs write intermediate results to disk, then read them back from disk for the next step. Views allow the intermediate result to be passed directly into the next procedure, saving two sets of I/O operations. Further savings are possible if several levels of views are nested.

Efficiency of storage

Results derived from other tables are logically redundant. Deriving these results as they are needed saves the storage space that would have been required to store them.

The only storage required for a view is for the program used to materialize the results.

Programs written with views can often run with less WORK library space, because intermediate results do not need to be stored.

Security

SAS can define different passwords for views and their underlying base tables. This makes it possible to restrict access to specified variables or observations, summarized data only, or any other appropriate subset of the data.

Multitasking

Version 6 of SAS only allows one DATA step or procedure to run at a time, but programs in the form of views can be executed at any time, when referenced by Screen Control Language or Display Manager.

Data-oriented perspective

Designing systems based upon views encourages a focus on fundamental data structures rather than application-specific processing flows. This perspective encourages generalized coding and can make systems easier to maintain. Think of view-based systems like "plumbing" — a series of pipes and filters that transport and transform the data.

WHY NOT TO USE VIEWS

Even though computer power has increased dramatically in recent years, the ease of programming with views must still be balanced against efficiency issues. Views are usually very efficient, but can be very inefficient.

Efficiency: weaknesses of the SQL optimizer

Programs using SQL-based views rely upon the SQL optimizer to choose an efficient strategy to achieve the specified goal. Unfortunately, there is no way to override the optimizer if it makes a flawed decision. Creating intermediate tables breaks the task into segments, forcing the optimizer to execute the segments in the order shown.

An example of a weakness in the SAS implementation of SQL is that WHERE clauses applied to a GROUP BY variable in a summarization are not transformed into a WHERE clause on the input data set.

The following program illustrates a worst-case scenario:

```
proc sql;
  create view PAY as
  select    ID,
           mean(PAYCHECK) as AVG
  from      PAYROLL
  group by ID;
```

```
create table RESULT as
  select  EMPLOY.*,
         ( select AVG
           from PAY
           where ID=EMPLOY.ID
         ) as AVG
  from    EMPLOY;
quit;
```

SAS SQL is unable to apply the WHERE condition directly to the data set PAYROLL. Instead, for every value of ID in data set EMPLOY, the entire view PAY must be materialized.

It is much more efficient to store PAY as a table. The summarization only needs to be done once, and an index could be defined on the variable ID.

But the most efficient solution is to keep PAY defined as a view, but to use a join rather than a subquery:

```
proc sql;
  create table RESULT as
  select  EMPLOY.*,
         PAY.AVG
  from    EMPLOY left join PAY
         on EMPLOY.ID = PAY.ID;
quit;
```

The efficiency of each method was tested on SAS release 6.08 under CMS, using a data set PAYROLL containing 1000 observations (5 for each of 200 employees) and an index on ID:

Method	CPU	I/O
View with subquery	35.9	2104
Indexed table with subquery	1.1	236
View with join	0.2	104

This demonstrates the need to learn as much as you can about the behavior of SAS views, and to test different alternatives before making a final choice.

Efficiency: repeated uses

If the same results are used several places in a program, it is often more efficient to save a temporary copy of the results rather than executing the view each time.

Data consistency

Views always produce up-to-date results, whether you want them or not.

For example, suppose you want to do a PROC REPORT and a PROC GPLOT of the same view. It is conceivable that the base table data could change between the two procedures, leading to inconsistent results.

A good solution is to store the results of the view in a temporary data set, then use this data set throughout.

SOURCES OF DATA

Different types of views are required for different types of source data. This paper evaluates the different types of views which can be used to read each of the following types of base tables:

- SAS data files
- DBMS tables
- External flat files
- Other SAS data views

VIEWS BASED UPON SAS DATA FILES

When using SAS data files, there are two alternatives: SQL views or input DATA-step views.

Choice: SQL view

SQL is a standardized database query language supported by many vendors. In this paper, the term "SQL" refers to the SAS implementation.

SQL views are created using PROC SQL, introduced with base SAS release 6.06.

Choice: DATA-step view

DATA-step views (of the "input" variety) were introduced in release 6.07 of base SAS.

DATA-step views are created using the familiar SAS DATA-step language. For example:

```
data ADDRBOOK / view=ADDRBOOK;
  set PROSPECT (keep = NAME
                    ADDRESS PHONE STATUS);
  where STATUS = 'Active';
  drop STATUS;
run;
```

Issue: execution-time optimization

A tremendous advantage of SQL views over DATA-step views is that SQL views called from a SQL context are optimized in conjunction with that context. For example, suppose we define:

```
create view ADDRBOOK as
  select NAME,
         ADDRESS,
         PHONE
  from   PROSPECT
  where  STATUS = 'Active';
```

Now reference this view in the expression:

```
select PHONE
from   ADDRBOOK
where  NAME = 'Andy Norton';
```

The SQL optimizer combines the expression and view definition together into:

```
select PHONE
from   ( select NAME,
               ADDRESS,
               PHONE
        from   PROSPECT
        where  STATUS = 'Active'
        ) as ADDRBOOK
where  NAME = 'Andy Norton';
```

then transforms this into the equivalent but more efficient expression:

```
select PHONE
from   PROSPECT
where  STATUS = 'Active' and
       NAME = 'Andy Norton';
```

This allows an index on variable NAME of data set PROSPECT to be used to immediately locate 'Andy Norton'. Without this "push-down WHERE clause" feature, every observation of view ADDRBOOK would have to be materialized. DATA-step views have to do this, making them less efficient.

There are many other transformations that the SQL optimizer performs (see Kent, 1991), with further improvements to the optimizer planned for the future.

Unfortunately, as of the current release (6.08), WHERE clauses specified in procedures or DATA steps are not pushed down into SQL views (Kent, 1992).

Issue: dynamic data vectors

Consider the program:

```
data DEMOPLUS;
  set DEMO;
  AGE = (today() - BIRTHDT) / 365.25;
run;
```

As a DATA step, this program copies all variables on the data set DEMO to the data set DEMOPLUS. Unfortunately, this is not true of DATA-step views. If the structure of a base table changes after a view has been defined, the view may fail to execute properly.

SQL views do not have this problem. The * symbol is expanded to a variable list at execution time. So the following SQL view could be used without problems, even if the structure of data set DEMO changes:

```
proc sql;
  create view DEMOPLUS as
  select *,
         (today() - BIRTHDT)
         / 365.25 as AGE
  from   DEMO;
quit;
```

Issue: input data sequence

Many DATA-step programs work sequentially using sorted data. This is a burdensome requirement, because the data might need to be sorted by a SQL view before it can be processed by a DATA-step view.

SQL views automatically sort input data as needed.

Issue: language features

The DATA-step language is familiar to most SAS programmers. The flexibility of its algorithmic language includes the use of RETAIN and ARRAY operations which are not always feasible in SQL.

The set-oriented declarative approach of SQL is already familiar to programmers who have used other implementations of SQL, and is usually more concise than DATA steps.

Issue: efficiency

Behind the scenes, SQL views use a wide variety of techniques to achieve the desired result efficiently. Many of these techniques, such as hash joins, are unlikely to be used by the typical SAS programmer. As a result, SQL programs are often more efficient than DATA-step programs.

On the other hand, the simplicity of the sequential methods used by the DATA step can achieve speeds unapproachable by SQL for some common requests such as concatenating data sets.

One factor increasing the cost of SQL programs is the cost of the optimization process. This cost becomes proportionally less important as the size of the task becomes larger.

Issue: updatable views

In the future, it will be possible to update data through simple SQL views (SAS Institute, 1989, page 43). I do not expect updates to be supported by input DATA-step views.

Recommendation

SQL views are usually the best choice for views of SAS data sets, because of the sophistication of the SQL optimizer.

The general rule of thumb is to use SQL and stay in SQL. To work around the inability of SQL views to recognize WHERE clauses specified outside of SQL, create a temporary SQL view containing the WHERE clause.

DATA-step views provide the flexibility needed for LAG operations, extensive variable transformation, and reading external files. If WHERE clauses are not specified at execution, you may get better performance from a DATA-step view than from a SQL view.

VIEWS BASED UPON DBMS TABLES

When reading data from a Data Base Management System (DBMS) table, SAS offers a choice between SAS/ACCESS views and SQL pass-through views.

Choice: SAS/ACCESS view

SAS/ACCESS is marketed in different varieties for different data base management systems. My experience has been with SAS/ACCESS for ORACLE® on VMS™, and with SAS/ACCESS for SQL/DS™ on CMS. Views for other interfaces may have different properties.

SAS/ACCESS views are created using PROC ACCESS or the ACCESS window of Display Manager. They can only reference a single DBMS table and cannot compute additional variables.

Choice: SQL pass-through view

Version 6.07 of SAS/ACCESS introduces the "SQL Pass-Through" facility, which lets programmers embed DBMS code within SAS SQL expressions. The program can specify exactly what processing should take effect on the DBMS side and make use of any special features that the DBMS offers.

Issue: optimization

SAS/ACCESS views are optimized at execution time in the same fashion as SQL views. In fact, the technology is based upon SQL.

One advantage of SAS/ACCESS views is that WHERE clauses and BY statements are pushed down into the DBMS not just from SQL, but also from procedures or DATA steps.

SQL pass-through views never push WHERE clauses or BY statements down to the DBMS. DBMS code embedded within a SQL pass-through view is executed exactly as written, then the results are brought completely into SAS for further processing.

Issue: assigning tasks to the DBMS

The most efficient way to join two SAS/ACCESS views is to have the DBMS perform the join before converting the data to SAS files. Similarly, summarizations should be performed by the DBMS to reduce the volume of data that must be converted (SAS Institute, 1993, page 58). Programmers can use SQL pass-through views to specify that these operations should be performed by the DBMS.

Unfortunately, SAS/ACCESS views require that data be brought into SAS for joins or summarizations. This may improve in a future release of the SAS System (Kent, 1993).

Issue: missing value comparisons

Most Database Management Systems treat missing values differently than the SAS System. For example, if AGE is

missing, then SAS considers the expression

```
AGE < 21
```

to be true. Other DBMSs would not consider this expression to be true.

SAS/ACCESS often sends the WHERE clause to the DBMS to be processed (for efficiency reasons). Unfortunately, no adjustment is made for the difference in handling missing values.

For compatibility with SAS, you can specify the handling of missing values explicitly:

```
AGE < 21 or AGE is null
```

```
AGE < 21 and AGE is not null
```

Each of these expressions works identically in SAS and elsewhere.

Issue: sorting with missing values

A similar problem arises when an ORDER BY clause is sent to the DBMS for processing. SAS sorts missing values to the beginning of the file, but some DBMSs (such as SQL/DS) sort missing values to the end of the file. As a result, data sorted by a DBMS may cause SAS BY-statements to generate a "BY variables are not properly sorted" error.

Because SQL pass-through views execute the embedded DBMS code exactly as written, you can use them to force the sort to be processed on the SAS side:

```
proc sql;
  create view EMPLOY as
  select *
  from connection to sqlds
    ( select *
      from LIB.EMPLOY)
  order by SALARY;
quit;
```

Issue: dynamic data headers

This code specifies that all variables in the DBMS table are to be included in the SAS/ACCESS view:

```
proc access;
  create WORK.V.access;
  table USER1.T;
  create WORK.V.view;
  select all;
run;
```

The variables referenced by this view are determined when the view is defined. Later changes to the structure of base table USER1.T will not be reflected by the view and may cause errors.

SQL pass-through variable lists in views such as:

```
create view V as
  select *
  from connection to sqlds
    ( select *
      from USER1.T);
```

are determined at execution time and will always match the DBMS base table.

Issue: updatable views

In release 6.08 of SAS, SAS/ACCESS views are the only type of views that let you update the base table. SQL pass-through views are not updatable and I do not expect this to change.

Recommendation

In general, use SAS/ACCESS views, especially if WHERE clauses may be specified when the view is executed.

If you wish to update the DBMS from SAS, then you must use a SAS/ACCESS view.

Reserve SQL pass-through views for special cases in which summarizations or joins can be performed on the DBMS side. Also consider using SQL pass-through views if you need to sort by variables containing missing values.

VIEWS BASED UPON EXTERNAL FLAT FILES

If you are going to use an external flat file frequently, it is better to convert it into a SAS data set. This lets you create indexes, compress the data set, and transport it.

Choice: DATA-step view

When using data from external flat files, you have only one choice: input DATA-step views.

Recommendation

For data files that are rarely used by SAS applications or are updated frequently, DATA-step views are a convenient way to construct SAS data sets as needed.

SAS Institute is currently investigating development of DATA-step views as the primary tool for real-time data stream processing (Polzin, Rodriguez, and Sattler, 1992).

VIEWS BASED UPON OTHER SAS DATA VIEWS

Views may also reference other views, so there may be several layers of views above the base tables. When you use a chain of views, you are limited by the properties of the weakest link in the chain.

Choices: SQL view or DATA-step view

SAS data views are interchangeable with SAS data files, so you have the same choice as before: a SQL view or a DATA-step view.

Recommendation

This is where consistent use of SQL views really pays off. The entire structure of nested views is jointly optimized at execution time. Ideally, WHERE clauses can be pushed down through multiple levels of views to reach indexes defined on the base tables.

There is no need for the SQL processor to materialize intermediate views if the desired end result can be achieved more efficiently using some other strategy. Consider the code:

```
proc sql;
  create view VP as
    select VISIT.*,
           PATIENT.SEX,
           PATIENT.MEDCODE
    from   VISIT, PATIENT
    where  VISIT.PATIENT =
           PATIENT.PATIENT;
  create view VPM as
    select VP.*,
           MEDCODE.MEDNAME
    from   VP, MEDCODE
    where  VP.MEDCODE =
           MEDCODE.MEDCODE;
quit;
```

The view VP specifies that data sets VISIT (2000 observations) and PATIENT (100 observations) should be joined. The resulting 2000 observations are then joined with MEDCODE (3 observations).

It is more efficient to first join the two small data sets PATIENT and MEDCODE, and then join the resulting 100 observations with VISIT (2000 observations). The SAS SQL optimizer is supposed to perform this restructuring, but is not working properly in the current release (6.07 and 6.08).

CONCLUSION

No one type of view is best for every purpose. For optimal results you need to consider the properties of each choice in relation to your application.

REFERENCES

Kent, Paul (1991), "Inside the SQL Procedure's Query Optimizer," in *SAS Users Group International Proceedings of the Sixteenth Annual Conference*. Cary, NC: SAS Institute, Inc.

Kent, Paul (1992), "Fun with the SQL Procedure -- An Advanced Tutorial," in *SAS Users Group International Proceedings of the Seventeenth Annual Conference*. Cary, NC: SAS Institute, Inc.

Kent, Paul (1993), "Fun with SQL -- An Advanced Tutorial,"

in *SAS Users Group International Proceedings of the Eighteenth Annual Conference*. Cary, NC: SAS Institute, Inc.

Norton, Andrew A. (1992), "The View Review: PROC SQL Views, SAS/ACCESS® Views, and DATA Step Views," in *SAS Users Group International Proceedings of the Seventeenth Annual Conference*. Cary, NC: SAS Institute, Inc.

Polzin, Jeffrey A., Robert N. Rodriguez, and James F. Sattler (1992), "The SAS System Enters the Third Dimension: Using DATA Step Views for Real-Time Quality Control," in *SAS Users Group International Proceedings of the Seventeenth Annual Conference*. Cary, NC: SAS Institute, Inc.

Polzin, Jeffrey A. (1993), "Introduction to DATA Step Views," in *SAS Users Group International Proceedings of the Eighteenth Annual Conference*. Cary, NC: SAS Institute, Inc.

SAS Institute, Inc. (1989), *SAS Guide to the SQL Procedure: Usage and Reference, Version 6, First Edition*. Cary, NC: SAS Institute, Inc.

SAS Institute, Inc. (1990), *SAS Language: Reference, Version 6, First Edition*. Cary, NC: SAS Institute, Inc.

SAS Institute, Inc. (1993), *SAS Technical Report P-242, SAS Software: Changes and Enhancements, Release 6.08*. Cary, NC: SAS Institute, Inc.

ACKNOWLEDGEMENTS

Thanks to Paul Kent for his continuing help, and to the members of the SAS-L electronic users group for always being there in time of need. Thanks to Nancy Piper for reviewing a draft of this paper.

NOTES

A substantially different paper of the same title was delivered at the 1991 MidWest SAS Users Group conference and at the 1992 SAS Users Group International conference (Norton, 1992).

SAS and SAS/ACCESS are registered trademarks of SAS Institute, Inc. ORACLE is a registered trademark of Oracle Corporation. VMS is a trademark of Digital Equipment Corporation. SQL/DS is a trademark of International Business Machines Corporation.

The author may be contacted at:

Andrew A. Norton
Trilogy Consulting Corporation
5148 Lovers Lane
Kalamazoo, MI 49002
(616) 344-4100 voice
(616) 344-6849 fax
Internet: 76350.1604@compuserve.com