

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.

The aim of this paper is to encourage more sophistication in the development of views, and in training and documentation. The issues discussed are a determining factor in whether views can become the focus of a major change in programming perspective.

Different types of views have different properties. These differences are sometimes due to intrinsic limitations, sometimes due to pursuit of different goals, and sometimes simply mistakes or unfinished work.

This paper will introduce the different types of views, explore their behavior with respect to the different types of data set components, and conclude with a summary guide to selecting the right type of view for your application.

The answers provided in this paper should be considered as approximate. I do not have access to inside information. The best sources of information (especially for some of the SAS/ACCESS interfaces) are the SUGI Proceedings (SAS Institute, 1990a and 1991a). The descriptions of the SQL and DATA step view engines are primarily based upon experimentation (using release 6.06 of SAS* on VMS) and discussions with SAS Technical Support.

One advantage of the lack of detailed official specifications is that these features can be improved with less concern for compatibility. Be aware that anything undocumented is subject to change.

Types of views

In Version 6, the term "SAS data set" incorporates both "SAS data files" (also called tables) and "SAS data views" (SAS Institute, 1990a, page 201). SAS data files are equivalent to version 5 "SAS data sets". SAS data views contain instructions for obtaining data rather than the data itself. This processing is made transparent to SAS applications through the use of engines which provide a standard interface.

Views are commonly considered to be a substitute for data files, but they can equally well be considered as a substitute for programs. All of the steps between data input and data presentation can be views.

The SAS system currently supports three types of views: SQL views, SAS/ACCESS views, and DATA step views. The primary differences are the languages used to define the views, and the data which they can read. Unfortunately, there are additional considerations, as discussed below.

A. SQL views.

SQL views are created using the base SAS procedure PROC SQL. Many different vendors have implemented some form of SQL. In this paper, the term "SQL" refers to the SAS implementation of SQL.

SAS SQL views can access one or more SAS data sets (including any type of SAS view).

Other vendors also use SQL to define views which access tables in their systems. For example, ORACLE* views access ORACLE tables and views. ORACLE views may be accessed from the SAS System using SAS/ACCESS for ORACLE.

B. SAS/ACCESS views

SAS/ACCESS views are created using PROC ACCESS or the Display Manager ACCESS window, which use a limited form of SQL. There are different SAS/ACCESS products that interface with different data base management systems. My experience has been with SAS/ACCESS for ORACLE on VMS. Views for other interfaces may have different properties.

SAS/ACCESS views can access only a single DBMS table or view.

C. Input DATA step views

Input DATA step views will be a new feature in SAS release 6.07.03 (Polzin, 1991). I have been using the Experimental Release for release 6.06 on VMS, which can be obtained through SAS Technical Support.

Views are created using the SAS DATA step language, providing access to external files as well as SAS data sets. Many features are supported which are unavailable in SQL views, but there can be serious performance problems.

DATA step views may access multiple SAS data sets.

D. Future plans

Additional types of views are under consideration for future releases, including output DATA step views, which run when the data set is created rather than when it is read (Polzin, 1991).

Properties of different view types

Views are easy to incorporate into the SAS System because they mimic an existing concept: SAS data sets. The same commands and queries that pass between the SAS core and the data set engine also define the relationship of SAS data views to the SAS System. I have therefore organized my investigations into the properties of views according to the various components of the data set concept:

- Data set header (one per data set)
- Variable headers (one per variable)
- Data values (one per variables per observation)
- Indexes

A. Data set headers

SAS data files are relatively straightforward. If you can read the data, you can use the data. This is not true of SAS data views, which include hidden references to other data sets required for execution.

A.1 Standardized libnames

Libnames referenced internally within SQL and DATA step views must not change between view definition time and view execution time. The libname of the view itself does not matter.

SQL views provide additional flexibility. Internal data set references without a libname are assumed to refer to the library in which the view is stored. To refer to temporary datasets, WORK must be specified explicitly. If you standardize libnames you do not need to use this feature.

A.2 Granting of privileges

Standard SQL uses the GRANT statement to specify privileges on tables and views. This function is not yet supported for any type of SAS view (SAS Institute, 1989b, page 192). For example:

- 1) Read-only views cannot be created. Denying write access to the view descriptor only guards against modifications to the view descriptor, not the data to which it points. DBMS data can be protected from modification through SAS/ACCESS using security features of the DBMS.
- 2) Users with access to a SAS data set through a view also have access to that data set directly.
Passwords can be hidden within SAS/ACCESS views, so users can be allowed to access DBMS tables through views that they could not access directly. The view can be used to restrict access to particular rows and columns.

B. Variable Headers

DATA steps can manipulate selected variables without disturbing other variables on the input data sets, which pass through transparently. This feature allows generalized "filters" to be programmed, which can be used without modification on a variety of input data. Maintenance is reduced because programs need not be modified every time the structure of the input data changes.

DATA steps are compiled immediately before execution, so they can adapt to any changes that have been made to the input data structure. SAS views, by contrast, are defined in advance. If a view is defined using input data set attributes that are changed prior to execution, error messages may result. SQL views are easier to maintain than other types of views because they do not make use of input data set attributes during the definition phase. Their definitions can be modified in any order.

B.1 Dynamically constructed data set data vectors

In release 6.06, SAS/ACCESS views are fully bound to the data set attributes at view definition time. If the base table is modified, then all views referring to the modified fields must be redefined. (Changes in variable order are harmless, unlike ORACLE views).

If the base table has been modified so that it no longer matches a SAS/ACCESS view, an error message should be generated, but a bug in SAS/ACCESS for ORACLE (release 6.06) yields an empty data set with no

indication of an error.

DATA step views use only the type and length of explicitly referenced variables during the definition phase. Any input variables unreferenced by the DATA step code are determined at execution time. This is the same behavior as stored DATA steps.

The data vector of SQL views is dynamically materialized at execution time. Unfortunately the syntax of SQL prevents taking full advantage of this feature. The SELECT T.* construct (referring to all variables from data set T) is of limited use because the DROP, KEEP, and RENAME data set options cannot be used.

All three types of views should support inheritance of attributes from their base tables, but there is a bug in version 6.06 SQL views. If an attribute (format, informat, or label) is specified in a view, no other attributes are inherited from the base table.

B.2 Modification of views after they have been created

The attributes of an existing SAS/ACCESS view may be modified using the View Descriptor Display window. Unlike data files, PROC DATASETS cannot be used.

The attributes of variables in SQL or DATA step views cannot be modified without re-creating the view, because the data set data vector is not constructed until execution time. The DESCRIBE statement in PROC SQL lets you recover the code that created a particular SQL view. If you want to re-create a DATA step view, you need to have saved the original code separately.

B.3 DATA set options

Data set options may be used when reading any type of view, except when reading a SQL view within SQL. The inability to use DROP, KEEP, and RENAME in the definition of SQL views hinders the construction of "filters".

C. Data values

The data in SAS data files can be directly read and manipulated. The paradox of views is that they do not actually contain any data at all. Views are programs masquerading as data. They show their true nature when you work with only a section of data. Although each piece of a data file can be accessed individually, it is often difficult for the system to run only part of a program defining a view.

C.1 Dynamic Base Tables

Many potential problems can occur when the base tables of views are dynamically updated while the view is open. This issue arises primarily with interactive programs that keep a view open for a long time.

C.1.a. Dynamic Base Tables: Materialization of Data

Try an experiment. Create an SAS/ACCESS view. Then use Display Manager to open an FSBROWSE window and an FSEDIT window. Use the FSEDIT window to update the view and watch what happens in the FSBROWSE window.

If you want to use SAS/ACCESS in the future, don't let your manager watch with you. Some updates will be reflected in the FSBROWSE window while others will not be. SAS/ACCESS appears to be reliable for many applications, but this isn't one of them.

Views can be opened in EDIT mode, as in FSEDIT, or in INPUT (read) mode, as in FSBROWSE. EDIT mode always displays current values. After a view has been opened in INPUT mode, there is no guarantee that later updates will be posted (but they might be).

The cause of this strange behavior is that observations are sometimes stored in a buffer and re-used rather than re-reading a fresh copy of the observation.

This method works fine for batch procedures, which read all the data immediately. Using the buffer guarantees that each pass of multi-pass procedures (such as PROC PRINT with a BY statement) will get the same data, and increases efficiency.

Interactive procedures (such as FSBROWSE) process only as much data as is necessary to satisfy current needs (e.g. fill the screen), then pause for user interaction. Another user, or another Display Manager window, can update the base table during this pause. Such updates will not be applied to the buffer copy of observations already saved in the buffer.

The occasions on which fresh data are obtained are difficult to determine. For example, using the BACKWARD key in FSBROWSE sometimes forces the buffer to be reloaded. The safest assumption is that you may be receiving data as of any time since the view was opened.

The same considerations apply to SQL views and DATA step views when their base tables are updated. There appear to be two classes of prospective solutions:

- 1) Lock out updates to the base tables upon opening the view. This guarantees internal consistency, but is not suitable for many applications.
- 2) Force the data reading mechanism of UPDATE mode to be used, even for procedures or views that do not allow the data to be updated. This is on my wish-list for future enhancements.

C.1.b. Dynamic Base Tables: Updating through views

Using SAS/ACCESS views, you can update values, delete observations, and insert observations in the DBMS base table. The SAS/ACCESS engine does not support library functions such as creating, deleting, and renaming tables.

On the day this paper was due, I discovered that SAS/ACCESS to ORACLE does not lock observations displayed in FSEDIT. This can lead to data integrity failures: Open two FSEDIT screens on the same ACCESS view. Update the first field on the first screen. Then update the second field on the second screen. The first field will be changed back to its old value in the database.

Simple SQL views will be updatable in the future (SAS Institute, 1989b, page 43).

Input DATA step views are not updatable.

C.1.c. Dynamic Base Tables: Detecting End-of-File

Another experiment: open an FSEDIT window on a SAS data file. Open an FSBROWSE window on a DATA step view or SQL view that is based upon the same SAS data file.

After reading the last observation of the view, add more observations to the base table. The additional observations will not be detected by the view.

DATA step views and SQL views do not look for additional observations after the end of the input data has been reached. (In fact, DATA step views shut down after reading the second-to-the-last observation, because of the look-ahead buffer).

This restriction does not apply to SAS data files. For example, you can use FSBROWSE to inspect a data file receiving a real-time flow of input data, but SAS data views would not work.

SAS/ACCESS views do not have this problem.

C.2 Efficiency

To read tables efficiently, programmers use WHERE clauses and KEEP data set options to subset the rows and columns that need to be read. When using views, these specifications cannot be applied directly to the base tables, but must be pushed down through the layers of views to the base. If this can be done, then only the desired data need to be processed. Otherwise, then the entire table is processed at each layer, even though most of the results will never be used. This is the major factor affecting view efficiency.

C.2.a Efficiency: push-down observation subsetting (WHERE)

Indexes cannot be specified for views. To make use of indexes defined on the base tables, the WHERE clause must be passed through the layers of views to the base tables. If any view in the chain of views cannot push down the WHERE clause, then the base table indexes cannot be used, and the data must be scanned to satisfy the WHERE clause.

Even if the WHERE clause cannot be pushed all the way down to the base tables, the further down it is pushed, the sooner processing can be subset to just those data of interest.

DATA step views do not accept WHERE clauses and do not support any indexed retrieval capability.

SQL views only accept WHERE clauses from SQL views and SQL statements. Expressions involving variables modified by the view cannot be pushed down.

SQL views will not accept WHERE clauses from other procedures, including DATA steps, Screen Control Language, and SAS/FSP. This is a major limitation which ought to be removed.

SAS/ACCESS views accept any WHERE clause they are given, combining it with any WHERE clause of their own and passing it on to the DBMS server.

C.2.b Efficiency: Push-down variable subsetting (KEEP)

Variable subsetting (using DROP and KEEP statements and data set options, as well as SELECT clauses in SQL) can be pushed-down similar to observation subsetting, but is usually not as crucial.

SQL accepts subsetting of variables, and pushes the subsetting down further. Some SAS/ACCESS interfaces have views that accept subsetting, while others do not (van Wyk, 1991). DATA step views cannot push variable subsetting specifications down.

C.3 Direct Access

One serendipitous result of storing materialized views in buffers is that is that SQL and DATA step views of compressed data files may be accessed using SET POINT=, even though this method of access is unsupported by the base table. The original access to observation 400 (for example) may be very slow because the first 400 observations need to be materialized. After this has been accomplished, retrieval of one of the materialized observations (e.g. #200) is truly direct access.

SAS/ACCESS views of ORACLE tables do not allow the use of SET POINT=. The BACKWARD command of FSEDIT and FSBROWSE is supported, but is very inefficient when there is no buffer copy for the procedure to read directly.

C.4 Output Sequence

Access views may be read in any desired sequence, simply by specifying the appropriate BY statement. An ORDER BY clause is generated and passed to the DBMS server, which provides observations in the desired order (whether an index exists or not).

BY statements have no effect when reading a SQL view (unfortunately). To obtain a sequence different than that specified by the view, you can define a new view which specifies an ORDER BY clause to be applied to the old view.

DATA step views materialize observations in only one sequence, that specified within the DATA step.

SAS data files have an advantage over SQL views and DATA step views because they can be read in their physical sequence, or in order of any index.

C.5 Input sequence

SQL views and SAS/ACCESS views can accept input data in any sequence. DATA step views, like DATA steps, often require that input data be pre-sorted. I use SQL views to pre-sort the data within a chain of views.

C.6 Number of observations

The cost of obtaining some information outweighs the potential benefits. For example, it may not be feasible to determine the number of observations that a view will produce without actually materializing the data.

DATA step views and SQL views do not attempt to determine the number of observations that will be produced.

SAS/ACCESS views, in contrast, request that the database count the number of observations in the table or view. This is easy for tables, but can be very costly for views. This statistic appears in many contexts (such as PROC CONTENTS). I would prefer that SAS/ACCESS views followed the same conventions as other views, refusing to materialize views simply to count the number of observations. This can be easily programmed when needed.

D. Indexes

When retrieving data from SAS data files, programmers write queries that can be directly evaluated against those files. Queries against a view are specified in terms of the output from the view program. The program must be "inverted" to find the corresponding query to be applied to the input file.

SQL programs provide the most flexibility in this regard, because the sparse syntax of SQL eases the task of program inversion. The SQL engine is based upon an optimizer, which determines how to achieve the goal specified by the SQL code, given the resources available. The optimizer uses information about the input tables (Kent, 1991) including the size of the tables, indexes, and sort order (in release 6.07).

Although this information would also help the optimizer when the input data sets are views, it is not always available. For example, Kent (1991) states that "the SAS/ACCESS Interface does not surface index information". Lacking this information, the SQL optimizer might choose a less optimal strategy.

It is difficult to ascertain the strategies selected by the optimizer. The EXPLAIN statement of release

6.07 PROC SQL (Kent, 1991) will make it easier to determine what information is available, and how it is used.

Summary

The fact that different types of views have different properties has advantages as well as disadvantages. Not surprisingly, the best type of view to use depends upon the application.

SAS/ACCESS views should be selected when the data is stored in a DBMS supported by SAS/ACCESS. The performance of these views depends largely on the DBMS. The lack of any dynamic data vector capabilities imposes an extra maintenance burden when base table structures are modified. The awkward method of defining SAS/ACCESS views in release 6.06 will be improved in release 6.07 (SAS Institute, 1991b).

SQL views are usually the best choice for views of SAS data sets, because of the sophistication of the SQL optimizer and the dynamic data vector feature. SQL views are especially valuable when using PROC SQL or nested within other SQL views, because of the push-down row and column subsetting capability.

DATA step views provide the flexibility needed for LAG operations, extensive variable transformation, and reading external files. The lack of push-down row subsetting capabilities prohibits the use of DATA step views when indexed retrieval is necessary.

Views are important, and I hope this paper has given you a better understanding of the factors determining whether a particular implementation of the concept fulfills its potential.

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.
- Polzin, Jeff (1991), "An Insider's Sneak Preview of the Next DATA Step Release", in SAS Users Group International Proceedings of the Sixteenth Annual Conference. Cary, NC: SAS Institute Inc.
- SAS Institute Inc. (1989), SAS/ACCESS Interface to ORACLE: Usage and Reference, Version 6, First Edition. 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. (1990a), SAS Language: Reference, Version 6, First Edition. Cary, NC: SAS Institute Inc.
- SAS Institute Inc. (1990b), SAS Users Group International Proceedings of the Fifteenth Annual Conference. Cary, NC: SAS Institute, Inc.
- SAS Institute Inc. (1991a), SAS Users Group International Proceedings of the Sixteenth Annual Conference. Cary, NC: SAS Institute, Inc.
- SAS Institute Inc. (1991b), SAS Communications, volume XVII, No. 2 (Second Quarter 1991). Cary, NC: SAS Institute, Inc.
- van Wyk, Jana (1991), "Efficient Use of the SAS/ACCESS Interfaces to INGRES and Sybase", in SAS Users Group International Proceedings of the Sixteenth Annual Conference. Cary, NC: SAS Institute Inc.
- * SAS, SAS/ACCESS, and SAS/FSP are registered trademarks of SAS Institute, Inc.
ORACLE is a registered trademark of Oracle Corporation.

Acknowledgements

Special thanks to Mark Moorman, Stephen Vincent, Maureen Hayes, and Paul Kent for their help and good humor.

Thanks to Grant Blank for reviewing an earlier draft of the paper, and to Lorex Pharmaceuticals for providing an up-to-date computing environment.

The author may be contacted at:

Andrew A. Norton
Trilogy Consulting Corporation
1616 1/2 Forest Place
Evanston, IL 60201
(708) 982-8490