The Where, When, Who, Why, and How of Views
Lynn Palmer, State of California, Richmond, CA

ABSTRACT

Views are added code that filter, summarize, and add to data from files and databases. Added code means more processing, so when are views efficient? Additionally, databases contain tables of records: some tables match one to one, some don’t, how do you resolve the differences? Views can be inside a database, like a table, or out in SAS code, where will you put your view? Views seem vague, requiring more coding up front, so why bother? And who needs or wants views? These are some of the questions that need to be considered when deciding to use views with your database. By presenting specific situations where views can be applied and analysis of the efficiency, the “where, when, who, why, and how” will be addressed.

ARE VIEWS EFFICIENT?

Views are additional code that is being run when you run your program. Additional code means the program will run slower. However, already written and tested code saves programming time and in cases where you would write out a data set or file before running a program, you will save space. You will save time, as well, if the data set needs to be updated each time you run the program. Views allow you to access current data each time it is run. Views are also a way to save workspace while running your program.

I tested a simple view and the same code for a table on a Pentium IV PC accessing a database of about 1.3 million records across a LAN (Local Area Network) by doing a simple count of all the records in one table since for 2003. The table is not indexed on year. The code for the view is:

```
TITLE 'EXAMPLE OF VIEW CREATION';
LIBNAME MYLIB 'D:\SAS\';
PROC SQL;
CONNECT TO odbc AS MYCON (DSN = 'CON');
CREATE VIEW MYLIB.MYVIEW AS
(SELECT *
FROM CONNECTION TO MYCON
(SELECT COUNT(*) AS Count
FROM DB.dbo.tblTestDataAmino
WHERE (DB.dbo.tblTestDataAmino.Variable# > '000000021200311')
AND (SUBSTRING
(DB.dbo.tblTestDataAmino.Variable#, 11, 4) = '2003')));
DISCONNECT FROM MYCON;
```

The time for this code to run was 0.04 seconds real time and 0.03 seconds CPU time. I ran the same code using "table' instead of 'view' and it took real time 0.39 seconds and CPU time 0.01 seconds. I reused the view will the following code:

```
TITLE 'EXAMPLE OF VIEW USE';
LIBNAME MYLIB 'D:\SAS\';
PROC PRINT DATA=MYLIB.MYVIEW; RUN;
```

The time for this code to run was 0.04 seconds real time and 0.03 seconds CPU time. I ran the same code using “table’ instead of ‘view’ and it took real time 0.39 seconds and CPU time 0.01 seconds. I reused the view will the following code:

```
TITLE 'EXAMPLE OF VIEW USE';
LIBNAME MYLIB 'D:\SAS\';
PROC PRINT DATA=MYLIB.MYVIEW; RUN;
```

As you can see from this example, setting up the code to access a database can be complex. Using an already created view is only requires using the right
name. This fact can save time for staff without the same programming skills as you have.

When using SAS programs on our mainframe (an IBM MVS/OS system) where tapes of about 500,000 records each were used, I was continuously running out of workspace, even when I used tape to hold the work data sets. I found the solution was to create a data step view. When SAS creates a data step view, the data isn’t read into the program until the view is called. This allows you to manipulate and select the data in the data step without reading in the whole file into workspace. When the view is called only the data selected is used. If you are running many PROC steps, this method is not as efficient because the database or tapes are re-accessed each time. An example of code for a data step view is:

```
TITLE 'EXAMPLE OF DATA STEP VIEW';
DATA AA/ VIEW = AA;
INFILE DDNAME;
INPUT $1 VAR1 $CHAR1. $2 VAR2 $CHAR1.;
IF VAR1 EQ 'A' OR VAR2 EQ 'A';
RUN;
PROC FREQ DATA=AA; TABLE VAR1*VAR2; RUN;
```

Using a two-part name, i.e. MYLIB.AA, can save a data step view. If you are running many PROC steps, this method is not as efficient because the database or tapes are re-accessed each time.

**VIEWS AND DATABASE STRUCTURE**

Views are an ideal solution to accessing databases. In SAS, there are interface SAS views that are used with an Access descriptor and PROC SQL views. `CONNECT TO` and `USING LIBNAME` statements allow PROC SQL to interface with databases such as DB2, SQLServer, Oracle, and MSAccess. With DB2 and PROC SQL, you can access up to 16 tables at a time. Most databases have tables that are not one to one. Frequently, database contain tables with records that are one to many or many to many. This fact can create problems when writing out data using a view that reads from several different tables.

The main problem is the view will create a record for every possible data combination unless you control what is written out. This requires being aware of the database structure. You need to have an idea of how many records in one table might match with a record or records from another table. Then a decision by the users and management has to be made. Are you going to keep two versions of the address? If there is an AKA, are you going to include it or drop it? Which multiple records are needed, which can be limited?

For example, in our database, address is in one table and name is in another. When a study is being done where the person has more than one name or more than one address, the analyst tries for all the addresses and names. The result is for a person with 2 names and 2 addresses (often due to misspellings) will be 4 records written out. The first name with the first address, the first name with the second address, the second name with the first address, and the second name with the second address. This result, called a Cartesian product, is not what is expected. There are ways to handle this duplication.

The first step is to look at the database. In one month worth of data from our database, we found between 4 and 12 records just from the name and address table combinations for about 300 cases out of 30,000. The next step is to decide which to use. In order to write out an unduplicated data from the view, the data will have to be ‘flattened’. Holding space for 3 or 4 names and 3 or 4 addresses is cumbersome. A decision might be made to use the first and second or last. Flattening data requires planning ahead and writing 2 or more of the same field with different information into one record. One way to flatten data from a database is to use in-line views.

**In-line views**

In-line views are queries made in the `FROM` statement of PROC SQL. They are named and used to select the desired data. These views can check for duplicate values, select the best information, and write out a flag or variable that indicates what was found when reading the database. The following code is an example of PROC SQL where an in-line view called, QRY1 and an in-line view called, QRY2 aid in selecting at most 2 names and 2 addresses into the same record. Flags are set to indicate how many of each was found.

```
PROC SQL;
CREATE VIEW MYVIEW AS
SELECT *
FROM (SELECT TABLE1.NAME AS NAME1,
      TABLE2.ADDR AS ADDR1
   FROM (SELECT COUNT (*)
         AS FLAG1
         FROM TABLE1
         WHERE TABLE1.VAR1=TABLE2.VAR1) QRY1,
   (SELECT COUNT (*)
    AS FLAG2
    FROM TABLE2
    WHERE TABLE2.VAR1=TABLE1.VAR1) QRY2
   WHERE QRY1.FLAG1=1 AND QRY2.FLAG2=1) QRY3
AND (SELECT TABLE1.NAME AS NAME2,
     TABLE2.ADDR AS ADDR2
    FROM QRY1, QRY2
    WHERE QRY1.FLAG1=2 AND QRY2.FLAG2=2) QRY4
WHERE QRY1.FLAG1 < 3 AND QRY2.FLAG2 < 3;
```

QRY3 and QRY4 need to be named but are not called. This program will write out NAME1, ADDR1, NAME2, ADDR2, FLAG1, and FLAG2. FLAG1 is the count of names and FLAG2 is the count of
addresses for that particular case. As you can see, selecting more than a couple of name and address fields can make the code very complex. The more complex the code, the less efficient the view. Planning for what is required and writing different views for different purposes is important. Several views do not take much space to store, one large many-purpose view will cost at run time.

**VIEW LOCATION**

A view can be situated inside of your database or outside in a SAS library or created temporarily in code. Views that are part of your database can be created with PROC SQL or with the database query mechanism, usually SQL code. The SQL code will be the version of SQL used by the database even when PROC SQL is used in a SAS program. Views inside the database have an advantage in that they can use the power of the database to retrieve and manipulate data. This fact allows them to run faster and more efficiently. Unless you are the database administrator, you will have to get permission to store the view in the database or have the administrator create the view (possibly run your SAS program for you).

Where a view is placed can effect its performance. A view that allows a summary of data from one database to be read when accessing another database is best placed into the database from which the data will be accessed. Running the view on the client or the server can also effect performance, not only of the view, but also of the server or the network. For example, we have several SQLServer databases on one server on a LAN. Data was being accessed using SAS and MSAccess. The MSAccess queries (which are SQL views stored in a PC application) were bring the network to a stand still. By placing frequently used views on the server inside the databases, processing was improved. Additionally, for the situation where data was needed from two of the databases, a view containing a selection of needed fields from one database was place into the second. This solution not only helped the MSAccess queries run faster, but also the SAS jobs.

SAS views outside of a database can be stored in SAS libraries. Stored SAS views have the ending, ‘SAS?VIEW’. A view stored in a SAS library has the member type, ‘view’. You can not write a table (data set) with the same name as your view to the same SAS library. You can use PROC CONTENTS and PROC DATASETS to manage your stored views. Views outside your code are referenced with a two-part name, the libref or ddname and the view name. The view code will print to the log if the key word, DESCRIBE is given after the view is called. For example:

```sas
DATA VIEW=LIBREF.VIEWNAME; DESCRIBE; RUN;
```

The option FEEDBACK can also be used. It will expand the SELECT * statement in the log when used with PROC SQL. PROC SQL views can be changed when being used using the MODIFY command; variables can also be removed by using the DROP command. Views outside your database have the advantage that you have control of them and they can be shared with other users by allowing access to your SAS library. Views can be created for groups of users and stored in a central location.

**CONNECT TO and USING LIBNAME**

PROC SQL can be used to connect to databases and to combine two or more databases in a view using a USING LIBNAME statement with the database engine and other parameters, like password. By using the CONNECTION TO and CONNECT TO statements with an ODBC link or the system link to your database, you can interface with a database using a PROC SQL. If you save your PROC SQL view with a two-part name to a library, then you can access your database with the view without rewriting the code. You can also share the view without revealing the connection code. The following is code for connecting to SQLServer:

```sas
PROC SQL;
CONNECT TO ODBC AS MYCON (DSN=YOURID)
CREATE VIEW AS LIBNAME.VIEWNAME
SELECT * FROM CONNECTION TO MYCON
    SELECT TABLE.FIELD FROM LIBNAME.TABLE
    WHERE SOME_CONDITION IS TRUE;
DISCONNECT FROM MYCON;
```

ODBC stands for Open Database Connectivity. It is an interface standard that provides a common application programming interface (API) for accessing databases. Many software products that run in the Windows operating environment adhere to this standard, giving users access to data that was created with other software. To create an ODBC link on your PC, go to ‘settings’, ‘control panel’, and ‘ODBC data sources’. There you will find a set of Microsoft menus to guide you in making a link to your database. You may have trouble setting up an ODBC link to a network server without the administrator’s help, however you can set up links to your MSAccess applications.

New with version 7 of SAS is the use of LIBNAME with a database engine to connect to your database. The ‘USING’ statement containing the LIBNAME statement gives this flexibility. The libref assigned inside the view is local to the view and will not conflict with other uses of the same libref in the same SAS session. The libref is de-assigned at the end of the query. An example of the code is:

```sas
PROC SQL;
CREATE VIEW MYVIEW AS
    SELECT * FROM MYLIB.TABLE TABLE
```
USING LIBNAME MYLIB DB2;

Multiple LIBNAME statements can be specified, separated by commas. And the USING statement must be the last one in your SELECT statement. The parameters USER=, PASS=, and PATH= add flexibility to this statement.

SECURITY

Passwords

Views can have passwords set when they are created which limit their use. Password protecting a view can affect the actual view or descriptor as well as the underlying data. There are three levels of security. The first is READ which allows read only of underlying data, allows source code of view to be printed to the log using DESCRIBE, and allows replacement of the view (overwriting). The second is WRITE allows writing to data/database. The third is ALTER which protects against reading, protects against source code being printed to the log using DESCRIBE, and protects against replacement.

To assign a password, you use the key words READ=, WRITE= or ALTER= with your password in parentheses. You can assign all three passwords and only give certain users the most appropriate one. For example:

```
CREATE VIEW MYLIB.MYVIEW
(READ=READ, WRITE=WRITE, ALTER=ADM) AS
SELECT *
```

```
 or
```

```
DATA MYVIEW/VIEW=MYVIEW(ALTER=MYPASS);
```

To use the view, the correct password must be used. It is given after the view name:

```
DATA NEW; SET VIEW(PW=MYPASS);
```

```
 or
```

```
PROC FREQ DATA=VIEW(PW=READ);
```

Passwords can add additional security beyond the security offered by creating different views for different groups of users.

Different Views

A different view for various groups of user is also an effective form of security. One view accessing the database can be for management revealing the salaries of staff, the next view can contain a subset of fields or summary variables that conceal the exact salary amounts and who earns them.

CONCLUSION

Views are an excellent solution for working with databases. Be aware of the possible pitfalls, plan ahead, and give views a try. You find a new tool to use in solving data access problems.

REFERENCES


SAS/ACCESS Interface to IMSDL/I: Reference; DATA; LIBNAME, SAS/ACCESS; The SQL Procedure; SAS On-line Documentation Version 8, SAS Institute Inc., Cary, NC, USA, 1999


SAS is a registered trademark or trademark of SAS Institute Inc. in the USA and other countries. ®Indicates USA registration.

IBM is a registered trademark or trademark of International Business Machines Corporation in the USA and other countries. ®Indicates USA registration.
AUTHOR
I have many years’ experience using SAS in a mainframe MVS and PC Windows environment. I develop views and SAS code for users on our staff and for researchers outside our organization to allow and manage access to our databases.

Lynn Palmer, State of California, DHS, GDB
850 Marina Bay Pkwy, F-175
Richmond, CA 94804-6403

Email: lpalmer@dhs.ca.gov,
      lynnpalmer@comcast.net

Phone: (510) 412-1510.