Using SAS Views and SQL Views
Lynn Palmer, State of California, Richmond, CA

ABSTRACT
Views are a way of simplifying access to your organization’s database while maintaining security. With new and easier ways to access data, more users are able to get their data directly from the organization database. This data often requires some knowledge of the “rules” for summarizing and presenting the information. These rules may not be readily available to or known by all staff. Additionally, database structures can be complex and difficult to navigate. One way to manage data access is to set up views that govern how the data is extracted. Views can also make it easier to access complex data stored in a database by summarizing the data for the user. You can create SAS views and SQL views that are transparent to your user and allow you to manage data access.

INTRODUCTION
What Are Views?
A view is like a stored query. It contains no data, but instead contains code that describes, defines, or selects stored data when it is needed. The data usually is in a database, however it could be in a SAS dataset or file. Views are named and used like a table, dataset, or file would be used. However, views are only code. The view is a subset or superset of the data from which it is created. The advantage of using views is they take little space, they access the file or database at execution reflecting the most current data, they can reference and join multiple sources of data reducing code, they can create temporary variables for statistics, they can be used to rename variables from the data source, and they can filter data making them good for security. Views can also be assigned passwords. Different views can access the same files for use by different groups of users.

There are two groups of views presented in this paper. The first group is the views created and used in SAS code: DATA step views, PROC SQL views, in-line views, and interface (SAS/ACCESS descriptor) views. The second group of views is those created inside your database, frequently with SQL, or with PROC SQL. These views are code residing inside the database. They can use the power of the database to process, update, and protect data. For many databases, your database administrator/creator must grant you permission to create these views or create them for you.

Why Use Views?
Views allow for a “virtual tier”, between the database tables and user code. This tier approach creates greater flexibility in your data system. Views can remain unchanged as system changes are made to the database, allowing the users to continue without disruption. Needed changes can be managed with views instead of changing the underlying database. Security can be adjusted or inserted by creating different views for various users.

In some data systems, the knowledge of how to operate in certain situations, or what should be interpreted from a particular event, or when to generate a special report is known only to one or two employees. By putting this knowledge into an organizational library of views, more staff can act independently, and staff can change without crisis.

VIEWS IN SAS CODE
There are three types of SAS data views: DATA step views, PROC SQL views, and SAS/ACCESS views (view descriptors). SAS data views are the member type, VIEW and are defined as native or interface. Native views are the views created in a DATA step or with PROC SQL. Interface views are created with SAS/ACCESS to read or write data to a database (DBMS) such as DB2, ORACLE, or ADABAS. From Version 7 forward, it has been possible to create native views that use the USING LIBNAME statement and therefore, are able to directly interface a database. Both native and interface views can be used with DATA or PROC steps.

SAS Data Views:
SAS DATA step views tell SAS how to process data using the code that you specify in the DATA step. Only one view can be created in a DATA step and the view name must match one of the DATA step names. This type of view can only read the underlying data. DATA step views can generate datasets when the view is executed. A view is executed when it is used in a PROC or DATA step. The view can be stored in a SAS library as a VIEW type, if you create it with a two-part name (library,viewname). The syntax is:

DATA VIEWNAME / VIEW=LIBREF.VIEWNAME;

To use a view, treat it like a dataset. Use the DATA= statement with PROC, and the SET statement with DATA steps. For example:

DATA NEW; SET LIBREF.VIEWNAME; _____ RUN; or
PROC FREQ DATA=LIBREF.VIEWNAME;

There are security features that can be used when creating a SAS DATA view. You can password your views. There is more on using passwords under the password section of this paper. The phrase (SOURCE=ENCRYPT) after the view name will encrypt your view, for example:

DATA VIEW_A/VIEW=LIB.VIEW_A(SOURCE=ENCRYPT);

The term, DESCRIBE, can be used to write a copy of the view source to your SAS log:

DATA AA; SET VIEWNAME; DESCRIBE; RUN;

Printing the source code to log will allow you to see the name of the underlying data file or database, and how the code in the view is a subset or a superset of that data.
Using DATA step views does reduce the efficiency of the program. Therefore, code using a view may run a little slower than code accessing the data source directly. This reduction may be reasonable, if you are realizing a saving of time by reusing code. Views can be used to reduce workspace, because SAS does not read the underlying data into a work dataset when a view is used. Data in large files can be selected into a subset using a view preventing the job from running out of space and failing. Working on an IBM mainframe, I use temporary DATA step views to select the data from tape files of 500,000 to million records. The code looks like this:

```sas
DATA VIEWNAME/VIEW=VIEWNAME; INFILE DDNAME;
INPUT @1 VARI SCHAR1. ...more input...;
IF VARI LE 9 AND VARI GT 0; RUN;
PROC FREQ DATA=VIEWNAME; TABLES ...more code...;
```

This use allows more workspace for processing. The view is not saved, once the program ends and the workspace is deleted the view is gone. The drawbacks are that you can't use a DATA view with PROC SORT without creating a dataset and if you are running many PROC statements using the view, your efficiency declines.

**SQL Views:**

SQL views are defined in PROC SQL using a CREATE VIEW statement. They are based on SAS datasets, sequential files, databases, and other views. They can contain code to connect to a database using the PROC SQL pass-through facility. They can be in the program code or stored in a SAS library. These views are more efficient because at execution they utilize the SQL Processor that optimizes the access strategy. The following is an example:

```sas
PROC SQL;
CREATE VIEW AS LIBNAME.VIEWNAME;
SELECT TABLE.FIELD
FROM LIBNAME.TABLE
WHERE SOME_CONDITION IS TRUE;
```

Views contain no data so they take little space to store. They are stored in a SAS library as a VIEW member. You can refer to tables (datasets) and views using a view which resides in the same SAS library without the library name reference. The view is referenced in your SAS code like a dataset or in the FROM clause like a table. For example:

```sas
PROC FREQ DATA=LIBNAME.VIEWNAME;
```

or

```sas
PROC SQL;
CREATE TABLE AS MYTABLE;
SELECT *
FROM LIBNAME.VIEWNAME;
```

Beginning in Version 7, a PROC SQL view can be used to update the underlying database data. To update, the view must be based only on one database table or a database view with no calculated fields based on the table. You can use the ORDER BY clause in your CREATE VIEW statement, however it may be more efficient to sort when needed when using a view.

**Connect to/Connection to:**

One of the important features of SQL views is that they can be used to connect to other software vendor’s products, such as DB2, ORACLE, MSAccess, MSExcel, or MSSQLServer. The CONNECTION TO, and CONNECT TO statements allow this access. You can place these statements in a view so that your users can access the database without struggling with the code. The following is code for connecting to DB2:

```sas
PROC SQL;
CONNECT TO DB2 AS MYCON (SSID=YOURID);
CREATE VIEW AS MYVIEW
SELECT *
FROM CONNECTION TO MYCON
SELECT TABLE.FIELD
FROM LIBNAME.TABLE
WHERE SOME_CONDITION IS TRUE;
DISCONNECT FROM MYCON;
```

**ODBC**

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 a 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. Use that ODBC link in the DSN= portion of your CONNECT TO statement. Your SAS code will look like this:

```sas
PROC SQL;
CONNECT TO ODBC AS MYCON (DSN=’MYLINK’);
CREATE VIEW AS MYVIEW
SELECT *
FROM CONNECTION TO MYCON
(SELECT *
FROM TABLE T1
WHERE T1.APP_DATE BETWEEN '02/01/2003' AND '02/08/2003');
DISCONNECT FROM MYCON;
```

**In-Line Views:**

In-line views are defined in a PROC SQL query and used as part of the query. These views select or manipulate data in your program for use by that program. Using these views can simplify complex code. For example, in the following program, I am only selecting records from Table2 that are true for some condition in Table1, instead of using one program to select the true records from Table 1 and using another program to search for those records in Table 2.

```sas
PROC SQL;
CREATE TABLE AS MYTABLE
SELECT *
FROM (SELECT TABLE1.FIELD
FROM LIBREF.TABLE1
WHERE SOME_CONDITION = TRUE) MYVIEW,
LIBREF.TABLE2
WHERE MYVIEW.FIELD = TABLE2.FIELD;
```

In this code, SQL is using the view, MYVIEW, described in the FROM statement in parentheses with a table, LIBREF.TABLE1, from the database. The in-line view must be assigned a name after the parentheses. This in-line view demonstrates more efficient programming by reducing the number of steps need to select data and link.
Libname

A LIBNAME statement with a libref describing the path to your database will allow querying, updating, or deleting data in your database. The LIBNAME statement can be embedded in your PROC SQL view. Additionally, the SAS/ACCESS view libref can be used with LIBNAME automatically connecting you to your database. This use of LIBNAME is new with version 7.

The “USING” statement containing the LIBNAME statement gives this flexibility. The libref that is 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.

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. In the following example, a connection is made and the libref, MYREF, is assigned to an ORACLE database.

CREATE VIEW MYLIB.VIEW1 AS
SELECT *
FROM MYREF.TABLE TABLE
USING LIBNAME MYREF ORACLE
USER=USERNAME
PASS=PASSWORD
PATH=’PATH TO THE DATABASE/URL’;

{PATH=’C:/MYDIRECTORY/MYDATABASE’;}

Access Descriptor Interface Views:

SAS/ACCESS views are views that interface with the data of other software products. A descriptor is created for your database table or file using the SAS/ACCESS product for your non-SAS database or files. These descriptors can only access one table or file at a time. The view is based on this descriptor. The descriptor contains the appropriate engine for the database or files that you are accessing with the view.

In these views, you can format fields with SAS formats. You can assign different names to the database fields in the view and change the field length. You can limit the fields available to your users and you can include SAS code to change field values or create new fields from the fields in the underlying database table. Additionally, with this type of view, you can change the data in the underlying database table by updating, modifying, inserting or deleting.

The SAS/ACCESS descriptor for the table can be created interactively or in batch. The following code will create a descriptor and its view for a MSExcel worksheet:

PROC ACCESS DBMS=XLS;
CREATE LIBREF.TABLE.ACCESS;
PATH=’C:/YOURFILE.XLS’;
GETNAMES=YES;
LIST=ALL;
CREATE LIBREF.TABLE.VIEW;
SELECT FIELD;

If you are going to access a view repeatedly, you can improve performance by extracting the information you need into a SAS dataset. By using SET with the DATA step or OUT= with a PROC step, a SAS dataset can be created from the view. Repeated referencing of a view in the same SAS program/session will lead to use of more resources than necessary to get the job done.

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, and protects against replacement.

To assign a password, you use the key words READ, WRITE or ALTER and an equals sign with your password in parentheses. For example:

CREATE VIEW MYLIB.MYVIEW(WRITE=MYPASS) AS
SELECT *
or
DATA MYVIEW/VIEW=MYVIEW(ALTER=MYPASS);

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

DATA NEW; SET VIEW(PW=MYPASS);

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

SQL Views

DB2 Views:

Views in DB2 are SQL code. They can be created using PROC SQL, if you are granted permission to create views in your organization’s database. The SQL features available depend on the database not SAS. These views can be very powerful and efficient as they are part of the database and can utilize the database Processing tools. Code to produce a DB2 view is as follows:

PROC SQL;
CONNECT TO DB2 AS MYCON (SSID = YOURID);
CREATE VIEW AS FULL.NAME.MYVIEW
SELECT TABLE.FIELD
FROM FULL.TABLE.NAME TABLE
WHERE SOME_CONDITION = TRUE;
DISCONNECT FROM MYCON;

These views can also be programmed in SQL executed by
the database administrator. The view will be stored in the database like the tables in the database. However, unlike the tables, the views will not require space.

**MSAccess Queries:**

You can set up MSAccess queries to act like views in a MSAccess database. The query will contain SQL code (Microsoft®'s version). You can access this query the same way that you access a table using the PROC SQL pass through facility (CONNECT TO) and ODBC. These queries can be created visually using MSAccess menus or by programming SQL code in MSAccess. The following code will allow you to use a query in MSAccess with SAS. You will first need to set up the ODBC connection (MYLINK) on your PC (see above).

```sql
PROC SQL;
CONNECT TO ODBC AS MYCON (DSN='MYLINK');
CREATE TABLE MYTABLE AS
SELECT *
FROM CONNECTION TO MYCON
(SELECT QUERYNAME.FIELD
FROM QUERYNAME);
DISCONNECT FROM MYCON;
```

**MSSQLServer Views:**

SQL Server views are very similar to MSAccess queries. They can be connected to with an ODBC connection. The user uses the view like a table in the database. These views can change data in the underlying database. They allow for the processing to be done inside the database using the database which provides greater efficiency. A view in one SQLServer database can access another SQLServer database.

```sql
CREATE VIEW dbo.VIEW_1 AS
SELECT Vари, VAR_ДТ AS SYS_DATE, VAR2
FROM SERVERNAME.SYSNAME.dbo.TABLE
WHERE (Vари LIKE '%АВС%' or
Vари LIKE '%CDE%' and
VAR2 = 'Some character string');
```

In our office, the are production databases and study databases. Frequently, they need to be used together. We were using two PROC SQL CONNECTION TO statements to access our data in different databases. Linking the two resulting tables was required. By placing a view of the production database that limits the fields and the time period into the study database, users are able to access both databases with one connection.

```sql
CREATE VIEW VIEW_2 AS
SELECT *
FROM DATA_1.dbo.VIEW_1
WHERE SYS_DATE > '1/6/2002';
```

Such a simple view reduced complaints, and errors dramatically.

**EXAMPLES OF VIEWS**

Views as an excellent tool to build a virtual layer between your database and your users. An example is the use of DB2 views in our research unit. The administrator of our DB2 database named the fields and tables in the database with 16 character names to reflect the source names from another database. Our unit writes SAS code to retrieve data from DB2. We share our SAS code within the unit. We like standardized names for fields and we needed eight character names for fields and tables before Version 7. So, we developed SAS names for the fields, grouped in ways that make analysis easier. We, then, created views with eight character names in DB2 that renamed and regrouped the fields. These views don’t take up storage space, they allows us to find fields faster, and they run as fast as using the underlying tables without the views.

Our DB2 database administrator changed the structure of the underlying tables in our database, combining several together for more efficiency. He recreated the views by changing only the table names in the code of the views, so the views appear to be unchanged to the users. This use of views allowed him to make the changes without having to disturb our method of working or causing our code to be rewritten. Thereby, avoiding a battle over how and where fields are grouped and named.

In our data, one client can have many records. Which record to analyze can become difficult to determine without detailed coding. A view stored in a SAS library with the indicator pointing to the correct record to choose for analysis allows analysts to avoid coding and errors when doing client analysis.

By using this view, NEW, the dataset, OLD is checked for missing dates and those records with errors are filtered out of the NEW dataset. At the same time, a dataset of the records with missing dates is created, ERRORS.

```sql
DATA NEW ERRORS / VIEW = NEW;
SET OLD;
IF DATE = ., THEN OUTPUT ERRORS;
ELSE OUTPUT NEW;
RUN;
```

The dataset, ERRORS is created only when the view is executed and there are missing dates.

**CONCLUSION**

The examples above are simple, but the application of views can be as complex as your situation needs. Views can be a powerful tool when accessing data in your database. Views can allow your users to proceed without misusing the data in your database.

**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.

MSWindows®, MSDOS®, MSVisual Basic®, MSWord® and MSExcel®, MSSQLServer® are products of the Microsoft Corporation. Copyright © 1983-2002 Microsoft Corporation. All rights reserved.

Other brand and product names are registered trademarks or trademarks of their respective companies.

AUTHOR

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.