Views — Your Window on Data
Paul Kent
SAS Institute Inc., Cary, NC

Abstract
This paper describes the creation and storage of user-defined views of physical data, as implemented by various components of the SAS® System. It addresses both the similarities and differences among:

* Views created using the features of the native database management system (DBMS).
* Views created using the ACCESS procedure to associate SAS names and other information with data stored in an external DBMS such as ORACLE®, System 2000®, or IBM’s DB2®.
* Views created using the SQL procedure, SAS Institute’s implementation of Structured Query Language (SQL). PROC SQL is available with Release 6.06 of the SAS System.

The SAS System is developing “database ability” — indexes, WHERE clauses, SAS data set compression, and PROC SQL are examples of SAS Institute’s efforts to upgrade the database-like facilities available to the SAS user. The SAS/ACCESS family of products opens gateways to more and more external databases.

There is a lot of common ground among these diverse database systems. This paper focuses on the ability to establish user views of data stored in some database structure. It introduces the similarities and differences between the views provided by the various systems and reconciles the terminology used by those systems.

Database terminology
Unfortunately, the database world did not standardize its usage of terms to describe the components of a database. This table summarizes some of the terminology used by some database systems.

<table>
<thead>
<tr>
<th>DB2</th>
<th>Oracle</th>
<th>Rdb</th>
<th>S2K</th>
<th>SAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>tables, rows, columns, views</td>
<td>tables, rows, columns, views</td>
<td>relations, records, fields, views</td>
<td>segments, occurrences, fields, userview</td>
<td>datasets, observations, variables, views(2)</td>
</tr>
<tr>
<td>... and so on ...</td>
<td>... and so on ...</td>
<td>... and so on ...</td>
<td>... and so on ...</td>
<td>... and so on ...</td>
</tr>
</tbody>
</table>

In some systems, a view is a subset of the entire database where the subset you access is determined by your user ID when you log in to that database. SQL-based views are not limited to a subset of an existing structure, they may create new columns using some expression, and may be derived from more than one real database table.

The SAS System has two kinds of views: those that describe components of an external DBMS created by the ACCESS procedure and those that are SQL-based views created by the SQL procedure.

An Example
Our company records information about its employees in a DBMS. If we are "nouveau-DP" we might use a relational DBMS and store the data in tables that have been normalized to minimize the redundancy and interdependency of data stored in the rows of the tables.

The Employee Table records information about employees in our company:

<table>
<thead>
<tr>
<th>name</th>
<th>dob</th>
<th>phone</th>
<th>dept</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>paul</td>
<td>41feb72</td>
<td>7284</td>
<td>IPO</td>
<td>1000</td>
</tr>
<tr>
<td>mary</td>
<td>10feb75</td>
<td>7734</td>
<td>PUB</td>
<td>1200</td>
</tr>
<tr>
<td>fred</td>
<td>24aug68</td>
<td>7554</td>
<td>CGR</td>
<td>875</td>
</tr>
<tr>
<td>tom</td>
<td>18aug77</td>
<td>7791</td>
<td>TND</td>
<td>980</td>
</tr>
<tr>
<td>john</td>
<td>15may74</td>
<td>7602</td>
<td>MKT</td>
<td>1375</td>
</tr>
</tbody>
</table>

The Roadrace Table records information about employees who have participated in local running events:

<table>
<thead>
<tr>
<th>name</th>
<th>racename</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>paul</td>
<td>cary roadrace</td>
<td>52:21</td>
</tr>
<tr>
<td>mary</td>
<td>cary roadrace</td>
<td>32:21</td>
</tr>
<tr>
<td>mary</td>
<td>old reliable</td>
<td>42:07</td>
</tr>
<tr>
<td>tom</td>
<td>cary roadrace</td>
<td>22:11</td>
</tr>
<tr>
<td>tom</td>
<td>old reliable</td>
<td>32:34</td>
</tr>
<tr>
<td>tom</td>
<td>run 4 roses</td>
<td>27:31</td>
</tr>
</tbody>
</table>

On the other hand, you may have a hierarchical DBMS like System 2000. After consideration of the parent-child relationships among the data that you wish to store, you could organize the data with the EMPLOYEE as the root level and ROADRACE and JOBDetail segments as children of the appropriate EMPLOYEE record.
Views In general

Most DBMSs provide a facility for defining views on the data stored within that DBMS. The methods used to define the view vary from system to system. Relational DBMSs use SQL for the most part, but some have their own query language. Hierarchical DBMS views are usually limited to subsets of the physical data structure, although some can have links from one physical structure to another.

Once created, users reference database views as if they were tables. Some DBMSs place restrictions on updating data through views, but read access is almost always the same as accessing a "real" table.

Views facilitate accessing data from a vantage point not presented by the real fields stored in the database. A security-minded person could establish a view that excluded the salary field of the employee table. This can be accomplished in some hierarchical databases by creating a password with access to the first four fields of the employee segment only. The user of an SQL based DBMS could create a view like this:

```sql
CREATE VIEW SAFE_EMP AS
SELECT NAME, DOB, PHONE, DEPT FROM EMPLOYEE;
```

Views may span more than one base table. A person trying to determine if there is a correlation between age and performance in the road races would like to see that information presented logically as a single table. It would be nice if the DBMS could transform date of birth (DOB) into current age, obviating the need for a data step to compute this.

In a relational DBMS, this involves a join between the two example tables and can be coded like this:

```sql
CREATE VIEW RUNDATA AS
SELECT E.NAME, RACENUM, TIME, ((TODAY() - DOB) / 365 AS AGE
FROM EMPLOYEE E, ROADRACE R
WHERE E.NAME = R.RNAME;
```

Views can be considered as pipelines that facilitate the flow of data between the source (physical data storage) and the consumer (a SAS procedure in many cases). Views can:

- transform data before the procedure sees it.
- connect data sets and procedures.
- connect data sets to other data sets on some common attribute.

Views are cheap to store because they only store how to materialize the data. They do not store a redundant set of the data values — these are computed as the view is accessed. Because a view is always materialized from the underlying physical data at the time of the request, data presented to the procedure by the view mechanism are always up-to-date.

Access to DBMS entities from the SAS System

Traditionally the SAS user has accessed their DBMS data by creating a flat file of that data and then importing that file into a SAS data set using the data step and the INFILE statement.

Release 79.6 of the SAS System included a data step interface to the IMS database that removed the need for the intermediate flat file, although you had to be a capable DL/I programmer to use it.

Version 5 of the SAS System added procedures like DB2EXT, ORACLEXT, and S2K. These procedures are menu-driven and require no DBMS programming skills. Although they do not require an intermediate flat file, they must make a SAS data set that is a copy of the data selected from the DBMS.

Version 6 of the SAS System introduces the concept of "IO Engines", where the DBMS data are available to the SAS procedure directly, avoiding the requirement to create a SAS data set that (redundantly) contains a copy of the data stored by the DBMS.

Because the DBMS does not always store all the dictionary information (formats, labels...) and the SAS System cannot process names longer than eight characters yet, SAS/ACCESS software must store this information in a descriptor for the database entity. This "roadmap" that describes all the mappings between DBMS entities and the SAS notation for them is stored in a master descriptor by SAS/ACCESS software. It contains all the Operating System- and DBMS-specific information required to connect to a specific database in the DBMS.

SAS/ACCESS software allows you to set up views of this master descriptor of a database entity. Currently, with Release 6.05 of the DBMS, a master descriptor (and hence a SAS/ACCESS view on a master descriptor) can only reference a single database entity. Of course, if the DBMS supports views you can access multiple DBMS entities by creating a view in the DBMS, and a SAS/ACCESS software view to access that DBMS view.

In simple terms, SAS/ACCESS software allows data from other DBMS's to masquerade as SAS data sets. The SAS procedures that operate on data materialized by SAS/ACCESS are indifferent to the source of that data. As far as the procedure is concerned, its just operating on a SAS data set.

PROC SQL features for creating Views on data.

The SQL procedure processes SAS data sets using SQL. The user can enter SQL statements to be evaluated in immediate mode by PROC SQL, or create views that can be processed by other SAS procedures.

Because SAS/ACCESS creates views that masquerade as SAS data sets, PROC SQL can create views of these SAS/ACCESS views. This is useful when data resides in more than one table (since SAS/ACCESS software is currently restricted to one database entity) or when data resides in two completely different databases.

Putting it all together

This scenario is complex but allows one to get a feel for the permutations possible when using SAS/ACCESS software to create views on external DBMS data, coupled with views created by PROC SQL. In practice, you probably do not need to have a view defined at all three levels.

Suppose we have:

1. Database data in two tables A and B
2. Database view of [1] combining A and B
3. SAS/ACCESS master descriptor of [2]
4. SAS/ACCESS subdescriptor (view) of [3]

The ability to reference the data directly from the DBMS is a desirable feature, but there are times when creating an intermediate SAS data set is a better approach. It is hard to give guidelines for appropriate use of either method, but I am sure that you will experiment with the alternatives.
There is some overhead in materializing the data through all these levels. If you wanted to perform 10 consecutive PROC GCHARTs on this data, it would be more efficient to make a copy of the data in a temporary SAS dataset, and then process that data set. This process is not difficult, because the view libname.SQLVIEW created in [5] above behaves as if it were a SAS data set. This simple data step suffices to create a temporary copy:

DATA WORK.ANALYSIS;
SET libname.SQLVIEW;
RUN;

Where to keep a view?

How should one decide where to store a view when there are so many possibilities? Unfortunately, there is no definitive answer because there are many factors to consider. I will summarize some of the alternatives here.

If the data are all in the same DBMS, then you can consider many options. The merits of these options follow this decision chart.

**Can DBMS support View?**

**YES:** create view in DBMS, and use SAS/ACCESS to process DBMS view

**NO:**

**Can SAS/ACCESS support View?**

**YES:** create view using SAS/ACCESS

**NO:**

**Can PROC SQL support View?**

**YES:** create SAS/ACCESS view on each table, and a PROC SQL view that joins data as desired

Create the view in the DBMS. This approach is probably the most effective because the DBMS has the best opportunity to optimize queries against that view. Sometimes it is not appropriate to create the view in the database, you may not be authorized to do that, or you may not use the view enough to warrant storing it in the database.

Create a SAS/ACCESS view. This may be a better approach if you are not authorized to create views in the DBMS or you change the view often. SAS/ACCESS software is menu-driven, so it may be easier to use it to create views. SAS/ACCESS software presents the entire query to the database, so there is a good chance that the view will be optimized appropriately by the DBMS. (Release 6.06 will not support more than one table per SAS/ACCESS view. This feature will be available in a later release of the SAS System.)

Create a PROC SQL view. This may be your only option. The DBMS may not be able to join the tables (and if this is so, SAS/ACCESS is unlikely to support it either). You may want to use PROC SQL features that are not supported by the underlying DBMS; outer joins, set operators like EXCEPT and INTERSECT, and access to the wide range of SAS data step functions are features rare in DBMSs today. The DBMS does not get a chance to optimize the query — PROC SQL does its own query optimization, but probably does not match that of the DBMS which has access to many of the storage characteristics of the tables in question.

**But my data is in Different DBMSs**

If the data upon which the view is based are not in the same DBMS, your options are limited. Some DBMSs can support views of data in heterogeneous systems. You could create a DBMS view in this fashion, then access that view with SAS/ACCESS software.

The other alternative is to create a SAS/ACCESS view for each individual table, and then create a PROC SQL view that joins those two (or more) SAS/ACCESS views.

**Summary**

This paper introduced the view concept as implemented by external DBMSs, the ACCESS procedure, and the SQL procedure. It showed that these views can be used to present a logical table to the user that is some transformation of the stored data.

As more external DBMSs are supported by the ACCESS procedure and support for remote SAS data set access over a network is available, views will become increasingly useful vehicles for combining data stored in heterogeneous sources.

SAS and System 2000 are registered trademarks of SAS Institute Inc., Cary, NC.

SAS/ACCESS is a trademark of SAS Institute Inc., Cary, NC.

ORACLE is a registered trademark of Oracle Corporation, Belmont, CA.

DB2 is a trademark of International Business Machines Corporation.