

Frame Your View of Data with the SQL Procedure

Kirk Paul Lafler, Software Intelligence Corporation

ABSTRACT

The SQL view is a powerful feature of the relational model. Data becomes visible as a framed picture through which you can see information. This paper presents the many features of an SQL view and how it can be used as a building block by users and application programmers alike. A tutorial approach will be used to illustrate the virtues found in building, using, and exploring the numerous practical applications of viewing data.

INTRODUCTION

Like the join operation, the SQL view is a standout feature of the relational model. Views behave in many ways as a table, even though they are not a table. A table stores data while a view contains a set of instructions. With this in mind, views cannot be updated as tables can. Consequently, views are considered "read-only".

An internal table is constructed whenever a view is referenced by another SQL statement, DATA step, or PROC step. Since a view always retrieves data from the most current table(s), the view doesn't suffer from having out-of-date information. Once the internal table is constructed, it is then processed by the SQL procedure.

That's exactly what makes a view so indispensable as a data management tool. Because views are used to display data from one or more underlying base tables and not to store data, views are not considered to be "real" tables. Since they do not exist as independent entities as SQL tables do, they can be created without the fear of data redundancy. Consequently, as you have probably already guessed, data storage requirements aren't affected either.

VIEW ADVANTAGES - WHY USE THEM

There are many reasons for constructing and using views. A few of the more common are presented here.

① **Minimize, or perhaps eliminate, the need to know the table or tables underlying structure.** Often a great degree of knowledge is required to correctly

identify and construct the particular table interactions that are necessary to satisfy a requirement. When this prerequisite knowledge is not present, a view becomes a very attractive alternative. Once the view is constructed, user personnel simply execute the view resulting in the baseline tables to be correctly processed. Consequently, data integrity and control is maintained since a common set of instructions (view) are used by all.

② **Reduce the amount of typing for longer requests.** Often, a query will involve many lines of instruction combined with logical and comparison operators. When this occurs, there are any number of places where a typographical error or inadvertent use of a comparison operator may present an incorrect picture of your data. The construction of views are advantageous in these circumstances, since a simple call to a view virtually eliminates the problems resulting from a lot of typing.

③ **Knowledge of SQL language syntax is non-existent or lacking among user personnel.** When this is the case, user personnel need only execute the desired view using simple calls (or select choices from a menu).

④ **Provide security to sensitive parts of a table.** Security measures can be realized by designing and constructing views designating what pieces of a table's information is available for viewing. Since data should always be protected from unauthorized use, views can provide some level of protection (one should also consider and use security measures at the operating system level).

⑤ **Change and customization independence.** Occasionally, table and/or process changes may be necessary. When this happens, it is advantageous to make it a painless process for user personnel. When properly designed and constructed, a view can be modified without the slightest hint or impact to user personnel, with the one exception that results and/or output may appear differently. Consequently, change independence can be directly influenced by using views.

TYPES OF VIEWS

Views can be typed or categorized according to their purpose and construction method. Joe Celko has this to say about views, *"Views can be classified by the type of SELECT statement they use and the purpose they are meant to serve."* To classify views in a SAS System environment, one looks at how the SELECT statement is constructed. The following classifications are useful when describing a view's capabilities.

① Single-Table Views

Views constructed from a single table are often used to control or limit what is accessible from that table. These views generally limit what columns, rows, and/or both are viewed.

② Ordered Views

Views constructed with an ORDER BY clause arrange one or more rows of data in some desired way.

③ Grouped Views

Views constructed with a GROUP BY clause divide a table into sets for conducting data analysis. Grouped views are more often than not used in conjunction with aggregate functions (see aggregate views below).

④ Distinct Views

Views constructed with the DISTINCT keyword tell the SAS System how to handle duplicate rows in a table.

⑤ Aggregate Views

Views constructed using aggregate and statistical functions tell the SAS System what rows in a table you want summary values for.

⑥ Joined-Table Views

Views constructed from a join on two or more tables use a connecting column to match or compare values. Consequently, data can be retrieved and manipulated to assist in data analysis.

⑦ Nested Views

Views can be constructed from other views, although extreme care should be taken to build views from base tables.

CREATING VIEWS

When creating a view, its name must be unique and follow SAS naming conventions. Also, the view cannot reference itself since it doesn't already exist.

The following example illustrates the process of creating an SQL view.

```
PROC SQL;
  CREATE VIEW PERM.COLLGRAD AS
  SELECT LASTNAME, SSN, DOB
  FROM PERM.EMPLOYEE
  WHERE EDUC > 16
  ORDER BY LASTNAME;
QUIT;
```

In this example the CREATE VIEW statement tells the SAS System that a view is to be created using the instructions and conditions specified in the SELECT statement. The resulting view for all intensive purposes looks and behaves like a real table. Although in this case, something similar to a temporary internal table is created.

VIEWING THE VIEW CONTENTS

The DESCRIBE statement is used to describe the view in greater detail. The information is directed to the SAS System Log. The following example illustrates how the statement is specified.

```
PROC SQL;
  DESCRIBE VIEW PERM.COLLGRAD;
QUIT;
```

In the next illustration, the result from executing the DESCRIBE statement is presented.

NOTE: View PERM.COLLGRAD is defined as:

```
SELECT LASTNAME, SSN, DOB
FROM PERM.COLLGRAD
WHERE EDUC > 16
ORDER BY LASTNAME;
```

USING VIEWS

Using a view is as easy as using a table, since view names and table names can be used in the SELECT

statement as well as many procedures. The only exception to using a view in other steps would be where an in-line view were created (in-line views are discussed in a later section). The following example illustrates how a view can be used as if it were a SAS dataset.

```
PROC SQL;
  TITLE1 'Using View to Print Graduates';
  SELECT *
  FROM PERM.COLLGRAD;
QUIT;
```

Views can act as input SAS System datasets in all SAS procedures. Please keep in mind though, that its the underlying view that dictates what is displayed as output, not the dataset. The following example illustrates using the view called PERM.COLLGRAD in the PRINT procedure.

```
PROC PRINT DATA=PERM.COLLGRAD
  NOOBS N;
  TITLE1 'Using View to Print Graduates';
RUN;
```

The next example illustrates using the same view in the MEANS procedure.

```
PROC MEANS DATA=PERM.COLLGRAD;
  TITLE1 'Using a View for Statistics';
RUN;
```

CONSTRUCTING IN-LINE VIEWS

In-line views are created as a query expression nested in a FROM clause. By nesting the view expression within the FROM clause of a query, processing can be performed in a single step. In-line views are not assigned names as views created with the CREATE VIEW statement. They can only be referenced in the query in which it is defined. Consequently, they cannot be referenced as tables in other queries or SAS procedures.

In the following in-line view example, employees that are college graduates with more than ten days of vacation are listed.

```
PROC SQL;
  TITLE1 'In-Line View Example';
  SELECT LASTNAME, NUMVAC
  FROM
    (SELECT LASTNAME, SSN, DOB,
      SUM(VAC) AS NUMVAC
    FROM PERM.EMPLOYEE
    WHERE EDUC > 16
    GROUP BY LASTNAME)
  WHERE NUMVAC > 10
  ORDER BY LASTNAME;
QUIT;
```

In the previous in-line example, the inner query is processed first. An intermediate virtual table is created from the view's SELECT statement, by using the WHERE clause expression to evaluate each row and select any rows that match the condition.

The SUM function in the inner SELECT statement adds values in the VAC (days of vacation) column for each group identified in the GROUP BY clause.

The outer SELECT statement, using the internal table, evaluates each row with the WHERE expression. Then the desired columns are selected. Finally, the rows of data are ordered by the LASTNAME column. The final result is then displayed.

Vacation Days for Graduates

<u>LASTNAME</u>	<u>NUMVAC</u>
Adams	13
Jones	11
Kingman	16
Thompson	14

DROPPING VIEWS OUT OF VIEW

For views that are no longer wanted, the DROP VIEW statement is used. After a view is deleted, all references to that view in other queries, views, and statements will also have to be deleted. The following example illustrates dropping PERM.COLLGRAD.

```
PROC SQL;
  DROP VIEW PERM.COLLGRAD;
QUIT;
```

CONCLUSION

The concept of a view is a relatively easy one to understand. By constructing picture frames or windows, data becomes visible for someone to see. Although views behave in many ways as a table, they contain nothing more than a set of directions to the underlying base table or tables.

It is fair to say that views offer a great deal of flexibility for programmers and users alike. Views can be designed to look as if custom databases were constructed, but without the typical problems associated with data redundancy and storage overhead. So try to examine each of your applications to see how views can be constructed to offer greater flexibility for data analysis and manipulation. And don't be surprised if you find that the database you thought you were using turns out to be a view.

ACKNOWLEDGMENTS

The author would like to thank the SESUG Leadership, especially Melissa Garreans, John Alden Life Insurance Company and Andy Parks for asking me to participate at SESUG and for their support and encouragement during the development of this paper.

REFERENCES

Bowman, Judith S., Sandra L. Emerson, and Marcy Darnovsky (1993). *The Practical SQL Handbook: Using Structured Query Language*, Second Edition. Addison-Wesley Publishing Company, pps. 269-291.

Celko, Joe (1995). *Joe Celko's SQL for Smarties: Advanced SQL Programming*. The Morgan Kaufmann Series in Data Management Systems Series. Morgan Kaufmann Publishers, Inc., pps. 207-225.

Date, C.J. (1993). *A Guide to the SQL Standard*, Third Edition. Addison-Wesley Publishing Company, pps. 167-180.

Lafler, Kirk Paul (1991), "Using the SQL Procedure", *Proceedings of the Sixteenth Annual SAS Users Group International Conference*, pps. 815-818.

Lafler, Kirk Paul (1993), "Using the SQL Procedure", *Proceedings of the Eighteenth Annual SAS Users Group International Conference*, pps. 430-439.

Lorie, Raymond A. and Jean-Jacques Daudenarde (1991). *SQL and Its Applications*. Englewood Cliffs, NJ: Prentice-Hall, pps. 144-153.

Norton, Andrew A. (1994), "The View Review: PROC SQL Views, SAS/ACCESS Views, and DATA-Step Views", *Proceedings of the Nineteenth Annual SAS Users Group International Conference*, pps. 1434-1439.

Rabb, Merry G. (1991), *Using the SQL Procedure in Data Management Applications*, *Proceedings of the Sixteenth Annual SAS Users Group International Conference*, pps. 600-604.

AUTHOR'S BIOGRAPHY

Kirk Paul Lafler is senior consultant and founder of Software Intelligence Corporation with eighteen years experience using the SAS System. His areas of expertise include information-based systems analysis, design and development, computer training, and programming using base-SAS, SAS/SQL, SAS/FSP, SAS/AF, Screen Control Language, and SAS/EIS software.

The author welcomes comments and suggestions. He can be reached at:

Kirk Paul Lafler
Software Intelligence Corporation
P.O. Box 1390
Spring Valley, California 91979-1390
(619) 670-SOFT or (619) 670-7638
Compuserve: 72642,226