Differences and Interrelationships between the SQL Procedure and SAS/ACCESS® Software

Mark E. Stranieri, SAS Institute Inc., Cary, NC

SECTION I: SAS/ACCESS VS PROC SQL

SAS/ACCESS Defined

SAS/ACCESS software is many different SAS products, each providing an interface to a different database system. For example, the SAS/ACCESS Interface to DB2 allows the user to view, extract, manipulate and load data to and from a DB2 database. Similarly, the SAS/ACCESS Interface to ORACLE provides an interface to an ORACLE database. SAS/ACCESS software will always include the ACCESS procedure. In addition, many of the SAS/ACCESS interfaces include the DBLOAD procedure. The ACCESS procedure includes a series of interactive windows and pull-down menus that are used to create view descriptors. View descriptors are a type of SAS data set that view or look at data contained in other vendor database tables (third-party tables, see Fig. 1). Each time a view descriptor is referenced, it provides a real-time interface to the third-party database. Therefore, any changes made to third-party tables are immediately visible when using a view descriptor of that table. For example, if a third-party table called EMP is updated every time a new employee is hired, a view descriptor of that table will automatically reflect the updates. In addition, PROC ACCESS allows for the extraction of data from third-party tables to native SAS datasets. PROC ACCESS also provides several other secondary functions that vary, depending on the specific SAS/ACCESS interface.

A second procedure that is included with many SAS/ACCESS interface products is PROC DBLOAD. PROC DBLOAD allows users to create third-party database tables which duplicate or subset the structure and data of a SAS data set, PROC SQL view, or another view descriptor (see Section 2 for examples). PROC DBLOAD can be run either interactively in SAS display manager or as a SAS batch program. The DBLOAD procedure automatically converts SAS data types into third-party data types, just as PROC ACCESS performs the reciprocal conversions. The DBLOAD procedure may also allow SAS users to issue other SOL commands to the third-party database. For example, using the SAS/ACCESS Interface to ORACLE, you can easily issue the following types of SOL commands:

- CREATE and DROP database objects (tables, views, indexes, etc.)
- ALTER tables
- UPDATE tables and views
- INSERT into tables and views
- GRANT and REVOKE privileges to database objects
- COMMIT and ROLLBACK
- SAVEPOINT

The SELECT statement is absent from the above list because queries to a third-party database can be issued by referencing view descriptors in SAS procedures. Two procedures, PROC APPEND and PROC SQL, allow data in third-party database tables to be updated by referencing view descriptors. Any data contained in the view descriptors can be treated just like SAS data sets by most SAS procedures (see Table 1).

PROC ACCESS in combination with PROC DBLOAD allows users to define, manipulate, control and query third-party database data (and database objects) within a SAS program.

In summary, SAS/ACCESS software offers a full-featured and trans-
parent interface to other vendor database products.

PROC SQL Defined
PROC SQL, which is part of base SAS software, is the SAS System's implementation of structured query language, or SQL (pronounced "sequel" in this paper). SQL was developed by IBM® and has since been adopted by the American National Standards Institute (ANSI) as the standard language for relational database management systems (RDBMS). SQL is a comprehensive language that is intended for use as both a database management tool and a user application tool. SQL is a non-procedural language, and therefore does not operate on single data records, but rather it allows for group processing. Therefore, many types of applications are much faster in SQL, and involve less program code than conventional record-by-record processing. PROC SQL is particularly useful when merging two or more data sets on a key value (JOIN). PROC SQL can be used as a stand-alone language to create, update, insert, and delete SAS data sets and data. However, it is best used in combination with the entire SAS language. All SAS functions and many SAS SQL enhancements are available under PROC SQL. Likewise, any database entity created by PROC SQL is available for use in other SAS procedures, including tables, indexes and views. As mentioned in the Introduction, are SAS data sets. Indexes can be created on tables to improve the performance of programs that use those tables. PROC SQL views are a type of data set that view or retrieve data contained in or derived from other tables, views, and view descriptors. These views do not contain actual data. They contain a SQL SELECT statement that is executed whenever the view is used. Therefore, any data added, changed or deleted from a view's underlying table(s) will automatically be reflected in that view. A SELECT statement performs simple or complex queries on one or more tables. For example, the following SELECT statement queries the ALLEMP table for the first, middle, and last names of all employees that are in department A.

```
PROC SQL;
SELECT FNAME, INI, LNAME FROM ALLEMP WHERE DEPNO = 'A';
```

To create a view of this query, simply add a CREATE VIEW statement as follows.

```
PROC SQL;
CREATE VIEW A_EMP AS SELECT FNAME, INI, LNAME FROM ALLEMP WHERE DEPNO = 'A';
```

PROC SQL queries may also be complex, and involve multi-table joins, correlations, multiple levels of subqueries, summary functions, SAS functions and SAS macros. For example, the ALLEMP table in the previous example could be a view that combines and summarizes information from other tables, views and view descriptors using a SQL JOIN statement (see Section 2 for example programs). In summary, PROC SQL is the SAS implementation of a database language that can be used alone, or in conjunction with native SAS statements and procedures.

Similarities between SAS/ACCESS Views and PROC SQL Views

Conceptual Similarities

After reviewing the definitions presented previously it may appear that there are few similarities between PROC ACCESS view descriptors and PROC SQL views. It is true that in many respects PROC SQL views are vastly different from view descriptors; however, there is one overriding conceptual similarity between the two types of views, they are both SAS data views (see Figure 1). In addition, there are several other important similarities between the two.

Figure 1 illustrates that PROC ACCESS view descriptors and PROC SQL views are members of a single type of SAS data set, the SAS data view. The definition of a SAS data view is simply that views do not contain data. More precisely, a data view contains information that describes, or points to data contained elsewhere. This definition is true for view descriptors and PROC SQL views and will continue to be true as new view types are added to the SAS System. This concept of having a virtual table available for use in the SAS System can be carried to an extreme, since SQL views can select data from other data views. In addition, many third-party databases, such as DB2 allow similar views. For example, a PROC SQL view called LOCALEMP might point to a subset or superset of the data contained in the SAS EMPLOYEE table. Moreover, a PROC ACCESS view descriptor called FORGNEMP might point to a DB2 table called FOREIGN EMPLOYEES. A second SQL view called ALLEMP might point to a JOIN of the SQL view LOCALEMP with the ACCESS view descriptor FOREIGN_EMPLOYEE, thus pointing to both a SAS data set and a third-party table simultaneously (see Fig. 2). Since both DB2 and the SAS System allow views of views, a very deep view is possible but probably not desirable.

The PROC SQL view ALLEMP displayed in Figure 2 does not contain any data observations, but rather stores a SQL SELECT statement that joins data from the views specified. Likewise, those views contain a stored query to a SAS data set and to a DB2 table. If the following SAS statement were then issued:

```
PROC PRINT DATA=ALLEMP; RUN;
```

the output would display data from the third-party table (FOREIGN_EMPLOYEE) joined with data from the SQL view LOCALEMP. If changes were made to the underlying tables, the views would reflect those changes since each time a view is used, it re-accesses the underlying tables. This type of JOIN, and others are detailed in Section 2.

Functional Similarities of SAS Data Views

Since both the PROC ACCESS view descriptor, and the PROC SQL view fall under the domain of SAS data views, there are many functions and attributes common to both. Here are some examples:

- SAS procedures can use data views as input data sets (with some restrictions)
- SAS labels, formats and informats can be used on view columns
- SAS functions can be used on views
- Views are transparent to the user (they appear to be SAS data sets)
- Views can be merged and joined with other views
- Views can subset their underlying table(s)
- Views automatically reflect changes to their underlying tables
- The extension .VEW is used for all data views in a SAS library

In addition, many of the third-party databases that SAS/ACCESS software interfaces with use SQL as their primary database language. The SAS System allows the use of SQL as a database language with PROC SQL. Therefore, SQL can be used on ordinary SAS data sets, on PROC ACCESS view descriptors, on PROC SQL views and by many third-party databases on their own tables and views.

Differences between SAS/ACCESS Views and PROC SQL Views

Once you have become familiar with both PROC ACCESS view descriptors and PROC SQL views, many differences become appar-
ent. Although there are many differences between the two types of views, there are five which characterize the views as distinct data objects. Those five differences follow.

1. A PROC ACCESS view descriptor interfaces with another vendor tables: a SQL view selects data from SAS data sets and data views. A PROC SQL view (and other SAS procedures) may look at third-party database objects only through view descriptors.

2. A PROC ACCESS view descriptor looks at data contained in a single third-party database object, a PROC SQL view looks at data contained in one or more SAS System data objects.

3. PROC ACCESS is currently an interactive procedure, and view descriptors are created interactively. PROC SQL views can be created interactively using SAS display manager, or in batch mode using a SAS program. Once a PROC ACCESS view descriptor has been created it can be used in a SAS program. It is the goal of SAS Institute to provide for batch view descriptor creation in Release 6.07 of the SAS System for many SAS/ACCESS interfaces.

4. Third-party database tables may have data added, updated and deleted through a PROC ACCESS view descriptor. In Release 6.06 of the SAS System Proc SQL views are read-only entities. Ironically, this difference allows PROC SQL INSERT, UPDATE and DELETE statements to be used on PROC ACCESS view descriptors, but not on PROC SQL views.

5. SQL views take full advantage of SAS functions and operators, as well as SQL functions and operators. Therefore a SQL view is very robust, and can involve JOIN statements, group functions, unions, intersections, and character and number functions. A PROC ACCESS view descriptor is limited by the third-party database object that it interfaces with. However, view descriptors can subset the number of columns viewed, and can use the Selection Criteria Window to conditionize the data returned. If you are familiar with the third-party database, you can use native database statements to perform data, and data structure operations before a view descriptor is created. If available, PROC DBLOAD can be used to write and execute native database statements (see Section 2 for examples).

SECTION II: METHODS OF USING DATA VIEWS

This section primarily consists of PROC ACCESS, PROC DBLOAD and PROC SQL programs that demonstrate the wide variety of uses for these procedures individually and together. Example programs included in this section demonstrate how to combine and move data between different databases.

Creating Multi-Database Views

It is likely that many user sites have data contained in several databases. It follows that a multi-database site would have a need to join or merge the data from several of these databases to perform a single analysis or create a single analysis or report.

The following example SAS programs demonstrate how to construct PROC SQL views that logically join or merge data from different databases.

Joining SAS Tables with Third-Party Database Tables

Before constructing the final PROC SQL view it is necessary to define the entities that will be involved. First a SAS table will be created, then a PROC ACCESS view descriptor. It is beyond the scope of this paper to fully explain DATA Step and SAS Procedure language conventions. For more information on DATA Step and SAS procedure conventions see References, at the end of this paper.

The following data step creates a SAS table called LOCALEMP. The SAS file ref EMPDATA points to a flat file containing data as specified in the input statement. SASLIB is a SAS library used exclusively for SAS tables and their corresponding indexes.

```
DATA SASLIB.LOCALEMP;
FILENAME EMPDATA "C:\EMPDATA\EMPS.DAT";
INPUT FNAME $1-15 INI $16-17 LNAME $18-32 EMPNO $33-43
DEPNO $44-45 POSITION $46-60 LEV $61-63 STARTDATE $64-66;
FORMAT FNAME $15. INI $2. LNAME $15. EMPNO $11. DEPNO $2.
POSITION $15. LEV $3. STARTDATE $6.
RUN;
```

Next, presuppose a third-party table contained in an ORACLE database that is similar in structure to LOCALEMP, but contains data of employees at a different site, called NYCEMP. Assume the following CREATE statement was used to create the ORACLE table.

```
CREATE TABLE NYCEMP
  (FNAME CHAR(15), INI CHAR(2), LNAME CHAR(15),
   EMPNO CHAR(11), DEPNO CHAR(2), POSITION CHAR(15),
   LEV CHAR(3), STARTDATE DATE);
```

Note that the third-party database does not have to be ORACLE, or even SQL based.

A PROC ACCESS view descriptor also called NYCEMP is now created interactively to interface with the ORACLE table. The following box inset summarizes the steps used to create the view descriptor. For further information on creating view descriptors refer to the appropriate SAS/ACCESS software interface guide.

```
CREATING A VIEW DESCRIPTOR INTERACTIVELY

1. Issue the following from SAS display manager, or in line mode: ORALIS is a SAS library used exclusively for ORACLE
   PROC ACCESS AD=ORALIS.NYCEMP
   FUNCTION=C;
   RUN;

2. Select the correct third-party engine (ORACLE) if your site
   has more than one SAS/ACCESS engine

3. Enter Table name, User name, and Password in the Access
   Descriptor Identification Window (Table=NYCEMP).

4. Save/end the Access Descriptor Display Window with all
   columns selected.

5. End the Access Descriptor Identification Window.

6. Issue the following from the SAS program editor or in line mode:
   PROC ACCESS AD=ORALIS.NYCEMP;
   RUN;

7. Enter ORALIS and NYCEMP for the view descriptor Library
   and Member names respectively.

8. Select all columns

9. Save/end the View Descriptor Display Window

```

```
```
A SAS table and a SAS view descriptor have now been created. The
table, SASLIB.LOCALEMP contains employee data from the local
company site. The view descriptor, ORAU8.NYCEMP interfaces
with the third-party table also called NYCEMP which contains em-
ployee data from the New York City office.

Now suppose you need a report once a month of all employees that
work in New York City, and locally that have been working for the
company five years or longer, grouped by department. Since this is
a fairly complex query that relies on current data, a PROC SQL
view is a good choice for the report. The following SAS pro-
gram creates a PROC SQL view and stores it in the SAS library VEWU8.

PROC PRINT is then used to output the view.

PROC SQL;
CREATE VIEW VEWU8.FIVYREMP AS
SELECT L.FNAME || ' ' || L.LNAME AS NAME,
L.DEPNO,
L.STARTDAT * 86400 LABEL = 'START DATE',
FORMAT = DATETIME7,
ROUND(DATE0 - L.STARTDAT)/365.25 AS YEARS
FROM SASLIB.LOCALEMP AS L;
WHERE ROUND((DATE0 - L.STARTDAT)/365.25) >= 5
UNION
SELECT C.FNAME, C.INI,
C.LNAME || ' ' || C.DEPNO,
C.STARTDAT,
ROUND(DATE0 - C.STARTDAT)/31557600
FROM ORAU8.NYCEMP AS C
WHERE ROUND((DATE0 - C.STARTDAT)/31557600) >= 5
ORDER BY 3;

PROC PRINT DATA = VEWU8.FIVYREMP; RUN;
The above SOL program demonstrates the versatility of PROC SQL
by using a combination of SAS and SQL functions, includ-
ing concatenation symbols ( || ), DATE0, DATETIME0, and ROUND. The
concatenation symbols create a single column and title from the
three name columns (FNAME, INI and LNAME). The program as-
sumes 365.25 days in a year 31,557,600 seconds in a year and
86,400 seconds in a day. A view is created that performs a union
of the SAS table (LOCALEMP) with the ORACLE table (NYCEMP).

It's worth noting that the employees starting date subtracted
from the current date (or data-time) in years is greater than or equal
to 5. This PROC SQL view does not physically contain data, instead,
the union is performed when the view is referenced in a SAS pro-
gram or in a SAS/FS/SP application and can then be treated as a
read-only SAS data set.

Multi-Database Table Joins
To provide an example of joining data between two different third-
party databases a second database, Rdb/VMS, is added to the sce-
nerio. The Rdb/VMS database has another employee table
called CHGEMP that contains data from the Chicago office. This
Rdb/VMS table is similar in structure to NYCEMP but does not con-
tain a one character department code called DEPNO. Instead
it contains the full department name in a 25 character column called
DEPT. Assume the Rdb/VMS CHGEMP table was created with the fol-
loving SQL statement:

CREATE TABLE CHGEMP
(FNAME CHAR(15),
INI CHAR(2),
LNAME CHAR(15),
DEPT CHAR(25),
POSITION CHAR(15),
LEV CHAR(3),
STARTDAT DATE);

In order to change the Rdb/VMS CHGEMP department names into
ORACLE NYCEMP coded names, a translation table that contains
both the 25 character department name and the 2 character code
is needed. The ORACLE database has the DEPNO translation table
that is defined as follows:

CREATE TABLE DEPT
(DEPNAME CHAR(25), DEPNO CHAR(2));

To create a view that joins the Rdb/VMS CHGEMP table with the
ORACLE DEPT table in order to substitute the full department name
DEPNAME with the department code DEPNO, perform the following
two steps:

1. Create a PROC ACCESS view descriptor for each of the tables
involved in the JOIN using the method summarized in the box
inset. Create one view descriptor called RDBUB.CHGEMP for the
Rdb/VMS CHGEMP table, and one view descriptor called
ORALIB.DEP for the ORACLE DEPT translation table.

PROC SQL;
CREATE VIEW VEWU8.CHGEMP AS
SELECT C.FNAME, C.INI,
C.LNAME, C.DEPNO,
C.STARTDAT,
ROUND(DATE0 - C.STARTDAT)/31557600
FROM RDBLIB.CHGEMP AS C,
ORALIB.DEP AS D
WHERE UPCASE(C.DEPT) = UPCASE(D.DEPNO);

Before continuing it may be useful to review Table 2 which summa-
izes all database objects that have been used or created in exam-
amples.

SAS Programs That Manipulate Third-Party
Databases and View Them
Manipulating third-party database objects is very straightforward
provided you are familiar with the third-party database language
and that the SAS/ACCESS interface being used includes the PROC
DBLOAD SOL option. The DBLOAD SOL option and its interactive
counterpart, the Query Entry window, is included with many of the
SAS/ACCESS interfaces, including but not limited to the SAS/
ACCESS interfaces to DB2, SOL/DS, ORACLE, and Database Man-
ger. The following batch SAS programs demonstrate the use of
the DBLOAD SOL option. Identical third-party database programs
can be submitted interactively from the PROC DBLOAD Query Entry
window.

Creating Third-Party Data Objects
Create a database table called ALLEMP that is a copy of NYCEMP.
The ALLEMP table will be used combine the data from the three
existing employee tables SASLIB.LOCALEMP, ORALIB.NYCEMP,
and RDBLIB.CHGEMP.

PROC DBLOAD DBMS = ORACLE;
USER = username;
PASSWORD = password;
SQL CREATE TABLE ALLEMP AS
SELECT * FROM NYCEMP;
RUN;

Altering and Inserting Data Into Third-Party Data Objects
Before copying data from the other tables, column CITY must be
added so that each row will be associated with the correct city.
Since only data from NYC is currently in the table, an update of
every row to have the value "NYC" in the newly created CITY column will correctly identify each row. The following program first adds the column CITY, then updates each row giving CITY a value of NYC.

```sas
PROC DBLOAD DBMS=ORACLE;
  USER=username;
  PASSWORD=password;
  SQL ALTER TABLE ALLEMP
    ADD CITY CHAR(20);
  SQL UPDATE ALLEMP
    SET CITY='NYC';
RUN;
```

Comparing Third-Party Tables

The data from the New York City table is now contained in two ORACLE tables. Before deleting the old table, compare the data between the two to assure there are no differences. To compare the two tables, do the following:

- Create a view descriptor of ALLEMP in the ORALIB SAS library.
- Deselect the new CITY column.
- Run the following PROC COMPARE program:

```sas
PROC COMPARE BASE=ORALIB.NYCEMP
  COMPARE=ORALIB ALLEMP;
RUN;
```

- Re-select the CITY column in the ORALIB ALLEMP view descriptor.

After checking the SAS output to be sure there are no differences, delete the ORAUB.NYCEMP view descriptor and the ORACLE NYCEMP table as follows:

```sas
PROC SOL;
  DROP VIEW ORALIB.NYCEMP;
  PROC DBLOAD DBMS=ORACLE;
  USER=username;
  PASSWORD=password;
  SQL DROP TABLE NYCEMP;
RUN;
```

Moving Data between Databases

This section demonstrates copying data from one third-party database to another.

The last task that must be performed to create a single table containing the data from all three databases is to move the data from the SAS LOCALEMP table and the RDB/VMS CHGEMP table into the ORACLE ALLEMP table.

```
PROC APPEND and PROC SQL can both be used to add data to third-party tables. In order to demonstrate both procedures, PROC APPEND will be used to move the data from LOCALEMP and PROC SQL will be used to move the data from CHGEMP.
```

First, append all the SAS local data in LOCALEMP to the ORACLE table ALLEMP.

```
PROC APPEND BASE=ORALIB LOCALEMP
  DATA=SASLIB ALLEMP;
RUN;
```

Next, set the CITY column equal to "ATL" (the local city is Atlanta) wherever the CITY column is currently NULL (or MISSING)

```
PROC SOL;
  UPDATE ORALIB ALLEMP
    SET CITY='ATL'
    WHERE CITY IS NULL;
RUN;
```

The next example uses PROC SQL to copy the RDB/VMS CHGEMP table and insert a value for the new CITY column in a single step.

```
PROC SOL;
  INSERT INTO ORALIB ALLEMP
    (FNAME,INI,
    UNAME,EMPNO,DEPT,
    POSITION,LEV,STARTDAT,
    CITY)
  SELECT FNAME,INI,UNAME,EMPNO,DEPT,
    POSITION,LEV,STARTDAT,
    'CHGEMP'
  FROM RDB/VMS CHGEMP;
```

Conclusion

SAS/ACCESS software offers a full-featured and transparent interface to other vendor database products. The SAS procedures included with SAS/ACCESS software allow users to define, manipulate, control and query third-party database data and database objects interactively and within batch SAS programs.

PROC SQL is the SAS System implementation of a database language that can be used alone, or in conjunction with the entire SAS language to define and manage SAS databases. Future releases of the SAS System will likely increase the power and efficiency of current methods, however, the concept of SAS data views, and SQL as an integrated database language will remain constant.

References


Digital Equipment Corporation (July 1980) VAX SQL Users Guide

International Business Machines (March 1990) IBM Database 2 Reference, Third Edition


Acknowledgements

Thanks to John West for help with illustrations.

Thanks to Curtis Yao, John West and Kathy Conard for technical review.

SAS, SAS/ACCESS, SAS/PS, and SYSTEM 2000 are registered trademarks of SAS Institute Inc, Cary, NC, USA.

ADABAS is a registered trademark of Software AG.

CA-DATACOM/DB is a registered trademark of Computer Associates.

dBASE is a registered trademark of Ashton-Tate Corporation.

DB2, SQL/DS and IBM are registered trademarks of International Business Machines.

INGRES is a registered trademark of Behavioral Technology Inc.

Lotus is a registered trademark of Lotus Development Corporation.

ORACLE is a registered trademark of Oracle Corporation.

Rdb/VMS is a registered trademark of Digital Equipment Corporation.
### Table 1 - Procedure Functions on Database Objects

<table>
<thead>
<tr>
<th>Function Code Definitions:</th>
<th>PROC SQL view</th>
<th>PROC ACCESS view</th>
<th>3rd Party tables</th>
<th>SAS tables (data sets)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C - Create (view or table)</td>
<td>DR,RC,O</td>
<td>C,DR,EC,RC,O</td>
<td>DR,EC,C,A,U,D,O</td>
<td>R</td>
</tr>
<tr>
<td>D - Delete data</td>
<td>RC</td>
<td>RC</td>
<td>AA,RA</td>
<td>A,R</td>
</tr>
<tr>
<td>O - Other operations</td>
<td>O</td>
<td>O</td>
<td>RA,AA,UA,DA</td>
<td>C,R,A,U,D,O,RS</td>
</tr>
<tr>
<td>R - Read data</td>
<td>RC</td>
<td>RC</td>
<td>RC,DR,O</td>
<td>RA</td>
</tr>
<tr>
<td>U - Update data</td>
<td>RC</td>
<td>RC,EC</td>
<td>RC,DR,O</td>
<td>RA,OA,U,C,RS,O</td>
</tr>
</tbody>
</table>

### Function Code Definitions:

- **A** - Add data
- **C** - Create (view or table)
- **D** - Delete data
- **O** - Other operations
- **R** - Read data
- **U** - Update data

### Table 2 - Database Structures Used and Created in Examples

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDBLIB</td>
<td>SAS Library</td>
<td>Contains PROC ACCESS view and access files for Rdb/VMS tables</td>
</tr>
<tr>
<td>ORALIB</td>
<td>SAS Library</td>
<td>Contains PROC ACCESS view and access files for ORACLE tables</td>
</tr>
<tr>
<td>VEWLIB</td>
<td>SAS Library</td>
<td>Contains PROC SQL views</td>
</tr>
<tr>
<td>SASLIB</td>
<td>SAS Library</td>
<td>Contains SAS tables and corresponding indexes</td>
</tr>
<tr>
<td>SASLIB.LOCALEMP</td>
<td>SAS table</td>
<td>Contains local site (Atlanta) employee data</td>
</tr>
<tr>
<td>RDBLIB.CHGEMP</td>
<td>View descriptor</td>
<td>Interfaces with the Rdb/VMS CHGEMP table</td>
</tr>
<tr>
<td>CHGEMP</td>
<td>Rdb/VMS table</td>
<td>Contains Chicago site employee data</td>
</tr>
<tr>
<td>ORALIB.NYCEMP</td>
<td>View descriptor</td>
<td>Interfaces with the ORACLE NYCEMP table</td>
</tr>
<tr>
<td>NYCEMP</td>
<td>ORACLE table</td>
<td>Contains New York City site employee data</td>
</tr>
<tr>
<td>ORALIB.DEPT</td>
<td>View descriptor</td>
<td>Interfaces with the ORACLE DEPT table</td>
</tr>
<tr>
<td>DEPT</td>
<td>ORACLE table</td>
<td>Translation table for department codes</td>
</tr>
<tr>
<td>EMPDATA</td>
<td>SAS file</td>
<td>References an input file for SASLIB.LOCALEMP</td>
</tr>
<tr>
<td>VEWLIB.FIVYREMP</td>
<td>PROC SOL view</td>
<td>Combines data from the SAS table SASLIB.LOCALEMP with the ORACLE table NYCEMP using ORALIB.NYCEMP</td>
</tr>
<tr>
<td>VEWLIB.CHGEMP</td>
<td>PROC SOL view</td>
<td>Joins the ORACLE table DEPT with the Rdb/VMS table CHGEMP using ORALIB.DEPT and RDBLIB.CHGEMP respectively</td>
</tr>
</tbody>
</table>

559
Figure 1: Diagram of SAS Data Sets

Figure 2: Diagram of a SQL view