CHAPTER

14

Using DBMS Data with SAS/ ACCESS Version 6 Procedures

Introduction

This topic presents examples of accessing and updating DBMS data through Version 6 view descriptors. In Version 7 and later, there are several changes in the ways that the SAS/ACCESS procedures work. For more information, see “Version 8 Compatibility for Version 6 Procedures” on page 99.

Note: It is recommended that you use the new SAS/ACCESS LIBNAME statement to access your DBMS data more easily and directly and to take full advantage of Version8 enhancements. See “SAS/ACCESS LIBNAME Statement” on page 27 for more information about the SAS/ACCESS LIBNAME statement. △

For information about using view descriptors efficiently in SAS programs, see “Performance and Efficient View Descriptors” on page 125.

Note: See your DBMS chapter to determine whether the ACCESS and DBLOAD procedures are available for your DBMS. △

Tips for Running ACCESS and DBLOAD Procedure Examples

As you work through the examples, notice that the descriptors can be created in several ways. In some cases, the ASSIGN=YES statement is specified and SAS variable names and formats are assigned when the access descriptor is created. In other cases, the ASSIGN statement is omitted and editing statements, such as RENAME and UNIQUE, are specified when the view descriptors are created. How you create descriptors depends on your site's needs and practices.

When you run the examples, you need only to create an access descriptor or a view descriptor once per example. If you rerun the examples, you do not need to re-create the descriptors.

Reviewing Variables

Before retrieving or updating DBMS data that is described by a view descriptor, you might want to review the attributes of the data's variables. You can use the CONTENTS or DATASETS procedure to display a view descriptor's variable and format information. You can use these procedures with view descriptors in the same way that you use them with other SAS data sets.

This example uses the DATASETS procedure to display information about the view descriptor VLIB.USACUST, which describes the data in the DB2 table SASDEMO.CUSTOMERS.

```
options linesize=80;

proc access dbms=db2;
/* create access descriptor  */
   create adlib.customr.access;
   table=sasdemo.customers;
   ssid=db2;
   assign=yes;
   rename customer=custnum
          firstorder=firstord;
   format firstorder date9.;
   list all;

/* create vlib.usacust view */
   create vlib.usacust.view;
   select customer state zipcode name
          firstorder;
   subset where customer like '1%';
run;

/* example             */
proc datasets library=vlib memtype=view;
   contents data=usacust;
run;
```

Output 14.1 on page 186 shows the results of this example.

Output 14.1 Using the DATASETS Procedure with a View Descriptor

```
                              DATASETS PROCEDURE

        Data Set Name: VLIB.USACUST          Observations:          .
        Member Type:   VIEW                  Variables:             5
        Engine:        DB2                   Indexes:               0
        Created:       .                     Observation Length:    0
        Last Modified: .                     Deleted Observations:  0
        Protection:                          Compressed:            NO
        Data Set Type:                       Sorted:                NO
        Label:

            -----Alphabetic List of Variables and Attributes-----

    #    Variable    Type    Len    Pos    Format    Informat    Label
    ------------------------------------------------------------------------
    1    CUSTNUM     Char      8      0    $8.       $8.         CUSTOMER
    5    FIRSTORD    Num       8     88    DATE9.    DATE9.      FIRSTORDER
    4    NAME        Char     60     24    $60.      $60.        NAME
    2    STATE       Char      2      8    $2.       $2.         STATE
    3    ZIPCODE     Char      5     16    $5.       $5.         ZIPCODE
```

In the DATASETS procedure output, the VLIB.USACUST view descriptor has five variables: CUSTNUM, FIRSTORD, NAME, STATE, and ZIPCODE. The variables are listed in alphabetic order, and the # column in the listing shows the order of each variable in VLIB.USACUST. The **`Label`** field in the DATASETS procedure lists the complete names of the DBMS columns. Note that descriptors do not support SAS names longer than eight characters.

The information displayed by the DATASETS procedure does not include any selection criteria that might be specified for the view descriptor. To see selection criteria, you must review the code that created the view descriptor.

You can use the ACCESS procedure's UPDATE statement to change the attributes of a view descriptor. See "ACCESS Procedure Syntax" on page 104 for more information.

Printing Data

The DATASETS procedure shows you the SAS variables and their attributes, but it does not display the data values. You can use the PRINT procedure to print all or some of the data values.

In this example, you use the PRINT procedure to print data that is described by the VLIB.CUSORDR view descriptor; VLIB.CUSORDR gets its data from the table SASDEMO.ORDERS.

Note: The OBS= data set option limits the output to five observations. △

```
proc access dbms=db2;
/* create access descriptor    */
   create adlib.order.access;
   table=sasdemo.orders;
   ssid=db2;
   assign=no;
   list all;

/* create vlib.cusordr view  */
   create vlib.cusordr.view;
```

```
   select ordernum stocknum shipto;
   rename ordernum ordnum;
   format ordernum 5.0
          stocknum 4.0;
run;

/* example             */
proc print data=vlib.cusordr(obs=5);
title 'Five Observations Described by VLIB.CUSORDR';
run;
```

Output 14.2 on page 188 shows the results of this example.

Output 14.2 Results of Using the OBS= Option

```
        Five Observations Described by VLIB.CUSORDR

         OBS    ORDERNUM    STOCKNUM    SHIPTO

           1     11269         9870      19876078
           2     11270         1279      39045213
           3     11271         8934      18543489
           4     11272         3478      29834248
           5     11273         2567      19783482
```

In addition to the OBS= option, the SAS system option FIRSTOBS= also works with view descriptors. However, because of the way data is stored in a relational DBMS table, there is no true first observation, and the FIRSTOBS= option might not consistently retrieve the same observation. In addition, the FIRSTOBS= option might not improve performance significantly because each row might still be read and its position must be calculated.

Calculating Statistics

You can also use SAS statistical procedures on DBMS data. This section shows examples using the MEANS and RANK procedures. See Chapter 12, "Using DBMS Data in Version 7 and Version 8," on page 151 for an example of the FREQ procedure.

Using the MEANS Procedure

In your analysis of recent orders, suppose you also want to calculate some statistics for each U.S. customer. From the DB2 table SASDEMO.ORDERS, the view descriptor VLIB.USAORDR selects a subset of observations that have a SHIPTO value beginning with a 1, indicating a U.S. customer. The observations are also ordered by the SHIPTO variable.

The following example generates the means and sums of the length of material ordered (in yards) and the fabric charges (in dollars) for each U.S. customer. Also included are the number of observations (N) and the number of missing values (NMISS). The MAXDEC= option specifies the number of decimal places (0-8) for PROC MEANS to use in printing the results.

```
proc access dbms=db2;
/* create access descriptor    */
```

```
    create adlib.order.access;
    ssid=db2;
    table=sasdemo.invoice;
    assign=yes;
    rename dateorderd = dateord
           processdby = procesby;
    format dateorderd  date9.
           shipped     date9.
           ordernum    5.0
           length      4.0
           stocknum    4.0
           takenby     6.0
           processdby  6.0
           fabcharges  12.2;
    list all;

/* create vlib.usaordr view  */
    create vlib.usaordr.view;
    select ordernum stocknum length
           fabcharges shipto;
    subset where shipto like '1%';
run;

/* example             */
proc means data=vlib.usaordr mean
     sum n nmiss maxdec=0;
   by shipto;
   var length fabcharg;
title 'Data Described by VLIB.USAORDR';
run;
```

Output 14.3 on page 189 shows the output for this example.

Output 14.3 Statistics on Fabric Length and Charges for Each U.S. Customer

```
                Data Described by VLIB.USAORDER


---------------------- SHIPTO=14324742 -----------------------

 Variable  Label               Mean            Sum   N  Nmiss
 ------------------------------------------------------------
 LENGTH    LENGTH              1095           4380   4      0
 FABCHARG  FABCHARGES       1934460        3868920   2      2
 ------------------------------------------------------------

---------------------- SHIPTO=14898029 -----------------------


 Variable  Label               Mean            Sum   N  Nmiss
 ------------------------------------------------------------
 LENGTH    LENGTH              2500           5000   2      0
 FABCHARG  FABCHARGES       1400825        2801650   2      0
 ------------------------------------------------------------

---------------------- SHIPTO=15432147 -----------------------


 Variable  Label               Mean            Sum   N  Nmiss
 ------------------------------------------------------------
 LENGTH    LENGTH               725            2900   4      0
 FABCHARG  FABCHARGES        252149          504297   2      2
 ------------------------------------------------------------

---------------------- SHIPTO=18543489 -----------------------


 Variable  Label               Mean            Sum   N  Nmiss
 ------------------------------------------------------------
 LENGTH    LENGTH               303            1820   6      0
 FABCHARG  FABCHARGES       11063836       44255344   4      2
------------------------------------------------------------

---------------------- SHIPTO=19783482 -----------------------


 Variable  Label               Mean            Sum   N  Nmiss
 ------------------------------------------------------------
 LENGTH    LENGTH               450            1800   4      0
 FABCHARG  FABCHARGES         252149         1008594   4      0
 ------------------------------------------------------------

---------------------- SHIPTO=19876078 -----------------------


 Variable  Label               Mean            Sum   N  Nmiss
 ------------------------------------------------------------
 LENGTH    LENGTH               690            1380   2      0
 FABCHARG  FABCHARGES             .               .   0      2
 ------------------------------------------------------------
```

The BY statement causes the interface view engine to generate a DBMS-specific SQL ORDER BY clause so that the data from this table is returned as if it were sorted.

Using the RANK Procedure

You can also use more advanced statistical procedures on DBMS data. The following example uses the RANK procedure to calculate the order of birthdays for a set of employees who are listed in the DB2 table SASDEMO.EMPLOYEES. The OUT= option

creates a SAS data file, DLIB.RANKEXAM, from the view descriptor VLIB.EMPS so that the data in the SAS file can be sorted by the SORT procedure. The RANKS statement assigns the name DATERANK to the new variable (in the SAS data file) that is created by the procedure.

```
proc access dbms=db2;
/* create access descriptor  */
   create adlib.employ.access;
   database=sample;
   table=sasdemo.employees;
   drop salary;
   rename birthdate birthdat;
   list all;

/* create vlib.emps view     */
   create vlib.emps.view;
   select empid jobcode birthdate lastname;
   format birthdate date9.
          empid     6.0;
   subset where jobcode=602;
run;

/* example          */
proc rank data=vlib.emps out=dlib.rankexam;
   var birthdat;
   ranks daterank;
run;

proc sort data=dlib.rankexam;
   by lastname;
run;

proc print data=dlib.rankexam(drop=jobcode);
title 'Order of Dept 602 Employee Birthdays';
run;
```

The DROP= data set option is used on the PROC PRINT statement because the JOBCODE variable is not needed in the output. Output 14.4 on page 192 shows the result of this example.

Output 14.4 Ranking of Employee Birthdays

Order of Dept 602 Employee Birthdays

OBS	EMPID	BIRTHDAT	LASTNAME	DATERANK
1	456910	24SEP1958	ARDIS	5
2	237642	13MAR1959	BATTERSBY	6
3	239185	28AUG1964	DOS REMEDIOS	7
4	321783	03JUN1940	GONZALES	2
5	120591	12FEB1951	HAMMERSTEIN	4
6	135673	21MAR1966	HEMESLY	8
7	456921	12MAY1967	KRAUSE	9
8	457232	15OCT1968	LOVELL	11
9	423286	31OCT1969	MIFUNE	12
10	216382	24JUL1968	PURINTON	10
11	234967	21DEC1972	SMITH	13
12	212916	29MAY1933	WACHBERGER	1
13	119012	05JAN1951	WOLF-PROVENZA	3

Reading and Updating Data with the SQL Procedure

In Version 6, the SAS System's SQL procedure enabled you to retrieve and update data from DBMS tables and views. You could read and display DBMS data by specifying a view descriptor or other SAS data set in the SQL procedure's SELECT statement. In Version 7, you can specify librefs based on DBMS data that you create by using the SAS/ACCESS LIBNAME statement. See Chapter 12, "Using DBMS Data in Version 7 and Version 8," on page 151 for examples.

If you are using descriptors, you can specify them in the SQL procedure's INSERT, DELETE, and UPDATE statements. You can also use these statements to modify SAS data files. The ability to update data in a DBMS table or through a DBMS view by using descriptors is subject to the following conditions:

- As in other PROC and DATA steps, you can use only a view descriptor or other SAS data set in an SQL procedure statement, not an access descriptor.
- You can usually only browse data retrieved using a view descriptor that is based on a DBMS view. Most DBMS views have a number of restrictions on when you can use them to update their underlying data. (The primary restriction is that a DBMS view can be used only to update data when the view is defined on a single DBMS table.)
- If you did not create the DBMS table or view, you must be granted the appropriate DBMS privileges before you can select, insert, delete, or update the data.
- A DBMS trigger might prevent you from updating observations in a DBMS table. Refer to your DBA to determine if triggers are used in your DBMS tables.
- Some DBMSs require the SQL procedure's UNDO_POLICY option before you can use the INSERT, DELETE, or UPDATE statements with a DBMS view descriptor. For example:

```
proc sql undo_policy=none;
```

  Refer to your DBMS chapter to see if this requirement applies to your DBMS.

Here is a summary of some of the SQL procedure statements, when used with view descriptors:

SELECT retrieves, manipulates, and displays data described by a view descriptor. A SELECT statement is usually referred to as a *query* because it queries the table for information.

DELETE deletes rows from a DBMS table that is described by a view descriptor. If a view descriptor is based on an updatable DBMS view, rows can also be deleted from the view's underlying table.

INSERT inserts rows into a DBMS table.

UPDATE updates the data values in a DBMS table.

Because the SQL procedure is based on the Structured Query Language, it works somewhat differently than some SAS procedures. For example, the SQL procedure executes without a RUN statement when a procedure statement is submitted. The SQL procedure also displays any output automatically without using the PRINT procedure.

By default, PROC SQL uses the LABEL option to display output. LABEL displays SAS variable labels, which default to DBMS column names. If you prefer to use SAS variable names in your output, specify NOLABEL in the OPTIONS statement.

Reading Data with the SQL Procedure

You can use the SQL procedure's SELECT statement to display data that is described by a view descriptor. In the following example, the query uses the VLIB.PRODUCT view descriptor to retrieve a subset of the data in the ORACLE SPECPROD table.

The asterisk (*) in the SELECT statement indicates that all the columns in VLIB.PRODUCT are retrieved. The WHERE clause retrieves a subset of the rows. The ORDER BY clause causes the data to be presented in ascending order according to the table's FIBERNAME column. Both the WHERE clause and the ORDER BY clause are passed to the DBMS for processing.

Note: The following SQL procedure examples assume that the DBMS tables have not been updated by other examples. △

```
proc access dbms=oracle;
/* create access descriptor  */
   create adlib.product.access;
   user=scott;  orapw=tiger;
   path='myorapath';
   table=specprod;
   assign=yes;
   rename productid=prodid
          fibername=fibernam;
   format productid  4.
          weight     e16.9
          fibersize  e20.13
          width      e16.9 ;
   list all;

/* create view descriptor   */
   create vlib.product.view;
   select all;
   list view;
 run;

options nodate linesize=120;
title 'DBMS Data Retrieved with a SELECT
       Statement';
proc sql;
```

```
select *
  from vlib.product
  where cost is not null
  order by fibernam;
quit;
```

Output 14.5 on page 194 displays the query's output. Note that the SQL procedure displays the DBMS table's column names, not the SAS variable names.

Output 14.5 DBMS Data Retrieved with a PROC SQL Query

DBMS Data Retrieved with a SELECT Statement

PRODUCTID	WEIGHT	FIBERNAME	FIBERSIZE	COST	PERUNIT	WIDTH
1279	1.278899910E-01	asbestos	6.3476000000000E-10	1289.64	m	2.227550050E+02
2567	1.258500220E-01	fiberglass	5.1880000000000E-11	560.33	m	1.205000000E+02
8934	1.429999950E-03	gold	2.3800000000000E-12	100580.33	cm	2.255999760E+01

Updating Data with the SQL Procedure

You can use the SQL procedure's UPDATE statement to update the data in a DBMS table.

The following UPDATE statements update the values in the Oracle Rdb table EMPLOYEES. Because you are referencing a view descriptor, you use the SAS variable names in the UPDATE statement; however, the SQL procedure displays the Oracle Rdb column names.

Note: The following examples use a previously created view descriptor, VLIB.EMPEEOC, which is based on data that is contained in the EMPLOYEES table. △

```
proc sql;
update vlib.empeeoc
   set salary=26678.24,
       gender='M',
       birthdat='28AUG64'dt
   where empid='123456';

options linesize=120;
title 'Updated Data in EMPLOYEES Table';
select empid, hiredate, salary, dept,
       jobcode, gender, birthdat, lastname
   from vlib.empeeoc
   where empid='123456';
quit;
```

Output 14.6 on page 194 displays the updated row of data retrieved from the view descriptor VLIB.EMPEEOC.

Output 14.6 Oracle Rdb Data Updated with the UPDATE Statement

```
                          Updated Data in EMPLOYEES Table

 EMPID  HIREDATE          SALARY  DEPT    JOBCODE  GENDER  BIRTHDATE  LASTNAME
-------------------------------------------------------------------------------
123456   04APR1989    $26,678.24  ACC043     1204  M         28AUG64  VARGAS
```

Deleting Data with the SQL Procedure

You can use the SQL procedure's DELETE statement to delete rows from a DBMS table. In the following example, the row that contains the value 346917 in the EMPID column is deleted from the Oracle Rdb table EMPLOYEES.

```
proc sql undo_policy=none;
delete from vlib.empeeoc
   where empid='346917';
quit;
```

A message is written to the SAS log to indicate that the row has been deleted, as shown in Output 14.7 on page 195.

Output 14.7 Message Displayed in the SAS Log When a Row Is Deleted

```
6688
6689   /*=========================*/
6690   /* Example for Output */
6691   /* shows in a SAS log.     */
6692   /*=========================*/
6693  proc sql undo_policy=none;
6694  delete from vlib.empeeoc
6695     where empid='346917';

NOTE: 1 row was deleted from VLIB.EMPEEOC.

6707  quit;
```

If you have many rows to delete, you could use a macro variable for EMPID instead of the individual EMPID values to change the values more easily.

```
%let empid='346917';

proc sql;
delete from vlib.empeeoc
   where empid=&empid;
quit;
```

CAUTION:

Use a WHERE clause in the DELETE statement. If you omit the WHERE clause from the DELETE statement, you delete *all* the data in the SAS data file or DBMS table. △

Inserting Data with the SQL Procedure

You can use the SQL procedure's INSERT statement to add rows to a DBMS table. In the following example, the row that contains the value 346917 in the EMPID column is inserted back into the Oracle Rdb table EMPLOYEES.

Note: The following examples use a previously created view descriptor, VLIB.ALLEMP, which is based on data contained in the EMPLOYEES table. △

```
proc sql undo_policy=none;
insert into vlib.allemp
   values(346917,'02MAR87'd,46000.33,'SHP013',
          204,'F','15MAR1955'DT,'SHIEKELESLAM',
          'SHALA','Y.','8745');
quit;
```

A message is written to the SAS log to indicate that the row has been inserted, as shown in Output 14.8 on page 196.

Output 14.8 Message Displayed in the SAS Log When a Row Is Inserted

```
6698
6699   /*=========================*/
6700   /* Example for Output */
6701   /* shows in a SAS log.     */
6702   /*=========================*/
6703  proc sql undo_policy=none;
6704  insert into vlib.allemp
6705     values(346917,'02MAR87'd,46000.33,'SHP013',204,'F',
6706    '15MAR1955'DT, 'SHIEKELESLAM','SHALA','Y.','8745');

NOTE: 1 row was inserted into VLIB.ALLEMP.

6707  quit;
```

Selecting and Combining Data By Using the SQL Procedure

Joining Data from Various Sources

The SQL procedure provides another way to select and combine data. For example, suppose you have three data sets: two view descriptors, VLIB.CUSPHON and VLIB.CUSORDR, which are based on the ORACLE tables CUSTOMERS and ORDERS, respectively, and a SAS data file, DLIB.OUTOFSTK, which contains product names and numbers that are out of stock.

Note: See the appendix for a description of DLIB.OUTOFSTK. △

You can use the SQL procedure to create a view that joins the data from these three sources and displays their output. The SAS WHERE or subsetting IF statements would not be appropriate in this case because you want to compare variables from several sources, rather than simply merging or concatenating the data.

The following SAS statements select and combine data from the view descriptors and the SAS data file to create a PROC SQL view, SLIB.BADORDR. SLIB.BADORDR

retrieves customer and product information that the sales department uses to notify customers of unavailable products.

Note: Although this example shows you an alternate way to join data by using access descriptors, SAS/ACCESS now provides more efficient ways to join your data by using the LIBNAME statement. △

```
proc access dbms=oracle;

/* create access descriptor  */

   create adlib.customr.access;
   user=scott;  orapw=tiger;
   path='myorapath';
   table=customers;
   list all;

/* create vlib.cusphon view */

   create vlib.cusphon.view;
   select customer phone name;
   rename customer = custnum;
run;

/* create access descriptor  */

proc access dbms=oracle;
   create adlib.orders.access;
   user=scott;  orapw=tiger;
   path='myorapath';
   table=orders;
   list all;

/* create vlib.cusordr view  */

   create vlib.cusordr.view;
   select ordernum stocknum shipto;
   rename ordernum ordnum;
   format ordernum 5.0
          stocknum 4.0;
run;

proc sql;
   create view slib.badordr as
      select distinct cusphon.custnum,
        cusphon.name, cusphon.phone,
        cusordr.stocknum, outofstk.fibernam
             as product
        from vlib.cusphon, vlib.cusordr,
             dlib.outofstk
        where cusordr.stocknum=outofstk.fibernum
             and cusphon.custnum=cusordr.shipto;
quit;
```

The CREATE VIEW statement incorporates a WHERE clause as part of its SELECT clause. The DISTINCT keyword eliminates any duplicate rows of customer numbers that occur when companies order an unavailable product more than once.

Note: It is recommended that you *not* include an ORDER BY clause in a CREATE VIEW statement. This causes the output data to be sorted every time the PROC SQL view is submitted and might have a negative impact on performance. It is more efficient to add an ORDER BY clause to a SELECT statement that displays your output data, as shown below. △

```
options linesize=120;
title 'Data Described by SLIB.BADORDR';

select * from slib.badordr
   order by custnum, product;
```

This SELECT statement uses the PROC SQL view SLIB.BADORDR to display joined ORACLE and SAS data in ascending order by the CUSTNUM column and then by the PRODUCT (that is, FIBERNAM) column. The data is ordered by PRODUCT because one customer might have ordered more than one product. To select all the columns from the view, use an asterisk (*) in place of column names. When an asterisk is used, the columns are displayed in the order specified in the SLIB.BADORDR view. Output 14.9 on page 198 shows the data described by the SLIB.BADORDR view.

Output 14.9 Data Described by the PROC SQL View SLIB.BADORDR

Data Described by SLIB.BADORDER

CUSTOMER	NAME	PHONE	STOCKNUM	PRODUCT
15432147	GREAT LAKES LABORATORY EQUIPMENT MANUFACTURERS	616/582-3906	4789	dacron
18543489	LONE STAR STATE RESEARCH SUPPLIERS	512/478-0788	8934	gold
29834248	BRITISH MEDICAL RESEARCH AND SURGICAL SUPPLY	(0552)715311	3478	olefin
31548901	NATIONAL COUNCIL FOR MATERIALS RESEARCH	406/422-3413	8934	gold
43459747	RESEARCH OUTFITTERS	03/734-5111	8934	gold

Although the query uses SAS variable names like CUSTNUM, you may notice that the output uses DBMS column names like CUSTOMER. By default, PROC SQL displays SAS variable labels, which default to DBMS column names. (You can use the NOLABEL option to change this default.)

Creating New Columns and Using the GROUP BY Clause

Instead of creating a new PROC SQL view, you might want to summarize your data and create new columns in a report. Although you cannot use the ACCESS procedure to create new columns, you can easily do this by using the SQL procedure with data that is described by a view descriptor.

This example uses the SQL procedure to retrieve and manipulate data from the view descriptor VLIB.ALLEMP, which is based on the DB2 table SASDEMO.EMPLOYEES. When this *query* (as a SELECT statement is often called) is submitted, it calculates and displays the average salary for each department. The query enables you to manipulate your data and display the results without creating a SAS data set.

Because this example reports on employees' salaries, the view descriptor VLIB.ALLEMP is assigned a SAS System password (MONEY) using the DATASETS

procedure. Because of the READ= level of protection, the password must be specified in the PROC SQL SELECT statement before you can see the DB2 data accessed by VLIB.ALLEMP.

In the following example, the DISTINCT keyword in the SELECT statement removes duplicate rows. The AVG function in the SQL procedure is equivalent to the SAS MEAN function.

```
options linesize=80;

proc access dbms=db2;
/* create access descriptor  */
   create adlib.employ.access;
   ssid=db2;
   table=sasdemo.employees;
   assign=yes;
   format empid     6.0
          salary    dollar12.2
          jobcode   5.0
          birthdate date9.
          hiredate  date9.;
   list all;
run;

/* create vlib.allemp view */
proc access dbms=db2
            accdesc=adlib.employ;
   create vlib.allemp.view;
   select all;
run;

/* assign a password  */
proc datasets library=vlib  memtype=view;
   modify allemp (read=money);
run;

/* example            */
title 'Average Salary Per ACC Department';
proc sql;
   select distinct dept,
          avg(salary) label='Average Salary'
          format=dollar12.2
      from vlib.allemp(pw=money)
      where dept like 'ACC%'
      group by dept;
```

The columns are displayed in the order specified in the SELECT clause of the query. Output 14.10 on page 199 shows the result of the query.

Output 14.10 Data Retrieved by an SQL Procedure Query

```
                Average Salary Per ACC Department

                                    Average
                          DEPT       Salary
                          --------------------
                          ACC013    $54,591.33
                          ACC024    $55,370.55
                          ACC043    $75,000.34
```

To delete a password on an access or view descriptor or any SAS data set, put a slash after the password:

```
/* delete the password  */
proc datasets library=vlib  memtype=view;
   modify allemp (read=money/);
run;
```

Using Advanced PROC SQL Features

This example combines a number of PROC SQL components including summary functions, a HAVING clause with a subquery, and an inline view within that subquery. The reason you use each component and the way in which each is evaluated are described following the example. The example displays the employees who took the most orders that have already shipped. It uses the VLIB.ALLEMP and VLIB.ALLORDR view descriptors to join data from the tables SASDEMO.EMPLOYEES and SASDEMO.ORDERS.

```
options ls=80;

proc access dbms=db2;
/* create access descriptor  */
   create adlib.employ.access;
   database=sample;
   table=sasdemo.employees;
   assign=yes;
   format empid     6.0
          salary    dollar12.2
          jobcode   5.0
          birthdate date9.
          hiredate  date9.;

/* create vlib.allemp view */
   create vlib.allemp.view;
   select all;
   list view;
run;

proc access dbms=db2;
/* create access descriptor    */
   create adlib.order.access;
   database=sample;
   table=sasdemo.orders;
   assign=yes;
```

```
   rename dateorderd = dateord
          processdby = procesby;
   format dateorderd datetime9.
          shipped datetime9.
          ordernum      5.0
          length        4.0
          stocknum      4.0
          takenby       6.0
          processdby    6.0
          fabcharges    12.2;

/* create vlib.allordr view   */
   create vlib.allordr.view;
   select all;
   list view;
run;


proc sql;
title 'Employees Who Took the Most Orders That
Shipped';
select distinct lastname label "Took Orders",
takenby, count(shipped) as ordship
   from vlib.allemp, vlib.allordr
   where takenby=empid
   group by takenby
   having ordship=
      (select max(ordship)
          from (select distinct takenby,
                   count(shipped) as ordship
                   from vlib.allemp, vlib.allordr
                   where takenby=empid
                   group by takenby));
quit;
```

You begin to evaluate the query at its innermost level. Here the query begins by evaluating the inline view:

```
          from (select distinct takenby,
                   count(shipped) as ordship
                   from vlib.allemp, vlib.allordr
                   where takenby=empid
                   group by takenby));
```

The inline view lists employees who have taken orders and counts the number of orders that have been shipped. A column alias ORDSHIP is assigned to the number of orders and is used elsewhere in the query.

```
TAKENBY   ORDSHIP
-----------------
119012         6
212916         0
234967         0
321783         6
456910         5
```

The SELECT MAX(ORDSHIP) clause uses the results of the inline view to determine the highest number of orders taken and shipped, 6. When this amount is supplied to the outer query, it evaluates as if the query were written:

```
proc sql;
select distinct lastname label "Took Orders",
   takenby, count(shipped) as ordship
   from vlib.allemp, vlib.allordr
   where takenby=empid
   group by takenby
   having ordship=6;
```

The first part of the outer query adds the names of the employees who took the orders and joins the data from the two view descriptors with matching EMPLOYEE numbers (**where takenby=empid**). The COUNT function computes the number of shipped orders that were taken by each employee to reduce the number of rows to one per employee. The HAVING expression then selects rows if the number of orders is 6. Output 14.11 on page 202 shows the results of the PROC SQL query.

Output 14.11 Data on Orders Shipped

```
        Employees Who Took the Most Orders That Shipped

             Took Orders          TAKENBY   ORDSHIP
             ---------------------------------------
             GONZALES              321783          6
             WOLF-PROVENZA         119012          6
```

Accessing Tables That Are Located on Different Nodes or Databases

In a networking environment, you can often access data from DBMS tables that is stored on different machines or in different databases. When using the SAS/ACCESS Interface to SYBASE, for example, you use the SERVER= and DATABASE= statements to specify the locations of the tables that you want to access. Use TABLE= to specify the names of the SYBASE tables.

In the following example, you create access descriptors and view descriptors for SYBASE tables that have different owners and are stored in databases that reside on different machines. The USER= and PASSWORD= statements identify the owners of the EMPLOYEES and INVOICE tables and their passwords.

After creating the descriptors, you use the SQL procedure to join the tables' data. Because the database identification information is stored permanently in each descriptor, PROC SQL can use the view descriptors to access and join the remote SYBASE data:

```
/* create access descriptor  */
proc access dbms=sybase;
   create work.employ.access;
   server=server1;
   database=personnel;
   user=carmen;
   password=aria;
   table=employees;

/* create vlib.emp_acc view  */
   create vlib.emp_acc.view;
```

```
   select all;
   format empid 6.0
          salary dollar12.2
          jobcode 5.0
          hiredate date9.
          birthdate date9.;
   subset where DEPT like 'ACC%';
   list all;
run;

/* create access descriptor  */
proc access dbms=sybase;
   create work.invoice.access;
   server=server2;
   database=inventory;
   user=joachim;
   password=machauf;
   table=invoice;

/* create vlib.sainv view   */
   create vlib.sainv.view;
   select all;
   rename invoicenum invnum
          amtbilled  amtbilld
          amountinus amtinus;
   format invoicenum 5.
          billedby   6.
          amtbilled  15.2
          amountinus 15.2
          billedon date9.
          paidon   date9.;
   subset where COUNTRY in ('Argentina','Brazil');
   list all;
run;
```

SYBASE objects, such as table names and columns, are case sensitive. The WHERE clauses in PROC ACCESS SUBSET statements are passed to SYBASE exactly as you type them, so you must use the correct case for SYBASE column names. The database identification statements and column names in all other statements are converted to uppercase unless they are enclosed in quotes.

```
options linesize=120;
title 'South American Invoices and Who Submitted
       Them';

proc sql;
   select invnum, country, billedon,
          paidon, billedby, lastnam, firstnam
      from vlib.emp_acc, vlib.sainv
      where emp_acc.empid=sainv.billedby;
quit;
```

Output 14.12 on page 204 shows the results of the PROC SQL query.

Output 14.12 Data Joined from Tables in Different Databases

South American Invoices and Who Submitted Them

INVOICENUM	COUNTRY	BILLEDON	PAIDON	BILLEDBY	LASTNAME	FIRSTNAME
12476	Argentina	24DEC1998	.	135673	HEMESLY	STEPHANIE
11270	Brazil	05OCT1998	18OCT1998	239185	DOS REMEDIOS	LEONARD
11285	Argentina	10OCT1998	30NOV1998	239185	DOS REMEDIOS	LEONARD
11280	Brazil	07OCT1998	20OCT1998	423286	MIFUNE	YUKIO
12051	Brazil	02NOV1998	.	457232	LOVELL	WILLIAM
12471	Brazil	27DEC1998	.	457232	LOVELL	WILLIAM

Updating DBMS Data with the MODIFY Statement

The MODIFY statement extends the capabilities of the DATA step by enabling you to modify data accessed by a view descriptor or a SAS data file without creating an additional copy of the data. To use the MODIFY statement with a view descriptor, you must have UPDATE privileges on the view's underlying DBMS table.

A DBMS trigger may prevent you from modifying observations in a DBMS table. Refer to your DBMS documentation to see if triggers are used in your DBMS.

You can specify either a view descriptor or a SAS data file as the master data set in the MODIFY statement. In the following example, the master data set is the view descriptor VLIB.MASTER, which describes data in the ORACLE table ORDERS. You also create a transaction data file, DLIB.TRANS, that you use to update the master data set (and therefore, the ORDERS table). The SAS variable names, formats, and informats of the transaction data file must correspond to those described by the view descriptor VLIB.MASTER.

Using the VLIB.MASTER view descriptor, the MODIFY statement updates the ORDERS table with data from the DLIB.TRANS data file. The SAS System reads one observation (or row) of the ORDERS table for each iteration of the DATA step, and performs any operations that the code specifies. In this case, the IF-THEN statements specify whether the information for an order is to be updated, added, or deleted.

```
proc access dbms=oracle;
/* create access descriptor  */
   create adlib.orders.access;
   user=scott;  orapw=tiger;
   path='myorapath';
   table=orders;
   assign=yes;
   rename dateorderd = dateord
          processdby = procesby;
   format dateorderd date9.
          shipped date9.
          ordernum      5.0
          length        4.0
          stocknum      4.0
          takenby       6.0
          processdby    6.0
          fabcharges    12.2;

/* create vlib.master view   */
   create vlib.master.view;
```

```
   select all;
run;

data dlib.trans;
/* Obs. 1 specifies Update for */
/* ORDERNUM=12102              */
   ordernum=12102;
      shipped='05DEC1998'd;
      type='U';
      output;

/* Obs. 2 specifies Update for */
/* ORDERNUM=12160              */
   ordernum=12160;
      shipped=.;
      takenby=456910;
      type='U';
      output;

/* Obs. 3 specifies Add for new */
/* ORDERNUM=13000               */
   ordernum=13000;
      stocknum=9870;
      length=650;
      fabcharg=.;
      shipto='19876078';
      dateord='18JAN1999'd;
      shipped='29JAN1999'd;
      takenby=321783;
      procesby=120591;
      specinst='Customer agrees to
                certain limitations.';
      type='A';
      output;

/* Obs. 4 specifies Delete for */
/* ORDERNUM=12465              */
   ordernum=12465;
      type='D';
      output;
 run;

/* MODIFY statement example */
data vlib.master;
   modify vlib.master dlib.trans;
   by ordernum;
   select (_iorc_);
/* No match in MASTER - Add */
      when (%sysrc(_dsenmr)) do;
         if type='A'
            then output vlib.master;
         _error_ = 0;
      end;
/* Match located - Update or Delete */
```

```
      when (%sysrc(_sok)) do;
         if type='U'
            then replace vlib.master;
         else if type='D'
            then remove vlib.master;
      end;
/* Traps unexpected outcomes */
      otherwise do;
      put 'Unexpected ERROR condition:
          _IORC_ = ' _iorc_ ;
/* This dumps all vars in the PDV */
         put _all_;
         _error_ = 0;
      end;
   end;
run;

/* prints the example's output */
options linesize=120;

proc print data=vlib.master;
   where ordernum
      in(12102 12160 13000 12465);
   title 'ORACLE Data Updated with
         the MODIFY Statement';
run;
```

The DATA step uses the SYSRC macro to check the value of the _IORC_ automatic variable. It also prevents an error message from being generated when no match is found in the VLIB.MASTER file for an observation that is being added. It prevents the error message by resetting the _ERROR_ automatic variable to 0. The PRINT procedure specifies a WHERE statement so only the observations that are included in the transaction data set are displayed. The observation with ORDERNUM **12465** is deleted by the MODIFY statement, so it does not appear in the results. The results of this example are shown in Output 14.13 on page 206.

Output 14.13 Revising DBMS Data with a MODIFY Statement

DBMS Data Updated with the MODIFY Statement

OBS	ORDERNUM	STOCKNUM	LENGTH	FABCHARG	SHIPTO	DATEORD	SHIPPED	TAKENBY	PROCESBY
1	13000	9870	650	.	19876078	18JAN1999	29JAN1999	321783	120591
2	12160	3478	1000	.	29834248	19NOV1998	.	456910	.
3	12102	8934	110	11063836.00	18543489	15NOV1998	05DEC1998	456910	.

OBS	SPECINST
1	Customer agrees to certain limitations.
2	Customer agrees to pay in full.

In this example, any column value that you specify in the transaction data set carries over to any subsequent observations if the values for the subsequent observations are missing. For example, the first observation sets the value of SHIPPED to **05DEC98**. The second observation sets the value to MISSING. If the value of SHIPPED were not set to MISSING in the second observation, the value **05DEC98** would be incorrectly supplied.

Therefore, you might want to create your transaction data set in a specific order to minimize having to reset variables.

There are some differences in the ways you use a MODIFY statement to update a SAS data file and to update DBMS data through a view descriptor. When a view descriptor is used as the master data set in a MODIFY statement, the following conditions apply:

- The POINT= option cannot be used because observation numbers are not available in a relational DBMS table.
- The NOBS= option displays the largest positive integer value available on the host operating system.
- The KEY= option cannot be used because the DBMS determines whether to use DBMS indexes that have been assigned to columns in tables.
- Each DBMS statement that is issued, whether an INSERT, DELETE, or UPDATE, is a separate transaction and is saved in the DBMS table. You cannot undo (or reverse) these changes without re-editing.

Updating a SAS Data File with DBMS Data

You can update a SAS data file with DBMS data that is described by a view descriptor just as you can update a SAS data file with data from another SAS data file.

Suppose you have a SAS data set, DLIB.BIRTHDAY, that contains employee ID numbers, last names, and birthdays. (See Appendix 1, "Sample Data," on page 217 for a description of DLIB.BIRTHDAY.) You want to update this data set with data described by VLIB.EMPBDAY, a view descriptor that is based on the DB2 table SASDEMO.EMPLOYEES. To perform this update, enter the following SAS statements:

```
options linesize=80;

proc access dbms=db2;
/* create access descriptor   */
   create adlib.employ.access;
   ssid=db2;
   table=sasdemo.employees;
   assign=yes;
   format empid 6.
          salary dollar12.2
          jobcode 5.
          hiredate date9.
          birthdate date9.;
   list all;

/* create view descriptor   */
   create vlib.empbday.view;
   select empid birthdate lastname
          firstname phone;
run;

proc sort data=dlib.birthday;
   by lastname;
run;

/* examples               */
proc print data=dlib.birthday;
```

```
   format birthdat date9.;
   title 'DLIB.BIRTHDAY Data File';
run;

proc print data=vlib.empbday;
   format birthdat date9.;
   title 'Data Described by VLIB.EMPBDAY';
run;

data dlib.newbday;
   update dlib.birthday vlib.empbday;
   by lastname;
run;

proc print;
   format birthdat date9.;
   title 'DLIB.NEWBDAY Data File';
run;
```

When the UPDATE statement references the view descriptor VLIB.EMPBDAY, and a BY statement is used in the DATA step, the BY statement causes the interface view engine to generate an ORDER BY clause for the variable LASTNAME. Thus, the ORDER BY clause causes the DBMS data to be presented to the SAS System in sorted order for use in updating the DLIB.NEWBDAY data file. However, the SAS data file DLIB.BIRTHDAY must be sorted before the update because the UPDATE statement expects both the original file and the transaction file to be sorted by the same BY variable.

Output 14.14 on page 208, Output 14.15 on page 208, and Output 14.16 on page 209 show the results of the PRINT procedures.

Output 14.14 Data File to Be Updated, DLIB.BIRTHDAY

DLIB.BIRTHDAY Data File

OBS	EMPID	BIRTHDAT	LASTNAME
1	127845	25DEC1949	MEDER
2	459287	05JUN1939	RODRIGUES
3	254896	06APR1951	TAYLOR-HUNYADI

Output 14.15 AS/400 Data Described by the View Descriptor VLIB.EMPBDAY

Data Described by VLIB.EMPBDAY

OBS	EMPID	BIRTHDAT	LASTNAME	FIRSTNAM	PHONE
1	119012	05JAN1951	WOLF-PROVENZA	G.	3467
2	120591	12FEB1951	HAMMERSTEIN	S.	3287
3	123456	.	VARGAS	CHRIS	
4	127845	25DEC1948	MEDER	VLADIMIR	6231
5	129540	31JUL1965	CHOULAI	CLARA	3921
6	135673	21MAR1966	HEMESLY	STEPHANIE	6329
7	212916	29MAY1933	WACHBERGER	MARIE-LOUISE	8562
8	216382	24JUL1968	PURINTON	PRUDENCE	3852
9	234967	21DEC1972	SMITH	GILBERT	7274
10	237642	13MAR1959	BATTERSBY	R.	8342
11	239185	28AUG1964	DOS REMEDIOS	LEONARD	4892
12	254896	06APR1954	TAYLOR-HUNYADI	ITO	0231
13	321783	03JUN1940	GONZALES	GUILLERMO	3642
14	328140	02JUN1956	MEDINA-SIDONIA	MARGARET	5901
15	346917	15MAR1955	SHIEKELESLAM	SHALA	8745
16	356134	25OCT1965	DUNNETT	CHRISTINE	4213
17	423286	31OCT1969	MIFUNE	YUKIO	3278
18	456910	24SEP1958	ARDIS	RICHARD	4351
19	456921	12MAY1967	KRAUSE	KARL-HEINZ	7452
20	457232	15OCT1968	LOVELL	WILLIAM	6321
21	459287	05JAN1939	RODRIGUES	JUAN	5879
22	677890	24APR1970	NISHIMATSU-LYNCH	CAROL	6245

Output 14.16 Data in the Updated Data File DLIB.NEWBDAY

DLIB.NEWBDAY Data File

OBS	EMPID	BIRTHDAT	LASTNAME	FIRSTNAM	PHONE
1	456910	24SEP1958	ARDIS	RICHARD	4351
2	237642	13MAR1959	BATTERSBY	R.	8342
3	129540	31JUL1965	CHOULAI	CLARA	3921
4	239185	28AUG1964	DOS REMEDIOS	LEONARD	4892
5	356134	25OCT1965	DUNNETT	CHRISTINE	4213
6	321783	03JUN1940	GONZALES	GUILLERMO	3642
7	120591	12FEB1951	HAMMERSTEIN	S.	3287
8	135673	21MAR1966	HEMESLY	STEPHANIE	6329
9	456921	12MAY1967	KRAUSE	KARL-HEINZ	7452
10	457232	15OCT1968	LOVELL	WILLIAM	6321
11	127845	25DEC1948	MEDER	VLADIMIR	6231
12	328140	02JUN1956	MEDINA-SIDONIA	MARGARET	5901
13	423286	31OCT1969	MIFUNE	YUKIO	3278
14	677890	24APR1970	NISHIMATSU-LYNCH	CAROL	6245
15	216382	24JUL1968	PURINTON	PRUDENCE	3852
16	459287	05JAN1939	RODRIGUES	JUAN	5879
17	346917	15MAR1955	SHIEKELESLAM	SHALA	8745
18	234967	21DEC1972	SMITH	GILBERT	7274
19	254896	06APR1954	TAYLOR-HUNYADI	ITO	0231
20	123456	.	VARGAS	CHRIS	
21	212916	29MAY1933	WACHBERGER	MARIE-LOUISE	8562
22	119012	05JAN1951	WOLF-PROVENZA	G.	3467

Appending Data with the APPEND Procedure

You can append data from any data set to a SAS data file or view descriptor. Specifically, you can append DBMS data described by one view descriptor to another, or you can append a SAS data file to a view descriptor (and therefore to the DBMS table).

The following example uses the APPEND procedure's FORCE option to append a SAS data file with extra variables to the view descriptor VLIB.SQLEMPS. You must be granted DBMS-specific INSERT privileges to add rows to the table SASDEMO.EMPLOYEES.

You can append data to a table that is referenced by a view descriptor even if the view descriptor contains a subset of columns and a subset of rows. If a DBMS column is defined as NOT NULL, some restrictions apply when appending data.

The FORCE option forces PROC APPEND to concatenate two data sets even though they may have some different variables or variable attributes. The SAS data file, DLIB.TEMPEMPS, has DEPT, FAMILYID, and GENDER variables that have not been selected in the view descriptor VLIB.SQLEMPS. The extra variables are dropped from DLIB.TEMPEMPS when it and the BASE= data set, VLIB.SQLEMPS, are concatenated. A message is displayed in the SAS log indicating that the variables are dropped.

```
/* create access descriptor    */
proc access dbms=oracle;
    create adlib.employ.access;
    user=scott;  orapw=tiger;
    path='myorapath';
    table=sasdemo.employees;
    assign=no;
    drop salary;
    list all;

/* create view descriptor    */
    create vlib.sqlemps.view;
    select empid hiredate lastname
           firstname middlename;
    format empid 6.0
           hiredate date9.;
run;

proc print data=vlib.sqlemps;
/* examples                  */
title 'Data Described by VLIB.SQLEMPS';
run;

proc print data=dlib.tempemps;
title 'Data in DLIB.TEMPEMPS Data File';
run;
```

The view descriptor VLIB.SQLEMPS is displayed in Output 14.17 on page 210, and the SAS data file DLIB.TEMPEMPS is displayed in Output 14.18 on page 211.

Output 14.17 Data Described by VLIB.SQLEMPS

Data Described by VLIB.SQLEMPS

OBS	EMPID	HIREDATE	LASTNAME	FIRSTNAM	MIDDLENA
1	119012	01JUL1968	WOLF-PROVENZA	G.	ANDREA
2	120591	05DEC1980	HAMMERSTEIN	S.	RACHAEL
3	123456	04APR1989	VARGAS	CHRIS	J.
4	127845	16JAN1967	MEDER	VLADIMIR	JORAN
5	129540	01AUG1982	CHOULAI	CLARA	JANE
6	135673	15JUL1984	HEMESLY	STEPHANIE	J.
7	212916	15FEB1951	WACHBERGER	MARIE-LOUISE	TERESA
8	216382	15JUN1985	PURINTON	PRUDENCE	VALENTINE
9	234967	19DEC1988	SMITH	GILBERT	IRVINE
10	237642	01NOV1976	BATTERSBY	R.	STEPHEN
11	239185	07MAY1981	DOS REMEDIOS	LEONARD	WESLEY
12	254896	04APR1985	TAYLOR-HUNYADI	ITO	MISHIMA
13	321783	10SEP1967	GONZALES	GUILLERMO	RICARDO
14	328140	10JAN1975	MEDINA-SIDONIA	MARGARET	ROSE
15	356134	14JUN1985	DUNNETT	CHRISTINE	MARIE
16	423286	19DEC1988	MIFUNE	YUKIO	TOSHIRO
17	456910	14JUN1978	ARDIS	RICHARD	BINGHAM
18	456921	19AUG1987	KRAUSE	KARL-HEINZ	G.
19	457232	15JUL1985	LOVELL	WILLIAM	SINCLAIR
20	459287	02NOV1964	RODRIGUES	JUAN	M.
21	677890	12DEC1988	NISHIMATSU-LYNCH	CAROL	ANNE
22	346917	02MAR1987	SHIEKELESLAM	SHALA	Y.

Output 14.18 Data in DLIB.TEMPEMPS

Data in DLIB.TEMPEMPS Data File

OBS	EMPID	HIREDATE	DEPT	GENDER	LASTNAME	FIRSTNAM	MIDDLENA	FAMILYID
1	765111	04MAY1998	CSR011	M	NISHIMATSU-LYNCH	RICHARD	ITO	677890
2	765112	04MAY1998	CSR010	M	SMITH	ROBERT	MICHAEL	234967
3	219776	15APR1998	ACC024	F	PASTORELLI	ZORA		.
4	245233	10APR1998	ACC013		ALI	SADIQ	H.	.
5	245234	10APR1998	ACC024	F	MEHAILESCU	NADIA	P.	.
6	326721	01MAY1998	SHP002	M	CALHOUN	WILLIS	BEAUREGARD	.

The APPEND procedure also accepts a WHERE= data set option or a SAS WHERE statement to retrieve a subset of the observations. In this example, a subset of the observations from DLIB.TEMPEMPS is added to VLIB.SQLEMPS by using a SAS WHERE statement; the WHERE statement applies only to the DATA= data set.

```
proc append base=vlib.sqlemps
            data=dlib.tempemps force;
   where hiredate <= '30APR1998'd;
run;

proc print data=vlib.sqlemps;
 title 'Subset of SAS Data Appended
        to a View Descriptor';
run;
```

Output 14.19 on page 211 shows VLIB.SQLEMPS with three rows from DLIB.TEMPEMPS appended to it.

Output 14.19 Subset of Data Appended with the FORCE Option

Subset of SAS Data Appended to a View Descriptor

OBS	EMPID	HIREDATE	LASTNAME	FIRSTNAM	MIDDLENA
1	119012	01JUL1968	WOLF-PROVENZA	G.	ANDREA
2	120591	05DEC1980	HAMMERSTEIN	S.	RACHAEL
3	123456	04APR1989	VARGAS	CHRIS	J.
4	127845	16JAN1967	MEDER	VLADIMIR	JORAN
5	129540	01AUG1982	CHOULAI	CLARA	JANE
6	135673	15JUL1984	HEMESLY	STEPHANIE	J.
7	212916	15FEB1951	WACHBERGER	MARIE-LOUISE	TERESA
8	216382	15JUN1985	PURINTON	PRUDENCE	VALENTINE
9	234967	19DEC1988	SMITH	GILBERT	IRVINE
10	237642	01NOV1976	BATTERSBY	R.	STEPHEN
11	239185	07MAY1981	DOS REMEDIOS	LEONARD	WESLEY
12	254896	04APR1985	TAYLOR-HUNYADI	ITO	MISHIMA
13	321783	10SEP1967	GONZALES	GUILLERMO	RICARDO
14	328140	10JAN1975	MEDINA-SIDONIA	MARGARET	ROSE
15	356134	14JUN1985	DUNNETT	CHRISTINE	MARIE
16	423286	19DEC1988	MIFUNE	YUKIO	TOSHIRO
17	456910	14JUN1978	ARDIS	RICHARD	BINGHAM
18	456921	19AUG1987	KRAUSE	KARL-HEINZ	G.
19	457232	15JUL1985	LOVELL	WILLIAM	SINCLAIR
20	459287	02NOV1964	RODRIGUES	JUAN	M.
21	677890	12DEC1988	NISHIMATSU-LYNCH	CAROL	ANNE
22	346917	02MAR1987	SHIEKELESLAM	SHALA	Y.
23	219776	15APR1998	PASTORELLI	ZORA	
24	245233	10APR1998	ALI	SADIQ	H.
25	245234	10APR1998	MEHAILESCU	NADIA	P.

When you use PROC APPEND with a view descriptor as the BASE= file, the DBMS issues DBMS-specific INSERT statements and places the new rows into the DBMS table wherever the free space exists. The Screen Control Language (SCL) APPEND function behaves in the same way, that is, the DBMS determines where the rows are inserted into the table. This approach is contrary to how SAS usually performs an append. When the BASE= file is a SAS data file, the data is appended to the end of the data file.

See Output 14.20 on page 212 for a copy of the SAS log screen and the messages about the FORCE option.

Output 14.20 SAS Log with Messages about the FORCE Option

```
10504
10505
10506   /*==========================*/
10507   /* Example for Output  */
10508   /*==========================*/
10509  proc append base=vlib.sqlemps data=dlib.tempemps force;
10510     where hiredate <= '30APR98'd;
10511  run;

NOTE: Appending DLIB.TEMPEMPS to VLIB.SQLEMPS.
WARNING: Variable DEPT was not found on BASE file.
WARNING: Variable GENDER was not found on BASE file.
WARNING: Variable FAMILYID was not found on BASE file.
NOTE: FORCE is specified, so dropping/truncating will occur.
NOTE: 3 observations added.
NOTE: The data set VLIB.SQLEMPS has . observations and 5
      variables.
```

Because the BASE= data set is a view descriptor in this example, PROC APPEND generates a DBMS-specific SQL INSERT statement for the rows to be appended to the DBMS table.

The number of observations in the EMPLOYEES table is not displayed in the SAS log because when the view descriptor is opened by the SAS/ACCESS engine, the number of rows in the underlying table is not known.

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS/ACCESS® Software for Relational Databases: Reference, Version 8*, Cary, NC: SAS Institute Inc., 1999.

SAS/ACCESS® Software for Relational Databases: Reference, Version 8

ISBN 1–58025–558–2

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st printing, October 1999