CHAPTER

6

Using PC Files Data in SAS Programs

Introduction

One advantage of using SAS/ACCESS software is that it enables the SAS System to read and write PC files data directly from SAS programs. This chapter presents examples in which PC files data-accessed through view descriptors are used as input data for SAS programs, and it also shows you how to use SAS procedures and the DATA step to review and update PC file data that are described by SAS/ACCESS view descriptors.

The examples in this chapter use DIF, DBF, WK*n*, and XLS data. The PC file format is identified in each example and any file-specific issues are described in the example. Throughout the examples, the SAS terms *variable* and *observation* are used instead of column and row because this chapter illustrates SAS System procedures and the SAS

DATA step. The examples in this chapter show how to create access descriptors and view descriptors and then use the view descriptors in SAS procedures and DATA steps. For more information on the SAS language and procedures that are used in the examples, refer to the books listed at the end of each section. For information about using view descriptors efficiently in SAS programs, see "Performance and Efficient View Descriptors" on page 31 and Chapter 2, "ACCESS Procedure Reference," on page 11.

In examples that update DBF file data, examples that are re-run will not work the same because the data have been modified. In this case, submit the PCFFDBL.SAS file to re-create the PC files tables.

See Appendix 1, "Sample Data," on page 151 for all the PC files on which the access and view descriptors are defined. This appendix also includes the SAS data files that are used in this chapter, as well as the SAS statements that created them.

Running the Examples in This Chapter

The examples in this chapter use data in different PC file formats. The PC file data are identified in each example and any file-specific issues are described in the example.

The examples in this chapter show

- how to create access descriptors and view descriptors
- how to use the view descriptors in SAS procedures and DATA steps.

As you work through the examples, notice that you can create the descriptors in a number of 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 only need to create an access descriptor or a view descriptor one time per example. If you re-run the examples, you do not need to re-create the descriptors.

The macro file (PCFFMAC.SAS) provided with the files contains macros that enable any SAS/ACCESS interface for a PC format to create database-description statements. Use the macro file with PCFFDBL.SAS (creates PC files), PCFFSAMP.SAS (contains samples) and PCFFSCL.SAS (contains SAS/AF examples). To adapt the PCFFMAC.SAS file for use at your site, insert your PC file format in the first line of the code. See the comments in the PCFFMAC.SAS file for more information.

If you run the examples individually instead of running the entire examples file, you must preface them with LIBNAME statements to identify where your SAS data libraries are stored. In these examples, the libname DLIB is used for SAS data files; the libname SLIB is used for PROC SQL views; the libname ADLIB is used when creating access descriptors; and the libname VLIB is used when creating view descriptors.

The files that create the PC files tables, descriptors, and the examples are shipped with your SAS/ACCESS software. See "Sample Data in This Book" on page 6 for more information about these files.

Reviewing Variables

Before retrieving or updating the PC files data that are described by a view descriptor, you may 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 much the same way 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 CUSTOMERS.WK3 file.

```
options linesize=80;

proc access dbms=wk3;
   create adlib.customr.access;
   /* create access descriptor  */
   path="c:\sasdemo\customer.wk3";
   worksheet=a;
   range='a1..j22';
   getnames=yes;
   scantype=5;
   mixed=yes;
   assign=yes;
   rename customer = custnum;
   format firstorder date9.;
   list all;

   create vlib.usacust.view;
   /* create vlib.usacust view */
   select customer state zipcode name
          firstorder;
run;

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

Output 6.1 on page 59 shows the results of this example.

Output 6.1 Using the DATASETS Procedure with a View Descriptor

```
                          DATASETS PROCEDURE

      Data Set Name: VLIB.USACUST          Observations:          21
      Member Type:   VIEW                  Variables:             5
      Engine:        SASIOWK3              Indexes:               0
      Created:       .                     Observation Length:    83
      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     75    DATE9.    DATE9.      FIRSTORDER
  4    NAME        Char     60     15    $60.      $60.        NAME
  2    STATE       Char      2      8    $2.       $2.         STATE
  3    ZIPCODE     Char      5     10    $5.       $5.         ZIPCODE
```

As you can see from 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 names of the PC files columns. The FIRSTORDER column name has been truncated to the eight-character SAS variable name FIRSTORD because the SAS System uses only the first eight characters of a PC files column name when it assigns a default SAS variable name. See "ASSIGN" on page 16 for more information about how SAS variable names are assigned for PC files column names.

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 cannot use the MODIFY statement in the DATASETS procedure to change the attributes of a view descriptor.

For more information about the DATASETS procedure, see *SAS Procedures Guide*.

Charting Data

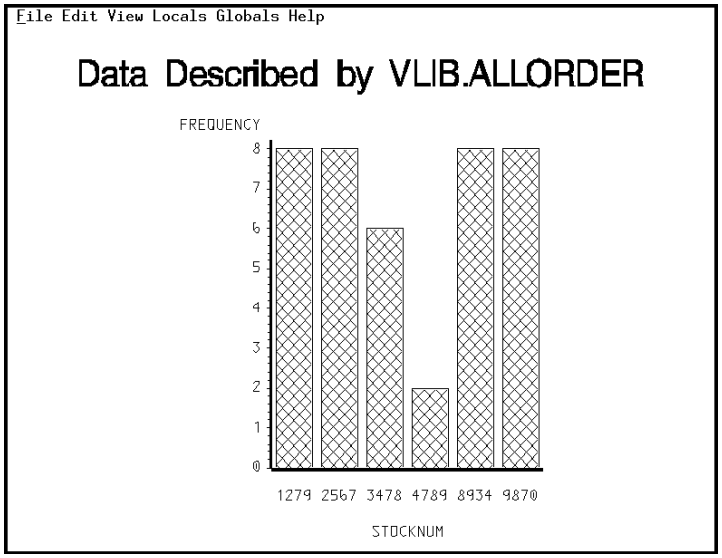
PROC GCHART programs work with data that are described by view descriptors just as they do with other SAS data sets. (Using this procedure requires a SAS/GRAPH license at your site.) The following example uses the view descriptor VLIB.ALLORDR to create a vertical bar chart of the number of orders per product. VLIB.ALLORDR describes the data in the ORDERS.XLS file. You need a SAS/GRAPH license to complete this exercise.

```
proc access dbms=xls;
   create adlib.order.access;
   /* create access descriptor   */
   path="c:\sasdemo\orders.xls";
   worksheet=sheet1;
   range='a1..j39';
   getnames=yes;
   scantype=5;
   mixed=yes;
   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.allordr.view;
   /* create vlib.allordr view */
   select all;
run;
```

```
proc gchart data=vlib.allordr;
   /* example                      */
   vbar stocknum / discrete;
title 'Data Described by VLIB.ALLORDR';
run;
```

Output 6.2 on page 61 shows the output for this example. STOCKNUM represents each product. The number of orders for each product is represented by the height of the bar.

Output 6.2 Vertical Bar Chart of Number of Orders per Product



For more information on the GCHART procedure, see *SAS/GRAPH Software: Reference*.

Calculating Statistics

You can also use SAS statistical procedures on PC files data. This section shows examples using the FREQ, MEANS, and RANK procedures.

Using the FREQ Procedure

Suppose you want to find the percentages of your invoices that went to each country so that you can decide where to increase your overseas marketing. The following example uses the view descriptor VLIB.INV to calculate the percentage of invoices for each country that appears in the INVOICE.DBF file.

```
proc access dbms=dbf;
   /* create access descriptor   */
   create adlib.invoice.access;
      path="c:\sasdemo\invoice.dbf";
   assign;
```

```
   rename invoicenum = invnum
          amtbilled = amtbilld ;
   format paidon      date9.
          invoicenum  5.0
          billedby    6.0;
   assign=yes;

   create vlib.inv.view;
   /* create vlib.inv view      */
   select invoicenum amtbilled
          country billedby paidon;
   list all;
run;

proc freq data=vlib.inv;
   /* example             */
   tables country;
   title 'Data Described by VLIB.INV';
run;
```

Output 6.3 on page 62 shows the one-way frequency table that this example generates.

Output 6.3 Frequency Table for Variable COUNTRY Described by View Descriptor VLIB.INV

Data Described by VLIB.INV 6

COUNTRY

COUNTRY	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Argentina	2	11.76	2	11.76
Australia	1	5.88	3	17.65
Brazil	4	23.53	7	41.18
USA	10	58.82	17	100.00

For more information on the FREQ procedure, see *SAS Language and Procedures: Usage* and *SAS Procedures Guide.*

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 ORDERS.XLS file, the view descriptor VLIB.USAORDR selects a subset of observations that have a SHIPTO value beginning with a 1, indicating a U.S. customer.

Using the OUT= option in the SORT procedure, the data from the DBF file are extracted, placed in a SAS data file, and then sorted.

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=xls;
    create adlib.order.access;
    /* create access descriptor    */
    path="c:\sasdemo\orders.xls";
    worksheet=shee1;
    getnames=yes;
    skiprows=2;
    scantype=5;
    mixed=yes;
    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;

proc sort data=vlib.usaordr out=work.usaorder;
    by shipto;
run;

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

Output 6.4 on page 64 shows the output for this example.

Output 6.4 Statistics on Fabric Length and Charges for Each U.S. Customer

```
                      Data Described by VLIB.USAORDR                      7

------------------------------- 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
          ------------------------------------------------------------
```

For more information on the MEANS procedure, see *SAS Procedures Guide*.

Using the RANK Procedure

You can also use more advanced statistical procedures on PC files data. The following example uses the RANK procedure to calculate the order of birthdays for a set of employees who are listed in the EMPLOYEES.DBF file. 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. The PRINT procedure then prints the data that are described by DLIB.RANKEXAM. You can also use the PRINT procedure to print all or some of the PC file data values described by view descriptors.

```
proc access dbms=dbf;
   create adlib.employ.access;
   /* create access descriptor  */
   path="c:\sasdemo\employees";
   drop salary;
   list all;

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

proc rank data=vlib.emps out=dlib.rankexam;
   /* example          */
   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;
```

Data stored in the DBF file must be extracted and placed in a SAS data set before they can be sorted with a SAS procedure. (This restriction also applies to data from other PC files.) The DROP= data set option is used in the PROC PRINT statement because the JOBCODE variable is not needed in the output. The JOBCODE variable is required in the SELECT statement so it can be used in the WHERE statement. The JOBCODE variable is then included in the view descriptor, even though it is not needed in the output. Output 6.5 on page 66 shows the result of this example.

Output 6.5 Ranking of Employee Birthdays

Order of Dept 602 Employee Birthdays

OBS	EMPID	BIRTHDAT	LASTNAME	DATERANK
1	456910	24SEP1953	ARDIS	5
2	237642	13MAR1954	BATTERSBY	6
3	239185	28AUG1959	DOS REMEDIOS	7
4	321783	03JUN1935	GONZALES	2
5	120591	12FEB1946	HAMMERSTEIN	4
6	135673	21MAR1961	HEMESLY	8
7	456921	12MAY1962	KRAUSE	9
8	457232	15OCT1963	LOVELL	11
9	423286	31OCT1964	MIFUNE	12
10	216382	24JUL1963	PURINTON	10
11	234967	21DEC1967	SMITH	13
12	212916	29MAY1928	WACHBERGER	1
13	119012	05JAN1946	WOLF-PROVENZA	3

When you use the PRINT procedure, you may want to take advantage of the SAS data set option OBS=, which enables you to limit the number of observations to be processed. This option is especially useful when the view descriptor describes a large amount of data, the SAS data file is large, or when you just want to see an example of the output. You cannot use OBS= if the view descriptor contains a WHERE clause in the SUBSET statement.

For more information on RANK, on other advanced statistical procedures, and for the PRINT procedure, see *SAS Procedures Guide*. For more information on the OBS= and FIRSTOBS= options, see *SAS Language Reference: Dictionary*.

Selecting and Combining Data

For many of your SAS programs, you may need to combine data from more than one view descriptor or to manipulate data that are accessed by a specific view descriptor. The following sections describe how you can select and combine data

- using the WHERE statement in a DATA step
- using the SQL procedure to create a new PROC SQL view
- using the SQL procedure to join data from various sources
- using a summary function in a PROC SQL query to create a new column in the output

Using the WHERE Statement

Suppose you have a view descriptor VLIB.ALLINV that lists invoices for all customers; VLIB.ALLINV is based on the INVOICE.DBF file. You can use a SET statement to create a SAS data file that contains information on customers who have not paid their bills and whose bills amount to at least $300,000.

```
proc access dbms=dbf;
   create adlib.invoice.access;
   /* create access descriptor   */
   path="c:\sasdemo\invoice.dbf";
   assign=yes;
   rename invoicenum = invnum
          amtbilled  = amtbilld
          amountinus = amtinus;
   format paidon date9.
          billedon date9.
          invoicenum  5.0
          billedby    6.0
          amtbilled   15.2
          amountinus  15.2;
   list all;

   create vlib.allinv.view;
   /* create vlib.allinv view  */
   select all;
run;

data notpaid(keep=invnum billedto amtinus billedon);
    /* example  */
   set vlib.allinv;
   where paidon is missing and amtinus>=300000;
run;
```

In the DATA step's WHERE statement, be sure to use SAS variable names, not PC files column names. Output 6.6 on page 67 shows the result of the new temporary SAS data file WORK.NOTPAID.

```
proc print data=notpaid;
   format amtinus dollar20.2;
title 'High Bills--Not Paid';
run;
```

Output 6.6 WORK.NOTPAID Data File Created Using a SAS WHERE Statement

High Bills--Not Paid

OBS	INVNUM	BILLEDTO	AMTINUS	BILLEDON
1	11271	18543489	$11,063,836.00	05OCT1998
2	12102	18543489	$11,063,836.00	17NOV1998
3	11286	43459747	$11,063,836.00	10OCT1998
4	12051	39045213	$2,256,870.00	02NOV1998
5	12471	39045213	$2,256,870.00	27DEC1998
6	12476	38763919	$2,256,870.00	24DEC1998

The first line of the DATA step uses the KEEP= data set option. This option works with view descriptors just as it works with other SAS data sets; it specifies that you want to include only the listed variables in the new SAS data file WORK.NOTPAID. However, you can still use the other view descriptor variables in other statements within the DATA step.

The SAS WHERE statement includes two conditions to be met. First, it selects only observations that have a missing value for the PAIDON variable. Second, the SAS WHERE statement requires that the amount in each bill be higher than a certain figure. You need to be familiar with the PC files data so that you can determine reasonable values for these expressions. For information on the SAS WHERE statement, refer to *SAS Language: Reference*.

Using the SQL Procedure

The SQL procedure implements the Structured Query Language in the SAS System. The SQL procedure follows the SQL convention of using the terms column and row for variable and observation This section provides examples of using the SQL procedure with PC files data.

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 CUSTOMERS.WK3 and ORDERS.XLS files, respectively, and a SAS data file, DLIB.OUTOFSTK, which contains product names and numbers that are out of stock. 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.

```
proc access dbms=wk3;
   create adlib.customr.access;
   /* create access descriptor  */
   path="c:\sasdemo\customers.wk3";
   worksheet=v;
   range='cus_phone';
   getnames=yes;
   skiprows=2;
   scantype=5;
   mixed=yes;
   list all;

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

proc access dbms=xls;
   create adlib.orders.access;
```

```
   /* create access descriptor  */
   path="c:\sasdemo\orders.xls";
   worksheet='sheet1';
   range='a1..j39';
   getnames=yes;
   skiprows=2;
   scantype=5;
   mixed=yes;
   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;
   /* example              */
   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;
```

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.

It is recommended that you *not* include an ORDER BY clause in a CREATE VIEW statement. Doing so causes the output data to be sorted every time the PROC SQL view is submitted, which may 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;
quit;
```

This SELECT statement uses the PROC SQL view SLIB.BADORDR to display joined WK3 and XLS data and SAS data in ascending order by the CUSTNUM column and then by the PRODUCT (that is, FIBERNAM) column. The data are ordered by PRODUCT because one customer may 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 6.7 on page 70 shows the data described by the SLIB.BADORDR view.

Output 6.7 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 PC files column names like CUSTOMER. By default, PROC SQL displays SAS variable labels, which default to PC files column names. (You can use the NOLABEL option to change this default.)

Creating New Columns with the GROUP BY Clause

Instead of creating a new PROC SQL view, you may 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 are 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 EMPLOYEE.DBF file. 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 DIF file data accessed by WORK.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=dbf;
   /* create access descriptor */
   create adlib.employ.access;
   path="c:\sasdemo\employee.dbf";
   assign=yes;
   format empid     6.0
          salary    dollar12.2
          jobcode   5.0
          birthdate date9.
          hiredate  date9.;
```

```
   list all;
run;

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

/* assign a password */
proc datasets library=work  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 work.allemp(pw=money)
      where dept like 'ACC%'
      group by dept;
quit;
```

The columns are displayed in the order specified in the SELECT clause of the query. Output 6.8 on page 71 shows the result of the query.

Output 6.8 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 descriptor or any SAS data set, put a slash after the password:

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

For more information about SAS System passwords, see "SAS System Passwords for SAS/ACCESS Descriptors" on page 14.

Reading PC Files Data into a SAS/AF Application

The following example shows you how to use a view descriptor in a SAS/AF application just as you would any other SAS data set. In this example, you create an application by using SAS/AF software and Screen Control Language (SCL). The application enables you to enter an employee ID (EMPID in the EMPLOYEE.DBF file) in order to display the employee's name and phone extension.

Follow these steps to create a SAS/AF application that uses a view descriptor:

1 Use PROC ACCESS to create the access descriptor ADLIB.EMPLOY and view descriptor VLIB.ALLEMP.proc access dbms=dbf;

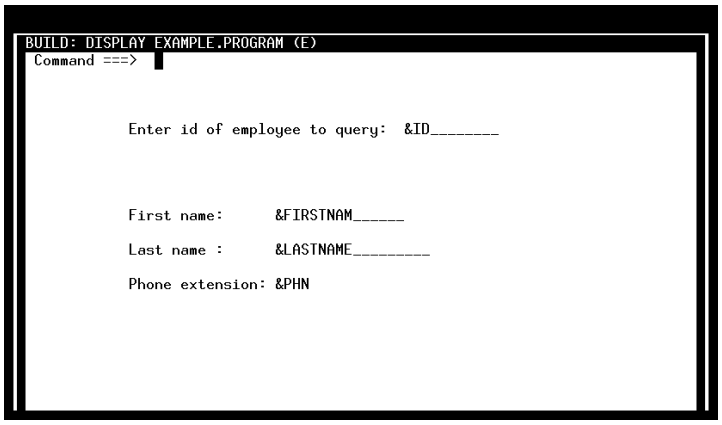
```
/* create access descriptor  */
create adlib.employ.access;
path='c:\sasdemo\employee.dbf';
assign=yes;
format empid     6.0
       salary    dollar12.2
       jobcode   5.0
       birthdate date9.
       hiredate  date9.;
list all;

/* create vlib.allemp view */
create vlib.allemp.view;

select all;
run;
```

2 To start building a SAS/AF PROGRAM entry for this application, submit the following statements:libname scllib *'SAS-data-library'*;

```
/* assign a libref */

 /* create the catalog entry */
proc build
 catalog=scllib.employee.example.program;
run;
```

3 Use the features available in the DISPLAY window to create the window shown in Display 6.1 on page 73. For more information about these features, refer to the online help information for SAS/AF software

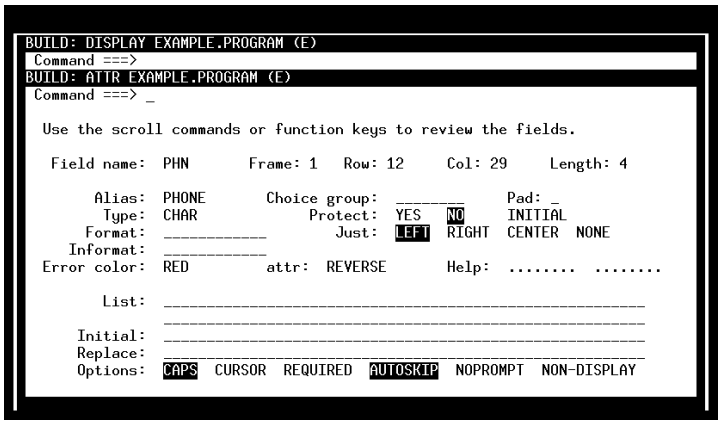
Display 6.1 DISPLAY Window for EXAMPLE Application



After designing the DISPLAY window, open the ATTR window (with the ATTR command) and change the default values for the **ID** and **PHN** attributes. The type for **ID** should be changed from **CHAR** to **NUM**.

The **&PHN** field is shown in Display 6.2 on page 73. Notice that the field name **PHN** is assigned to the variable **PHONE**, which is listed as the alias. This variable is only four characters wide (as shown in the **Length** field) but the name **PHONE** is five characters long. Therefore, a shorter field name is assigned–three characters (**PHN**) plus the **&**. For more information about window attributes, refer to the online help information for SAS/AF software.

Display 6.2 ATTR Window for EXAMPLE Application



Issue the SOURCE command to open the SOURCE window so that you can enter the SCL program into it, as shown in Example Code 6.1 on page 73.

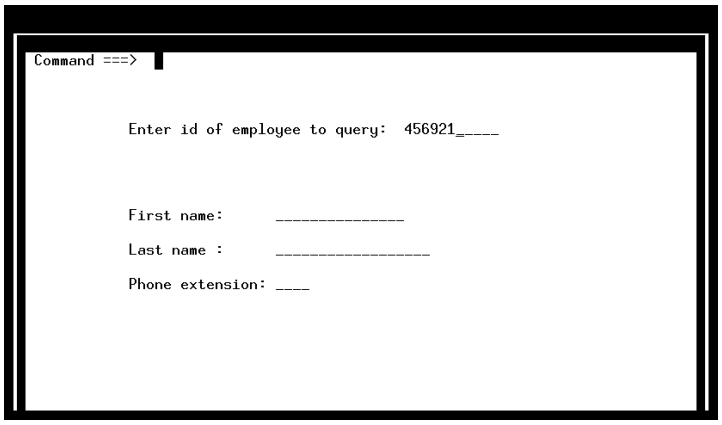
Example Code 6.1 SCL Program for the Source Window

```
/* SCL program to be added to the Source window */
INIT:

  /* Open the view descriptor VLIB.ALLEMP for reading (input mode) */
  dsid=open('vlib.allemp','i');
  call set(dsid);
```

```
  return;

MAIN:

  /* When an employee id is entered, subset the view    */
  /* descriptor and fetch in values for the employee's  */
  /* first name, last name, and phone extension         */
  if modified(id) and id ne _blank_ then
    do;
      rc=where(dsid,'empid = '||put(id,6.));
      rc=fetch(dsid);
      if rc ne 0 then
       _msg_ = 'Employee id not found. Please re-enter';
    end;
return;
TERM:

  /* Close the view descriptor                          */
  if (dsid > 0) then dsid=close(dsid);
return;
```

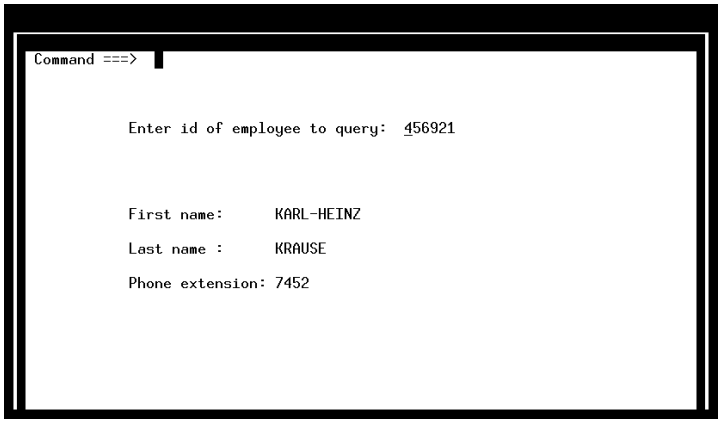
Compile and test the program. Your SAS/AF application window is displayed. Enter an employee's ID (EMPID) from EMPLOYEE.DBF, as shown in Display 6.3 on page 74.

Display 6.3 User's View of the Application



Press ENTER and the employee's name and phone extension are automatically supplied from the DBF file, as shown in Display 6.4 on page 75.

Display 6.4 PC Files Data Supplied by the Application



Browsing and Updating Data with SAS/FSP Procedures

If your site has SAS/FSP software as well as SAS/ACCESS software, you can use it to browse PC files data or to browse and update the data in DBF files. You have a choice of using three SAS/FSP procedures: FSBROWSE, FSEDIT, and FSVIEW.

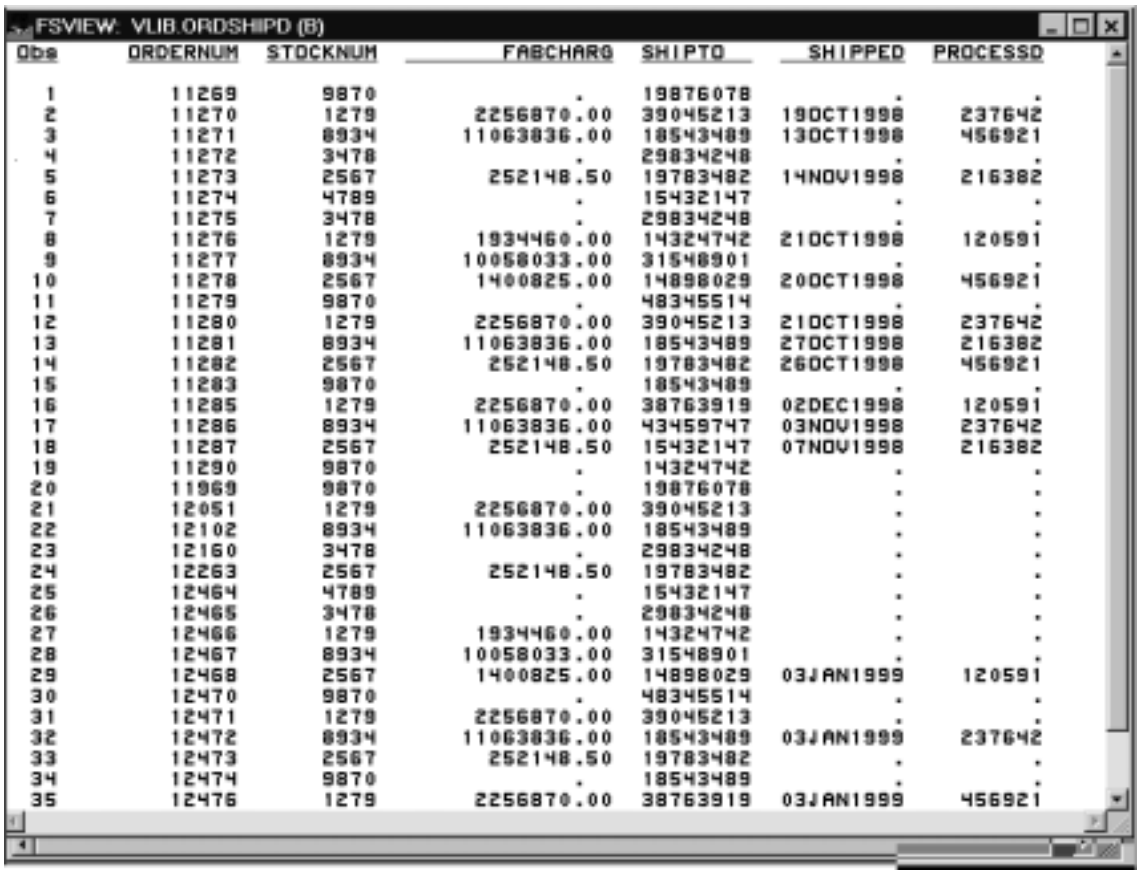
Note: Only DBF files can be edited. If you try to edit other PC file formats, SAS defaults to browse mode. △

Using the FSVIEW Procedure to Browse PC File Data

All of the SAS/FSP procedures work in a similar way and therefore only the FSVIEW procedure is shown. The FSVIEW procedure enables you to browse or edit data from a view descriptor or other SAS data set. The data are displayed in a tabular format similar to the format of the PRINT procedure output. You specify the PROC FSVIEW statement in the PROGRAM EDITOR window. Display 6.5 on page 76 creates a view descriptor, VLIB.ORDSHIPD, and specifies it in a PROC FSVIEW statement:

```
proc access dbms=dbf;
create adlib.orders.access;
path='c:\sasdemo\orders.dbf';
assign=yes;
drop delete_flg specinstr;
format shipped=date9.;
list all;
create vlib.ordshipd.view;
select ordernum stocknum fabcharges
  shipto shipped processby;
run;

proc fsview data=vlib.ordshipd;
run;
```

Display 6.5 Browsing Data in an FSVIEW Window



FSVIEW: VLIB.ORDSHIPD (B)

Obs	ORDERNUM	STOCKNUM	FABCHARG	SHIPTO	SHIPPED	PROCESSD
1	11269	9870	.	19876078	.	.
2	11270	1279	2256870.00	39045213	19OCT1998	237642
3	11271	8934	11063836.00	18543489	13OCT1998	456921
4	11272	3478	.	29834248	.	.
5	11273	2567	252148.50	19783482	14NOV1998	216382
6	11274	4789	.	15432147	.	.
7	11275	3478	.	29834248	.	.
8	11276	1279	1934460.00	14324742	21OCT1998	120591
9	11277	8934	10058033.00	31548901	.	.
10	11278	2567	1400825.00	14898029	20OCT1998	456921
11	11279	9870	.	48345514	.	.
12	11280	1279	2256870.00	39045213	21OCT1998	237642
13	11281	8934	11063836.00	18543489	27OCT1998	216382
14	11282	2567	252148.50	19783482	26OCT1998	456921
15	11283	9870	.	18543489	.	.
16	11285	1279	2256870.00	38763919	02DEC1998	120591
17	11286	8934	11063836.00	43459747	03NOV1998	237642
18	11287	2567	252148.50	15432147	07NOV1998	216382
19	11290	9870	.	14324742	.	.
20	11969	9870	.	19876078	.	.
21	12051	1279	2256870.00	39045213	.	.
22	12102	8934	11063836.00	18543489	.	.
23	12160	3478	.	29834248	.	.
24	12263	2567	252148.50	19783482	.	.
25	12464	4789	.	15432147	.	.
26	12465	3478	.	29834248	.	.
27	12466	1279	1934460.00	14324742	.	.
28	12467	8934	10058033.00	31548901	.	.
29	12468	2567	1400825.00	14898029	03JAN1999	120591
30	12470	9870	.	48345514	.	.
31	12471	1279	2256870.00	39045213	.	.
32	12472	8934	11063836.00	18543489	03JAN1999	237642
33	12473	2567	252148.50	19783482	.	.
34	12474	9870	.	18543489	.	.
35	12476	1279	2256870.00	38763919	03JAN1999	456921

Using the FSVIEW Procedure to Update PC File Data

If you are not a file's owner or creator, you must be granted update permissions on that file before you can edit it. Consult your site's DBA for more information about permissions.

There are two ways to use the FSVIEW procedure to open a window in editing mode. The most direct is to specify the **edit** command in the PROC FSVIEW statement:

```
proc fsview data=vlib.ordshipd edit;
run;
```

If you open an FSVIEW window in browse mode and then decide to edit the DBF data, you can follow these steps to change to edit mode:

1. Select the SAS icon in the top, left corner and then select the **Menu** item. Doing so opens a list of menus.
2. Select the **Edit** menu and then select the **Update...** item. An Update window opens and asks whether you want record-level or member-level locking on the DBF file.

You can now edit the DBF data. For information about using PROC FSVIEW, select the **Using this Window** item from the **Help** menu.

Using the SAS Viewer on PC File Data

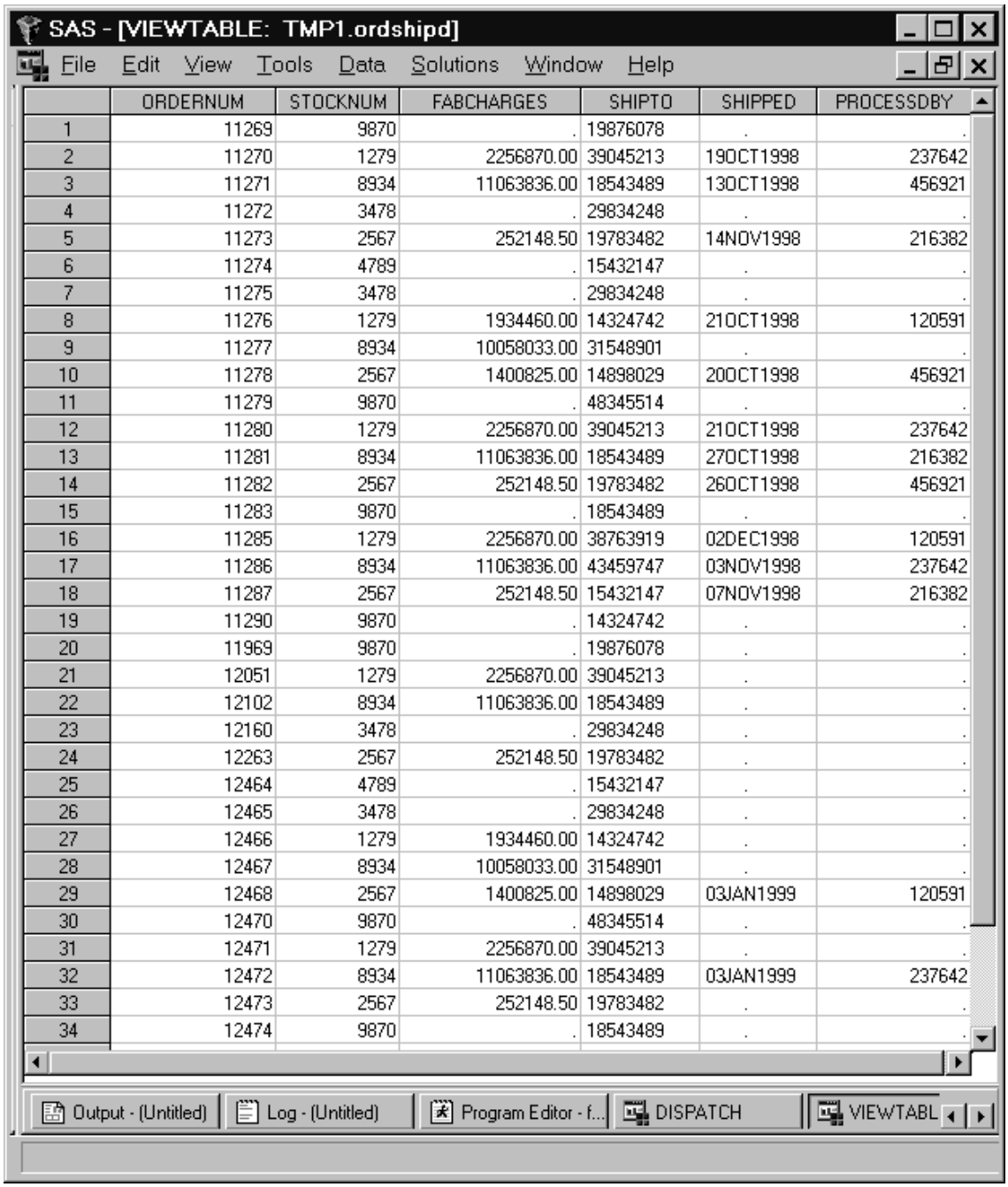
While your DBF data is displayed in an FSVIEW window, you can save it to a data file and then re-open that file using the SAS Viewer (VIEWTABLE window). Take these steps to save your FSVIEW output to a data file:

1. Select the SAS icon in the top, left corner and then select the **Menu** item. Doing so opens a listing of menus.
2. Select the **File** menu and then the **Save As** item.
3. A Save As window opens and asks you for the directory and filename information for the file that you want to save. In the **Save as type:** field, click on the down button to select **Data Files**.
4. Press the Save button. In this example, the output is stored to a file named VLIB.ORDSHP.

When your file is saved, you can go to the SAS Explorer window and double-click on the *libref.name* of your new file, in this case, VLIB.ORDSHP. Doing so opens the VIEWTABLE window, as shown in Display 6.6 on page 78:

You can browse or edit the PC file data from the VIEWTABLE window. For information about using this window, select the **Using this Window** item from the **Help** menu.

Display 6.6 Browsing or Editing Data through the VIEWTABLE Window



	ORDERNUM	STOCKNUM	FABCHARGES	SHIPTO	SHIPPED	PROCESSDBY
1	11269	9870	.	19876078	.	.
2	11270	1279	2256870.00	39045213	19OCT1998	237642
3	11271	8934	11063836.00	18543489	13OCT1998	456921
4	11272	3478	.	29834248	.	.
5	11273	2567	252148.50	19783482	14NOV1998	216382
6	11274	4789	.	15432147	.	.
7	11275	3478	.	29834248	.	.
8	11276	1279	1934460.00	14324742	21OCT1998	120591
9	11277	8934	10058033.00	31548901	.	.
10	11278	2567	1400825.00	14898029	20OCT1998	456921
11	11279	9870	.	48345514	.	.
12	11280	1279	2256870.00	39045213	21OCT1998	237642
13	11281	8934	11063836.00	18543489	27OCT1998	216382
14	11282	2567	252148.50	19783482	26OCT1998	456921
15	11283	9870	.	18543489	.	.
16	11285	1279	2256870.00	38763919	02DEC1998	120591
17	11286	8934	11063836.00	43459747	03NOV1998	237642
18	11287	2567	252148.50	15432147	07NOV1998	216382
19	11290	9870	.	14324742	.	.
20	11969	9870	.	19876078	.	.
21	12051	1279	2256870.00	39045213	.	.
22	12102	8934	11063836.00	18543489	.	.
23	12160	3478	.	29834248	.	.
24	12263	2567	252148.50	19783482	.	.
25	12464	4789	.	15432147	.	.
26	12465	3478	.	29834248	.	.
27	12466	1279	1934460.00	14324742	.	.
28	12467	8934	10058033.00	31548901	.	.
29	12468	2567	1400825.00	14898029	03JAN1999	120591
30	12470	9870	.	48345514	.	.
31	12471	1279	2256870.00	39045213	.	.
32	12472	8934	11063836.00	18543489	03JAN1999	237642
33	12473	2567	252148.50	19783482	.	.
34	12474	9870	.	18543489	.	.

Reading and Updating Data with the SQL Procedure

The SAS System's SQL procedure enables you to retrieve data from PC files and update data in DBF files. You can read and display PC files data by specifying a view descriptor or other SAS data set in the SQL procedure's SELECT statement.

To update DBF data, you can specify view descriptors in the SQL procedure's INSERT, DELETE, and UPDATE statements. You can also use these statements to

modify SAS data files. However, the ability to update data in a DBF file 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.
- If you did not create the DBF file, you must be granted the appropriate file access privileges before you can select, insert, delete, or update the data.
- You must also be granted the appropriate file access privileges before you select the data from DIF, WK*n*, or XLS files. The SAS/ACCESS interface to these files is read-only, so the SELECT statement is the only one of the four PROC SQL statements (in this section) that can reference a view descriptor based on DIF, WK*n*, or XLS data.

A summary of some of the SQL procedure statements follows:

SELECT	retrieves, manipulates, and displays PC file data that is described by a view descriptor. **SELECT** can also use data described by a PROC SQL or DATA step view or data in a SAS data file. A SELECT statement is usually referred to as a *query* because it queries the table for information.
DELETE	deletes rows from a SAS data file or from a DBF file that is described by a view descriptor. When you reference a view descriptor that is based on a DBF file in the DELETE statement, the records in the DBF file are marked for deletion.
INSERT	inserts rows into a DBF file or a SAS data file.
UPDATE	updates the data values in a DBF file or in a SAS data file.

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 PC files column names. If you prefer to use SAS variable names in your output, specify NOLABEL in the OPTIONS statement.

For more information about this procedure, its options, and example, see the SQL procedure chapter in *SAS Procedures Guide*.

Reading Data with the SQL Procedure

You can use the SQL procedure's SELECT statement to display PC files data that are described by a view descriptor or by another SAS data set. In the following example, the query uses the VLIB.PRODUCT view descriptor to retrieve a subset of the data in the SPECPROD.DIF file.

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.

```
proc access dmbs=dif;
   create adlib.product.access;
   /* create access descriptor  */
   path="c:\sasdemo\specprod.dif";
   diflabel;
   assign=yes;
   rename productid prodid;
```

```
   format productid  4.
          weight     e16.9
          fibersize  e20.13
          width      e16.9;
   list all;

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

options nodate linesize=120;
  title 'DIF File Data Retrieved with a SELECT
         Statement';

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

Output 6.9 on page 80 displays the query's output. Notice that the SQL procedure displays the DIF file's column names, not the SAS variable names.

Output 6.9 PC Files Data Retrieved with a PROC SQL Query

DIF File 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 DBF file, as illustrated by the following example. Because you are referencing a view descriptor, you use the SAS variable names in the UPDATE statement; however, the SQL procedure displays the DBF column names.

```
proc sql;
 update vlib.empeeoc
   set salary=26678.24,
       gender='M',
```

```
        birthdat='28AUG1959'd
    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 6.10 on page 81 displays the updated row of data retrieved from the view descriptor VLIB.EMPEEOC.

Output 6.10 DBF File 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         28AUG1959  VARGAS
```

Deleting Data with the SQL Procedure

You can use the SQL procedure's DELETE statement to delete rows from a DBF file. In the following example, the row that contains the value 346917 in the EMPID column is deleted from the EMPLOYEE.DBF.

```
proc sql;
  delete from vlib.empeeoc
   where empid=346917;
quit;
```

The deleted observation is marked with an asterisk (*) in the DELETE_FLG field. This is the only indicator you have that a record in a DBF field has been marked for deletion. If you have a number of rows to delete, you could use a macro variable EMPID instead of the individual EMPID values. Doing so would enable you 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 the DBF file. △

Inserting Data with the SQL Procedure

You can use the SQL procedure's INSERT statement to add rows to a DBF file. In the following example, the row that contains the value 346917 in the EMPID column is inserted back into the EMPLOYEE.DBF file.

```
proc sql;
  insert into vlib.allemp
   values('',346917,'02MAR1987'd,46000.33,'SHP013',204,
          'F','15MAR1950'd,'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 6.11 on page 82:

Output 6.11 Message Displayed in the SAS Log When a Row Is Inserted

```
6698
6699
6700
6701
6702
6703  proc sql;
6704  insert into vlib.allemp
6705     values('',346917,'02MAR1987'd,46000.33,
6706            'SHP013',204,'F','15MAR1950'd,
6707            'SHIEKELESLAM','SHALA','Y.',
6708            '8745');

NOTE: 1 row was inserted into VLIB.ALLEMP.

6709  quit;
```

Updating PC Files Data with the MODIFY Statement

The MODIFY statement extends the capabilities of the DATA step by enabling you to modify DBF file 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 DBF file.

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 ORDERS.DBF file. You also create a transaction data file, DLIB.TRANS, that you use to update the master data set (and therefore, the ORDERS.DBF 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.DBF table with data from the DLIB.TRANS data file. The SAS System reads one observation (or row) of the ORDERS.DBF 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 dmbs=dbf;
   /* create access descriptor  */
   create adlib.orders.access;

   path="c:\sasdemo\orders.dbf";
   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;
   ordernum=12102;
   /*Obs. 1 specifies Update for
     ORDERNUM=12102*/
      shipped='05DEC1998'd;
      type='U';
      output;

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

   ordernum=13000;
   /*Obs. 3 specifies Add for new
     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;

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

data vlib.master;
   /* MODIFY statement example */
   modify vlib.master dlib.trans;
   by ordernum;
   select (_iorc_);
      when (%sysrc(_dsenmr)) do;
   /* No match in MASTER - Add */
        if type='A'
           then output vlib.master;
        _error_ = 0;
      end;
      when (%sysrc(_sok)) do;
   /* Match located - Update or Delete */
        if type='U'
          then replace vlib.master;
        else if type='D'
          then remove vlib.master;
      end;
      otherwise do;
   /* Traps unexpected outcomes */
        put 'Unexpected ERROR condition:
            _IORC_ = ' _iorc_ ;
        put _all_;
   /* This dumps all vars in the PDV */
         _error_ = 0;
      end;
   end;
run;

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

proc print data=vlib.master;
   where ordernum in(12102 12160 13000 12465);
   title 'DBF File 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 that it displays only the observations that are included in the transaction data set. 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 6.12 on page 85.

Output 6.12 Revising PC Files Data with a MODIFY Statement

DBF File Data Updated with the MODIFY Statement

OBS	DELETE_F	ORDERNUM	STOCKNUM	LENGTH	FABCHARG	SHIPTO	DATEORD	SHIPPED	TAKENBY	PROCESBY
22		12102	8934	110	11063836.00	18543489	15NOV1998	05DEC1998	456910	.
23		12160	3478	1000	.	29834248	19NOV1998	.	456910	.
26	*	12465	3478	1000	.	29834248	23DEC1998	.	234967	.
39		13000	9870	650	.	19876078	18JAN1999	29JAN1999	321783	120591

OBS	SPECINST
22	
23	Customer agrees to pay in full.
26	
39	Customer agrees to certain limitations.

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 **05DEC1998**. The second observation sets the value to missing. If the value of SHIPPED was not set to missing in the second observation, the value **05DEC1998** would be incorrectly supplied. Therefore, you may want to create your transaction data set in a particular order to minimize having to reset variables.

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

- You cannot use the POINT= option because observation numbers are not available in a DBF file.
- The NOBS= option displays the largest positive integer value available on the host operating system.
- Each PC files statement that is issued, whether an INSERT, DELETE, or UPDATE, is a separate transaction and is saved in the DBF file. You cannot undo (or reverse) these changes without re-editing.

For more information about the MODIFY statement, see *SAS Language Reference: Dictionary*.

Updating a SAS Data File with PC Files Data

You can update a SAS data file with DBF file data that are 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 151 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 EMPLOYEE.DBF file. To perform this update, enter the following SAS statements:

```
options linesize=80;

proc access dbms=dbf;
   create adlib.employ.access;
   /* create access descriptor   */
   path="c:\sasdemo\employee.dbf";
   assign=yes;
   format empid 6.
          salary dollar12.2
          jobcode 5.
          hiredate date9.
          birthdate date9.;
   list all;

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

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

proc print data=dlib.birthday;
   /* examples                */
   format birthdat date9.;
   title 'DLIB.BIRTHDAY Data File';
run;

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

proc sort data=vlib.empbday out=work.empbirth;
   by lastname;
run;

data dlib.newbday;
   update dlib.birthday work.empbirth;
   by lastname;
run;
```

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

In this example, a PROC SORT statement with the OUT= option extracts DBF file data, places them in the SAS data file WORK.EMPBIRTH, and sorts the data by the LASTNAME variable. (When using a DATA step, PC files data must be extracted before you can sort them.) When the UPDATE statement references the SAS data file WORK.EMPBIRTH and you use a BY statement 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 DBF 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 6.13 on page 87, Output 6.14 on page 88, and Output 6.15 on page 89 show the results of the PRINT procedures.

Output 6.13 Data File to Be Updated, DLIB.BIRTHDAY

DLIB.BIRTHDAY Data File

OBS	EMPID	BIRTHDAT	LASTNAME
1	254196	06APR1949	CHANG
2	459288	05JAN1934	JOHNSON
3	127815	25DEC1943	WOLOSCHUK

Output 6.14 DBF File Data Described by the View Descriptor VLIB.EMPBDAY

Data Described by VLIB.EMPBDAY

OBS	EMPID	BIRTHDAT	LASTNAME	FIRSTNAM	PHONE
1	119012	05JAN1946	WOLF-PROVENZA	G.	3467
2	120591	12FEB1946	HAMMERSTEIN	S.	3287
3	123456	28AUG1959	VARGAS	CHRIS	
4	127845	25DEC1943	MEDER	VLADIMIR	6231
5	129540	31JUL1960	CHOULAI	CLARA	3921
6	135673	21MAR1961	HEMESLY	STEPHANIE	6329
7	212916	29MAY1928	WACHBERGER	MARIE-LOUISE	8562
8	216382	24JUL1963	PURINTON	PRUDENCE	3852
9	234967	21DEC1967	SMITH	GILBERT	7274
10	237642	13MAR1954	BATTERSBY	R.	8342
11	239185	28AUG1959	DOS REMEDIOS	LEONARD	4892
12	254896	06APR1949	TAYLOR-HUNYADI	ITO	0231
13	321783	03JUN1935	GONZALES	GUILLERMO	3642
14	328140	02JUN1951	MEDINA-SIDONIA	MARGARET	5901
15	346917	15MAR1950	SHIEKELESLAM	SHALA	8745
16	356134	25OCT1960	DUNNETT	CHRISTINE	4213
17	423286	31OCT1964	MIFUNE	YUKIO	3278
18	456910	24SEP1953	ARDIS	RICHARD	4351
19	456921	12MAY1962	KRAUSE	KARL-HEINZ	7452
20	457232	15OCT1963	LOVELL	WILLIAM	6321
21	459287	05JAN1934	RODRIGUES	JUAN	5879
22	677890	24APR1965	NISHIMATSU-LYNCH	CAROL	6245
23	346917	15MAR1950	SHIEKELESLAM	SHALA	

Output 6.15 Data in the Updated Data File DLIB.NEWBDAY

DLIB.NEWBDAY Data File

OBS	EMPID	BIRTHDAT	LASTNAME	FIRSTNAM	PHONE
1	456910	24SEP1953	ARDIS	RICHARD	4351
2	237642	13MAR1954	BATTERSBY	R.	8342
3	254196	06APR1949	CHANG		
4	129540	31JUL1960	CHOULAI	CLARA	3921
5	239185	28AUG1959	DOS REMEDIOS	LEONARD	4892
6	356134	25OCT1960	DUNNETT	CHRISTINE	4213
7	321783	03JUN1935	GONZALES	GUILLERMO	3642
8	120591	12FEB1946	HAMMERSTEIN	S.	3287
9	135673	21MAR1961	HEMESLY	STEPHANIE	6329
10	459288	05JAN1934	JOHNSON		
11	456921	12MAY1962	KRAUSE	KARL-HEINZ	7452
12	457232	15OCT1963	LOVELL	WILLIAM	6321
13	127845	25DEC1943	MEDER	VLADIMIR	6231
14	328140	02JUN1951	MEDINA-SIDONIA	MARGARET	5901
15	423286	31OCT1964	MIFUNE	YUKIO	3278
16	677890	24APR1965	NISHIMATSU-LYNCH	CAROL	6245
17	216382	24JUL1963	PURINTON	PRUDENCE	3852
18	459287	05JAN1934	RODRIGUES	JUAN	5879
19	346917	15MAR1950	SHIEKELESLAM	SHALA	8745
20	234967	21DEC1967	SMITH	GILBERT	7274
21	254896	06APR1949	TAYLOR-HUNYADI	ITO	0231
22	123456	28AUG1959	VARGAS	CHRIS	
23	212916	29MAY1928	WACHBERGER	MARIE-LOUISE	8562
24	119012	05JAN1946	WOLF-PROVENZA	G.	3467
25	127815	25DEC1943	WOLOSCHUK		

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 PC files data described by one view descriptor to another, or you can append a SAS data file. Because the SAS/ACCESS interface to DIF, WK*n*, and XLS files is read-only, you cannot append data *to* those files. You can however, append data *from* them to a DBF file or to a SAS data file.

The following example uses the APPEND procedure's FORCE option to append a SAS data file with extra variables to the data file referenced by the view descriptor VLIB.SQLEMPS. You must have DBF INSERT privileges in order to add rows to the EMPLOYEES.DBF file.

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 PC files column is defined as NOT NULL, some restrictions apply when appending data. For more information, see the APPEND procedure in *SAS Procedures Guide*.

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.

```
proc access dbms=dbf;
  /* create access descriptor    */
  create adlib.employ.access;
  path='c:\sasdemo\employee.dbf';
  assign=no;
  drop salary;
  list all;

  create vlib.sqlemps.view;
  /* create view descriptor    */
  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 6.16 on page 91, and the SAS data file DLIB.TEMPEMPS is displayed in Output 6.17 on page 91.

Output 6.16 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	346917	02MAR1987	SHIEKELESLAM	SHALA	Y.
16	356134	14JUN1985	DUNNETT	CHRISTINE	MARIE
17	423286	19DEC1988	MIFUNE	YUKIO	TOSHIRO
18	456910	14JUN1978	ARDIS	RICHARD	BINGHAM
19	456921	19AUG1987	KRAUSE	KARL-HEINZ	G.
20	457232	15JUL1985	LOVELL	WILLIAM	SINCLAIR
21	459287	02NOV1964	RODRIGUES	JUAN	M.
22	677890	12DEC1988	NISHIMATSU-LYNCH	CAROL	ANNE
23	346917	02MAR1987	SHIEKELESLAM	SHALA	Y.

Output 6.17 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 >= '01JAN1998'd;
run;

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

Output 6.18 on page 92 shows VLIB.SQLEMPS with three rows from DLIB.TEMPEMPS appended to it.

Output 6.18 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	346917	02MAR1987	SHIEKELESLAM	SHALA	Y.
16	356134	14JUN1985	DUNNETT	CHRISTINE	MARIE
17	423286	19DEC1988	MIFUNE	YUKIO	TOSHIRO
18	456910	14JUN1978	ARDIS	RICHARD	BINGHAM
19	456921	19AUG1987	KRAUSE	KARL-HEINZ	G.
20	457232	15JUL1985	LOVELL	WILLIAM	SINCLAIR
21	459287	02NOV1964	RODRIGUES	JUAN	M.
22	677890	12DEC1988	NISHIMATSU-LYNCH	CAROL	ANNE
23	346917	02MAR1987	SHIEKELESLAM	SHALA	Y.
24	765111	04MAY1994	NISHIMATSU-LYNCH	RICHARD	ITO
25	765112	04MAY1998	SMITH	ROBERT	MICHAEL
26	219776	15APR1998	PASTORELLI	ZORA	
27	245233	10APR1998	ALI	SADIQ	H.
28	245234	10APR1998	MEHAILESCU	NADIA	P.
29	326721	01MAY1998	CALHOUN	WILLIS	BEAUREGARD

See Output 6.19 on page 93 for a copy of the SAS log screen and the messages about the FORCE option.

Output 6.19 SAS Log with Messages about the FORCE Option

```
10504
10505
10506
10507
10508
10509  proc append base=vlib.sqlemps
10510       data=dlib.tempemps force;
10511     where hiredate <= '30APR1998'd;
10512  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 SQL INSERT statement for the rows to be appended to the DBF file.

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

For more information on the APPEND procedure, see *SAS Procedures Guide*.

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

SAS/ACCESS® Software for PC File Formats: Reference, Verison 8

ISBN 1-58025–544–2

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

1st printing, October 1999