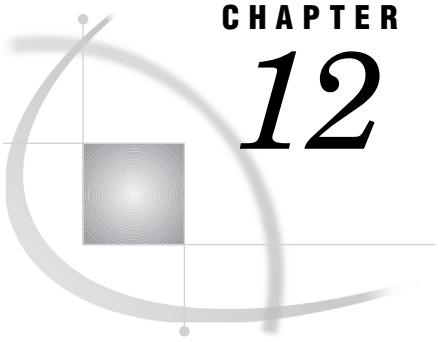


CHAPTER

# 12

# Using DBMS Data in Version 7 and Version 8



## Introduction

This topic contains examples of easier and more direct ways of using data stored in your database management system (DBMS), beginning in Version 7 of SAS/ACCESS software. These examples use ORACLE and DB2, but SAS/ACCESS also provides interfaces to many other widely used database management systems.

Refer to Chapter 13, "Using DBMS Data with the SQL Pass-Through Facility," on page 179 for SAS/ACCESS examples that use the SQL Procedure Pass-Through Facility, access descriptors, view descriptors, and the ACCESS and DBLOAD procedures.

## Running the Examples in This Section

The examples in this section all use the SAS/ACCESS LIBNAME statement to associate a libref directly with DBMS objects. When you assign a libref in SAS/ACCESS, you can customize the way your data is processed by using SAS LIBNAME statement options, including the new SAS/ACCESS LIBNAME options listed in "SAS/ACCESS LIBNAME Statement" on page 27 as well as additional DBMS-specific LIBNAME statement options listed in your DBMS chapter.

When you use the LIBNAME statement to create a libref that refers to DBMS data, you can refer to the DBMS objects within the libref, such as tables and views, by using the *libref.object-name* syntax. Because these objects are treated by SAS software as SAS data sets, you can specify how they are processed by using SAS data set options, including the new SAS/ACCESS data set options listed in "SAS/ACCESS Data Set Options" on page 43 and additional DBMS-specific data set options listed in your DBMS chapter.

If you specify a data set option that has the same name as a LIBNAME option that was specified during LIBNAME assignment, the data set option overrides the LIBNAME option.

The files that create the DBMS tables, descriptors, and the examples are shipped with your SAS/ACCESS software. See "Sample Data in This Book" on page 8 for more information about these files.

*Note:* The examples in this chapter use the SAS/ACCESS Interface to DB2 and the SAS/ACCESS Interface to ORACLE. Because the connection arguments, such as USER=, PASSWORD=, and DATABASE= vary depending on which DBMS you use, you must substitute the appropriate connection arguments for your DBMS if you run these examples on your DBMS. △

# Creating SAS Data Sets from DBMS Data

Once you have associated a SAS/ACCESS libref with your DBMS data, you can use the libref just as you would use any SAS libref. The following examples illustrate basic uses of the DATA step with librefs that refer to DBMS data.

## Using the PRINT Procedure with DBMS Data

In the following example, the libref MYDB2LIB is assigned with the DB2 engine to associate the libref with tables and views that reside on DB2. The PRINT procedure prints a phone list containing information for staff in New Jersey from the DB2 table STAFF. Information for staff from states other than New Jersey is not printed. The DB2 table STAFF is not modified.

Note that you can specify a libref that references DBMS data in the DATA= option.

```
libname mydb2lib db2 ssid=db2;

proc print data=mydb2lib.staff
  (keep=lname fname hphone state);
  where state = 'NJ';
```

```
  title 'New Jersey Phone List';
run;
```

Output for this example is shown in Output 12.1 on page 153.

**Output 12.1** Output Listing from the PRINT Procedure

```
                    New Jersey Phone List                    1

     Obs  LNAME            FNAME            STATE  HPHONE

       1  ALVAREZ           CARLOS            NJ    201/732-8787
       2  BAREFOOT          JOSEPH            NJ    201/812-5665
       3  DACKO             JASON             NJ    201/732-2323
       4  FUJIHARA          KYOKO             NJ    201/812-0902
       5  HENDERSON         WILLIAM           NJ    201/812-4789
       6  JOHNSON           JACKSON           NJ    201/732-3678
       7  LAWRENCE          KATHY             NJ    201/812-3337
       8  MURPHEY           JOHN              NJ    201/812-4414
       9  NEWKIRK           SANDRA            NJ    201/812-3331
      10  NEWKIRK           WILLIAM           NJ    201/732-6611
      11  PETERS            RANDALL           NJ    201/812-2478
      12  RHODES            JEREMY            NJ    201/812-1837
      13  ROUSE             JEREMY            NJ    201/732-9834
      14  VICK              THERESA           NJ    201/812-2424
      15  YANCEY            ROBIN             NJ    201/812-1874
```

## Combining DBMS Data and SAS Data

The following example shows how to read DBMS data into SAS and create additional variables to perform calculations or subsetting operations on the data.

This example creates the SAS data set WORK.HIGHWAGE from the DB2 table PAYROLL and adds a new variable, CATEGORY. The CATEGORY variable is based on the value of the salary column in the DB2 table PAYROLL. The PAYROLL table is not modified.

```
libname mydb2lib db2 ssid=db2;

data highwage;
  set mydb2lib.payroll(drop=sex birth hired);
  if salary>60000 then
    CATEGORY="High";
  else if salary<30000 then
    CATEGORY="Low";
  else
    CATEGORY="Avg";
run;

options obs=20;

proc print data=highwage;
  title "Salary Analysis";
  format salary dollar10.2;
run;
```

Partial output for this example is shown in Output 12.2 on page 153.

**Output 12.2** Combining DBMS Data and SAS Data

Salary Analysis 1

| OBS | IDNUM | JOBCODE | SALARY | CATEGORY |
|---|---|---|---|---|
| 1 | 1919 | TA2 | $34,376.00 | Avg |
| 2 | 1653 | ME2 | $35,108.00 | Avg |
| 3 | 1400 | ME1 | $29,769.00 | Low |
| 4 | 1350 | FA3 | $32,886.00 | Avg |
| 5 | 1401 | TA3 | $38,822.00 | Avg |
| 6 | 1499 | ME3 | $43,025.00 | Avg |
| 7 | 1101 | SCP | $18,723.00 | Low |
| 8 | 1333 | PT2 | $88,606.00 | High |
| 9 | 1402 | TA2 | $32,615.00 | Avg |
| 10 | 1479 | TA3 | $38,785.00 | Avg |
| 11 | 1403 | ME1 | $28,072.00 | Low |
| 12 | 1739 | PT1 | $66,517.00 | High |
| 13 | 1658 | SCP | $17,943.00 | Low |
| 14 | 1428 | PT1 | $68,767.00 | High |
| 15 | 1782 | ME2 | $35,345.00 | Avg |
| 16 | 1244 | ME2 | $36,925.00 | Avg |
| 17 | 1383 | BCK | $25,823.00 | Low |
| 18 | 1574 | FA2 | $28,572.00 | Low |
| 19 | 1789 | SCP | $18,326.00 | Low |
| 20 | 1404 | PT2 | $91,376.00 | High |

## Reading Data from Multiple DBMS Tables

You can use the DATA step to read data from multiple data sets, in this case, two DBMS tables. This example merges data from the two ORACLE tables STAFF and SUPERV in the SAS data set WORK.COMBINED. Notice that the PATH= statement includes an alias to the database, as required by ORACLE SQL*NET software.

```
libname myoralib oracle user=karin password=haggis
  path='airhrdata' schema=airport
  preserve_col_names=yes;

data combined;
  merge myoralib.staff myoralib.superv(in=super
    rename=(supid=idnum));
  by idnum;
  if super;
run;

proc print data=combined;
  title "Supervisor Information";
run;
```

*Note:* The PRESERVE_COL_NAMES=YES LIBNAME option retains the lowercased column names from ORACLE when creating the corresponding SAS variable names. For information on additional new LIBNAME and data set options, see "SAS/ACCESS LIBNAME Statement" on page 27 and "SAS/ACCESS Data Set Options" on page 43. △

Output for this example is shown in Output 12.3 on page 154.

**Output 12.3** Reading Data from Multiple DBMS Tables

Supervisor Information 1

| Obs | idnum | lname | fname | city | state | hphone | jobcat |
|---|---|---|---|---|---|---|---|
| 1 | 1106 | MARSHBURN | JASPER | STAMFORD | CT | 203/781-1457 | PT |
| 2 | 1118 | DENNIS | ROGER | NEW YORK | NY | 718/383-1122 | PT |
| 3 | 1126 | KIMANI | ANNE | NEW YORK | NY | 212/586-1229 | TA |
| 4 | 1352 | RIVERS | SIMON | NEW YORK | NY | 718/383-3345 | NA |
| 5 | 1385 | RAYNOR | MILTON | BRIDGEPORT | CT | 203/675-2846 | ME |
| 6 | 1401 | ALVAREZ | CARLOS | PATERSON | NJ | 201/732-8787 | TA |
| 7 | 1405 | DACKO | JASON | PATERSON | NJ | 201/732-2323 | SC |
| 8 | 1417 | NEWKIRK | WILLIAM | PATERSON | NJ | 201/732-6611 | NA |
| 9 | 1420 | ROUSE | JEREMY | PATERSON | NJ | 201/732-9834 | ME |
| 10 | 1431 | YOUNG | DEBORAH | STAMFORD | CT | 203/781-2987 | FA |
| 11 | 1433 | YANCEY | ROBIN | PRINCETON | NJ | 201/812-1874 | FA |
| 12 | 1442 | NEWKIRK | SANDRA | PRINCETON | NJ | 201/812-3331 | PT |
| 13 | 1564 | WALTERS | ANNE | NEW YORK | NY | 212/587-3257 | SC |
| 14 | 1639 | CARTER-COHEN | KAREN | STAMFORD | CT | 203/781-8839 | TA |
| 15 | 1677 | KRAMER | JACKSON | BRIDGEPORT | CT | 203/675-7432 | BC |
| 16 | 1834 | LEBLANC | RUSSELL | NEW YORK | NY | 718/384-0040 | BC |
| 17 | 1882 | TUCKER | ALAN | NEW YORK | NY | 718/384-0216 | ME |
| 18 | 1935 | FERNANDEZ | KATRINA | BRIDGEPORT | CT | 203/675-2962 | NA |
| 19 | 1983 | DEAN | SHARON | NEW YORK | NY | 718/384-1647 | FA |

## Using the DATA Step's UPDATE Statment with DBMS Data

You can also use the DATA step's UPDATE statement to create a SAS data set with DBMS data. This example creates the SAS data set WORK.PAYROLL with data from the ORACLE tables PAYROLL and PAYROLL2. The ORACLE tables are not modified.

Note that the columns in the two ORACLE tables must match; however, PAYROLL2 may have additional columns. Any additional columns in PAYROLL2 are added to the PAYROLL data set. Also, the UPDATE statement requires unique values for IDNUM to correctly merge the data from PAYROLL2.

```
libname myoralib oracle user=scott password=tiger path='myorapath';

data payroll;
  update myoralib.payroll
         myoralib.payroll2;
  by idnum;

proc print data=payroll;
  format birth datetime9. hired datetime9.;
  title 'Updated Payroll Data';
run;
```

Partial output from this example is shown in Output 12.4 on page 155.

**Output 12.4** Creating a SAS Data Set with DBMS Data by Using the UPDATE Statement

Updated Payroll Data 1

| Obs | IDNUM | SEX | JOBCODE | SALARY | BIRTH | HIRED |
|---|---|---|---|---|---|---|
| 1 | 1009 | M | TA1 | 28880 | 02MAR1959 | 26MAR1992 |
| 2 | 1017 | M | TA3 | 40858 | 28DEC1957 | 16OCT1981 |
| 3 | 1036 | F | TA3 | 42465 | 19MAY1965 | 23OCT1984 |
| 4 | 1037 | F | TA1 | 28558 | 10APR1964 | 13SEP1992 |
| 5 | 1038 | F | TA1 | 26533 | 09NOV1969 | 23NOV1991 |
| 6 | 1050 | M | ME2 | 35167 | 14JUL1963 | 24AUG1986 |
| 7 | 1065 | M | ME3 | 38090 | 26JAN1944 | 07JAN1987 |
| 8 | 1076 | M | PT1 | 69742 | 14OCT1955 | 03OCT1991 |
| 9 | 1094 | M | FA1 | 22268 | 02APR1970 | 17APR1991 |
| 10 | 1100 | M | BCK | 25004 | 01DEC1960 | 07MAY1988 |
| 11 | 1101 | M | SCP | 18723 | 06JUN1962 | 01OCT1990 |
| 12 | 1102 | M | TA2 | 34542 | 01OCT1959 | 15APR1991 |
| 13 | 1103 | F | FA1 | 23738 | 16FEB1968 | 23JUL1992 |
| 14 | 1104 | M | SCP | 17946 | 25APR1963 | 10JUN1991 |
| 15 | 1105 | M | ME2 | 34805 | 01MAR1962 | 13AUG1990 |
| 16 | 1106 | M | PT3 | 94039 | 06NOV1957 | 16AUG1984 |
| 17 | 1107 | M | PT2 | 89977 | 09JUN1954 | 10FEB1979 |
| 18 | 1111 | M | NA1 | 40586 | 14JUL1973 | 31OCT1992 |
| 19 | 1112 | M | TA1 | 26905 | 29NOV1964 | 07DEC1992 |
| 20 | 1113 | F | FA1 | 22367 | 15JAN1968 | 17OCT1991 |

# Using the SQL Procedure with DBMS Data

The previous examples showed how to read DBMS data and perform operations on the data in SAS. You can also perform operations on data directly in your DBMS by using the SQL procedure.

The following examples use the SQL procedure to query, update, and create DBMS tables.

## Querying a DBMS Table

This example uses the SQL procedure to query the ORACLE table PAYROLL. The PROC SQL query retrieves all job codes and provides a total salary amount for each job code.

```
libname myoralib oracle user=karin password=haggis
  path='airhrdata' schema=airport;

proc sql;
  select jobcode label='Jobcode',
         sum(salary) as total
         label='Total for Group'
         format=dollar11.2
  from myoralib.payroll
  group by jobcode;
quit;
```

Output for this example is shown in Output 12.5 on page 157.

**Output 12.5** Querying a DBMS Table

```
                    The SAS System    1

                                 Total for
                Jobcode              Group

                    BCK       $232,148.00
                    FA1       $253,433.00
                    FA2       $447,790.00
                    FA3       $230,537.00
                    ME1       $228,002.00
                    ME2       $498,076.00
                    ME3       $296,875.00
                    NA1       $210,161.00
                    NA2       $157,149.00
                    PT1       $543,264.00
                    PT2       $879,252.00
                    PT3        $21,009.00
                    SCP       $128,162.00
                    TA1       $249,492.00
                    TA2       $671,499.00
                    TA3       $476,155.00
```

The next example uses the SQL procedure to query flight information from the ORACLE table DELAY. The WHERE clause specifies that only flights to London and Frankfurt are retrieved.

```
libname myoralib oracle user=kurt password=freude
  path='fltdata' schema=airport;

 title 'Flights to London and Frankfurt';

 proc sql;
   select dates format=date9.,
     dest from myoralib.delay
   where (dest eq "FRA") or
     (dest eq "LON")
   order by dest;
quit;
```

*Note:* Interaction between the SQL procedure and the SAS/ACCESS engine ensures that both the WHERE clause and the ORDER BY clause are processed by the DBMS for optimized performance. △

Output for this example is shown in Output 12.6 on page 157.

**Output 12.6** Querying a DBMS Table

```
              Flights to London and Frankfurt

                          DATES  DEST

                       01MAR1998  FRA
                       04MAR1998  FRA
                       07MAR1998  FRA
                       03MAR1998  FRA
                       05MAR1998  FRA
                       02MAR1998  FRA
                       04MAR1998  LON
                       07MAR1998  LON
                       02MAR1998  LON
                       06MAR1998  LON
                       05MAR1998  LON
                       03MAR1998  LON
                       01MAR1998  LON
```

The next example uses the SQL procedure to query the DB2 table INTERNAT for information on international flights with over 200 passengers. Note that output can be sorted by using a PROC SQL query and that the TITLE, LABEL, and FORMAT key words are not ANSI standard SQL; they are SAS extensions that you can use in PROC SQL.

```
libname mydb2lib db2 ssid=db2;

proc sql;
   title  'International Flights by Flight Number';
   title2 'with Over 200 Passengers';
   select flight   label="Flight Number",
          dates    label="Departure Date"
                   format DATE9.,
          dest     label="Destination",
          boarded  label="Number Boarded"
     from mydb2lib.internat
    where boarded > 200
    order by flight;
quit;
```

Output for this example is shown in Output 12.7 on page 158.

**Output 12.7** Querying a DBMS Table

```
            International Flights by Flight Number
                  with Over 200 Passengers

          Flight  Departure                 Number
          Number       Date  Destination   Boarded
          ----------------------------------------
          219     04MAR1998  LON               232
          219     07MAR1998  LON               241
          622     07MAR1998  FRA               210
          622     01MAR1998  FRA               207
```

The next example uses the SQL procedure to query the DB2 table PAYROLL for information on all flight attendants, ordered by JOBCODE and SERVICE.

```
libname mydb2lib db2 ssid=db2;

proc sql;
   title  'Service Years and Salary';
   title2 'for Flight Attendants';
   select idnum     label='ID Number',
          jobcode   label='Job Code',
          salary    label='Salary'
                    format dollar7.,
          (today()-HIRED)/365.25 as service
                    label='Years Service'
                    format 4.1,
          hired     label='Hire Date'
                    format date9.
    from mydb2lib.payroll
    where jobcode like 'FA%'
    order by jobcode, service;
quit;
```

Partial output for this example is shown in Output 12.8 on page 159.

**Output 12.8** Querying a DBMS Table

```
                Service Years and Salary
                 for Flight Attendants

         ID      Job              Years
         Number  Code   Salary  Service  Hire Date
         ----------------------------------------
         1132    FA1   $22,413      3.9  22OCT1993
         1425    FA1   $23,979      4.6  28FEB1993
         1103    FA1   $23,738      5.2  23JUL1992
         1130    FA1   $23,916      5.3  05JUN1992
         1414    FA1   $23,644      5.4  12APR1992
         1113    FA1   $22,367      5.9  17OCT1991
         1094    FA1   $22,268      6.4  17APR1991
         1422    FA1   $22,454      6.5  06APR1991
         1116    FA1   $22,862      6.5  21MAR1991
         1970    FA1   $22,615      6.5  12MAR1991
```

## Querying Multiple DBMS Tables

You can also retrieve data from multiple DBMS tables in a single query by using the SQL procedure. This example joins the ORACLE tables STAFF and PAYROLL to query salary information for employees earning more than $40,000.

```
libname myoralib oracle user=michelle password=toys
  path='airhrdata' schema=hrdept;

title 'Employees with salary greater than $40,000';

options obs=20;

proc sql;
  select a.lname, a.fname, b.salary
    format=dollar10.2
  from myoralib.staff a, myoralib.payroll b
```

```
  where (a.idnum eq b.idnum) and
    (b.salary gt 40000);
quit;
```

*Note:* For optimized performance, the SAS/ACCESS engine passes the entire join to the DBMS for processing. △

Output for this example is shown in Output 12.9 on page 160.

**Output 12.9** Querying Multiple DBMS Tables

Employees with salary greater than $40,000

| LNAME | FNAME | SALARY |
|---|---|---|
| BAREFOOT | JOSEPH | $43,025.00 |
| BANADYGA | JUSTIN | $88,606.00 |
| BRANCACCIO | JOSEPH | $66,517.00 |
| BRADY | CHRISTINE | $68,767.00 |
| COHEN | LEE | $91,376.00 |
| CARTER-COHEN | KAREN | $40,260.00 |
| CASTON | FRANKLIN | $41,690.00 |
| FERNANDEZ | KATRINA | $51,081.00 |
| GRAHAM | ALVIN | $65,111.00 |
| GREGORSKI | DANIEL | $68,096.00 |
| HARRIS | CHARLES | $84,685.00 |
| HASENHAUER | CHRISTINA | $70,736.00 |
| HAVELKA | RAYMOND | $41,551.00 |
| HERRERO | CLYDE | $66,130.00 |
| KIMANI | ANNE | $40,899.00 |
| MARSHBURN | JASPER | $89,632.00 |
| MORGAN | ALFRED | $42,264.00 |
| NEWKIRK | SANDRA | $84,536.00 |
| NEWKIRK | WILLIAM | $52,270.00 |
| NEWTON | JAMES | $84,203.00 |

The next example uses the SQL procedure to join and query the DB2 tables MARCH, DELAY, and FLIGHT. The query retrieves information on delayed international flights during the month of March.

```
libname mydb2lib db2 ssid=db2;

title "Delayed International Flights in March";

proc sql;
  select distinct march.flight, march.dates,
      delay format=2.0
    from mydb2lib.march, mydb2lib.delay,
      mydb2lib.internat
  where march.flight=delay.flight and
      march.dates=delay.dates and
      march.flight=internat.flight and
      delay>0
  order by delay descending;
quit;
```

*Note:* For optimized performance, the SAS/ACCESS engine passes the entire join to the DBMS for processing. △

Output for this example is shown in Output 12.10 on page 161.

**Output 12.10** Querying Multiple DBMS Tables

Delayed International Flights in March

| FLIGHT | DATES | DELAY |
|---|---|---|
| 622 | 04MAR1998 | 30 |
| 219 | 06MAR1998 | 27 |
| 622 | 07MAR1998 | 21 |
| 219 | 01MAR1998 | 18 |
| 219 | 02MAR1998 | 18 |
| 219 | 07MAR1998 | 15 |
| 132 | 01MAR1998 | 14 |
| 132 | 06MAR1998 | 7 |
| 132 | 03MAR1998 | 6 |
| 271 | 01MAR1998 | 5 |
| 132 | 02MAR1998 | 5 |
| 271 | 04MAR1998 | 5 |
| 271 | 05MAR1998 | 5 |
| 271 | 02MAR1998 | 4 |
| 219 | 03MAR1998 | 4 |
| 271 | 07MAR1998 | 4 |
| 219 | 04MAR1998 | 3 |
| 132 | 05MAR1998 | 3 |
| 219 | 05MAR1998 | 3 |
| 271 | 03MAR1998 | 2 |

The next example uses the SQL procedure to retrieve the combined results of two queries to the ORACLE tables PAYROLL and PAYROLL2. An OUTER UNION in PROC SQL concatenates the data.

```
libname myoralib oracle user=charles password=mazyar
  path='airhrdept' schema=hrdept;

title "Payrolls 1 & 2";

proc sql;
  select *
     from myoralib.payroll
  outer union corr
     select *
        from myoralib.payroll2
        order by idnum, jobcode, salary;
quit;
```

Partial output for this example is shown in Output 12.11 on page 161.

**Output 12.11** Querying Multiple DBMS Tables

Payrolls 1 & 2 1

| IDNUM | SEX | JOBCODE | SALARY | BIRTH | HIRED |
|---|---|---|---|---|---|
| 1009 | M | TA1 | 28880 | 02MAR1959 | 26MAR1992 |
| 1017 | M | TA3 | 40858 | 28DEC1957 | 16OCT1981 |
| 1036 | F | TA3 | 39392 | 19MAY1965 | 23OCT1984 |
| 1036 | F | TA3 | 42465 | 19MAY1965 | 23OCT1984 |
| 1037 | F | TA1 | 28558 | 10APR1964 | 13SEP1992 |
| 1038 | F | TA1 | 26533 | 09NOV1969 | 23NOV1991 |
| 1050 | M | ME2 | 35167 | 14JUL1963 | 24AUG1986 |
| 1065 | M | ME2 | 35090 | 26JAN1944 | 07JAN1987 |
| 1065 | M | ME3 | 38090 | 26JAN1944 | 07JAN1987 |
| 1076 | M | PT1 | 66558 | 14OCT1955 | 03OCT1991 |
| 1076 | M | PT1 | 69742 | 14OCT1955 | 03OCT1991 |
| 1094 | M | FA1 | 22268 | 02APR1970 | 17APR1991 |
| 1100 | M | BCK | 25004 | 01DEC1960 | 07MAY1988 |
| 1101 | M | SCP | 18723 | 06JUN1962 | 01OCT1990 |
| 1102 | M | TA2 | 34542 | 01OCT1959 | 15APR1991 |
| 1103 | F | FA1 | 23738 | 16FEB1968 | 23JUL1992 |
| 1104 | M | SCP | 17946 | 25APR1963 | 10JUN1991 |
| 1105 | M | ME2 | 34805 | 01MAR1962 | 13AUG1990 |

## Updating DBMS Data

In addition to querying data, you can also update data directly in your DBMS. You can update rows, columns, and tables by using the SQL procedure.

The following example adds a new row to the DB2 table SUPERV.

```
libname mydb2lib db2 ssid=db2;

proc sql;
insert into mydb2lib.superv
  values('1588','NY','FA');
quit;

proc print data=mydb2lib.superv;
  title "New Row in AIRLINE.SUPERV";
run;
```

*Note:* Depending on how your DBMS processes inserts, the new row might not be added as the last physical row of the table. △

Output for this example is shown in Output 12.12 on page 162.

**Output 12.12** Updating DBMS Data

New Row in AIRLINE.SUPERV 1

| OBS | SUPID | STATE | JOBCAT |
|---|---|---|---|
| 1 | 1677 | CT | BC |
| 2 | 1834 | NY | BC |
| 3 | 1431 | CT | FA |
| 4 | 1433 | NJ | FA |
| 5 | 1983 | NY | FA |
| 6 | 1385 | CT | ME |
| 7 | 1420 | NJ | ME |
| 8 | 1882 | NY | ME |
| 9 | 1935 | CT | NA |
| 10 | 1417 | NJ | NA |
| 11 | 1352 | NY | NA |
| 12 | 1106 | CT | PT |
| 13 | 1442 | NJ | PT |
| 14 | 1118 | NY | PT |
| 15 | 1405 | NJ | SC |
| 16 | 1564 | NY | SC |
| 17 | 1639 | CT | TA |
| 18 | 1401 | NJ | TA |
| 19 | 1126 | NY | TA |
| 20 | 1588 | NY | FA |

The next example deletes all employees who work in Connecticut from the DB2 table STAFF.

```
libname mydb2lib db2 ssid=db2;

proc sql;
  delete from mydb2lib.staff
    where state='CT';
quit;

options obs=20;

proc print data=mydb2lib.staff;
   title "AIRLINE.STAFF After Deleting
       Connecticut Employees";
run;
```

*Note:* If you omit a WHERE clause when you delete rows from a table, all rows in the table are deleted. △

Output for this example is shown in Output 12.13 on page 163.

**Output 12.13** Updating DBMS Data

AIRLINE.STAFF After Deleting Connecticut Employees 1

| OBS | IDNUM | LNAME | FNAME | CITY | STATE | HPHONE |
|---|---|---|---|---|---|---|
| 1 | 1400 | ALHERTANI | ABDULLAH | NEW YORK | NY | 212/586-0808 |
| 2 | 1350 | ALVAREZ | MERCEDES | NEW YORK | NY | 718/383-1549 |
| 3 | 1401 | ALVAREZ | CARLOS | PATERSON | NJ | 201/732-8787 |
| 4 | 1499 | BAREFOOT | JOSEPH | PRINCETON | NJ | 201/812-5665 |
| 5 | 1101 | BAUCOM | WALTER | NEW YORK | NY | 212/586-8060 |
| 6 | 1402 | BLALOCK | RALPH | NEW YORK | NY | 718/384-2849 |
| 7 | 1479 | BALLETTI | MARIE | NEW YORK | NY | 718/384-8816 |
| 8 | 1739 | BRANCACCIO | JOSEPH | NEW YORK | NY | 212/587-1247 |
| 9 | 1658 | BREUHAUS | JEREMY | NEW YORK | NY | 212/587-3622 |
| 10 | 1244 | BUCCI | ANTHONY | NEW YORK | NY | 718/383-3334 |
| 11 | 1383 | BURNETTE | THOMAS | NEW YORK | NY | 718/384-3569 |
| 12 | 1574 | CAHILL | MARSHALL | NEW YORK | NY | 718/383-2338 |
| 13 | 1789 | CARAWAY | DAVIS | NEW YORK | NY | 212/587-9000 |
| 14 | 1404 | COHEN | LEE | NEW YORK | NY | 718/384-2946 |
| 15 | 1065 | COPAS | FREDERICO | NEW YORK | NY | 718/384-5618 |
| 16 | 1876 | CHIN | JACK | NEW YORK | NY | 212/588-5634 |
| 17 | 1129 | COUNIHAN | BRENDA | NEW YORK | NY | 718/383-2313 |
| 18 | 1988 | COOPER | ANTHONY | NEW YORK | NY | 212/587-1228 |
| 19 | 1405 | DACKO | JASON | PATERSON | NJ | 201/732-2323 |
| 20 | 1983 | DEAN | SHARON | NEW YORK | NY | 718/384-1647 |

## Creating a DBMS Table

You can create new tables in your DBMS by using the SQL procedure.

This example uses the SQL procedure to create the ORACLE table GTFORTY by using data from the ORACLE STAFF and PAYROLL tables.

```
libname myoralib oracle user=charles password=mazyar
  path='airhrdept' schema=hrdept;

proc sql;
  create table myoralib.gtforty as
  select lname as lastname,
         fname as firstname,
         salary as salary
           format=dollar10.2
  from myoralib.staff a,
       myoralib.payroll b
  where (a.idnum eq b.idnum) and
        (salary gt 40000);

options obs=20;

proc print data=myoralib.gtforty noobs;
  title 'Employees with salaries over $40,000';
run;
```

Output for this example is shown in Output 12.14 on page 164.

**Output 12.14** Creating a DBMS Table

```
          Employees with salaries over $40,000         1

       LASTNAME        FIRSTNAME        SALARY

       BAREFOOT        JOSEPH       $43,025.00
       BANADYGA        JUSTIN       $88,606.00
       BRANCACCIO      JOSEPH       $66,517.00
       BRADY           CHRISTINE    $68,767.00
       COHEN           LEE          $91,376.00
       CARTER-COHEN    KAREN        $40,260.00
       CASTON          FRANKLIN     $41,690.00
       FERNANDEZ       KATRINA      $51,081.00
       GRAHAM          ALVIN        $65,111.00
       GREGORSKI       DANIEL       $68,096.00
       HARRIS          CHARLES      $84,685.00
       HASENHAUER      CHRISTINA    $70,736.00
       HAVELKA         RAYMOND      $41,551.00
       HERRERO         CLYDE        $66,130.00
       KIMANI          ANNE         $40,899.00
       MARSHBURN       JASPER       $89,632.00
       MORGAN          ALFRED       $42,264.00
       NEWKIRK         SANDRA       $84,536.00
       NEWKIRK         WILLIAM      $52,270.00
       NEWTON          JAMES        $84,203.00
```

# Using Other SAS Procedures with DBMS Data

The following examples illustrate basic uses of other SAS procedures with librefs that refer to DBMS data.

## Using the MEANS Procedure

This example uses the PRINT and MEANS procedures on a SAS data set created from the ORACLE table INTERNAT. The MEANS procedure provides information on the largest number of passengers on each flight.

```
libname myoralib oracle user=anita password=traveler
  path='fltdata' schema=airport;

title 'Number of Passengers per
       Flight by Date';

proc print data=my_data noobs;
  var date boarded;
  by flight dest;
  sumby flight;
  sum boarded;
run;

title 'Maximum Number of
       Passengers per Flight';

proc means data=my_data fw=5 maxdec=1 max;
  var boarded;
```

```
  class flight;
run;
```

Partial output for this example is shown in Output 12.15 on page 166.

**Output 12.15** Using the PRINT and MEANS Procedures

```
                    Number of Passengers per Flight by Date

  ------------------------------ FLIGHT=132 DEST=YYZ -------------------------------

                                      DATE    BOARDED

                                  01MAR1998       115
                                  02MAR1998       106
                                  03MAR1998        75
                                  04MAR1998       117
                                  05MAR1998       157
                                  06MAR1998       150
                                  07MAR1998       164
                                  ---------    -------
                                     FLIGHT       884


  ------------------------------ FLIGHT=219 DEST=LON -------------------------------

                                      DATE    BOARDED

                                  01MAR1998       198
                                  02MAR1998       147
                                  03MAR1998       197
                                  04MAR1998       232
                                  05MAR1998       160
                                  06MAR1998       163
                                  07MAR1998       241
                                  ---------    -------
                                     FLIGHT      1338
```

```
                  Maximum Number of Passengers per Flight

                           The MEANS Procedure

                       Analysis Variable : BOARDED

                                    N
                     FLIGHT     Obs     MAXIMUM
                     132          7       164.0

                     219          7       241.0

                     271          6       177.0

                     622          6       210.0
```

## Using the DATASETS Procedure

This example uses the DATASETS procedure to view a list of DBMS tables, in this case, in a DB2 database.

*Note:* The MODIFY and ALTER statements in PROC DATASETS are not available for use with librefs that refer to DBMS data. △

```
libname mydb2lib db2 ssid=db2;

title "Table Listing for DB2";

proc datasets lib=mydb2lib;
  contents data=_all_ nods;
run;
```

Partial output for this example is shown in Output 12.16 on page 167.

**Output 12.16** Using the DATASETS Procedure

```
                    Table Listing for DB2

                     DATASETS PROCEDURE

                     -----Directory-----

                  Libref:        MYDB2LIB
                  Engine:        DB2
                  Filefmt:
                  Physical Name: DB2

                    #  Name      Memtype
                    --------------------
                    1  DELAY     DATA
                    2  INTERNAT  DATA
                    3  MARCH     DATA
                    4  PAYROLL   DATA
                    5  PAYROLL2  DATA
                    6  SCHEDULE  DATA
                    7  STAFF     DATA
                    8  SUPERV    DATA
```

## Using the CONTENTS Procedure

These examples show output from the CONTENTS procedure when it is run on a DBMS table. Note that PROC CONTENTS shows all of the SAS metadata derived from the DBMS table by the SAS/ACCESS engine.

```
libname mydb2lib db2 ssid=db2;

proc contents data=mydb2lib.delay;
run;
```

Output from this example is shown in Output 12.17 on page 167.

**Output 12.17** Using the CONTENTS Procedure

CONTENTS PROCEDURE

| | | | |
|---|---|---|---|
| Data Set Name: | AIRLINE.DELAY | Observations: | . |
| Member Type: | DATA | Variables: | 7 |
| 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 |
|---|---|---|---|---|---|---|---|
| 2 | DATES | Num | 8 | 8 | DATE9. | DATE9. | DATES |
| 7 | DELAY | Num | 8 | 64 | | | DELAY |
| 5 | DELAYCAT | Char | 15 | 32 | $15. | $15. | DELAYCAT |
| 4 | DEST | Char | 3 | 24 | $3. | $3. | DEST |
| 6 | DESTYPE | Char | 15 | 48 | $15. | $15. | DESTYPE |
| 1 | FLIGHT | Char | 3 | 0 | $3. | $3. | FLIGHT |
| 3 | ORIG | Char | 3 | 16 | $3. | $3. | ORIG |

## Using the RANK Procedure

This example uses the RANK procedure to rank flights in the DB2 table DELAY by number of minutes delayed.

```
libname mydb2lib db2 ssid=db2;

options obs=20;

proc rank data=mydb2lib.delay descending
    ties=low out=ranked;
  var delay;
  ranks RANKING;
run;

proc print data=ranked;
  title "Ranking of Delayed Flights";
  format delay 2.0;
run;
```

Output for this example is shown in Output 12.18 on page 168.

**Output 12.18** Using the RANK Procedure

```
                          Ranking of Delayed Flights                        1

 OBS FLIGHT     DATES ORIG DEST DELAYCAT        DESTYPE         DELAY RANKING

   1  114   01MAR1998 LGA  LAX  1-10 Minutes    Domestic          8       9
   2  202   01MAR1998 LGA  ORD  No Delay        Domestic         -5      42
   3  219   01MAR1998 LGA  LON  11+ Minutes     International    18       4
   4  622   01MAR1998 LGA  FRA  No Delay        International    -5      42
   5  132   01MAR1998 LGA  YYZ  11+ Minutes     International    14       8
   6  271   01MAR1998 LGA  PAR  1-10 Minutes    International     5      13
   7  302   01MAR1998 LGA  WAS  No Delay        Domestic         -2      36
   8  114   02MAR1998 LGA  LAX  No Delay        Domestic          0      28
   9  202   02MAR1998 LGA  ORD  1-10 Minutes    Domestic          5      13
  10  219   02MAR1998 LGA  LON  11+ Minutes     International    18       4
  11  622   02MAR1998 LGA  FRA  No Delay        International     0      28
  12  132   02MAR1998 LGA  YYZ  1-10 Minutes    International     5      13
  13  271   02MAR1998 LGA  PAR  1-10 Minutes    International     4      19
  14  302   02MAR1998 LGA  WAS  No Delay        Domestic          0      28
  15  114   03MAR1998 LGA  LAX  No Delay        Domestic         -1      32
  16  202   03MAR1998 LGA  ORD  No Delay        Domestic         -1      32
  17  219   03MAR1998 LGA  LON  1-10 Minutes    International     4      19
  18  622   03MAR1998 LGA  FRA  No Delay        International    -2      36
  19  132   03MAR1998 LGA  YYZ  1-10 Minutes    International     6      12
  20  271   03MAR1998 LGA  PAR  1-10 Minutes    International     2      25
```

## Using the TABULATE Procedure

This example uses the TABULATE procedure on the ORACLE table PAYROLL to display a chart of the number of employees for each job code.

```
libname myoralib oracle user=antonio password=porsche
  path='airhrdept' schema=hrdept;

title "Number of Employees by Jobcode";

proc tabulate data=myoralib.payroll format=3.0;
   class jobcode;
   table jobcode*n;
   keylabel n="#";
run;
```

Output for this example is shown in Output 12.19 on page 169.

**Output 12.19** Using the TABULATE Procedure

```
                        Number of Employees by Jobcode                    1

          ------------------------------------------------------------
          |                           jobcode                          |
          |------------------------------------------------------------|
          |BCK|FA1|FA2|FA3|ME1|ME2|ME3|NA1|NA2|PT1|PT2|PT3|SCP|TA1|TA2|TA3|
          |---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---|
          | # | # | # | # | # | # | # | # | # | # | # | # | # | # | # | # |
          |---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---|
          |  9| 11| 16|  7|  8| 14|  7|  5|  3|  8| 10|  2|  7|  9| 20| 12|
          ------------------------------------------------------------
```

## Using the APPEND Procedure

In this example, the DB2 table PAYROLL2 is appended to the DB2 table PAYROLL with the APPEND procedure. The PAYROLL table is updated on DB2.

*Note:* When you append data to a DBMS table, you are actually inserting rows into a table. The rows can be inserted into the DBMS table in any order. △

```
libname mydb2lib db2 ssid=db2;

proc append base=mydb2lib.payroll
            data=mydb2lib.payroll2;
run;

proc print data=mydb2lib.payroll;
  title 'PAYROLL After Appending
         PAYROLL2';
run;
```

*Note:* In cases where a DBMS table that you are using is in the same database space as a table that you are creating or updating, you must use the CONNECTION=SHARED LIBNAME option to prevent a deadlock. See "SAS/ACCESS LIBNAME Statement" on page 27 for more information on SAS/ACCESS LIBNAME options. △

Partial output for this example is shown in Output 12.20 on page 170.

**Output 12.20** Using the APPEND Procedure

PAYROLL After Appending PAYROLL2 1

| OBS | IDNUM | SEX | JOBCODE | SALARY | BIRTH | HIRED |
|---|---|---|---|---|---|---|
| 1 | 1919 | M | TA2 | 34376 | 12SEP1960 | 04JUN1987 |
| 2 | 1653 | F | ME2 | 35108 | 15OCT1964 | 09AUG1990 |
| 3 | 1400 | M | ME1 | 29769 | 05NOV1967 | 16OCT1990 |
| 4 | 1350 | F | FA3 | 32886 | 31AUG1965 | 29JUL1990 |
| 5 | 1401 | M | TA3 | 38822 | 13DEC1950 | 17NOV1985 |
| 6 | 1499 | M | ME3 | 43025 | 26APR1954 | 07JUN1980 |
| 7 | 1101 | M | SCP | 18723 | 06JUN1962 | 01OCT1990 |
| 8 | 1333 | M | PT2 | 88606 | 30MAR1961 | 10FEB1981 |
| 9 | 1402 | M | TA2 | 32615 | 17JAN1963 | 02DEC1990 |
| 10 | 1479 | F | TA3 | 38785 | 22DEC1968 | 05OCT1989 |
| 11 | 1403 | M | ME1 | 28072 | 28JAN1969 | 21DEC1991 |
| 12 | 1739 | M | PT1 | 66517 | 25DEC1964 | 27JAN1991 |
| 13 | 1658 | M | SCP | 17943 | 08APR1967 | 29FEB1992 |
| 14 | 1428 | F | PT1 | 68767 | 04APR1960 | 16NOV1991 |
| 15 | 1782 | M | ME2 | 35345 | 04DEC1970 | 22FEB1992 |
| 16 | 1244 | M | ME2 | 36925 | 31AUG1963 | 17JAN1988 |
| 17 | 1383 | M | BCK | 25823 | 25JAN1968 | 20OCT1992 |
| 18 | 1574 | M | FA2 | 28572 | 27APR1960 | 20DEC1992 |
| 19 | 1789 | M | SCP | 18326 | 25JAN1957 | 11APR1978 |
| 20 | 1404 | M | PT2 | 91376 | 24FEB1953 | 01JAN1980 |

# Charting Data

This example shows two ways to use the GCHART procedure with DBMS data. The first method uses descriptors. The second method uses the new SAS/ACCESS LIBNAME statement to accomplish the same task in an easier and more direct way. Note that descriptor names and variables are still limited to eight characters, but the LIBNAME statement accommodates member names and variable names up to 32 characters.

## Using the GCHART Procedure with Descriptors

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 Oracle Rdb table ORDERS.

```
proc access dbms=rdb;

/* create access descriptor    */

   create adlib.order.access;
   database='atlanta::disk1:[root]textile.rdb';
   table=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;
   list all;

/* create vlib.allordr view descriptor */

   create vlib.allordr.view;
   select all;
run;

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

Output for both of these examples is shown in Display 12.1 on page 172. STOCKNUM represents each product. The number of orders for each product is represented by the height of the bar. (In some operating environments, the increments on the vertical axis may be different.)

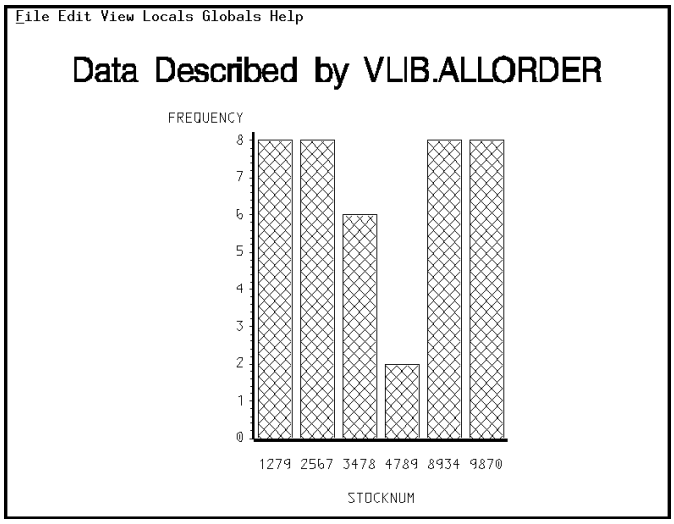
## Using the GCHART Procedure with a SAS/ACCESS LIBNAME

This example uses the SQL and GCHART procedures to chart data from the PROC SQL view WORK.ALLORDR, created from the ORACLE table ORDERS. This example differs from the previous example in that the SAS/ACCESS LIBNAME statement is used to define a SAS libref that references DBMS data. Descriptors are not used. Output from this example is identical to the previous example, except for the column names, which are limited to eight characters when you use descriptors but can be up to 32 characters when you use the SAS/ACCESS LIBNAME statement.

```
libname myoralib oracle user=dmitry pass=elvis
  path='txtdata' schema=textile;

proc gchart data=myoralib,orders;
  vbar stocknum / discrete;
  title 'Data Described by VLIB.ALLORDER';
run;
```

**Display 12.1** Output from PROC GCHART




# Calculating Statistics

This example shows two ways to use the FREQ procedure with DBMS data. The first method uses descriptors. The second method uses the new SAS/ACCESS LIBNAME statement, which fully supports the new Version 7 and Version 8 features, such as long names up to 32 characters, and accomplishes the same task in an easier and more direct way.

## Using the FREQ Procedure with Descriptors

The following example uses the view descriptor VLIB.INV to calculate the percentage of invoices for each country that appears in the table SASDEMO.INVOICE.

```
proc access dbms=oracle;

/* create access descriptor   */

   create adlib.invoice.access;
   user=scott orapw=tiger;
   path='myorapath';
   table=invoice;

/* create vlib.inv view      */

   create vlib.inv.view;
   select invoicenum amtbilled
          country
          billedby paidon;

   rename invoicenum = invnum
          amtbilled  = amtbilld;
   format paidon      date9.
          invoicenum  5.0
          billedby    6.0;
   list all;
run;

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

## Using the FREQ Procedure with a SAS/ACCESS LIBNAME

This example uses the FREQ procedure to calculate statistics on the PROC SQL view WORK.INV, created from the DB2 table INVOICE. This example differs from the previous example in that the SAS/ACCESS LIBNAME statement is used to define a SAS libref that references DBMS data. Descriptors are not used. Output from this example is identical to the previous example, except for the column names, which are limited to eight characters when you use descriptors but can be up to 32 characters when you use the SAS/ACCESS LIBNAME statement.

```
libname mydb2lib db2 ssid=db2;

proc freq data=mydb2lib.invoice(keep=invoicenum amtbilled country
billedby paidon);
  tables country;
  title 'Data Described by INV';
run;
```

Output 12.21 on page 173 shows the one-way frequency table that these examples generate.

**Output 12.21** Using the FREQ Procedure

```
                              Data Described by INV                        1
                                The FREQ Procedure

                                     COUNTRY

                                                              Cumulative    Cumulative
COUNTRY                    Frequency     Percent     Frequency      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
```

# Selecting and Combining Data

This example shows two ways to use the WHERE statement to subset DBMS data. The first method uses descriptors. The second method uses the new SAS/ACCESS LIBNAME statement to accomplish the same task in an easier and more direct way.

## Using the WHERE Statement with Descriptors

The view descriptor VLIB.ALLINV lists invoices for all customers and is based on the table INVOICE. You can use a DATA step 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=oracle;

/* create access descriptor   */

   create adlib.invoice.access;
   user=scott;  password=tiger;
   path='myorapath'
   table=invoice;
   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 work.notpaid(keep=invnum billedto amtinus billedon);
   set vlib.allinv;
   where paidon is missing and amtinus>=300000.00;
run;

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

In the DATA step's WHERE statement, be sure to use SAS variable names, not DBMS column names. The DATA statement uses the KEEP= data set option. This option specifies that you want to include only the listed variables in the new SAS data file WORK.NOTPAID. However, you can still reference the other view descriptor variables in other statements within the DATA step.

The SAS WHERE statement includes two conditions to be met. First, it specifies to select 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 must be familiar with the DBMS data so that you can determine reasonable values for these expressions.

When you are referencing a view descriptor in a SAS procedure or DATA step, it is more efficient to use a SAS WHERE statement rather than a subsetting IF statement. When possible, a WHERE statement's selection criteria is passed to the DBMS for processing and returns a subset of rows to the SAS System for further processing. In contrast, when you use a subsetting IF statement, every row is returned to the SAS System to be evaluated by the IF statement. For more information about how WHERE clauses are passed to the DBMS for processing, see "Using a Subset of the DBMS Data" on page 127.

Output for both of these examples is shown in Output 12.22 on page 175.

## Using the WHERE Statement with a SAS/ACCESS LIBNAME

This example uses a WHERE statement directly in the PRINT procedure to print only unpaid bills over $300,000. This example differs from the previous example in that the SAS/ACCESS LIBNAME statement is used to define a SAS libref that references the DBMS data. Descriptors are not used. Output from this example is identical to the previous example, except for the column names, which are limited to eight characters when you use descriptors but can be up to 32 characters when you use the SAS/ACCESS LIBNAME statement.

```
libname myoralib oracle user=dmitry pass=elvis
  path='txtdata' schema=textile;

proc sql;
  create view allinv as
       select paidon, billedon, invoicenum, amountinus, billedto
       from myoralib.invoice
       where paidon is null and amountinus>=300000.00;
quit;

proc print data=allinv(drop=paidon);
  format amountinus dollar20.2;
  title 'High Bills--Not Paid';
run;
```

**Output 12.22** Using the WHERE Statement

High Bills--Not Paid 1

| Obs | billedon | invoicenum | amountinus | billedto |
|---|---|---|---|---|
| 1 | 05OCT1998 | 11271 | $11,063,836.00 | 18543489 |
| 2 | 10OCT1998 | 11286 | $11,063,836.00 | 43459747 |
| 3 | 02NOV1998 | 12051 | $2,256,870.00 | 39045213 |
| 4 | 17NOV1998 | 12102 | $11,063,836.00 | 18543489 |
| 5 | 27DEC1998 | 12471 | $2,256,870.00 | 39045213 |
| 6 | 24DEC1998 | 12476 | $2,256,870.00 | 38763919 |

# Joining DBMS and SAS Data

This example shows two ways to combine SAS and DBMS data. The first method uses the SQL Procedure Pass-Through Facility. The second method uses the new SAS/ACCESS LIBNAME statement to accomplish the same task in an easier and more direct way.

## Combining a PROC SQL View with a SAS Data Set By Using the Pass-Through Facility

This example joins SAS data with CA-OpenIngres data that is retrieved by using a Pass-Through query in a PROC SQL SELECT statement.

In this example's PROC SQL CONNECT statement, the database name is **textile** and it is located on a remote network node named **atlanta**; the CA-OpenIngres server is specified by **star**.

Information on student interns is stored in the SAS data file, DLIB.TEMPEMPS. The CA-OpenIngres data is joined with DLIB.TEMPEMPS to determine whether any of the student interns have a family member who works in the CSR departments.

To join the data from DLIB.TEMPEMPS with the data from the Pass-Through query, you assign a table alias (QUERY1) to the query. Doing so enables you to qualify the query's column names in the WHERE clause.

```
options ls=120;

title 'Interns Who Are Family Members of
       Employees';

proc sql;
connect to ingres
   (database='atlanta::textile/star');
%put &sqlxmsg;

select tempemps.lastname, tempemps.firstnam,
       tempemps.empid, tempemps.familyid,
       tempemps.gender, tempemps.dept,
       tempemps.hiredate
   from connection to ingres
      (select * from employees) as query1, dlib.tempemps
   where query1.empid=tempemps.familyid;
%put &sqlxmsg;
```

```
disconnect from ingres;
quit;
```

*Note:* When SAS data is joined to DBMS data by using a Pass-Through query, PROC SQL cannot optimize the query. In this case it is much more efficient to use a SAS/ACCESS LIBNAME statement, as shown in the next example. Another way to increase efficiency is to extract the DBMS data and place it in a new SAS data file, assign SAS indexes to the appropriate variables, then to join the two SAS data files. △

Output for both of these examples is shown in Output 12.23 on page 177.

## Combining a PROC SQL View with a SAS Data Set By Using a SAS/ACCESS LIBNAME

This example creates a PROC SQL view, MYSASLIB.EMP_CSRALL, from the DB2 table EMPLOYEES and joins the view with a SAS data set to select only interns who are family members of existing employees.

```
libname mydb2lib db2 ssid=db2;
libname mysaslib "sas-data-library";
title 'Interns Who Are Family
       Members of Employees';

 create view mysaslib.emp_csrall as
 select * from mydblib.employees
  where dept in ('CSR010', 'CSR011', 'CSR004');

proc sql;
  select tempemps.lastname, tempemps.firstnam,
         tempemps.empid, tempemps.familyid,
         tempemps.gender, tempemps.dept,
         tempemps.hiredate
    from mydb2lib.employees as emp,
         mysaslib.tempemps as temps
    where emp.empid=temps.familyid;

quit;
```

**Output 12.23** Combining a PROC SQL View with a SAS Data Set

```
                Interns Who Are Family Members of Employees                1

  lastname          firstnam   empid    familyid    gender   dept     hiredate
  ------------------------------------------------------------------------------
  SMITH             ROBERT     765112   234967      M        CSR010   04MAY1998
  NISHIMATSU-LYNCH  RICHARD    765111   677890      M        CSR011   04MAY1998
```

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