CHAPTER 33

Accessing Data in a DBMS

Definition

SAS/ACCESS software
allows you to read and write data to and from other vendors' *database management systems (DBMS)*, as well as from some *PC file formats*. Depending on your DBMS, a SAS/ACCESS product might provide one or more of the following:

- a dynamic LIBNAME engine
- the SQL Pass-Through Facility
- the ACCESS procedure and interface view engine
- the DBLOAD procedure
- an interface DATA step engine.

These interfaces are described in this section. Each SAS/ACCESS product provides one or more of these interfaces for each supported DBMS. See Chapter 36, "SAS I/O Engines," on page 511 for more information about SAS engines.

Note: To use the SAS/ACCESS features described in this section, you must license SAS/ACCESS software. See the SAS/ACCESS documentation for your DBMS for full documentation of the features described in this section. △

Dynamic LIBNAME Engine

SAS/ACCESS LIBNAME Statement

Beginning in Version 7, you can associate a SAS libref directly with a database, schema, server, or group of tables and views, depending on your DBMS. To assign a libref to DBMS data, you must use the SAS/ACCESS LIBNAME statement, which has syntax and options that are different from the base LIBNAME statement. For example, to connect to an ORACLE database, you might use the following SAS/ACCESS LIBNAME statement:

```
libname mydblib oracle user=smith password=secret
   path='myoracleserver';
```

This LIBNAME statement connects to ORACLE by specifying the ORACLE connection options: USER=, PASSWORD=, and PATH=. In addition to the connection options, you can specify SAS/ACCESS LIBNAME options that control the type of database connection that is made. You can use additional options to control how your data is processed.

You can use a DATA step, SAS procedures, or the Explorer window to view and update the DBMS data associated with the libref, or use the DATASETS and CONTENTS procedures to view information about the DBMS objects.

See your SAS/ACCESS documentation for a full listing of the SAS/ACCESS LIBNAME options that can be used with librefs that refer to DBMS data.

Using Data Set Options with SAS/ACCESS Librefs

After you have assigned a libref to your DBMS data, you can use SAS/ACCESS data set options, and some of the base SAS data set options, on the data. The following example associates a libref with DB2 data and uses the SQL procedure to query the data:

```
libname mydb2lib db2;

proc sql;
   select *
      from mydb2lib.employees(drop=salary)
      where dept='Accounting';
quit;
```

The LIBNAME statement connects to DB2. You can reference a DBMS object, in this case, a DB2 table, by specifying a two-level name that is comprised of the libref and the DBMS object name. The DROP= data set option causes the SALARY column of the EMPLOYEES table on DB2 to be excluded from the data that is returned by the query.

See your SAS/ACCESS documentation for a full listing of the SAS/ACCESS data set options and the base data set options that can be used on data sets that refer to DBMS data.

Embedding a SAS/ACCESS LIBNAME Statement in a PROC SQL View

You can issue a SAS/ACCESS LIBNAME statement by itself, as shown in the previous examples, or as part of a CREATE VIEW statement in PROC SQL. The USING clause of the CREATE VIEW statement allows you to store DBMS connection information in a view by *embedding* a SAS/ACCESS LIBNAME statement inside the view. The following example uses an embedded SAS/ACCESS LIBNAME statement:

```
libname viewlib 'SAS-data-library';

proc sql;
   create view viewlib.emp_view as
      select *
         from mydblib.employees
         using libname mydblib oracle user=smith password=secret
            path='myoraclepath';
quit;
```

When PROC SQL executes the view, the SELECT statement assigns the libref and establishes the connection to the DBMS. The scope of the libref is local to the view and does not conflict with identically named librefs that might exist in the SAS session. When the query finishes, the connection is terminated and the libref is deassigned.

Note: You can also embed a base SAS LIBNAME statement in a PROC SQL view. △

SQL Procedure Pass-Through Facility

The SQL Procedure Pass-Through Facility is an extension of the SQL procedure that enables you to send DBMS-specific statements to a DBMS and to retrieve DBMS data. You specify DBMS SQL syntax instead of SAS SQL syntax when you use the Pass-Through Facility. You can use Pass-Through Facility statements in a PROC SQL query or store them in a PROC SQL view.

The Pass-Through Facility consists of three statements and one component:

- The CONNECT statement establishes a connection to the DBMS.
- The EXECUTE statement sends dynamic, non-query DBMS-specific SQL statements to the DBMS.
- The CONNECTION TO component in the FROM clause of a PROC SQL SELECT statement retrieves data directly from a DBMS.
- The DISCONNECT statement terminates the connection to the DBMS.

The following Pass-Through Facility example sends a query to an ORACLE database for processing:

```
proc sql;
   connect to oracle as myconn (user=smith password=secret
      path='myoracleserver');

   select *
      from connection to myconn
         (select empid, lastname, firstname, salary
            from employees
            where salary>75000);

   disconnect from myconn;
quit;
```

The example uses the Pass-Through CONNECT statement to establish a connection with an ORACLE database with the specified values for the USER=, PASSWORD=, and PATH= arguments. The CONNECTION TO component in the FROM clause of the SELECT statement allows data to be retrieved from the database. The DBMS-specific statement that is sent to ORACLE is enclosed in parentheses. The DISCONNECT statement terminates the connection to ORACLE.

To store the same query in a PROC SQL view, use the CREATE VIEW statement:

```
libname viewlib 'SAS-data-library';

proc sql;
   connect to oracle as myconn (user=smith password=secret
      path='myoracleserver');
      create view viewlib.salary as
         select *
            from connection to myconn
               (select empid, lastname, firstname, salary
                  from employees
                  where salary>75000);

      disconnect from myconn;
   quit;
```

ACCESS Procedure and Interface View Engine

The ACCESS procedure enables you to create *access descriptors*, which are SAS files of member type ACCESS. They describe data that is stored in a DBMS in a format that SAS can understand. Access descriptors enable you to create SAS/ACCESS views, called *view descriptors*. View descriptors are files of member type VIEW that function in the same way as SAS data views that are created with PROC SQL, as described in “Embedding a SAS/ACCESS LIBNAME Statement in a PROC SQL View” on page 488 and “SQL Procedure Pass-Through Facility” on page 489.

Note: If a dynamic LIBNAME engine is available for your DBMS, it is recommended that you use the SAS/ACCESS LIBNAME statement to access your DBMS data instead of access descriptors and view descriptors; however, descriptors continue to work in SAS software if they were available for your DBMS in Version 6. Some new SAS features, such as long variable names, are not supported when you use descriptors. △

The following example creates an access descriptor and a view descriptor in the same PROC step to retrieve data from a DB2 table:

```
libname adlib 'SAS-data-library';
libname vlib 'SAS'-data-library';

proc access dbms=db2;
   create adlib.order.access;
   table=sasdemo.orders;
   assign=no;
   list all;

   create vlib.custord.view;
   select ordernum stocknum shipto;
   format ordernum 5.
          stocknum 4.;
run;

proc print data=vlib.custord;
run;
```

When you want to use access descriptors and view descriptors, both types of descriptors must be created before you can retrieve your DBMS data. The first step, creating the access descriptor, allows SAS to store information about the specific DBMS table that you want to query.

After you have created the access descriptor, the second step is to create one or more view descriptors to retrieve some or all of the DBMS data described by the access descriptor. In the view descriptor, you select variables and apply formats to manipulate the data for viewing, printing, or storing in SAS. You use only the view descriptors, and not the access descriptors, in your SAS programs.

The interface view engine enables you to reference your view with a two-level SAS name in a DATA or PROC step, such as the PROC PRINT step in the example.

See Chapter 29, “SAS Data Views,” on page 455 for more information about views. See the SAS/ACCESS documentation for your DBMS for more detailed information about creating and using access descriptors and SAS/ACCESS views.

DBLOAD Procedure

The DBLOAD procedure enables you to create and load data into a DBMS table from a SAS data set, data file, data view, or another DBMS table, or to append rows to an existing table. It also enables you to submit non-query DBMS-specific SQL statements to the DBMS from your SAS session.

Note: If a dynamic LIBNAME engine is available for your DBMS, it is recommended that you use the SAS/ACCESS LIBNAME statement to create your DBMS data instead of the DBLOAD procedure; however, DBLOAD continues to work in SAS software if it was available for your DBMS in Version 6. Some new SAS features, such as long variable names, are not supported when you use the DBLOAD procedure. △

The following example appends data from a previously created SAS data set named INVDATA into a table in an ORACLE database named INVOICE:

```
proc dbload dbms=oracle data=invdata append;
  user=smith;
  password=secret;
  path='myoracleserver';
  table=invoice;
  load;
run;
```

See the SAS/ACCESS documentation for your DBMS for more detailed information about the DBLOAD procedure.

Interface DATA Step Engine

Some SAS/ACCESS software products support a DATA step interface, which allows you to read data from your DBMS by using DATA step programs. Some products support both reading and writing in the DATA step interface.

The DATA step interface consists of four statements:

- The INFILE statement identifies the database or message queue to be accessed.
- The INPUT statement is used with the INFILE statement to issue a GET call to retrieve DBMS data.
- The FILE statement identifies the database or message queue to be updated, if writing to the DBMS is supported.
- The PUT statement is used with the FILE statement to issue an UPDATE call, if writing to the DBMS is supported.

The following example updates data in an IMS database by using the FILE and INFILE statements in a DATA step. The statements generate calls to the database in the IMS native language, DL/I. The DATA step reads BANK.CUSTOMER, an existing SAS data set that contains information on new customers, and then it updates the ACCOUNT database with the data in the SAS data set.

```
data _null_;
  set bank.customer;
  length ssa1 $9;
  infile accupdt dli call=func dbname=db ssa=ssa1;
  file accupdt dli;
  func = 'isrt';
  db = 'account';
  ssa1 = 'customer';
  put @1   ssnumber $char11.
      @12  custname $char40.
      @52  addr1    $char30.
      @82  addr2    $char30.
      @112 custcity $char28.
      @140 custstat $char2.
      @142 custland $char20.
      @162 custzip  $char10.
      @172 h_phone  $char12.
      @184 o_phone  $char12.;
  if _error_ = 1 then
    abort abend 888;
run;
```

In SAS/ACCESS products that provide a DATA step interface, the INFILE statement has special DBMS-specific options that allow you to specify DBMS variable values and to format calls to the DBMS appropriately. See the SAS/ACCESS documentation for your DBMS for a full listing of the DBMS-specific INFILE statement options and the base INFILE statement options that can be used with your DBMS.

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS Language Reference: Concepts*, Cary, NC: SAS Institute Inc., 1999. 554 pages.

SAS Language Reference: Concepts

Copyright © 1999 SAS Institute Inc., Cary, NC, USA.

ISBN 1–58025–441–1

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

1st printing, November 1999