CHAPTER 7

Advanced Topics in SAS/ACCESS

Choosing a SAS/ACCESS Feature

In SAS/ACCESS software, there are often several ways to complete a given task. In addition to the SQL Procedure Pass-Through Facility, you can now use the SAS/ACCESS LIBNAME statement to access DBMS tables and views directly. Also, the ACCESS and DBLOAD procedures provided in Version 6 of SAS/ACCESS are still supported.

Note: Some of these features might not be available in your SAS/ACCESS interface. See your DBMS chapter for more information. △

Table 7.1 on page 86 provides a list of common tasks and the features that you can use to accomplish them. When choosing a feature to use for complex or data-intensive operations, you might want to test more than one method to determine the most efficient one for your particular task.

Note: The following abbreviations are used in the table:

- *LIBNAME statement* refers to the Version 7 and later SAS/ACCESS LIBNAME statement.
- *Pass-Through* refers to the SQL Procedure Pass-Through Facility.
- *View descriptors* refer to view descriptors that are created in the ACCESS procedure.

△

Table 7.1 SAS/ACCESS Features for Specific Tasks

If you want to...	you can use one of these SAS/ACCESS features:
read DBMS tables or views	LIBNAME statement
	Pass-Through
	view descriptors
create DBMS tables	LIBNAME statement
	DBLOAD procedure
	Pass-Through EXECUTE statement
update. delete, or insert rows into DBMS tables	LIBNAME statement
	view descriptors
	Pass-Through EXECUTE statement
append data to DBMS tables	DBLOAD procedure with APPEND option
	LIBNAME statement and APPEND procedure
	Pass-Through EXECUTE statement
list DBMS tables	LIBNAME statement and SAS Explorer window
	LIBNAME statement and DATASETS procedure
	LIBNAME statement with CONTENTS procedure
	LIBNAME statement and PROC SQL dictionary tables[1]
delete DBMS tables or views	LIBNAME statement and SQL procedure's DROP TABLE statement
	LIBNAME statement and DATASETS procedure's DELETE statement
	Pass-Through EXECUTE statement

1 See the SQL dictionary tables documentation for more information.

Comparing SAS/ACCESS Features

It is recommended that you use the new SAS/ACCESS LIBNAME statement to access your DBMS data easily and transparently. However, if your SAS/ACCESS interface does not support the LIBNAME statement, you might need to choose between the SQL Procedure Pass-Through Facility or view descriptors.

SAS/ACCESS LIBNAME Statement

When you use the SAS/ACCESS LIBNAME statement, the LIBNAME engine sends and receives data between the DBMS and the SAS System.

Beginning in Version 7, the LIBNAME statement is usually the fastest and most direct method of accessing DBMS data. Here are some of the tasks it performs:

- The LIBNAME statement provides transparent access to DBMS data. Once a libref is associated with a group of DBMS tables or views, you can use this libref just as you would use any SAS libref.
- You can use member and variable names of up to 32 characters.
- Automatic conversion of SAS to DBMS data types, and DBMS to SAS data types, allows you to move data easily between SAS and your DBMS. You can use SAS/ACCESS LIBNAME and data set options, including the DBTYPE= data set option, to further control the data conversion process.
- For extra security, you can prompt the user for DBMS connection information at runtime by using the DBPROMPT= libname and data set options. Locally stored environment variables can also be used from a SAS client to establish a connection to a DBMS.

The SAS/ACCESS LIBNAME statement has the following advantages over the SQL Procedure Pass-Through Facility and Version 6 view descriptors:

- Significantly fewer lines of SAS code are required to perform operations on your DBMS. For example, a single LIBNAME statement establishes a connection to your DBMS, allows you to specify how your data is processed, and allows you to easily browse your DBMS tables in SAS.
- You do not need to know your DBMS's SQL language to access and manipulate data on your DBMS. You can use SAS procedures, such as PROC SQL, or DATA step programming on any libref that references DBMS data. You can read, insert, update, delete, and append data, as well as create and drop DBMS tables by using normal SAS syntax.
- The LIBNAME statement provides more control over DBMS operations such as locking, spooling, and data type conversion through the many LIBNAME options and data set options.
- The LIBNAME engine optimizes the processing of joins and WHERE clauses by passing these operations directly to the DBMS to take advantage of your DBMS's indexing and other processing capabilities. See "SAS/ACCESS LIBNAME Statement" on page 27 for more information.

SQL Procedure Pass-Through Facility and View Descriptors

If you cannot use the LIBNAME statement, you can also use either the SQL Procedure Pass-Through Facility or SAS/ACCESS view descriptors to access DBMS data. There are certain cases where it is more efficient to use one method or the other, and choosing the best method can enhance your program's performance.

In both methods, the SAS/ACCESS interface view engine sends and receives data between the DBMS and the SAS System.

The SQL Procedure Pass-Through Facility and SAS/ACCESS view descriptors perform these tasks equally well:

- View descriptors and SQL Procedure Pass-Through Facility queries both offer direct access to DBMS data. SQL Procedure Pass-Through Facility queries can also be stored as PROC SQL views. When you refer to a view descriptor or a PROC SQL view (that is, a stored Pass-Through query) in a SAS program, the DBMS connection is automatically made, and the DBMS data is retrieved for your use.
- You can assign SAS variable names and formats to DBMS columns and data types by using either SAS/ACCESS descriptors or SQL Procedure Pass-Through Facility queries. You use the RENAME= and FORMAT= statements when you create a descriptor. When you create a SQL Procedure Pass-Through Facility query (or PROC SQL view), you can use column aliases or modifiers in the SELECT clause, or you can use a table alias and column list in the FROM clause.

- You can retrieve a subset of data from a DBMS table or view by using either view descriptors or SQL Procedure Pass-Through Facility queries. Both methods use a WHERE statement or clause. Both methods also enable you to group or sort the data that you retrieve by using GROUP BY and ORDER BY clauses.
- For extra security, you can assign SAS System passwords to either SAS/ACCESS descriptors or to SQL Procedure Pass-Through Facility queries that are stored as PROC SQL views.

There are some advantages to using either the SQL Procedure Pass-Through Facility or SAS/ACCESS descriptors. Use the following information to help you decide between the two methods:

- Advantages of the SQL Procedure Pass-Through Facility over view descriptors
  - You can use 32–character names in the SQL Pass-Through Facility; whereas, view descriptors are still limited to 8–character names.
  - SQL Procedure Pass-Through Facility statements enable the DBMS to optimize queries, particularly when you join tables. The DBMS optimizer can take advantage of indexes on DBMS columns to process a query more quickly and efficiently.
  - SQL Procedure Pass-Through Facility statements enable the DBMS to optimize queries when the queries have summary functions (such as AVG and COUNT), GROUP BY clauses, or columns created by expressions (such as the COMPUTED function). The DBMS optimizer can use indexes on DBMS columns to process the queries more quickly.
  - You control the SELECT statement when you use the SQL Procedure Pass-Through Facility, whereas the interface view engine generates the SELECT statement in a view descriptor.
  - On some DBMSs, you can use SQL Procedure Pass-Through Facility statements with SAS/AF applications to handle the transaction processing of the DBMS data. Using a SAS/AF application gives a user complete control of COMMIT and ROLLBACK transactions. SQL Procedure Pass-Through Facility statements give you better access to DBMS return codes.
- Advantage of the SQL Pass-Through Facility over the SAS/ACCESS LIBNAME Statement
  - The SQL Procedure Pass-Through Facility accepts all the extensions to SQL provided by your DBMS; whereas, you can use only ANSI standard SQL in the LIBNAME statement.
- Advantages of SAS/ACCESS view descriptors over the SQL Pass-Through Facility
  - SAS WHERE and BY statements can be passed to the DBMS for processing. Inthe case of view descriptors vs. PROC SQL views, view descriptors enable you to pass additional WHERE clauses to the DBMS by specifying the WHERE clauses in the selection criteria or in your SAS program.
  - In the SAS/ACCESS interfaces to CA-OpenIngres, ORACLE, Oracle Rdb, and SYBASE, you can pass certain SAS data set options (such as KEEP, RENAME, and DROP) to the DBMS. You can also pass certain sorting requests with a BY statement or ORDER BY clause to the DBMS for processing. For more information about using a BY statement with a view descriptor, see “Sorting DBMS Data” on page 125.

Creating SAS Views

Depending on which SAS/ACCESS feature you use, there are several ways to create a SAS view. A SAS view is a stored query that retrieves rows from a SAS data set or

DBMS table. Table 7.2 on page 89 lists the easiest way to create a view, depending on which feature you use.

Table 7.2 Creating SAS Views

If you use this SAS/ACCESS feature to access DBMS data...	you should use this SAS view technology...
SAS/ACCESS LIBNAME statement	PROC SQL view or DATA step view of the DBMS table
SQL Procedure Pass-Through Facility	PROC SQL view with CONNECTION TO component
ACCESS Procedure	SAS/ACCESS view, which is a PROC ACCESS view descriptor

Using SAS/ACCESS LIBNAME and Data Set Options to Improve Performance

Many of the new SAS/ACCESS LIBNAME options improve performance when you access data on your DBMS. This section provides details on some of these options. See "SAS/ACCESS LIBNAME Statement" on page 27 and "SAS/ACCESS Data Set Options" on page 43 for additional information.

Note: Some of the options discussed in this section might not be available for your DBMS. See your DBMS chapter for more information. △

Passing WHERE Clauses to the DBMS

The DBKEY= data set option should be used in contexts in which you want to join a large DBMS table and a relatively small SAS data set. If these conditions are not satisfied, the options will not improve performance and, in some cases, they might impact performance negatively.

Here is an example of how performance is improved by using this option:

```
data keyvalues;
   deptno=30; output;
   deptno=10; output;
run;

libname dblib oracle user=testuser password=testpass
  path='myorapath';

proc sql;
   select bigtab.deptno, bigtab.loc
   from dblib.dept bigtab,
        keyvalues smallds
   where bigtab.deptno = smallds.deptno;
quit;
```

In this example, the SQL procedure retrieves all the rows in the large ORACLE table DEPT and applies the WHERE clause during SQL procedure processing in SAS.

Processing can be both CPU-intensive and I/O-intensive, if MYTABLE is large. Use the DBKEY= option with the previous example and compare performance:

```
proc sql;
   select bigtab.deptno, bigtab.loc
   from  dblib.dept(dbkey=deptno) bigtab,
         keyvalues smallds
   where bigtab.deptno = smallds.deptno;
quit;
```

In this example, the DBKEY= option instructs the SQL procedure to pass the WHERE clause to the SAS/ACCESS engine in a form similar to **`where deptno=host-variable`**. The engine then passes this optimized query to the DBMS server. The *host-variable* is substituted, one at a time, with DEPTNO values from the observations in the SAS data file KEYVALUES. As a result, only rows that match the WHERE clause are retrieved from the DBMS. Without this option, PROC SQL retrieves all the rows from the DEPT table.

The SQL statement that is created by the SAS/ACCESS engine and passed to the DBMS is similar to the following;

```
select deptno, loc
   from bigtab.deptno
   where deptno=:hostvariable;
```

The *host-variable* takes the value of the DEPTNO variable from the SAS data file KEYVALUES. The number of SELECT statements issued is equal to the number of rows in the data file. Therefore, for improved performance, the SAS data file should contain relatively fewer rows than the DBMS table to which it is being joined.

The DBKEY= option can also be used in a SAS DATA step, with the KEY= option in the SET statement, to improve the performance of joins. Specify a value of KEY=DBKEY in this situation. The following DATA step creates a new data file by joining the data file KEYVALUES with the DBMS table MYTABLE. The variable DEPTNO is used with the DBKEY= option to cause a WHERE clause to be issued by the SAS/ACCESS engine.

```
data sasuser.new;
   set sasuser.keyvalues;
   set dblib.mytable(dbkey=deptno) key=dbkey;
run;
```

You can also use the DBINDEX= option instead of the DBKEY= option if you know that the DBMS table has one or more indexes that use the column(s) on which the join is being performed. Use DBINDEX=*index-name* if you know the name of the index, or use DBINDEX=YES if you do not know the name.

Sorting DBMS Data

Sorting DBMS data can be resource-intensive, whether it is done using the SORT procedure, a BY statement, or an ORDER BY clause in an SQL procedure's SELECT statement. You should sort data only when sorted data are needed for your program. The following list includes guidelines and information about sorting data:

- Say you have a data set where the libref references a relational DBMS table, and you are using that data set in a DATA step or procedure. If you specify a BY statement in a DATA or PROC step, be sure that the BY variable is associated with an indexed DBMS column. You can specify the LIBNAME option, DBINDEX=YES to try to use indexes defined on DBMS columns.

If you reference this same data set in a SAS program and the program includes a BY statement for a variable that corresponds to a column in the DBMS table, the SAS/ACCESS LIBNAME engine automatically generates an ORDER BY clause for that variable. Thus, the ORDER BY clause causes the DBMS to sort the data before the SAS procedure or DATA step uses the data in a SAS program. If the DBMS table is very large, this sorting can adversely affect your performance. Use a BY variable that is based on an indexed DBMS column to help reduce this negative impact.

- The outermost BY or ORDER BY clause overrides any embedded BY or ORDER BY clauses, including those specified in a WHERE clause. In the following example, the EXEC_EMPLOYEES data set includes a BY statement that sorts the data by the variable SENIORITY. However, when that data set is used in the following PROC SQL query, the data is ordered by the SALARY column and not by SENIORITY.

```
libname mydblib oracle user=sasha password=bee path="hrdb_023"
schema=hrdept;
data exec_employees;
   set mydblib.staff (keep lastname firstname idnum);
   by seniority;
   where salary >= 150000;
run;

proc sql;
select * from exec_employees
  order by salary;
```

Passing Joins to the DBMS

In versions prior to Version 7, an SQL query involving one or more DBMS tables (or view descriptors, in Version 6) was processed by SAS as if the DBMS tables were individual SAS data files. The SQL procedure fetched all the rows from each DBMS table and performed the join within the SAS. This algorithm performed poorly, especially if the tables were large and SAS and the DBMS communicated over a network.

A more efficient method is to let the DBMS perform the join and return only the results of the join to the client, in this case, SAS. In Version 7 and later, this is how joins are processed which provides a major performance enhancement for programs that perform joins across tables in a single DBMS.

For example, assume two large DBMS tables named TABLE1 and TABLE2 have a column named DEPTNO, and you want to retrieve the rows from an inner join of these tables where the DEPTNO value in TABLE1 is equal to the DEPTNO value in TABLE2.

```
proc sql;
   select tab1.deptno, dname from
      dblib.table1 tab1
      dblib.table2 tab2
      where tab1.deptno = tab2.deptno
         using libname dblib oracle user=scott
            password=tiger path='myserver';
quit;
```

In Version 8, this join between two tables within the same library (where the libref references a database) is detected by the SQL procedure and passed by the SAS/ACCESS engine directly to the DBMS. The DBMS processes the inner join between the

two tables, and only the resulting rows are passed back to the SAS System. Both inner and outer joins between two or more DBMS tables are supported in this enhancement.

Defining a Schema

Some SAS/ACCESS engines support LIBNAME options that restrict or qualify the scope, or schema, of the tables in the libref. For example, the DB2 engine supports the AUTHID= and LOCATION= options, and the ORACLE engine supports the SCHEMA= and DBLINK= options. The options vary depending on which SAS/ACCESS interface you are using. See your DBMS chapter to determine which options are available for your DBMS.

The following example uses the SAS/ACCESS Interface to ORACLE:

```
libname myoralib oracle user=testuser
password=testpass
   path='myoraserver' schema=testgroup;

proc datasets lib=myoralib;
run;
```

In this example, the MYORALIB libref is associated with the ORACLE schema named TESTGROUP. The DATASETS procedure lists only the tables and views that are accessible to the TESTGROUP schema. Any reference to a table that uses the libref MYORALIB is passed to the ORACLE server as a qualified table name; for example, if the SAS program reads a table by specifying the SAS data set MYORALIB.TESTTABLE, the SAS/ACCESS engine passes the following query to the server:

```
select * from "testgroup.testtable"
```

Dynamic Prompting for DBMS Connection Options

In Version 6 of SAS/ACCESS software, users stored their DBMS connection information, such as usernames and passwords, in SAS programs, or in encrypted form within an access or view descriptor. Beginning in Version 7, you can specify DBMS connection information in a more dynamic, interactive manner by using the LIBNAME option, DBPROMPT=. When you specify DBPROMPT=YES on the LIBNAME statement, as in the following example, a dialog window appears to prompt the user to enter connection information, such as a username and password.

```
libname myoralib oracle dbprompt=yes defer=no;
```

When this LIBNAME statement is issued, a prompt window appears. The window contains the DBMS connection options that are valid for the SAS/ACCESS engine that is being used, in this case, ORACLE.

This feature provides several advantages. DBMS account passwords are protected because they do not need to be stored in a SAS program or descriptor file. Also, when a password or username changes, the SAS program does not need to be modified. Another advantage is that the same SAS program can be used by any valid username and password combination that is specified during execution. You can also use connection options in this interactive manner when you want to run a program on a production server instead of testing a server without making modifications to your code. By using the prompt window, the new server name can be specified dynamically.

Note: This option is not available in the SAS/ACCESS Interface to DB2 Under OS/390. △

Controlling DBMS Connections

Because the overhead of executing a connection to a DBMS server can be resource-intensive, SAS/ACCESS engines support the CONNECTION= and DEFER= options to control when a DBMS connection is made, and how many connections are executed within the context of your SAS/ACCESS application. For most SAS/ACCESS engines, a connection to a DBMS begins one transaction, or work unit, and all statements issued in the connection execute within the context of the active transaction.

The CONNECTION= LIBNAME option allows you to specify how many connections are executed when the library is used and which operations on tables are shared within a connection. By default, the value is CONNECTION=SHAREDREAD, which means that a SAS/ACCESS engine executes a *shared read* DBMS connection when the library is assigned. Every time a table in the library is read, the read-only connection is used. However, if an application attempts to update data using the libref, a separate connection is issued, and the update occurs in the new connection. As a result, there is one connection for read-only transactions and a separate connection for each update transaction.

In the following example, the SAS/ACCESS engine issues a connection to the DBMS when the libref is assigned. The PRINT procedure reads the table by using the first connection. When the SQL procedure updates the table, the update is performed with a second connection to the DBMS.

```
libname myoralib user=testuser password=testpass
   path='myoraserver';

proc print data=myoralib.mytable;
run;

proc sql;
  update myoralib.mytable set acctnum=123
    where acctnum=456;
quit;
```

The following example uses the SAS/ACCESS Interface to DB2 under OS/390. The LIBNAME statement executes a connection by way of the DB2 Call Attach Facility to the DB2 DBMS server:

```
libname mydb2lib db2 authid=testuser;
```

If you want to assign more than one SAS libref to your DBMS server, and if you do not plan to update the DBMS tables, SAS/ACCESS provides an option to allow you to optimize the way the engine performs connections. Your SAS librefs can share a single read-only connection to the DBMS if you use the CONNECTION=GLOBALREAD option. The following example shows you how to use the CONNECTION= option with the ACCESS= option to control your connection and to specify read-only data access.

```
libname mydblib1 db2 authid=testuser
   connection=globalread access=readonly;
```

If you do not want the connection to occur when the library is assigned, you can delay the connection to the DBMS by using the DEFER= option. When you specify DEFER=YES on the LIBNAME statement, the SAS/ACCESS engine connects to the

DBMS the first time a DBMS object is referenced in a SAS program, as in the following example:

```
libname mydb2lib db2 authid=testuser defer=yes;
```

Note: If you use DEFER=YES to assign librefs to your DBMS tables and views in an AUTOEXEC program, the processing of the AUTOEXEC file will be faster because the connections to the DBMS are not made every time SAS is invoked. △

Customizing DBMS Connect and Disconnect Exits

You can specify DBMS commands or stored procedures to be executed immediately after a DBMS connection or before a DBMS disconnect by using the LIBNAME options DBCONINIT= and DBCONTERM=, as in the following example:

```
libname myoralib oracle user=testuser
   password=testpass path='myoraserver'
   dbconinit="EXEC MY_PROCEDURE";

proc sql;
   update myoralib.mytable set acctnum=123
      where acctnum=567;
quit;
```

When the libref is assigned, the SAS/ACCESS engine connects to the DBMS and passes a command to the DBMS to execute the stored procedure MY_PROCEDURE. By default, a new connection to the DBMS is made for every table that is opened for updating, so MY_PROCEDURE is executed a second time after a connection is made to update the table MYTABLE.

To execute a DBMS command or stored procedure *only* after the first connection in a library assignment, you can use the DBLIBINIT= option. Similarly, the DBLIBTERM= option enables you to specify a command to be executed prior to the disconnection of only the first library connection, as in the following example.

```
libname myoralib oracle user=testuser password=testpass
   dblibinit="EXEC MY_INIT" dblibterm="EXEC MY_TERM";
```

Note: DBLIBINIT= and DBCONINIT= are related LIBNAME options. For more information, see Chapter 3, "SAS/ACCESS LIBNAME Statement". △

Locking, Transactions, and Currency Control

Beginning in Version 7, SAS/ACCESS provides options that allow you to control some of the row, page, or table locking operations that are performed by the DBMS and the SAS/ACCESS engine as your programs are executed. For example, the SAS/ACCESS Interface to ORACLE has the READ_LOCK_TYPE=, UPDATE_LOCK_TYPE=, READ_ISOLATION_LEVEL=, UPDATE_ISOLATION_LEVEL=, and LOCKWAIT= options. By default, the SAS/ACCESS ORACLE engine does not lock any data when reading rows from ORACLE tables. However, you can override this behavior by using these options.

If you want to lock the data pages of a table while the SAS System is reading the data to prevent other processes from updating the table, you can use the READLOCK_TYPE= option, as in the following example:

```
libname myoralib oracle user=testuser pass=testpass
   path='myoraserver' readlock_type=table;

data work.mydata;
   set myoralib.mytable(where=(colnum > 123));
run;
```

In this example, the SAS/ACCESS ORACLE engine obtains a TABLE SHARE lock on the table so that the data cannot be updated by other processes while your SAS program is reading it.

In the following example, ORACLE acquires row-level locks on rows read for update in the tables in the libref.

```
libname myoralib oracle user=testuser password=testpass
   path='myoraserver' updatelock_type=row;
```

Note: Each SAS/ACCESS interface supports specific options; see your DBMS chapter to determine which options are supported for your DBMS. △

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