CHAPTER

6

SQL Procedure's Interaction with SAS/ACCESS Software

Introduction

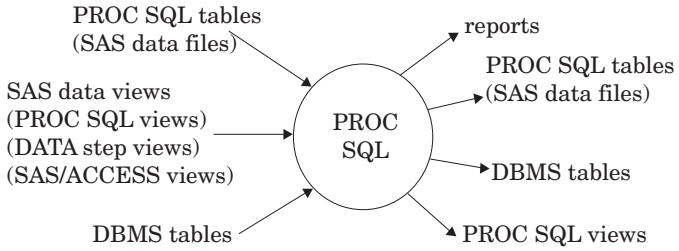
The SQL procedure implements the Structured Query Language (SQL) for the SAS System. SQL is a standardized, widely used language that retrieves and updates data in tables and views based on those tables.

The SAS System's SQL procedure enables you to

- retrieve and manipulate data that is stored in tables or views.
- create tables, views, and indexes (other than relational DBMS indexes) on SAS variables in SAS data sets.
- add or modify the data values in a table's columns or insert and delete rows. You can also modify the table itself by adding, modifying, or dropping columns, although this does not apply to relational DBMS tables.
- send DBMS-specific SQL statements to a relational database management system (RDBMS) and to retrieve RDBMS data.
- create SAS macro variables that contain values from rows in a query's result.

Figure 6.1 on page 66 summarizes the variety of source material that you can use with PROC SQL and what the procedure can produce:

Figure 6.1 PROC SQL Input and Output



Version 7 and Version 8 Enhancements

In Version 7 and later, the SQL Procedure has several enhancements to the ways in which it interacts with SAS/ACCESS software.

- After you assign a libref to a relational DBMS (RDBMS) using the SAS/ACCESS LIBNAME statement, you can reference this new libref in a PROC SQL statement to query, update, or delete RDBMS data.
- You can embed LIBNAME information in a PROC SQL view by using the new USING LIBNAME syntax; therefore, every time the PROC SQL view is processed, you automatically connect to the RDBMS and can access its data. For more information, see the SQL Procedure's CREATE VIEW statement in *SAS Procedures Guide*.
- In the Pass-Through Facility's CONNECT statement, you can now specify arguments that indicate whether a connection to a relational database is unique or shared, whether you want to be prompted by a window in order to enter your DBMS-specific connection information, and so on. For more information, see "CONNECT Statement Arguments" on page 75
- You can specify a simple PROC SQL view—a view that is based on one table—in any SAS application to update, insert, or delete the view's underlying data. For an example of this new feature, see the SQL Procedure's UPDATE statement in *SAS Procedures Guide*.
- Joins are passed to the RDBMS to process whenever possible. For example, before implementing a join, PROC SQL checks to see if the RDBMS can do the join. If it can, PROC SQL passes the join to the RDBMS. This increases performance by reducing data movement and translation.
- You can use the DQOUTE=ANSI option to enable you to use names that are not normally permissible in SAS as table names or column names.
- You can use the DBINDEX= and DBKEY= data set options to specify an index name or the column names that make up the index in order to improve performance. For more information, see "SAS/ACCESS Data Set Options" on page 43.

See Chapter 12, "Using DBMS Data in Version 7 and Version 8," on page 151 for additional examples of easier and more direct ways of using DBMS data in SAS/ACCESS software in Version 7 and later.

What Are PROC SQL Tables?

A PROC SQL *table* is synonymous with a SAS data file and has a member type of DATA. You can use PROC SQL tables as input into DATA steps and procedures.

You create PROC SQL tables from SAS data files, from SAS data views, from relational DBMS tables using the LIBNAME statement, or from relational DBMS tables using the SQL Procedure Pass-Through Facility. The SQL Procedure Pass-Through Facility is described in "SQL Procedure Pass-Through Facility" on page 72.

In PROC SQL terminology, a *row* in a table is the same as an *observation* in a SAS data set. A *column* is the same as a *variable*.

Note: In this chapter, the term *table* refers to PROC SQL tables, relational DBMS tables, SAS data files, and relational DBMS views unless otherwise noted. △

What Are Views?

A *SAS data view* defines a virtual data set that is named and stored for later use. A view contains no data but describes or defines data that are stored elsewhere. There are three types of SAS data views:

- PROC SQL views
- SAS/ACCESS views (also referred to as *view descriptors*)
- DATA step views.

You can specify views in queries as if they were tables. The view derives its data from the tables or views that are listed in its FROM clause. The data accessed by a view are a subset or superset of the data in its underlying table(s) or view(s).

A PROC SQL view is a SAS file of type VIEW created by PROC SQL. A PROC SQL view contains no data. It is a stored query expression that reads data values from its underlying files, which can include SAS data files, SAS/ACCESS views, DATA step views, other PROC SQL views, or relational DBMS data. When executed, a PROC SQL view's output can be a subset or superset of one or more underlying files. Beginning in Version 7, you can reference a simple PROC SQL view to update its underlying data.

SAS/ACCESS views and DATA step views are similar to PROC SQL views in that they are both data definitions of member type VIEW. SAS/ACCESS views describe data in DBMS tables from other software vendors. DATA step views are stored, compiled DATA step programs.

PROC SQL views can be used to update their underlying data if the view is based on only one DBMS table or on a DBMS view that is based on only one DBMS table and has no calculated fields. DATA step views can only read their underlying data.

You can use all types of views as input into DATA steps and procedures.

Note: In this chapter, the term *view* refers to PROC SQL views, DATA step views, and SAS/ACCESS views, unless otherwise noted. △

What Is the SQL Query Window?

The SQL Query Window is a graphical user interface (GUI) to the features that are offered by the SAS SQL procedure. It enables you to query data without writing programming statements. To invoke the QUERY window and access its online documentation, issue the **`QUERY`** command from your SAS session's command box or command line.

Passing Joins to the DBMS

Prior to Version 7 of the SAS System, an SQL query involving one or more DBMS tables or view descriptors was processed by the SQL procedure as if the DBMS tables were individual SAS files. For view descriptors, the SQL procedure fetched all the rows from each DBMS table and then performed the join processing within SAS.

Although the SQL Procedure Pass-Through Facility has always passed joins to the DBMS, it is now possible to pass joins to the DBMS without using Pass-Through. Beginning in Version 7, the LIBNAME engine allows you to pass joins to the DBMS without using Pass-Through but with the same performance benefits. The DBMS server will perform the join and return only the results of the join to the SAS software. This will provide a major performance enhancement for many of your programs that perform joins across tables in a single DBMS. Both inner and outer joins are supported in this new enhancement.

In this example, two large DBMS tables, TABLE1 and TABLE2, have a column named DEPTNO. An inner join of these tables is performed where the DEPTNO value in TABLE1 is equal to the DEPTNO value in TABLE2. This join will be detected by the SQL Procedure and passed by the SAS/ACCESS engine directly to the DBMS server. The resulting rows will be passed back to the SAS System.

```
proc sql;
select tab1.deptno, dname from
   mydblib.table1 tab1,
   mydblib.table2 tab2
where tab1.deptno=tab2.deptno
   using libname mydblib oracle user=testuser
     password=testpass path=myserver;
```

If you want to perform a join between a large DBMS table and a relatively small SAS data file, you may want to specify the DBKEY= data set option. The DBKEY= data set option causes the SQL Procedure to pass a WHERE clause to the DBMS so that only the rows that match the WHERE condition are retrieved from the DBMS table. Also, if DEPTNO has an ORACLE index defined on it, using DBKEY= will greatly enhance the join's performance. In this example, the DBKEY= option causes only the rows that match DEPTNO to be retrieved. Without this option, the SQL Procedure would retrieve all the rows from TABLE1.

```
libname mydblib oracle user=testuser
password=testpass;
proc sql;
select tab1.deptno, loc from
   mydblib.table1 (dbkey=deptno) tab1,
   sasuser.sasds tab2
   where tab1.deptno=tab2.deptno;
```

For more information on this data set option, see Chapter 4, "SAS/ACCESS Data Set Options" .

Long Names and Case Sensitivity in the SQL Procedure and Pass-Through Facility

Beginning in Version 7 of SAS software, SAS variable names and member names can be up to 32 characters long. Column names can be case-sensitive also. Some

DBMSs allow case sensitive column and table names as well as names with special characters such as an Oracle column named **`Amount Budgeted$`**. Therefore, special consideration should be used when the names of DBMS objects, such as tables and columns, are used with SAS/ACCESS and the SQL Procedure Pass-Through Facility. See your DBMS chapter for more information about the SAS System and how it interfaces with your DBMS names.

When your SAS/ACCESS engine is reading column names that do not conform to the SAS naming conventions, unsupported characters, such as spaces, are replaced with underscores (_). This is the default behavior. For example, the column name **`Amount Budgeted$`** becomes the SAS variable name **`Amount_Budgeted_`**. If the DBMS name is longer than 32 characters, then the name is truncated. If necessary, numbers are applied to the end of the name to make it unique. See "The DQUOTE=ANSI PROC SQL Option" on page 69 to override this default renaming algorithm.

In the following example, a connection is made to an ORACLE database and a view, MYVIEW, is created from the table, MYTABLE. The output produced by PROC CONTENTS would show that the ORACLE column names, that were processed by the SQL Pass-Through view of MYTABLE, were renamed to different SAS variable names: "Amount Budgeted$" becomes "Amount_Budgeted_" and "Amount Spent$" becomes "Amount_Spent_".

```
proc sql;
  connect to oracle (user=testuser pass=testpass);
  create view myview as
    select * from connection to oracle
      (select "Amount Budgeted$", "Amount Spent$"
        from mytable);
quit;
proc contents data=myview;
run;
```

See Chapter 2, "SAS Names and Support for DBMS Names," on page 9 for more information about SAS names and DBMS names.

The DQUOTE=ANSI PROC SQL Option

The PROC SQL option DQUOTE=ANSI can be specified to enable support for DBMS names with special characters. When you specify DQUOTE=ANSI, table and column names in your SQL statements can be enclosed in double quotes to preserve any special characters.

In the following example, a connection is made to an ORACLE database. By specifying DQUOTE=ANSI and double quoting the SAS names in the SELECT statement, the special characters are preserved in the output.

```
proc sql dquote=ansi;
  connect to oracle (user=testuser pass=testpass);
  create view myview as
    select "Amount Budgeted$", "Amount Spent$"
    from connection to oracle
      (select "Amount Budgeted$", "Amount Spent$"
        from mytable);
quit;
proc contents data=myview;
run;
```

Output from this example would show that "Amount Budgeted$" remains "Amount Budgeted$" and "Amount Spent$" remains "Amount Spent$".

You also can use the global system option VALIDVARNAME= ANY to override the SAS naming conventions. See Chapter 2, “SAS Names and Support for DBMS Names,” on page 9 for more information.

See Chapter 2, “SAS Names and Support for DBMS Names,” on page 9 for more information about SAS names and DBMS names.

Version 6 Names Compatibility

If you have existing PROC SQL Pass-Through applications that were written using Version 6 of the SAS System, you can still run them by specifying the global system option VALIDVARNAME=V6. In Version 6, SAS variable names were uppercased and truncated to 8 characters. The following example shows how the PROC SQL Pass-Through facility works in Version 6 compatibility mode.

```
options validvarname=v6;
proc sql;
  connect to oracle (user=testuser pass=testpass);
  create view myview as
    select amount_b amount_s
    from connection to oracle
      (select "Amount Budgeted$", "Amount Spent$"
        from mytable);
quit;
options validvarname=v6;
proc contents data=myview;
run;
```

Output from this example would show that "Amount Budgeted$" becomes "AMOUNT_B" and "Amount Spent$" becomes "AMOUNT_S" .

See Chapter 2, “SAS Names and Support for DBMS Names,” on page 9 for more information.

SQL Procedure Syntax

PROC SQL <*option(s)*>;

ALTER TABLE *SQL-table-name*
<*constraint-clause*> <, *constraint-clause*>…>;
<**ADD** column-definition <,column-definition>…>
<**MODIFY** column-definition
<,column-definition>…>
<**DROP** *column* <,*column*>…>;

CREATE <**UNIQUE**> **INDEX** *index-name*
ON *table-name (column* <, *column*>…);

CREATE TABLE *table-name* (column-definition <,column-definition>…);
(column-specification, ...<*constraint-specification*> ,...);

CREATE TABLE *table-name* **LIKE** *table-name*;

CREATE TABLE *table-name* **AS** query-expression
<**ORDER BY** *order-by-item* <,*order-by-item*>…>;

CREATE VIEW *SQL-view-name* **AS** query-expression

<**ORDER BY** *order-by-item* <,*order-by-item*>…>;
<**USING** *libname-clause*
<, *libname-clause*>...>;
DELETE
FROM *table-name*|*sas/access-view*|*SQL-view-name* <**AS** *alias*>
<**WHERE**sql-expression>;
DESCRIBE TABLE*table-name*<,*table-name*>… ;
DESCRIBE TABLE CONSTRAINTS *table-name*<, *table-name*>… ;
DESCRIBE VIEW *SQL-view-name*<, *proc-sql-view*>… ;
DROP INDEX *index-name* <,*index-name*>…
FROM *SQL-table-name*;
DROP TABLE *table-name* <,*table-name*>… ;
DROP VIEW *SQL-view-name* <,*SQL-view-name*>…;
INSERT INTO *table-name*|*sas/access-view*|*SQL-view-name*
<(*column*<,*column*>…)>
SET *column*=sql-expression
<,*column*=sql-expression>…
<**SET** *column*=sql-expression
<,*column*=sql-expression>…>;
INSERT INTO *table-name*|*sas/access-view*|*SQL-view-name*
<(*column*<,*column*>…)>
VALUES (*value* <, *value*>…)
<**VALUES** (*value* <, *value*>…)>…;
INSERT INTO *table-name*|*sas/access-view*|*SQL-view-name*
<(*column*<,*column*>…)> query-expression;
RESET <*option(s)*>;
SELECT <DISTINCT> *object-item* <,*object-item*>…
<**INTO** :*macro-variable-specification*
<, :*macro-variable-specification*>…>
FROM *from-list*
<**WHERE** sql-expression>
<**GROUP BY** *group-by-item*
<,*group-by-item*>…>
<**HAVING** sql-expression>
<**ORDER BY** *order-by-item*
<,*order-by-item*>…>;
UPDATE *table-name*|*sas/access-view*|*view-name* <**AS** *alias*>
SET *column*=sql-expression
<,*column*=sql-expression>…
<**SET***column*=sql-expression
<,*column*=sql-expression>…>
<**WHERE** sql-expression>;
VALIDATEquery-expression;

Complete PROC SQL syntax information can be found in the *SAS Procedures Guide*.

To do this	Use this statement
Modify, add, or drop columns	ALTER TABLE*
Establish a connection with a DBMS	LIBNAME, CONNECT
Create an index on a column	CREATE INDEX*

To do this	Use this statement
Create a table	CREATE TABLE
Create a PROC SQL view	CREATE VIEW*
Delete rows from a table or view (does not apply to DATA step views)	DELETE
Display a definition of a table or view	DESCRIBE
Terminate the connection with a DBMS	LIBNAME CLEAR, DISCONNECT
Delete tables, views, or indexes	DROP
Add rows to a table or view (does not apply to DATA step views)	INSERT
Reset options that affect the procedure environment without restarting the procedure	RESET
Select rows from a table	SELECT
Modify values	UPDATE
Verify the accuracy of your query	VALIDATE*

* Does not apply to DBMS tables. To add a column to a DBMS table, you use the DBMS-specific SQL ALTER statement.

SQL Procedure Pass-Through Facility

The SQL Procedure Pass-Through Facility is an extension to the SQL procedure that enables you to send DBMS-specific statements to a database management system and to retrieve DBMS data directly. However, because the SQL Procedure Pass-Through Facility uses a SAS/ACCESS engine to connect to the DBMS, you must have SAS/ACCESS software in order to use it. You specify DBMS syntax instead of SAS SQL syntax when you are using the SQL Procedure Pass-Through Facility.

Beginning in Version 7 of SAS/ACCESS software, there are a number of changes in the way the SQL Procedure Pass-Through Facility works. For more information, see "SQL Procedure Pass-Through Facility Statements" on page 74.

This section provides general reference information for the SQL Procedure Pass-Through Facility. See your DBMS chapter for detailed information about using your database's arguments and options. See Chapter 13, "Using DBMS Data with the SQL Pass-Through Facility," on page 179 for usage examples.

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

- The CONNECT statement establishes a connection with 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 with the DBMS.

You can use the SQL Procedure Pass-Through Facility statements in a PROC SQL query, or you can store them in a PROC SQL view. When you create a PROC SQL view, any arguments that you specify in the corresponding CONNECT statement are stored also. Thus, when the PROC SQL view is used in a SAS program, the SAS System can establish the appropriate connection to the DBMS.

See the *SAS Procedures Guide* for information on changes and enhancements to the SQL procedure.

Syntax for the SQL Procedure Pass-Through Facility

To connect to a DBMS and send it a DBMS-specific, nonquery SQL statement, use the statements that comprise the SQL Procedure Pass-Through Facility:

PROC SQL;
  <**CONNECT TO** *dbms-name* <AS *alias*><
    <(*connect-statement-argument-1=value*
    …<*connect-statement-argument-n=value*>)>>
    <(*dbms-argument-1=value*
    …<*dbms-argument-n=value*>)>>;
  EXECUTE (*dbms-SQL-statement*)
    BY *dbms-name*|*alias*;
  <**DISCONNECT FROM** *dbms-name*|*alias*;>
<**QUIT**;>

For information about the CONNECT, EXECUTE, and DISCONNECT statements as well as the CONNECTION TO component, see "SQL Procedure Pass-Through Facility Statements" on page 74.

To connect to a DBMS and query the DBMS data, use the SAS/ACCESS LIBNAME statement or use the PROC SQL Pass-Through statement CONNECTION TO in this form:

PROC SQL;
  <**CONNECT TO** *dbms-name* <AS *alias*><
    <(*connect-statement-argument-1=value*
    …<*connect-statement-argument-n=value*>)>>
    <(*dbms-argument-1=value*
    …<*dbms-argument-n=value*>)>>;
  SELECT *column-list*
    FROM CONNECTION TO *dbms-name*|*alias*
      (*dbms-query*)
      optional PROC SQL clauses;
  <**DISCONNECT FROM** *dbms-name*|*alias*;>
<**QUIT**;>

To do this	Use this statement
Establish a connection with a DBMS	CONNECT
Terminate the connection with a DBMS	DISCONNECT

To do this	Use this statement
Send a DBMS-specific, non-query SQL statement to a DBMS	EXECUTE
Query a DBMS using a DBMS-specific SQL statement	SELECT and FROM CONNECTION TO

SQL Procedure Pass-Through Facility Statements

The following sections describe the PROC SQL statements and component that comprise the SQL Procedure Pass-Through Facility. The statements are listed in alphabetical order, followed by the CONNECTION TO component.

For examples that illustrate the use of the SQL Procedure Pass-Through Facility, refer to the following:

- "Retrieving DBMS Data with a Pass-Through Query" on page 179
- "Using a Pass-Through Query in a Subquery" on page 182.

CONNECT Statement

Establishes a connection with the DBMS

Contains DBMS-specific arguments

Optional statement for some DBMSs

Syntax

CONNECT TO *dbms-name* <AS *alias*> <(<*connect-statement-arguments*> <*database-connection-arguments*>)>;

The CONNECT statement establishes a connection with the DBMS. You establish a connection to send DBMS-specific SQL statements to the DBMS or to retrieve DBMS data. The connection remains in effect until you issue a DISCONNECT statement or terminate the SQL procedure.

Using the CONNECT statement is optional for some DBMSs. However, if it is not specified, the default values for all of the database connection arguments are used.

Any return code or message that is generated by the DBMS is available in the macro variables SQLXRC and SQLXMSG after the statement executes. See "SQL Procedure Pass-Through Facility Return Codes" on page 83 for more information on these macro variables.

Arguments

You use the following arguments with the CONNECT statement:

dbms-name
identifies the database management system to which you want to connect. You must specify a DBMS name, which is listed in your DBMS chapter. You may also specify an optional alias in the CONNECT statement.

alias
specifies an optional alias for the Pass-Through connection that has 1 to 32 characters. If you specify an alias, the keyword AS must appear before the alias. If an alias is not specified, the DBMS name is used as the name of the Pass-Through connection.

connect-statement-arguments
specifies arguments that indicate whether you can make multiple connections, shared or unique connections, and so on to the database. These arguments are optional but if they are included, they must be enclosed in parentheses. For details, see "CONNECT Statement Arguments" on page 75.

database-connection-arguments
specifies the DBMS-specific arguments that are needed by PROC SQL to connect to the DBMS. These arguments are optional for most databases, but if they are included, they must be enclosed in parentheses. See your DBMS chapter for information about these arguments.

CONNECT Statement Arguments

The following list describes the general arguments that are used for the PROC SQL Pass-Through CONNECT statement.

CONNECTION=SHARED | GLOBAL
indicates whether multiple connect statements for a DBMS can use the same connection. See your DBMS chapter for DBMS-specific arguments for CONNECTION=.

Default value: SHARED

The CONNECTION= option enables you to control the number of connections, and therefore transactions, that your SAS/ACCESS engine executes and supports for each CONNECT statement.

This option is supported by the SAS/ACCESS engines that support multiple, simultaneous connections to the DBMS. For most of these SAS/ACCESS engines, there must be a connection, also known as an attach, to the DBMS server before any data can be accessed. Typically, each DBMS connection has one transaction, or work unit, active in the connection. This transaction is affected by any SQL COMMITs or ROLLBACKs that are performed within the connection while executing the Pass-Through statements.

The values for CONNECTION= are as follows:

SHARED
When CONNECTION=SHARED, the CONNECT statement makes one connection to the DBMS. All Pass-Through statements that use this alias share this connection.

SHARED is the default value for CONNECTION= .

GLOBAL
When CONNECTION=GLOBAL, multiple CONNECT statements that use identical values for all options can share the same connection to the DBMS.

In the following example, the two CONNECT statements, MYDBONE and MYDBTWO, share the same connection to the DBMS because CONNECTION=GLOBAL. Only the first CONNECT statement actually makes the connection to the DBMS while the last DISCONNECT statement is the only statement that disconnects from the DBMS.

```
proc sql;
```

```
connect to oracle as mydbone
   (user=testuser pw=testpass
     path='abc'
     connection=global);

...SQL Pass-Through statements referring
   to mydbone...

connect to oracle as mydbtwo
   (user=testuser pw=testpass
     path='abc'
     connection=global);

...SQL Pass-Through statements referring
   to mydbtwo...

disconnect from mydbone;
disconnect from mydbtwo;
```

CONNECTION_GROUP=*connection_group_name*
specifies a connection that can be shared among several CONNECT statements in the SQL Procedure Pass-Through Facility.

Default value: none

By specifying the name of a connection group, you can share one DBMS connection among several different CONNECT statements. The connection to the DBMS can be shared only if each CONNECT statement specifies the same CONNECTION_GROUP= value and specifies identical DBMS connection options.

When CONNECTION_GROUP= is specified, it implies that the value of the CONNECTION= option will be GLOBAL.

DBCONINIT=<'>*DBMS-user-command*<'>
specifies a user-defined initialization command to be executed immediately after the connection to the DBMS.

You can specify any DBMS command that can be passed by the SAS/ACCESS engine to the DBMS and that does not return a result set or output parameters. The command executes immediately after the DBMS connection is established successfully. If the command fails, a disconnect occurs, and the CONNECT statement fails. You must specify the command as a single, quoted string, unless it is an environment variable.

See "SAS/ACCESS LIBNAME Statement" on page 27 for an example.

DBCONTERM=<'>*DBMS-user-command*<'>
specifies a user-defined termination command to be executed before the disconnect from the DBMS that occurs with the DISCONNECT statement.

Default value: none

The termination command that you select can be a script, stored procedure, or any DBMS SQL language statement that might provide additional control over the interaction between the SAS/ACCESS engine and the DBMS. You can specify any valid DBMS command that can be passed by the SAS/ACCESS engine to the DBMS and that does not return a result set or output parameters. The command executes immediately before SAS terminates each connection to the DBMS. If the command fails, SAS provides a warning message but the disconnect still occurs. You must specify the command as a quoted string.

See "SAS/ACCESS LIBNAME Statement" on page 27 for an example.

DBPROMPT=YES | NO
specifies whether SAS displays a window that prompts the user to enter DBMS connection information prior to connecting to the DBMS.

Default value: NO

If you specify DBPROMPT=YES, SAS displays a window that interactively prompts you for the DBMS connection options when the CONNECT statement is executed. Therefore, it is not necessary to provide connection options with the CONNECT statement. If you do specify connection options with the CONNECT statement and you specify DBPROMPT=YES, the connection option values are displayed in the window. These values can be overridden interactively.

If you specify DBPROMPT=NO, SAS does not display the prompting window.

The DBPROMPT= option interacts with the DEFER= option to determine when the prompt window appears. If DEFER=NO, the DBPROMPT window opens when the CONNECT statement is executed. If DEFER=YES, the DBPROMPT window opens the first time a Pass-Through statement is executed. The DEFER= option normally defaults to NO but defaults to YES if DBPROMPT=YES. You can override this default by explicitly setting DEFER=NO.

See "SAS/ACCESS LIBNAME Statement" on page 27 for an example.

DEFER=NO | YES
determines when the connection to the DBMS occurs.

Default value: NO

If DEFER=YES, the connection to the DBMS occurs when the first Pass-Through statement is executed. If DEFER=NO, the connection to the DBMS occurs when the CONNECT statement occurs.

VALIDVARNAME=V6
indicates that only those variable names considered valid SAS variable names in Version 6 of SAS software are considered valid. Specify this connection argument if you want Pass-Through to operate in Version 6 compatibility mode.

When V6 is specified in SQL Pass-Through code, the SAS/ACCESS engine for the DBMS truncates column names to eight characters, as it did in Version 6. If required, numbers are appended to the ends of the truncated names to make them unique. Setting this option overrides the value of the SAS system option, VALIDVARNAME= during (and only during) the SQL Pass-Through connection.

The following example shows how the Pass-Through Facility uses VALIDVARNAME=V6 as a connection argument. Using this option causes the output to show the DBMS column "Amount Budgeted$" as AMOUNT_B and "Amount Spent$" as AMOUNT_S.

```
proc sql;
connect to oracle (user=gloria password=teacher
                   validvarname=v6)
create view budget2000 as
  select amount_b, amount_s
  from connection to oracle
     (select "Amount Budgeted$", "Amount Spent$"
         from annual_budget);
quit;
proc contents data=budget2000;
run;
```

In this example, if you had *omitted* VALIDVARNAME=V6 as a connection argument, you would have had to have added it in an OPTIONS= statement in order for PROC CONTENTS to work:

```
options validvarname=v6;
proc contents data=budget2000;
run;
```

Thus, using it as a connection argument saves you coding later.

Example

The following example connects to a SYBASE server and assigns the alias SYBCON1 to it. SYBASE is a case sensitive database; therefore, the database objects are in uppercase, as they were created.

```
proc sql;
   connect to sybase as sybcon1
      (server=SERVER1 database=PERSONNEL
         user=testuser password=testpass
         connection=global);
%put &sqlxmsg &sqlxrc;
```

DISCONNECT Statement

Terminates a connection to the DBMS

Optional statement

Syntax

DISCONNECT FROM *dbms-name* | *alias*

The DISCONNECT statement ends the connection with the DBMS. If the DISCONNECT statement is omitted, an implicit DISCONNECT is performed when PROC SQL terminates. The SQL procedure continues to execute until you submit a QUIT statement, another SAS procedure, or a DATA step.

Any return code or message that is generated by the DBMS is available in the macro variables SQLXRC and SQLXMSG after the statement executes. See “SQL Procedure Pass-Through Facility Return Codes” on page 83 for more information on these macro variables.

Arguments

You use one of the following arguments with the DISCONNECT statement:

dbms-name
specifies the database management system from which you want to disconnect. You must specify a DBMS name, which is listed in your DBMS chapter, or use an alias in the DISCONNECT statement.

Note: If you used the CONNECT statement to connect to the DBMS, the DISCONNECT statement’s DBMS name or alias must match the name or alias that you specified in the CONNECT statement. △

alias
specifies an alias that was defined in the CONNECT statement.

Example

The following example disconnects the user from a DB2 database with the alias DBCON1 and terminates the SQL procedure:

```
proc sql;
   connect to db2 as dbcon1 (ssid=db2a);
   ....
   disconnect from dbcon1;
quit;
```

EXECUTE Statement

Sends DBMS-specific, non-query SQL statements to the DBMS

Contains DBMS-specific arguments
Optional statement

Syntax

EXECUTE (*DBMS-specific-SQL-statement*) BY *dbms-name* | *alias*;

The EXECUTE statement sends dynamic non-query, DBMS-specific SQL statements to the DBMS and processes those statements.

In some SAS/ACCESS interfaces, you can issue an EXECUTE statement directly without first explicitly connecting to a DBMS (see "CONNECT Statement" on page 74). If you omit the CONNECT statement, an implicit connection is performed (by using default values for all database connection arguments) when the first EXECUTE statement is passed to the DBMS. See your DBMS chapter for details.

The EXECUTE statement cannot be stored as part of an SQL Procedure Pass-Through Facility query in a PROC SQL view.

Arguments

(*DBMS-specific–SQL-statement*)
a dynamic non-query, DBMS-specific SQL statement. This argument is required and must be enclosed in parentheses. However, the SQL statement cannot contain a semicolon because a semicolon represents the end of a statement in the SAS System. The SQL statement may be case-sensitive, depending on your DBMS, and it is passed to the DBMS exactly as you type it.

Any return code or message that is generated by the DBMS is available in the macro variables SQLXRC and SQLXMSG after the statement executes. See "SQL Procedure Pass-Through Facility Return Codes" on page 83 for more information on these macro variables.

dbms-name
identifies the database management system to which you direct the DBMS-specific SQL statement. The name for your DBMS is listed in your DBMS chapter. The

keyword BY must appear before the *dbms-name* argument. You must specify either a DBMS name or an alias in the EXECUTE statement.

alias
specifies an alias that was defined in the CONNECT statement. (You cannot use an alias if the CONNECT statement was omitted.)

Useful Statements to Include in EXECUTE Statements This section lists some of the statements that you can pass to the DBMS by using the SQL Procedure Pass-Through Facility's EXECUTE statement.

CREATE
creates a DBMS table, view, index, or other DBMS objects, depending on how the statement is specified.

DELETE
deletes rows from a DBMS table.

DROP
deletes a DBMS table, view, or other DBMS objects, depending on how the statement is specified.

GRANT
gives users the authority to access or modify objects such as tables or views.

INSERT
adds rows to a DBMS table.

REVOKE
revokes the access or modification privileges that were given to users by the GRANT statement.

UPDATE
modifies the data in columns of a row in a DBMS table.

For more information and restrictions on these and other SQL statements, see your DBMS-specific SQL documentation.

The following example grants UPDATE and INSERT authority to user TESTUSER on the Oracle Rdb table ORDERS. Because the CONNECT statement is omitted, an implicit connection, that uses the default database, is made to Oracle Rdb.

```
proc sql;
   execute (grant update, insert on orders
     to [qa,testuser]) by rdb;
%put &sqlxmsg;
```

CONNECTION TO Component

Retrieves and uses DBMS data in a PROC SQL query or view.

Contains DBMS-specific arguments

Optional component

Syntax

CONNECTION TO *dbms-name* | *alias (DBMS-query)*

The CONNECTION TO component specifies the DBMS connection that you want to use or that you want to create (if you have omitted the CONNECT statement). CONNECTION TO then enables you to retrieve DBMS data directly through a PROC SQL query.

You use the CONNECTION TO component in the FROM clause of a PROC SQL SELECT statement:

```
PROC SQL;
   SELECT column-list
      FROM CONNECTION TO dbms-name (DBMS-query)
           other optional PROC SQL clauses
QUIT;
```

CONNECTION TO can be used in any FROM clause, including those in nested queries (that is, subqueries).

You can store a SQL Procedure Pass-Through Facility query in a PROC SQL view and then use that view in SAS programs. When you create a PROC SQL view, any options that you specify in the corresponding CONNECT statement are stored too. Thus, when the PROC SQL view is used in a SAS program, the SAS System can establish the appropriate connection to the DBMS.

On many DBMSs, you can issue a CONNECTION TO component in a PROC SQL SELECT statement directly without first connecting to a DBMS (see "CONNECT Statement" on page 74). If you omit the CONNECT statement, an implicit connection is performed when the first PROC SQL SELECT statement that contains a CONNECTION TO component is passed to the DBMS. Default values are used for all DBMS connection arguments. See your DBMS chapter for details.

Because DBMSs and the SAS System have different naming conventions, some DBMS column names might be changed when you retrieve DBMS data through the CONNECTION TO component. See "Long Names and Case Sensitivity in the SQL Procedure and Pass-Through Facility" on page 68 for more information.

Arguments

dbms-name
identifies the database management system to which you direct the DBMS-specific SQL statement. The name for your DBMS is listed in your DBMS chapter.

alias
specifies an alias, if one was defined in the CONNECT statement.

(DBMS-query)
specifies the query that you are sending to the DBMS. The query can use any DBMS-specific SQL statement or syntax that is valid for the DBMS. However, the query cannot contain a semicolon because a semicolon represents the end of a statement in the SAS System.

You must specify a *dbms-query* argument in the CONNECTION TO component, and the query must be enclosed in parentheses. The query is passed to the DBMS

exactly as you type it; therefore, if your DBMS is case sensitive, you must use the correct case for DBMS object names.

On some DBMSs, the *dbms-query* argument can be a DBMS stored procedure. However, if the stored procedure contains more than one query, only the first query is processed.

Example

The following example sends an ORACLE SQL query, shown in italics, to the ORACLE database for processing. The results from the ORACLE SQL query serve as a virtual table for the PROC SQL FROM clause. In this example, MYCON is a connection alias.

```
proc sql;
connect to oracle as mycon (user=testuser
   password=testpass path='myorapath');
%put &sqlxmsg;

select *
   from connection to mycon
    (select empid, lastname, firstname,
               hiredate, salary
          from employees where
            hiredate>='31-DEC-88');
%put &sqlxmsg;

disconnect from mycon;
quit;
```

The SAS %PUT macro displays the &SQLXMSG macro variable for error codes and information from the DBMS. See "SQL Procedure Pass-Through Facility Return Codes" on page 83 for more information.

The following example gives the query a name and stores it as the PROC SQL view SLIB.HIRES88:

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

proc sql;
connect to oracle as mycon (user=testuser
password=testpass path='myorapath');
%put &sqlxmsg;

create view slib.hires88 as
   select *
      from connection to mycon
         (select empid, lastname, firstname,
           hiredate, salary
            from employees where
               hiredate>='31-DEC-88');
%put &sqlxmsg;

disconnect from mycon;

quit;
```

SQL Procedure Pass-Through Facility Return Codes

As you use the PROC SQL statements that are available in the SQL Procedure Pass-Through Facility, any error conditions are written to the SAS log. The SQL Procedure Pass-Through Facility generates return codes and messages that are available to you through the following two SAS macro variables:

SQLXRC
: contains the DBMS return code that identifies the DBMS error.

SQLXMSG
: contains descriptive information about the DBMS error that is generated by the DBMS.

The contents of the SQLXRC and SQLXMSG macro variables are printed in the SAS log using the %PUT macro. The automatic macro variables SQLXRC and SQLXMSG are reset after each SQL Procedure Pass-Through Facility statement has been executed.

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