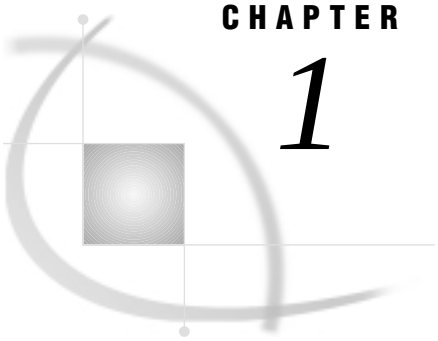

CHAPTER

1

Informix Chapter, First Edition

Introduction

This chapter introduces SAS System users to Informix, which is a relational database management system. It accompanies and should be used with *SAS/ACCESS Software for Relational Databases: Reference* (order #57204).*

This chapter focuses on the terms and concepts that help you use the SAS/ACCESS Interface to Informix. The SAS/ACCESS Interface to Informix connects you to Informix data through the LIBNAME statement and the SQL Procedure Pass-Through Facility.

For general information about database management systems, including information for the database administrator about how the SAS/ACCESS interfaces work, refer to Appendix 2, "DBMS Overview and Information for the Database Administrator". For more information about Informix, refer to your Informix documentation.

Default Environment

When you access Informix tables by using the SAS/ACCESS Interface to Informix, the default Informix read isolation level is set for committed reads, and SAS spooling is on. Committed reads allow you to read rows unless another user or process is updating

the rows; reading in this manner does not lock the rows. SAS spooling guarantees that you will get identical data each time you reread a row because SAS buffers the rows after you read them the first time. This default environment is suitable for most users; however, this chapter describes how to set the locking and read isolation level options if the default environment is unsuitable for your needs.

For more details on Informix locking, see your Informix documentation. To see the SQL statements, including locking statements, that SAS issues to the Informix server, include the following option in your code:

```
option sastrace=',,,d';
```

See Chapter 5, "Macro Variables and System Options" for more information about the SASTRACE= option.

Note: If you use quotes in your Informix SQL statements, your DELIMIDENT environment variable should be set to DELIMIDENT=YES, or your statements could be rejected by Informix. Because some of the SAS options that preserve case generate SQL statements that contain quotes, DELIMIDENT=YES should be set in your environment. △

SAS/ACCESS LIBNAME Statement

Chapter 3, "SAS/ACCESS LIBNAME Statement" describes options that you specify in the LIBNAME statement to associate a SAS libref with a DBMS database, schema, server, or group of tables and views. The following section describes DBMS-specific options and option values that you can use in the SAS/ACCESS Interface to Informix.

LIBNAME Statement: Informix Specifics

Associates a SAS libref with a DBMS database, schema, server, or group of tables and views.

Valid: Anywhere

Syntax

LIBNAME *libref SAS/ACCESS-engine-name*
<*SAS/ACCESS-engine-connection-options*>
<*SAS/ACCESS-LIBNAME-options*>;

Arguments

libref
is any SAS name that serves as an alias to associate the SAS System with a database, schema, server, or group of tables and views.

SAS/ACCESS-engine-name
is a SAS/ACCESS engine name for your DBMS, in this case, **informix**. SAS/ACCESS engines are implemented differently in different operating environments. The engine name is required.

SAS/ACCESS-engine-connection-options
are options that you specify to connect to a particular database; these options are different for each database. If the connection options contain characters that are not allowed in SAS names, enclose the values of the options in quotation marks. If you specify the appropriate system options or environment variables for your database, you can often omit the connection options. See your Informix documentation for details.

SAS/ACCESS-LIBNAME-options
are options that apply to the processing of objects and data in a DBMS, such as its tables or indexes. For example, the LOCKTABLE= option enables you to lock or unlock tables in a libref. Support for many of these options is DBMS specific.

Some SAS/ACCESS LIBNAME options have the same names as SAS/ACCESS data set options. When you specify an option in the LIBNAME statement, it applies to objects and data that are referenced by the libref. A SAS/ACCESS data set option applies only to the data set on which it is specified. If a like-named option is specified in both the SAS/ACCESS LIBNAME statement and after a data set name (which references a DBMS table or view), the SAS System uses the value that is specified later, on the data set name. See “Data Set Options: Informix Specifics” on page 6 for more information.

Details The LIBNAME statement associates a libref with a SAS/ACCESS engine to access tables or views in a database management system (DBMS). The SAS/ACCESS engine enables you to connect to a particular DBMS and to specify a DBMS table or view name in a two-level SAS name. For example, in MYDBLIB.EMPLOYEES_Q2, MYDBLIB is a SAS libref that points to a particular group of DBMS objects, and EMPLOYEES_Q2 is a DBMS table name. When you specify MYDBLIB.EMPLOYEES_Q2 in a DATA step or procedure, you dynamically access the DBMS table. In Version 7 and higher, SAS software supports reading, updating, creating, and deleting DBMS tables.

To disassociate or clear a libref from a DBMS, use a LIBNAME statement, specifying the libref (for example, MYDBLIB) and the CLEAR option as follows:

```
libname mydblib CLEAR;
```

The database engine will disconnect from the database and close any threads or resources that are associated with that connection.

See for more information on arguments that you can use in the LIBNAME statement.

SAS/ACCESS Engine Connection Options The SAS/ACCESS engine connection options for Informix are as follows:

USER=<'>*Informix-user-name*<'>
specifies the Informix user name that you use to connect to the database that contains the tables and views that you want to access. If you omit the USER= option, your operating system account name is used, if applicable to your operating environment.

USING=<'>*Informix-password*<'>
specifies the password that is associated with the Informix user. If you omit the password, Informix uses the password in the /etc/password file.

USING can also be specified with the PASSWORD= and PWD=aliases.

DATABASE=<'>*database-name*<'>
specifies the name of the Informix database that contains the tables and views that you want to access. If you omit the DATABASE= option, the value of the SAS environment variable DBDATABASE is used as the database name. An error occurs if neither the DATABASE= option nor the DBDATABASE environment variable is set. See "Using the DBDATABASE Environment Variable" on page 19 for more information.

DATABASE= can also be specified with the DB= alias.

SERVER=<'>*server-name*<'>
specifies the server with which to connect. This server accesses the database that contains the tables and views that you want to access. If you omit the SERVER= option, the value of the environment variable INFORMIXSERVER is used as the server name. An error occurs if neither the SERVER= option nor the INFORMIXSERVER environment variable is set.

You can use the DBDATABASE environment variable to specify database/server combinations, such as *database@server* or *//server/database*.

SAS/ACCESS LIBNAME Options When you specify any of the following options in the LIBNAME statement, the option is applied to all objects (such as tables, views, and indexes) in the database that the libref represents.

The SAS/ACCESS interface to Informix supports all of the SAS/ACCESS LIBNAME options listed in Chapter 3, "SAS/ACCESS LIBNAME Statement" , except for DBMAX_TEXT=. In addition to the supported options, the following LIBNAME options are used only in the interface to Informix or have Informix–specific aspects to them:

LOCKTABLE= on page 4
LOCKTIME= on page 4
LOCKWAIT= on page 4
PRESERVE_COL_NAMES= on page 5
PRESERVE_TAB_NAMES= on page 5
READ_ISOLATION_LEVEL= on page 5
SCHEMA= on page 6
SPOOL= on page 6

LOCKTABLE=EXCLUSIVE | SHARE
places exclusive or shared locks on tables. You may lock tables only if you are the owner or have been granted the necessary privilege.

If you omit LOCKTABLE=, no locking occurs. If you specify LOCKTABLE=EXCLUSIVE, other users are prevented from accessing each table that you open in the libref.

If you specify LOCKTABLE=SHARE, other users or processes can read data from the tables, but they cannot update the data.

LOCKTABLE= can also be specified with the TABLELOCK= alias.

LOCKTIME=*integer*
specifies the number of seconds to wait until rows are available for locking.

You must specify LOCKWAIT=YES for LOCKTIME= to have an effect. If you omit the LOCKTIME= option and use LOCKWAIT=YES, SAS suspends your process indefinitely until a lock can be obtained.

See also: LOCKWAIT=.

LOCKWAIT=YES | NO
specifies whether to wait until rows are available for locking.

By default, the SAS/ACCESS Interface to Informix returns an error if another user holds a lock on the rows that you want to lock. If you specify

LOCKWAIT=YES, SAS waits until rows are available for locking, or until the number of seconds specified by using the LOCKTIME= option has passed. In the latter case, an error is returned.

If you specify LOCKWAIT=NO or omit this option, SAS does not wait and returns an error to indicate that the lock is not available.

Note: If you specify LOCKWAIT= and do not limit the wait time by using the LOCKTIME= option, your process might suspend indefinitely if the lock cannot be obtained. △

See also: LOCKTIME=.

PRESERVE_COL_NAMES=YES | NO

preserves spaces, special characters, and mixed case in column names.

If you omit PRESERVE_COL_NAMES=, the default value is PRESERVE_COL_NAMES=NO, which means that column names are created and referenced in lowercase. If you want to preserve the case or allow characters that are not supported in SAS names, such as '$', in your column names, set PRESERVE_COL_NAMES=YES.

For a full description of this option, refer to Chapter 3, "SAS/ACCESS LIBNAME Statement".

PRESERVE_TAB_NAMES=YES | NO

preserves spaces, special characters, and mixed case in table names.

If you omit PRESERVE_TAB_NAMES=, the default value is PRESERVE_TAB_NAMES=NO, which means that table names are created and referenced in lowercase. If you want to preserve case or allow characters that are not supported in SAS names, such as '$', in your object names, including table names, schema names, and link names, set PRESERVE_TAB_NAMES=YES.

For a full description of this option, refer to Chapter 3, "SAS/ACCESS LIBNAME Statement".

READ_ISOLATION_LEVEL=COMMITTED_READ | REPEATABLE_READ | DIRTY_READ | CURSOR_STABILITY

specifies the method of read locking for Informix to use when it reads tables and views.

If you omit the READ_ISOLATION_LEVEL= option, the default value is READ_ISOLATION_LEVEL= COMMITTED_READ, which retrieves only committed rows. No locks are acquired, and rows can be locked exclusively for update by other users or processes.

If you specify READ_ISOLATION_LEVEL=REPEATABLE_READ, you acquire a shared lock on every row that is selected during the transaction. Other users or processes can also acquire a shared lock, but no other process can modify any row that is selected by your transaction. If you repeat the query during the transaction, you reread the same information. The shared locks are released only when the transaction commits or rolls back. Another process cannot update or delete a row that is accessed by using a repeatable read.

If you specify READ_ISOLATION_LEVEL=DIRTY_READ, you retrieve committed and uncommitted rows that might include *phantom rows*, which are rows that are created or modified by another user or process that might subsequently be rolled back. This type of read is most appropriate for tables that are not frequently updated.

If you specify READ_ISOLATION_LEVEL=CURSOR_STABILITY, you acquire a shared lock on the selected row. Another user or process can acquire a shared lock on the same row, but no process can acquire an exclusive lock to modify data in the row. When you retrieve another row or close the cursor, the shared lock is released.

Note: For current Informix releases, READ_ISOLATION_LEVEL= is only valid when transaction logging is enabled. If transaction logging is not enabled, an

error is generated when you use this option. Also, locks placed when READ_ISOLATION_LEVEL=REPEATABLE READ or CURSOR_STABILITY are *not freed* until the libref is cleared.

In most situations, spooling, which is on by default, provides the data consistency you need. However, if you want to use READ_ISOLATION_LEVEL=REPEATABLE_READ or CURSOR_STABILITY, it is recommended that you assign a separate libref with this option, and that you clear the libref when you have finished working with the tables. This technique minimizes the negative performance impact on other users that occurs when you lock the tables. To clear the libref, include the following code:

```
LIBNAME libref CLEAR;
```

△

SCHEMA=*username*
allows you to view another user's database tables and views.

If you omit the SCHEMA= option, you can view only your own tables and views. SCHEMA= can also be specified with the OWNER= alias.

SPOOL=YES | NO
specifies whether SAS creates a utility spool file during read operations that are performed with the specified libref.

If you omit SPOOL=, the default value is SPOOL=YES, which means that SAS performs spooling. If you specify SPOOL=NO, SAS does not perform spooling.

For a full description of this option, refer to Chapter 3, "SAS/ACCESS LIBNAME Statement".

Example: Specifying a LIBNAME Statement to Access Informix Data

In this example, the libref MYDBLIB uses the Informix engine to connect to an Informix database. The SAS/ACCESS engine connection options are USER=, PASSWORD=, DATABASE=, and SERVER=.

```
libname mydblib informix user=testuser
   using=testpass database=testdb
   server=testserver;

proc print data=mydblib.customers;
   where gender='M';
run;
```

Data Set Options: Informix Specifics

Chapter 4, "SAS/ACCESS Data Set Options" describes the SAS/ACCESS options that you can use when you specify a SAS data set in a DATA or PROC step; in this case, the SAS data set accesses data from a DBMS table or view. The following section describes the DBMS-specific options and option values that you can use in the SAS/ACCESS Interface to Informix. A data set option applies only to the SAS data set, or DBMS object, on which it is specified.

Unless otherwise noted, when you omit a SAS/ACCESS data set option, and you have specified a like-named LIBNAME option in the LIBNAME statement, the value of the LIBNAME option applies to all data sets within the libref.

Note: Not all LIBNAME options have corresponding data set options. Refer to Chapter 3, "SAS/ACCESS LIBNAME Statement" , Chapter 4, "SAS/ACCESS Data Set

Options" , and this chapter for a full listing of SAS/ACCESS LIBNAME options, data set options, and Informix-specific LIBNAME and data set options. △

The SAS/ACCESS interface to Informix supports all of the SAS/ACCESS data set options listed in Chapter 4, "SAS/ACCESS Data Set Options" , except for DBMAX_TEXT=, READ_LOCK_TYPE, and UPDATE_LOCK_TYPE. In addition to the supported options, the following data set options are used only in the interface to Informix or have Informix–specific aspects to them:

"DBNULL=" on page 7
"DBTYPE=" on page 7
"LOCKTABLE=" on page 7
"SASDATEFMT=" on page 8
"SCHEMA=" on page 8

DBNULL=

Indicates whether NULL is a valid value for the specified column(s).

Default value: YES

Details For a full description of this option, refer to Chapter 4, "SAS/ACCESS Data Set Options".

DBTYPE=

Specifies data type(s) to override the default DBMS data type(s) during output processing.

Default value: CHAR(size) for characters, where size is derived from the SAS variable length; FLOAT for numbers.

Details For a full description of this option, refer to Chapter 4, "SAS/ACCESS Data Set Options".

LOCKTABLE=

Places exclusive or shared locks on tables.

Default value: NONE

Syntax

LOCKTABLE=EXCLUSIVE | SHARE

EXCLUSIVE
locks a table exclusively.

SHARE
locks a table in shared mode.

Details You may lock tables only if you are the owner or have been granted the necessary privilege. If you omit LOCKTABLE=, no locking occurs. LOCKTABLE=EXCLUSIVE prevents other users from accessing each table that you open in the libref. LOCKTABLE=SHARE allows other users or processes to read data from the tables, but does not allow them to update data.

LOCKTABLE= can also be specified with the TABLELOCK= alias.

SASDATEFMT=

Specifies the SAS date format to use to convert SAS date values.

Default value: DATETIME

Details For a full description of this option, refer to Chapter 4, "SAS/ACCESS Data Set Options".

SCHEMA=

Allows you to view another user's database tables and views.

Default value: None

Syntax

SCHEMA=*owner-name*

owner-name
specifies the user name of the owner of the tables and the views that you want to access.

Details SCHEMA= can also be specified with the OWNER= alias.

SQL Procedure Pass-Through Facility: Informix Specifics

The SQL Procedure Pass-Through facility consists of three PROC SQL statements and one component. See the following Informix-specific sections for details:

CONNECT“CONNECT Statement” on page 9, EXECUTE“EXECUTE Statement” on page 10, and CONNECTION TO“CONNECTION TO Component” on page 13. For a complete description of the Pass-Through Facility, see Chapter 6, "SQL Procedure's Interaction with SAS/ACCESS Software".

CONNECT Statement

Establishes a connection with the DBMS

Optional statement

Syntax

CONNECT TO INFORMIX <AS *alias*> <(*Informix-connection-arguments*)>;

Arguments

alias
specifies an optional alias that has 1 to 32 characters. If you specify an alias, the keyword AS must appear before the alias.

(*Informix-connection-arguments*)
specifies the arguments that are needed by PROC SQL in order to connect to the Informix. These arguments must be enclosed in parentheses.

Informix Connection Arguments

The CONNECT statement establishes a connection with the DBMS. You establish a connection to send 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.

You can connect to only one Informix database. However, you can specify multiple CONNECT statements if they all connect to the same Informix database. If you use multiple connections, you must use an *alias* to identify the different connections. If you omit an alias, **informix** is automatically used.

When you omit a CONNECT statement, an implicit connection is performed when the first EXECUTE statement or CONNECTION TO component is passed to the DBMS.

The CONNECT statement is optional when connecting to an Informix database if the DBDATABASE environment variable has been set to include both the database and server specification. See “Using the DBDATABASE Environment Variable” on page 19 for more information.

Any return code or message that is generated by the DBMS or by the SQL Procedure Pass-through facility is available in the macro variables SQLXRC and SQLXMSG after the statement executes. See Chapter 6, "SQL Procedure's Interaction with SAS/ACCESS Software" for more information on these macro variables.

Informix uses the following database connection arguments. These arguments must be enclosed in parentheses.

DATABASE= | DB=<">*database-name*<">
specifies the name of the Informix database and optionally, database server, to which PROC SQL connects. The database name allows optional single or double

quotes. If the name contains special characters, national characters, or semicolons, you must enclose the name in quotes.

By using a full pathname, you can specify a database as well as a database server in the DATABASE argument.

You can set default values for the DATABASE= argument and therefore, the argument is optional. See "Using the DBDATABASE Environment Variable" on page 19 for more information.

SERVER=<'>*server-name*<'>
specifies the server with which to connect. This server accesses the database that contains the tables and views that you want to access. If you omit the SERVER= option, the value of the SAS environment variable INFORMIXSERVER is used as the server name. An error occurs if neither the SERVER= option nor the SAS INFORMIXSERVER environment variable is set.

You can use the DBDATABASE environment variable to specify database/server combinations, such as *database@server* or *//server/database*.

USER=<'>*Informix-user-name*<'>
specifies the Informix user name that you use to connect to the database that contains the tables and views that you want to access. If you omit the USER= option, your operating system account name is used, if applicable to your operating environment.

USING=<'>*Informix-password*<'>
specifies the password that is associated with the Informix user. If you omit the password, Informix uses the password in the /etc/password file.

USING can also be specified with the PASSWORD= and PWD=aliases.

Example This example connects to the Informix database **stores7** by using the **online** server. The database name is quoted because it includes special characters.

```
proc sql;
   connect to informix
   (user=SCOTT password=TIGER database='//online/stores7');
```

Note: You can use the DBDATABASE environment variable to specify database/server combinations, such as *database@server* or *//server/database*. △

EXECUTE Statement

Sends DBMS-specific, nonquery SQL statements to the DBMS.

Optional statement

Syntax

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

Arguments

(*DBMS-specific-SQL-statement*)
specifies a dynamic, DBMS-specific SQL statement that does not select data. This argument is required and must be enclosed in parentheses. The statement is case sensitive and is passed to the DBMS exactly as you type it.

Note: If you use quotes in your Informix SQL Pass-through statements, your DELIMIDENT environment variable must be set to DELIMIDENT=YES, or your statements will be rejected by Informix. △

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 Chapter 6, "SQL Procedure's Interaction with SAS/ACCESS Software" for more information on these macro variables.

dbms-name
identifies the database management system to which you direct the DBMS-specific SQL statement, in this case, **`informix`**. 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.)

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 connecting to a DBMS. If you omit the CONNECT statement, an implicit connection is performed (by using default values for all connection arguments) when the first EXECUTE statement is passed to the DBMS.

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

Useful Statements to Include in EXECUTE Statements

This section lists some of the statements that you can pass to the DBMS by using the Pass-Through facility's EXECUTE statement.

Note: The statements passed using the EXECUTE statement cannot contain a semicolon (;) because to SAS software a semicolon represents the end of a 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 modifies 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 Informix SQL documentation.

Special Informix Considerations

The Pass-Through Facility recognizes two types of stored procedures in Informix that perform only database functions. The method for executing the two types of stored procedures is different.

- Procedures that return no values to the calling application:

  Stored procedures that do not return values can be executed directly by using the Informix SQL EXECUTE statement. Stored procedure execution is initiated with the Informix EXECUTE PROCEDURE statement. The following example executes the stored procedure **make_table.** The stored procedure has no input parameters and returns no values.

```
execute (execute procedure make_table())
   by informix;
```

- Procedures that return values to the calling application:

  Stored procedures that return values must be executed by using the PROC SQL SELECT statement with a CONNECTION TO component. The following example executes the stored procedure **read_address**, which has one parameter,

```
"Putnum".
```

  The values that are returned by **read_address** serve as the contents of a virtual table for the PROC SQL SELECT statement.

```
select * from connection to informix
        (execute procedure read_address ("Putnum"));
```

  For example, when you try to execute a stored procedure that returns values from a PROC SQL EXECUTE statement, you get the following error message:

```
execute (execute procedure read_address
   ("Putnum")) by informix;

ERROR: Informix EXECUTE Error: Procedure
   (read_address) returns too many values.
```

Examples

The following example grants UPDATE and INSERT authority to user **gomez** on the Informix ORDERS table. Because the CONNECT statement is omitted, an implicit connection is made that uses a default value of **informix** as the connection alias and default values for the DATABASE and SERVER arguments. Informix is a case-sensitive database; therefore, the database object **ORDERS** is in uppercase, as it was created.

```
proc sql;
   execute (grant update, insert on ORDERS
   to gomez) by informix;
quit;
```

The next example connects to Informix and drops (that is, removes) the table **tempdata** from the **stores7** database. The alias **temp5** that is specified in the CONNECT statement is used in the EXECUTE statement's BY clause.

```
proc sql;
   connect to informix as temp5
   (database='//online/stores7');
   execute (drop table tempdata) by temp5;
   disconnect from temp5;
quit;
```

CONNECTION TO Component

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

Optional component

Syntax

CONNECTION TO INFORMIX | *alias (DBMS-SQL-query)*

Arguments

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

(DBMS-SQL-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 to the SAS System a semicolon represents the end of a statement.

You must specify a *dbms-SQL-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 DMBS object names, enclosing them in quotes, if necessary. Quoted character strings are limited to 200 characters.

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

The CONNECTION TO component specifies the DBMS connection that you want to use or that you want to establish (if you have omitted the CONNECT statement). CONNECTION TO 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-SQL-query) other optional PROC SQL clauses*;

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

You can store a Pass-Through 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. 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 connection arguments.

Because DBMSs and the SAS System have different naming conventions, some DBMS column names may be truncated when you retrieve DBMS data through the CONNECTION TO component. Default SAS variable names follow these rules:

- If the column name is longer than 32 characters, the SAS System uses only the first 32 characters. If truncating results in duplicate names, sequential numbers (starting with zero) are appended to the ends of the names.
- If the column name contains characters that are invalid in SAS names (such as national characters), the SAS System replaces the invalid characters with underscores (_). For example, the column name **func$** becomes the SAS variable name **func_**.

Examples

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

```
proc sql;
connect to informix as dbcon
   (user=testuser using=testpass db=testdb
    server=testserver);

select *
   from connection to dbcon
      (select empid, lastname, firstname,
      hiredate, salary
          from employees
          where hiredate>='31JAN88');

disconnect from dbcon;
quit;
```

The following example gives the previous query a name and stores it as the PROC SQL view SLIB.HIRES88. The CREATE VIEW statement appears in italics.

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

proc sql;
connect to informix as mycon
   (user=testuser using=testpass db=testdb
     server=testserver);

create view slib.hires88 as
  select *
    from connection to mycon
```

```
      (select empid, lastname, firstname,
       hiredate, salary from employees
       where hiredate>='31JAN88');

disconnect from mycon;
quit;
```

The next example connects to Informix and executes the stored procedure **testproc**. The **select *** clause displays the results from the stored procedure.

```
proc sql;
   connect to informix as mydb
      (database='//dbserver/corpdb');
   select * from connection to mydb
      (execute procedure testproc('123456'));
   disconnect from mydb;
quit;
```

Informix Command Restrictions for the Pass-Through Facility

Informix SQL contains extensions to the ANSI-89 standards. Some of these extensions, such as LOAD FROM and UNLOAD TO, are restricted from use by any applications other than the Informix DB-Access product. Specifying these extensions in the PROC SQL EXECUTE statement generates this error: **-201 A syntax error has occurred**.

Informix Naming Conventions

- Table and column names must begin with a letter or an underscore followed by letters, numbers, or underscores. However, if the name appears within quotes and the PRESERVE_TAB_NAMES option has been set to PRESERVE_TAB_NAMES=YES when applicable, it may start with any character.
- Table and column names can contain up to 18 characters.

Informix Data Types and SAS Representations and Formats

Every column in a table has a name and a data type. The *data type* tells Informix how much physical storage to set aside for the column and the form in which the data is stored. Informix data types fall into three categories: types for character data, types for numeric data, and types for abstract values such as dates. Each of these types is described in the following sections.

This section describes how the LIBNAME statement and the SQL Procedure Pass-Through Facility treat each of these types during input operations.

Character Data

CHAR*(n)*, NCHAR*(n)*
contains character string data from 1 to 32,767 characters in length and can include tabs and spaces.

VARCHAR(*m*, *n*), NVARCHAR*(m, n)*
contains character string data from 1 to 255 characters in length.

TEXT
contains unlimited text data, depending on memory capacity.

Numeric Data

DECIMAL, MONEY, NUMERIC
contains numeric data with definable scale and precision. The amount of storage that is allocated depends on the size of the number.

FLOAT, DOUBLE PRECISION
contains double-precision numeric data up to 8 bytes.

INTEGER
contains an integer up to 32 bits (from -2^{31} to 2^{31}—1).

REAL, SMALLFLOAT
contains single-precision, floating-point numbers up to 4 bytes.

SERIAL
stores sequential integers up to 32 bits.

SMALLINT
contains integers up to 2 bytes.

Abstract Data

DATE
contains a calendar date in the form of a signed integer value.

DATETIME
contains a calendar date and time of day stored in 2 to 11 bytes, depending on precision.

INTERVAL
contains a span of time stored in 2 to 12 bytes, depending on precision.

NULL and Default Values

If you do not indicate a default value for a column, the default value is NULL. You can specify the keywords NOT NULL after the data type of the column when you create a table to prevent NULL values from being stored in the column.

LIBNAME Statement Data Conversions

The following table shows the default SAS variable formats that the LIBNAME statement applies to Informix data types during input operations. You can override these default data types by using the DBTYPE= data set option on a specific data set.

Table 1.1 LIBNAME Statement: Default SAS Formats for Informix Data Types

Informix Column Type	Default SAS Format
CHAR(*n*)	$*n*
DATE	DATE9.
DATETIME**	DATETIME24.5
DECIMAL	none
DOUBLE PRECISION	none
FLOAT	none
INTEGER	none
INTERVAL	$*n*
MONEY	none
NCHAR(*n*)	$*n*
	NLS support required
NUMERIC	none
NVARCHAR(*m*,*n*)*	$*m*
	NLS support required
REAL	none
SERIAL	none
SMALLFLOAT	none
SMALLINT	none
TEXT*	$*n*
VARCHAR(*m*,*n*)*	$*m*

* Only supported by Informix-Online databases

** If the Informix field qualifier specifies either HOUR, MINUTE, SECOND, or FRACTION as the largest unit, the value is converted to a SAS TIME value. All others, such as YEAR, MONTH, or DAY, are converted to a SAS DATETIME value.

The following table shows the default Informix data types that the LIBNAME statement applies to SAS variable formats during output operations.

Table 1.2 LIBNAME Statement: Default Informix Data Types for SAS Variable Formats

SAS Variable Format	Informix Data Type
$*w*.	CHAR(w).
w. with SAS format name of NULL	FLOAT
w.d with SAS format name of NULL	FLOAT
all other numerics	FLOAT
datetime*w.d*	DATETIME YEAR TO FRACTION(5)

SAS Variable Format	Informix Data Type
date*w*.	DATE
time.	DATETIME HOUR TO FRACTION(5)

SQL Procedure Pass-Through Facility Data Conversions

The SQL Procedure Pass-Through Facility uses the same default conversion formats as the LIBNAME statement. See "LIBNAME Statement Data Conversions" on page 16 for the conversion tables.

Overview of Informix Servers

There are two Informix database servers, the Informix-Online and Informix-SE server. *Informix-Online database servers* can support many users and provide tools that ensure high availability, high reliability, and that support critical applications. *Informix-SE database servers* are designed to manage relatively small databases that are used privately by individuals or shared among a small number of users.

Specifying Databases and Servers

To connect to an Informix database, the Pass-Through Facility executes an Informix SQL DATABASE statement. The value that you specify in the DATABASE= argument of the CONNECT statement is passed as a parameter to the Informix SQL DATABASE statement. Your Informix environment must be properly configured in order for this DATABASE= statement to execute correctly. The environment variables that are required for local or remote processing must be set correctly. For a full explanation of database name and path specifications, see your Informix documentation.

To connect to an Informix-SE database, you must either be in the directory that contains the database or you must specify the full pathname of the database in the DATABASE= argument. The following example connects to the database **mydb** in the current directory.

```
proc sql;
   connect to informix(db=mydb);
```

The following example connects to the database **groupdb** in the directory **/usr/projects** on the remote Informix-SE server **rmtse**. Notice that the DB= argument is quoted because the SAS System does not accept slashes (/) in names.

```
proc sql;
   connect to informix
   (db='//rmtse/usr/projects/groupdb');
```

For Informix-Online, only the database server name and the database name are required. This example connects to the database **corpdb** that resides on the **online** server.

```
proc sql;
   connect to informix
   (db='//online/corpdb');
```

Using the DBDATABASE Environment Variable

The Pass-Through Facility supports the environment variable DBDATABASE, which is an extension to the Informix environment variable. If you set DBDATABASE, you can omit the CONNECT statement. The value of DBDATABASE is used instead of the DATABASE= argument in the CONNECT statement. The syntax for setting DBDATABASE is like the syntax of the DATABASE= argument.

```
Bourne shell: DBDATABASE='//online/corpdb'
               export DBDATABASE

C shell:      setenv DBDATABASE //online/corpdb
```

If you set DBDATABASE, you can issue a PROC SQL SELECT or EXECUTE statement without first connecting to Informix with the CONNECT statement. If you omit the CONNECT statement, an implicit connection is performed when the SELECT or EXECUTE statement is passed to Informix. If you create a PROC SQL Pass-Through view without an explicit CONNECT statement, the view can dynamically connect to different databases, depending on the value of the DBDATABASE environment variable.

If you specify both the DBDATABASE environment variable and the DATABASE= argument in the CONNECT statement, the DATABASE= argument takes precedence.

You can also use the DBDATABASE environment variable to specify database/server combinations, such as *database@server* instead of *//server/database*.

Using Fully Qualified Table Names

Informix supports a connection to only one database. If you have data that span multiple databases, you must use fully qualified table names to work within the Informix single-connection constraints.

In the following example, the tables **tab1** and **tab2** reside in different databases, **mydb1** and **mydb2**, respectively.

```
proc sql;
   connect to informix
   (database=corpdb server=online);

   create view tab1v as
      select * from connection
      to informix
         (select * from mydb1.tab1);

   create view tab2v as
      select * from connection
      to informix
         (select * from mydb2.tab2);
quit;

data getboth;
   merge tab1v tab2v;
   by common;
run;
```

Because the tables reside in separate databases, you cannot connect to each database with a PROC SQL CONNECT statement and then retrieve the data in a single step. Using the fully qualified table name (that is, *database.table*) enables you to use any

Informix database in the CONNECT statement and access Informix tables in the *same or different* databases in a single SAS procedure or DATA step.

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS/ACCESS® Software for Relational Databases: Reference, Version 8 (Informix Chapter)*, Cary, NC: SAS Institute Inc., 1999.

SAS/ACCESS® Software for Relational Databases: Reference, Version 8 (Informix Chapter)

ISBN 1-58025-537-X

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

1st printing, October 1999