CHAPTER 1

# SYBASE Chapter, First Edition

## Introduction

This chapter introduces SAS System users to SYBASE, 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 SYBASE. Then it describes the statements and options that are specific to SYBASE, the ACCESS and DBLOAD procedures, and the SQL procedure's CONNECT statement.

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 SYBASE, refer to your SYBASE documentation.

## SAS/ACCESS LIBNAME Statement

Chapter 3, "SAS/ACCESS LIBNAME Statement" describes options that you specify in the SAS/ACCESS 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 SYBASE.

## LIBNAME Statement: SYBASE 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, **sybase**. 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 SYBASE 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 BULK_BUFFER= option enables you to specify the number of bulk rows of DBMS data that the DBMS can write to the buffer. 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 engine 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: SYBASE Specifics" on page 7 for more information on data set options.

**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, therefore, 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. Beginning in Version 7, 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 free threads or resources that are associated with that connection.

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

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

USER= on page 3

PASSWORD= on page 3

DATABASE= on page 3

SERVER= on page 4

USER=<'>*SYBASE-user-name*<'>

specifies the SYBASE user name (also called the login name) that you use to connect to the database that contains the tables and views that you want to access. If the username contains spaces or nonalphanumeric characters, you must enclose the user name in quotation marks.

PASSWORD=<'>*SYBASE-password*<'>

specifies the password that is associated with the SYBASE user name.

If you omit the password, a default password of NULL is used. If the password contains spaces or nonalphanumeric characters, you must enclose the password in quotation marks.

PASSWORD= can also be specified with the SYBPW=, PASS=, and PW= aliases.

DATABASE=<'>*database-name*<'>

specifies the name of the SYBASE database that contains the tables and views that you want to access.

If the database name contains spaces or non-alphanumeric characters, you must enclose it in quotation marks. If you omit DATABASE=, the default database for your SYBASE user name is used.

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 the server name contains lowercase, spaces, or nonalphanumeric characters, you must enclose it in quotation marks.

If you omit SERVER=, the default action for your operating system occurs. On UNIX systems, the value of the environment variable DSQUERY is used if it has been set.

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

CONNECTION= on page 4

DBLINK= on page 4

ENABLE_BULK= on page 5

MAX_CONNECTS= on page 5

PACKETSIZE= on page 5

QUOTED_IDENTIFIER= on page 5

READ_BUFFER= on page 5

READ_ISOLATION_LEVEL= on page 5

READ_LOCK_TYPE= on page 6

SCHEMA= on page 6

UPDATE_ISOLATION_LEVEL= on page 6

UPDATE_LOCK_TYPE= on page 6

CONNECTION=SHAREDREAD | UNIQUE | GLOBALREAD

indicates whether multiple opens for read access can use the same physical connection.

*Note:* When you use the SQL procedure, you must specify CONNECTION= UNIQUE to create tables with the SELECT statement and to perform correlated subquery joins. You must also specify CONNECTION= UNIQUE when you use the APPEND procedure to append data from one table to another. △

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

DBLINK=*database-link*

allows you to link to another database within the same server to which you are connected.

If you omit DBLINK=, SAS can only access objects in your default database.

DBLINK= and SCHEMA= on page 6 can be used together, so that a SYBASE object can be referenced as *libref.schema-owner's-name.object*.

ENABLE_BULK=YES | NO

enables the connection to process bulk copy when you load data into a SYBASE table. Bulk copy groups rows so that they are inserted as a unit into the SYBASE table. Using bulk copy can improve performance.

Default value: YES

ENABLE_BULK=YES enables the connection to perform a bulk copy of SAS data into SYBASE. Specifying ENABLE_BULK=NO might save some memory but it also disables the bulk copy ability for that libref.

If you use both the LIBNAME option, ENABLE_BULK=, and the data set option, BULKCOPY=, the values of the two options must be the same or an error is returned. However, since the default value of ENABLE_BULK= is YES, you do not have to specify ENABLE_BULK= in order to use the BULKCOPY= data set option.

*Note:* In V7 and previous releases, this option was called BULKCOPY=. In Version 8, an error is returned if you try to specify BULKCOPY=. △

MAX_CONNECTS=*numeric-value*

allows you to specify the maximum number of simultaneous connections that SYBASE allows.

If you omit MAX_CONNECTS=, the maximum number of connections defaults to 25. Note that increasing the number of connections has a direct impact on memory.

PACKETSIZE=*numeric-value*

allows you to specify the packet size for SYBASE to use. You may specify any multiple of 512, up to the limit of the maximum network packet size setting on your server.

If you omit PACKETSIZE=, the packet size defaults to the current server setting. You can query the default network packet value in ISQL by using the SYBASE **sp_configure** command.

QUOTED_IDENTIFIER=YES | NO

allows you to specify table and column names with embedded spaces and special characters. This option is used in place of the PRESERVE_TAB_NAMES= and PRESERVE_COL_NAMES= options, which have no effect on the SYBASE engine.

Default value: NO.

QUOTED_IDENTIFIER= can also be specified with the QUOTED= alias.

READ_BUFFER=*numeric-value*

specifies the number of bulk rows of DBMS data to write to the buffer.

Default value: 100

This option improves performance by specifying a number of rows that can be held in memory for efficient input into SYBASE. A higher number signifies that more rows can be held in memory and accessed quickly during read operations. If you omit READ_BUFFER=, the default value is READ_BUFFER=100. The maximum number of rows allowed is dependent on the amount of memory that is available to your system.

READ_BUFFER= can also be specified with the BUFFSIZE= alias.

READ_ISOLATION_LEVEL=1 | 2 | 3

specifies which level of read isolation locking for SYBASE to use when it reads tables and views.

If you omit READ_ISOLATION_LEVEL=, the default value is READ_ISOLATION_LEVEL=1, which is the SYBASE default transaction isolation level that prevents dirty reads. Refer to your SYBASE documentation for more information.

READ_ISOLATION_LEVEL= can also be specified with the READ_ISO_LVL= and R_ISOLVL= aliases.

READ_LOCK_TYPE=NOLOCK
: specifies the method of locking to use during the reading of tables and views.

  In the SAS/ACCESS Interface to SYBASE, the default value and only valid value is READ_LOCK_TYPE=NOLOCK.

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

SCHEMA=*schema-name*
: allows you to view another user's database tables and views. If you omit SCHEMA=, you can view only your own tables and views.

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

UPDATE_ISOLATION_LEVEL=1 | 3
: specifies which level of read isolation locking for SYBASE to use when it reads tables and views for update.

  If you omit UPDATE_ISOLATION_LEVEL=, the default value is UPDATE_ISOLATION_LEVEL=1, which is the SYBASE default cursor isolation level. SYBASE uses a shared or update lock on base table pages that contain rows that represent a current cursor position. This option applies only when UPDATE_LOCK=PAGE. Refer to your SYBASE documentation for more information.

  UPDATE_ISOLATION_LEVEL= can also be specified with the CURSOR_ISOLATION_LEVEL=, CURSOR_ISO_LVL=, UPD_ISO_LVL=, and U_ISOLVL= aliases.

  *Note:* This option applies to updates only when UPDATE_LOCK_TYPE=PAGE because cursor updating is in effect. It does not apply when UPDATE_LOCK_TYPE=NOLOCK. △

UPDATE_LOCK_TYPE=NOLOCK | PAGE
: specifies the method of locking to use during the updating of tables and views.

  If you specify UPDATE_LOCK_TYPE=NOLOCK, SAS/ACCESS uses SYBASE browse mode updating, in which the table that is being updated must have a primary key and a timestamp column. If you specify UPDATE_LOCK_TYPE=PAGE, SAS/ACCESS uses a cursor that can be updated. When you use UPDATE_LOCK_TYPE=PAGE, it is recommended that the table have a defined primary key. If you omit UPDATE_LOCK_TYPE=, the default value is UPDATE_LOCK_TYPE=PAGE.

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

## Example 1: Specifying a LIBNAME Statement to Access SYBASE Data

In this example, the libref MYDBLIB connects to a SYBASE database. The SAS/ACCESS engine connection options are USER=, PASSWORD=, DATABASE=, and SERVER=.

```
libname mydblib sybase user=testuser
   password=testpass database=testdb
   server=testserver;

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

### Example 2: Appending SAS Data to a SYBASE Table Using BULKCOPY=

In this example, rows from the SAS data set, HR2.HIRES90S, are being added to the SYBASE table, MIDHIRES.STAFF. By specifying the BULKCOPY=YES data set option, the rows are inserted into the SYBASE tables as a unit. The BULKCOPY= option enables the connection to perform a bulk copy and, therefore, the rows of data are appended more quickly.

```
libname midhires sybase user=alhafez password=football server=hr_svr
               database=personnel;
libname hr2 'Your-SAS-Data-Library';
proc append base=midhires.staff(bulkcopy=yes) data=hr2.hires90s;
             where HIREDATE between '01JAN1990'd and '31DEC1997'd;
run;
```

### See Also

The SQL Chapter in the *SAS Procedures Guide* for enhancements in Version 7 and Version 8.

## Data Set Options: SYBASE 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 SYBASE. 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 SYBASE-specific LIBNAME and data set options. △

The SAS/ACCESS data set options for SYBASE are as follows:

“AUTOCOMMIT=” on page 8

“BULK_BUFFER=” on page 8

“BULKCOPY=” on page 8

“DBLINK=” on page 9

“READ_BUFFER=” on page 9

“READ_ISOLATION_LEVEL=” on page 10

“READ_LOCK_TYPE=” on page 10

“SEGMENT_NAME=” on page 11

“UPDATE_ISOLATION_LEVEL=” on page 11

“UPDATE_LOCK_TYPE=” on page 12

## AUTOCOMMIT=

**Specifies whether to enable the SYBASE autocommit capability.**

**Default value:** YES

### Syntax

**AUTOCOMMIT=**YES | NO

**YES**
enables the SYBASE autocommit capability.

**NO**
does not enable the SYBASE autocommit capability.

**Details** If AUTOCOMMIT=YES, all updates, inserts, and deletes are committed immediately after they are executed and no rollback is possible. If AUTOCOMMIT=NO, SAS performs the commit after processing the number of rows that are specified by using DBCOMMIT=, or the default number of rows, if DBCOMMIT= is not specified.

## BULK_BUFFER=

**Specifies the number of bulk rows that the SAS/ACCESS engine can buffer for output.**

**Default value:** 100

**Alias:** BULKBUFF=

### Syntax

**BULK_BUFFER=***numeric-value*

***numeric-value***
specifies the number of rows to be buffered for output.

**Details** This option improves performance by specifying the number of rows that can be held in memory for efficient retrieval from SYBASE. A higher number signifies that more rows can be held in memory and accessed quickly during output operations. The maximum number of rows that are allowed depends on the amount of memory that is available to your system.

## BULKCOPY=

**Calls the SYBASE bulk copy utility.**

**Default value:** NO

**Aliases:** BULK=, DBNOLOG=

### Syntax

**BULKCOPY=**YES | NO

**YES**
calls the SYBASE bulk copy utility when you load data into a SYBASE table.

**NO**
does not call the SYBASE bulk copy utility.

**Details** This utility groups rows so that they are inserted as a unit into a SYBASE table. Using this utility can improve performance.

If you specify BULKCOPY=YES, the SYBASE bulk copy utility is enabled and the connection to do the bulk copying is enabled. The LIBNAME option ENABLE_BULK= works in conjunction with the BULKCOPY= data set option. The LIBNAME ENABLE_BULK= option is YES by default, but the BULKCOPY= data set option is NO by default. Therefore, you must set BULKCOPY=YES to perform the bulk load.

If you specify BULKCOPY=NO, or omit the BULKCOPY= option (leaving the default value of NO), the bulk copy utility is not enabled.

For examples that use the BULKCOPY= options, see "Example 1: Specifying a LIBNAME Statement to Access SYBASE Data" on page 6.

BULKCOPY= can also be specified with the BULK= and DBNOLOG= aliases.

## DBLINK=

**Allows you to link to another database within the same server to which you are connected.**

**Default value:** None

### Syntax

**DBLINK=***database-link*

***database-link***
specifies the name of the SYBASE database to which you want to link.

### Details

The database server that you specify must be on a server to which you are connected. The DBLINK= dataset option overrides value of the LIBNAME option DBLINK=.

## READ_BUFFER=

**Specifies the number of bulk rows of SYBASE data to write to the buffer.**

**Default value:** 100

### Syntax

**READ_BUFFER=***numeric-value*

***numeric-value***
specifies the number of rows which can be held in the buffer during read operations from SAS.

**Details** This option improves performance by specifying a number of rows that can be held in memory for input into SAS. The maximum number of rows allowed is dependent on the amount of memory available to your system. Buffering data reads can decrease network activities and increase performance.

READ_BUFFER= can also be specified with the BUFFSIZE=, BUFSIZE=, and SYBBUFSIZE= aliases.

## READ_ISOLATION_LEVEL=

**Specifies which level of read isolation locking to use when reading data.**

**Default value:** 1

### Syntax

**READ_ISOLATION_LEVEL=**1 | 2 | 3

**1**
indicates that the SYBASE default transaction isolation level, which prevents dirty reads, is used.

**2**
indicates that SYBASE transaction level 2, which uses serialized reads, is used.

**3**
indicates that SYBASE transaction level 3, which also uses serialized reads, is used.

**Details** Refer to your SYBASE documentation for more information.

READ_ISOLATION_LEVEL can also be specified with the READ_ISO_LVL= and R_ISOLVL= aliases.

## READ_LOCK_TYPE=

**Specifies the method of locking to use during the reading of tables and views.**

**Default value:** NOLOCK

**Details** In the SAS/ACCESS Interface to SYBASE the only valid value is READ_LOCK_TYPE=NOLOCK.

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=***schema-name*

***schema-name***
specifies the name of the user whose database objects you want to view.

### Details

SCHEMA= can also be specified with the OWNER= alias. The SCHEMA= dataset option overrides the value of the LIBNAME option, SCHEMA=.

## SEGMENT_NAME=

**Allows you to control the segment in which you create a table.**

**Default value:** None

### Syntax

**SEGMENT_NAME=***segment-name*

***segment-name***
specifies the name of the segment in which to create a table.

SEGMENT_NAME can also be specified with the SEGMENT= alias.

## UPDATE_ISOLATION_LEVEL=

**Specifies which level of read isolation locking to use when updating data.**

**Default value:** 1

## Syntax

**UPDATE_ISOLATION_LEVEL=**1 | 3

**1**
indicates that the SYBASE default cursor isolation level is used.

**3**
indicates that SYBASE cursor isolation level 3, which uses serialized reads, is used.

**Details** SYBASE uses a shared or update lock on base table pages that contain rows representing a current cursor position. This option applies only when UPDATE_LOCK=PAGE. Refer to your SYBASE documentation for more information.

*Note:* This option applies to updates only when UPDATE_LOCK_TYPE=PAGE because cursor updating is in effect. It does not apply when UPDATE_LOCK_TYPE=NOLOCK. △

UPDATE_ISOLATION_LEVEL= can also be specified with the CURSOR_ISOLATION_LEVEL=, CURSOR_ISO_LVL=, UPD_ISO_LVL= and U_ISOLVL= aliases.

## UPDATE_LOCK_TYPE=

**Specifies the method of locking to use during the updating of tables and views.**

**Default value:** PAGE

**Details** If you specify UPDATE_LOCK_TYPE=NOLOCK, SAS/ACCESS uses SYBASE browse mode updating, in which the table that is being updated must have a primary key and timestamp. If you specify UPDATE_LOCK_TYPE=PAGE, SAS/ACCESS uses a cursor that can be updated. When you use UPDATE_LOCK_TYPE=PAGE, it is recommended that the table have a defined primary key.

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

## ACCESS Procedure: SYBASE Specifics

Chapter 9, "ACCESS Procedure Reference" describes the generic options and procedure statements that enable you to create access descriptors, view descriptors, and SAS data files from DBMS data. The following section describes the SYBASE-specific statements that you use in the SAS/ACCESS Interface to SYBASE.

In Version 8 of the SAS/ACCESS Interface to SYBASE, you can do the following:

- create, update, and use Version 8 access descriptors and view descriptors
- update an access descriptor created in Version 6

- create a view descriptor that is based on an access descriptor created in Version 6
- use a view descriptor created in Version 6.

If you are using Version 6 of the SAS/ACCESS Interface to SYBASE, you can do the following:

- create and use Version 6 access and view descriptors
- use a view descriptor created in Version 8 that is based on an access descriptor created in Version 6
- use an access descriptor created in Version 6 and updated in Version 8.

*Note:* In Version 6, you cannot use access descriptors created in Version 8, and you cannot use view descriptors created in Version 8 that are based on Version 8 access descriptors. △

## ACCESS Procedure Statements for SYBASE

To create an access descriptor, you use the DBMS=SYBASE option and these database description statements in the PROC ACCESS step: USER=, PASSWORD=, DATABASE=, SERVER=, INTERFACE=, and SYBBUFSZ=. The database description statements supply DBMS-specific information to the SAS System. These statements must immediately follow the CREATE statement.

Database description statements are required only when you create access descriptors. Because DBMS information is stored in an access descriptor, you do not need to repeat this information when you create view descriptors.

The SAS/ACCESS Interface to SYBASE uses the following procedure statements in interactive line, noninteractive, or batch mode.

**PROC ACCESS** <*accdesc-options* | *viewdesc-options*>;

**CREATE** <*libref.*>*member-name.*ACCESS | VIEW <*password-option*>;

**UPDATE** <*libref.*>*member-name.*ACCESS | VIEW <*password-option*>;

**USER**= <'>*user-name*<'>;

**PASSWORD**= | PASS= | SYBPW= <'>*password*<'>;

**DATABASE**= <'>*database-name*<'>;

**SERVER**= <'>*server-name*<'>;

**INTERFACE**=<'>*file-name*<'>;

**SYBBUFSZ**= | SYBBUF= *number-of-rows*;

**TABLE**= <'>*table-name*<'>;

**ASSIGN** <=> YES | NO;

**DROP** <'>*column-identifier-1*<'> <...<'>*column-identifier-n* <'>>;

**FORMAT** <'>*column-identifier-1*<'> <=>*SAS-format-name-1*
<...<'>*column-identifier-n*<'><=> *SAS-format-name-n*>;

**LIST** <ALL | VIEW | <'>*column-identifier*<'>>;

**QUIT**;

**RENAME** <'>*column-identifier-1*<'> <=> *SAS-variable-name-1*
<...<'>*column-identifier-n*<'> <=> *SAS-variable-name-n*>;

**RESET** ALL | <'>*column-identifier-1*<'> <...<'>*column-identifier-n*<'>>;

**SELECT** ALL | <'>*column-identifier-1*<'> <...<'>*column-identifier-n*<'>>;

**SUBSET** *selection-criteria*;

**UNIQUE** <=> YES | NO;

*Note:* SYBASE is a case-sensitive database. Any DBMS objects that were created with lowercase names or whose names contain special characters must be enclosed in quotation marks. See “Case Sensitivity in SYBASE” on page 20 for more information. △

DATABASE=<’>*database-name*<’>;
: specifies the name of the SYBASE database that contains the table on which the access descriptor is based. If you omit this statement, the default database for your SYBASE user name is used.

INTERFACE=<’>*file-name*<’>;
: specifies the name and location of the interfaces file that is searched when you connect to the SYBASE server. The interfaces file contains names and access information for the available servers on the network.

  If you omit a user name, the default action for your operating system occurs. INTERFACE= is not used in some operating environments. Contact your DBA to see whether this statement applies to your computing environment.

USER= <’>*user-name*<’>;
: specifies the SYBASE user name (also called the login name) that you use when executing the ACCESS procedure. If you omit this statement, your operating system account name is used, if applicable to your operating environment.

PASSWORD=<’>*password*<’>;
: specifies the password for SYBASE. If you omit the password, a default password of NULL is used.

  PASSWORD= can also be specified with the PASS= and SYBPW= aliases.

SERVER=<’>*server-name*<’>;
: specifies the server with which to connect. This server accesses the database that contains the table on which a particular access descriptor is based. If the server name contains lowercase or special characters, you must enclose it in quotation marks.

  If you omit a server name, the default action for your operating system occurs. On UNIX systems, the value of the environment variable DSQUERY is used if it has been set.

SYBBUFSZ=*number-of-rows*;
: specifies the number of rows of SYBASE data to write to the buffer. If this statement is used, SAS/ACCESS creates a buffer whenever the associated view descriptor is used to browse DBMS data. (Buffering is not performed when the view descriptor is being used to *update* data.) The interface view engine uses SYBBUFSZ= to improve performance by reducing network traffic.

  The specified buffer size can be between 1 and 32,767 rows. If you omit this statement, no data is written to the buffer.

  SYBBUFSZ= can also be specified with the SYBBUF= aliases.

## ACCESS Procedure Example

In the following example, you create access descriptors and view descriptors for the EMPLOYEES and INVOICE tables. These tables have different owners and are stored in PERSONNEL and INVENTORY databases that reside on different machines. The USER= and PASSWORD= statements identify the owners of the SYBASE tables and their passwords.

SYBASE is a case-sensitive database. The PROC ACCESS database identification statements and the SYBASE column names in all of the statements except SUBSET are converted to uppercase unless the names are enclosed in quotation marks. The SUBSET statements are passed to SYBASE exactly as you type them, so you must use the correct case for the SYBASE column names.

```
libname vlib 'sas-data-library';

proc access dbms=sybase;
   create work.employee.access;
      server='server1';
      database='personnel';
      user='testuser1';
      password='testpass1';
      table=EMPLOYEES;
   create vlib.emp_acc.view;
      select all;
      format empid 6.;
      subset where DEPT like 'ACC%';
run;

proc access dbms=sybase;
   create work.invoice.access;
      server='server2';
      database='inventory';
      user='testuser2';
      password='testpass2';
      table=INVOICE;
      rename invoicenum=invnum;
      format invoicenum 6. billedon date9.
        paidon date9.;
   create vlib.sainv.view;
      select all;
      subset where COUNTRY in ('Argentina','Brazil');
run;

options linesize=120;
title 'South American Invoices and
        Who Submitted Them';

proc sql;
   select invnum, country, billedon, paidon,
          billedby, lastname, firstnam
      from vlib.emp_acc, vlib.sainv
      where emp_acc.empid=sainv.billedby;
```

## DBLOAD Procedure: SYBASE Specifics

Chapter 10, "DBLOAD Procedure Reference" describes the generic options and procedure statements that enable you to create a DBMS table and to insert data in it. The following section describes the DBMS-specific statements that you use in the SAS/ACCESS Interface to SYBASE.

### DBLOAD Procedure Statements for SYBASE

To create and load a SYBASE table, the SAS/ACCESS Interface uses the following statements in the DBLOAD procedure in line and batch mode.

The database description statements include USER=, PASSWORD=, DATABASE=, SERVER=, and INTERFACE= (for some installations). If specified, these statements must immediately follow the PROC DBLOAD statement. The order of the statements within this group does not matter. The TABLE= statement must follow the database description statements.

**PROC DBLOAD** <DBMS=SYBASE>
  <DATA= <*libref.*>*SAS-data-set*> <APPEND>;
 **USER**=<'>*user-name*<'>
 **PASSWORD**=| PASS= | SYBPW= <'>*password*<'>
 **DATABASE**=| DB= <'>*database-name*<'>;
 **SERVER**=<'>*server-name*<'>;
 **INTERFACE**=<'>*file-name*<'>;
 **BULKCOPY**=| BULK= Y | N;
 **TABLE**=<'>*table-name*<'>;
 **ACCDESC**=<*libref.*>*access-descriptor*;
 **COMMIT**=*commit-frequency*;
 **DELETE** *variable-identifier-1*<*...variable-identifier-n*>;
 **ERRLIMIT**=*error-limit*;
 **LABEL**;
 **LIMIT**=*load-limit*;
 **LIST** <ALL | COLUMN | *variable-identifier*>;
 **LOAD**;
 **NULLS** *variable-identifier-1*=Y | N <*...variable-identifier-n*=Y | N>;
 **QUIT**;
 **RENAME** *variable-identifier-1*=
  <'>*column-name-1*<'>
  <*...variable-identifier-n* =<'>*column-name-n*<'>>;
 **RESET** ALL | *variable-identifier-1* <*...variable-identifier-n*>;
 **SQL** *Transact-SQL-statement*;
 **TYPE** *variable-identifier-1='column-type-1'*
  <*...variable-identifier-n='column-type-n'*>;
 **WHERE** *SAS-where-expression*;

**RUN**;

*Note:* If a database object contains lowercase, national, or special characters, you must enclose it in quotation marks. SYBASE is usually set for case sensitivity. Therefore, any objects that were given lowercase names when they were created, or whose names contain special characters, must be enclosed in quotation marks. See "Case Sensitivity in SYBASE" on page 20 for more information. △

DATABASE=<'>*database-name*<'>;
: specifies the name of the database in which you want to store the new SYBASE table. If you omit this statement, the default database for your SYBASE user name is used.

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

INTERFACE=<'>*file-name*<'>;
: specifies the name and location of the interfaces file that is searched when you connect to the SYBASE server. The interfaces file contains names and access information for available servers on the network.

  If you omit this statement, the default action for your operating system occurs. INTERFACE= is not used in some operating environments. Contact your DBA to determine whether this statement applies to your operating environment.

USER= <'>*user-name*<'>;
: specifies the SYBASE user name (also called the login name) that you use when executing the DBLOAD procedure. If you omit a user name, your operating system account name is used, if applicable to your operating environment.

PASSWORD=<'>*password*<'>;
: specifies the password for SYBASE. If you omit a password, a default password of NULL is used.

  PASSWORD= can also be specified with the PASS= and SYBPW= aliases.

SERVER=<'>*server-name*<'>;
: specifies the server with which to connect. This server accesses the database in which your new table is stored. If the server name contains lowercase or special characters, you must enclose it in quotation marks.

  If you omit this statement, the default action for your operating system occurs. On UNIX systems, the value of the environment variable DSQUERY is used if it has been set.

BULKCOPY=Y | N
: uses the SYBASE bulk copy utility to insert rows into a SYBASE table. The default value is N.

  If you specify BULKCOPY=Y, BULKCOPY= calls the SYBASE bulk copy utility in order to load data into a SYBASE table. This utility groups rows so that they are inserted as a unit into the new table. Using the bulk copy utility can improve performance.

  You use the COMMIT= statement to specify the number of rows in each group; this argument must be a nonnegative integer. After each group of rows is inserted, the rows are permanently saved in the table. While each group is being inserted, if one row in the group is rejected, then all of the rows in that group are rejected.

  If you specify BULKCOPY=NO, rows are inserted into the new table using Transact-SQL INSERT statements. Refer to your SYBASE documentation for more information on the bulk copy utility.

  BULKCOPY= can also be specified with the BULK= alias.

## DBLOAD Procedure Example

The following example creates a new SYBASE table, EXCHANGE, from the DLIB.RATEOFEX data file. An access descriptor ADLIB.EXCHANGE is also created, which is based on the new table. The DBLOAD procedure sends a Transact-SQL GRANT statement to SYBASE. You must be granted SYBASE privileges to create new SYBASE tables or to grant privileges to other users.

*Note:* The DLIB.RATEOFEX data set is included in the sample data that is shipped with your software. △

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

proc dbload dbms=sybase data=dlib.rateofex;
   server='server1'; database='testdb';
   user='testuser'; password='testpass';
   table=EXCHANGE;
   accdesc=adlib.exchange;
   rename fgnindol=fgnindolar 4=dolrsinfgn;
   nulls updated=n fgnindol=n 4=n country=n;
   load;
run;
```

## SQL Procedure Pass-Through Facility: SYBASE Specifics

Chapter 6, "SQL Procedure's Interaction with SAS/ACCESS Software" describes the generic PROC SQL statements you use to connect to and disconnect from a DBMS, to send DBMS-specific statements to the DBMS, and to retrieve DBMS data for your SAS programs. The following section describes the SYBASE-specific arguments that you use in the CONNECT statement"Arguments to Connect to SYBASE" on page 18.

### Arguments to Connect to SYBASE

The CONNECT statement establishes a connection with SYBASE. You establish a connection to send Transact-SQL statements to SYBASE or to retrieve SYBASE data. You can connect multiple times to one or more servers. After you establish a connection, it remains in effect until you issue a DISCONNECT statement or terminate the SQL procedure with a QUIT statement or begin a new DATA step or PROC step.

The CONNECT statement is optional. If omitted, the default values for all of the connection arguments are used.

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

Connection arguments for SYBASE are all case-sensitive. They are passed to SYBASE exactly as you type them. See "Case Sensitivity in SYBASE" on page 20 for more information.

USER=*username*
: specifies the SYBASE user ID that accesses the database. If you omit a username, your operating system account name is used, if appropriate for your operating environment.

PASSWORD=*password*
: identifies the SYBASE password that is associated with the specified SYBASE user ID. If you omit a password, a default password of NULL is used.
  PASSWORD= can also be specified with the SYBPW= alias.

DATABASE=*database-name*
: identifies the database to use for the connection. If you omit this statement, the default database for your SYBASE username is used.
  DATABASE= can also be specified with the DB= alias.

INTERFACE=*file-name*
: specifies the name and location of the SYBASE interfaces file. The interfaces file contains the names and network addresses of all of the available servers on the network.
  If you omit this statement, the default action for your operating system occurs. INTERFACE= is not used in some operating environments. Contact your DBA to determine whether it applies to your operating environment.

SERVER=*server-name*
: identifies the server to which this connection is made. If you omit this statement, the default SYBASE action for your operating system occurs. On UNIX systems, the value of the environment variable DSQUERY is used if it has been set.

SYBBUFSZ=*number-of-rows*

specifies the number of rows of DBMS data to write to the buffer. If this statement is used, the SAS/ACCESS interface view engine creates a buffer that is large enough to hold the specified number of rows. This buffer is created when the associated database table is read. The interface view engine uses SYBBUFSZ= to improve performance.

If you omit this statement, no data is written to the buffer.

## Pass-Through Example

The following example retrieves a subset of rows from the SYBASE INVOICE table. Because the WHERE clause is specified in the DBMS query (the inner SELECT statement), the DBMS processes the WHERE expression and returns a subset of rows to the SAS System.

```
proc sql;
connect to sybase(server=SERVER1
                  database=INVENTORY
                  user=testuser password=testpass);
%put &sqlxmsg;

select * from connection to sybase
   (select * from INVOICE where BILLEDBY=457232);
%put &sqlxmsg;
```

*Note:* The SELECT statement that is enclosed in parentheses is sent directly to the database and therefore must be specified using valid database variable names and syntax. △

## SYBASE Naming Conventions

SYBASE database objects that can be named include tables, views, columns, indexes, and database procedures. Use the following SYBASE naming conventions:

- □ A name must be from 1 to 30 characters long (or 28 characters if quoted).
- □ A name must start with an alphabetic character, unless the name is surrounded by quotation marks.
- □ After the first character, a name may contain the letters A through Z (in uppercase or lowercase), the digits 0 through 9, the underscore (_), the dollar sign ($), the pound sign (#), the at sign (@), the yen sign (¥), and the monetary pound sign (£). If you use any of these characters, enclose the name in quotation marks.
- □ Embedded spaces are not permitted unless the name is surrounded by quotation marks.
- □ Embedded quotation marks are not permitted.
- □ A name cannot be a SYBASE reserved word unless the name is surrounded by quotation marks. See your SYBASE documentation for more information about reserved words.
- □ Case sensitivity is set when a server is installed. By default, the names of database objects are case sensitive. On a case-sensitive server, the names **`CUSTOMER`** and **`customer`** are different.

- Database names must be unique. For each owner within a database, names of database objects must be unique. Column names and index names must be unique within a table.

*Note:* By default, column and table names are notquoted in the SAS/ACCESS Interface to SYBASE. To quote the table and column names, you must use the LIBNAME statement QUOTED_IDENTIFIER= option when you assign a libref. △

When you use the DATASETS procedure to list your SYBASE tables, the table names appear exactly as they exist in the SYBASE data dictionary. If you specified the LIBNAME option, SCHEMA= , SAS/ACCESS lists the tables for the specified schema username.

To reference a table or other named object that you own or for the specified schema, refer to the table name (for example, CUSTOMERS). If you use the LIBNAME statement DBLINK= option, all references to the libref will refer to the specified database.

## Case Sensitivity in SYBASE

SAS names can be entered in either uppercase or lowercase. When you reference SYBASE objects through the SAS/ACCESS LIBNAME engine, objects are case-sensitive and require no quotes.

However, SYBASE database is generally set for case sensitivity, and special consideration should be given to the names of objects (such as tables and columns) when they are to be used in SAS software by the SAS/ACCESS procedures. The ACCESS procedure converts SYBASE object names to uppercase unless they are enclosed in quotation marks. Any SYBASE objects that were given lowercase names, or whose names contain national or special characters, must be enclosed in quotation marks. The only exceptions are the SUBSET statement in the ACCESS procedure and the SQL statement in the DBLOAD procedure. Arguments or values from these statements are passed to SYBASE exactly as you type them, with the case preserved.

In the SQL Procedure Pass-Through Facility, all SYBASE object names are case sensitive. The names are passed to SYBASE exactly as they are typed.

For more information about case-sensitivity and SYBASE names, see “SYBASE Naming Conventions” on page 19.

## SYBASE Data Types

Every column in a table has a name and a data type. The *data type* indicates to the DBMS how much physical storage to reserve for the column and the format in which the data is stored. SYBASE data types fall into four categories: types for character data, types for numeric data, types for abstract values, and user-defined data types. Each of these types is described in the following sections.

*Note:* SAS/ACCESS does not support the following SYBASE data types: BINARY, VARBINARY, IMAGE, NCHAR(*n*), and NVARCHAR(*n*). SAS/ACCESS provides an error message when it attempts to read a table that has at least one column that uses an unsupported data type. △

## Character Data

You must enclose all character data in single or double quotation marks.

CHAR(*n*)
: CHAR(*n*) is a character string that can have 1 to 255 letters, symbols, and numbers. You specify the maximum length of the string with *n*. Storage size is also *n*, regardless of the actual entry length.

VARCHAR(*n*)
: VARCHAR(*n*) is a varying-length character string that can have 1 to 255 letters, symbols, and numbers. You specify the maximum length of the string with *n*. Storage size is the actual entry length.

TEXT
: TEXT stores character data of variable length up to two gigabytes. SAS supports the TEXT data type provided in SYBASE; however, SAS only allows a maximum of 32,767 bytes of character data.

## Numeric Data

NUMERIC(*p,s*), DECIMAL(*p,s*)
: Exact numeric values have specified degrees of precision (*p*) and scale (*s*). NUMERIC data can have a precision of 1 to 38 and scale of 0 to 38, where the value of *s* must be less or equal to than the value of *p*. The DECIMAL data type is identical to the NUMERIC data type. The default precision and scale are (18,0) for the DECIMAL data type.

REAL, FLOAT
: Floating-point values consist of an integer part, a decimal point, and a fraction part, or scientific notation. The exact format for REAL and FLOAT data depends on the number of significant digits and the precision that your machine supports. You can use all arithmetic operations and aggregate functions with REAL and FLOAT except modulus. The REAL (4 byte) range is approximately 3.4E–38 to 3.4E+38, with 7-digit precision. The FLOAT (8 byte) range is approximately 1.7E–308 to 1.7E+308, with 15-digit precision.

TINYINT, SMALLINT, INT
: Integers contain no fractional part. The three integer data types are TINYINT (1 byte), which has a range of 0 to 255; SMALLINT (2 bytes), which has a range of -32,768 to +32,767; and INT (4 bytes), which has a range of -2,147,483,648 to +2,147,483,647.

BIT
: BIT data has a storage size of one bit and holds either a 0 or a 1; other integer values are accepted but are interpreted as 1. BIT data cannot be NULL and cannot have indexes defined on it.

## Abstract Data

SYBASE date and money data types are abstract data types and are described in this section. Refer to your documentation on Transact-SQL for more information about abstract data types.

SMALLDATETIME
: SMALLDATETIME data is 4 bytes long and consists of one small integer that represents the number of days after January 1, 1900, and one small integer that represents the number of minutes past midnight. The date range is from January 1, 1900, to December 31, 2079.

DATETIME
: DATETIME data has two 4-byte integers. The first integer represents the number of days after January 1, 1900, and the second integer represents the number of minutes past midnight. Values can range from January 1, 1753 to December 31, 9999.

  DATETIME values are input as quoted character strings in various alphabetic or numeric formats. Time data must be entered in the prescribed order (hours; minutes; seconds; milliseconds; AM, am, PM, pm) and must include either a colon or an AM/PM designator. Case is ignored, and spaces can be inserted anywhere within the value.

  When you input DATETIME values, the national language setting determines how the date values are interpreted. You can change the default date order with the SET DATEFORMAT statement. See your Transact-SQL documentation for more information.

  You can use SYBASE built-in date functions to perform some arithmetic calculations on DATETIME values.

TIMESTAMP
: TIMESTAMP data is used by SAS in UPDATE mode. If you select a column that contains TIMESTAMP data for input into SAS, the values are displayed in hex format.

SMALLMONEY
: SMALLMONEY data is 4 bytes long and can range from -214,748.3648 to 214,748.3647. When displayed, it is rounded up to two places.

MONEY
: MONEY data is 8 bytes long and can range from -922,337,203,685,477.5808 to 922,337,203,685,477.5807. When input, a dollar sign ($) must appear before the MONEY value. For negative values, the minus sign must follow the dollar sign. Commas are not allowed.

  MONEY values are accurate to a ten-thousandth of a monetary unit. However, when they are displayed, the dollar sign is omitted and MONEY values are rounded up to two places. A comma is inserted after every three digits.

  You can store values for currencies other than USA dollars, but no form of conversion is provided.

## User-Defined Data Types

You can supplement the SYBASE system data types by defining your own data types with the SYBASE system procedure **sp_addtype**. When you define your own data type for a column, you can specify a default value (other than NULL) for the column and define a range of allowable values for the column.

## NULL Values

SYBASE has a special value that is called NULL. NULL means that a value in a row is not known or is missing; it does not mean that the value is blank or zero. It is analogous to the SAS System's *missing value.*

By default, SYBASE columns are defined as NOT NULL. NOT NULL tells SYBASE not to add a row to the table unless the row has a value for the specified column.

If you want a column to accept NULL values, you must explicitly define it as NULL. Here is an example of a CREATE TABLE statement that defines all of the columns for a table to be NULL except for CUSTOMER. In this case, SYBASE only accepts a row that contains a value for CUSTOMER.

```
create table CUSTOMERS
   (CUSTOMER        char(8)    not null,
    STATE           char(2)        null,
    ZIPCODE         char(5)        null,
    COUNTRY         char(20)       null,
    TELEPHONE       char(12)       null,
    NAME            char(60)       null,
    CONTACT         char(30)       null,
    STREETADDRESS   char(40)       null,
    CITY            char(25)       null,
    FIRSTORDERDATE  datetime       null);
```

## LIBNAME Statement Data Conversions

Table 1.1 on page 23 shows the default SAS System variable formats that the libname statement assigns to SYBASE data types during input operations.

**Table 1.1** LIBNAME Statement: Default SAS Formats for SYBASE Server Data Types

| SYBASE Column Type | SAS Data Type | Default SAS Format |
|---|---|---|
| CHAR(*n* ) | character | $*n*. (*n* <= 255)<br>$255. (*n* > 255) |
| VARCHAR(*n* ) | character | $*n*. (*n* <= 255)<br>$255. (*n* > 255) |
| TEXT | character | $*n*. (*n* <= 32,767)<br>$32,767. (*n* > 32,767) |
| BIT | numeric | 1.0 |
| TINYINT | numeric | 4.0 |
| SMALLINT | numeric | 6.0 |
| INT | numeric | 11.0 |
| NUMERIC | numeric | *w*, *w.d* (if possible) |
| DECIMAL | numeric | *w*, *w.d* (if possible) |
| FLOAT | numeric | |
| REAL | numeric | |
| SMALLMONEY | numeric | DOLLAR12.2 |
| MONEY | numeric | DOLLAR24.2 |
| SMALLDATETIME | numeric | DATETIME22.3 |

| SYBASE Column Type | SAS Data Type | Default SAS Format |
|---|---|---|
| DATETIME | numeric | DATETIME22.3 |
| TIMESTAMP | hex | $HEX*w* |

Table 1.2 on page 24 shows the default SYBASE data types that the LIBNAME statement assigns to SAS variable formats during ouput operations.

**Table 1.2** LIBNAME STATEMENT: Default SYBASE Data Types for SAS Variable Formats

| SAS Variable Format | SYBASE Data Type |
|---|---|
| $*w*., $CHAR*w*., $VARYING*w*., $HEX*w*. | VARCHAR(*w*) |
| any datetime, date, or time format | DATETIME |
| any numeric with a SAS format name of *w.d* or *w*. | NUMERIC(p,s) |
| any other numeric | FLOAT |

You can override these default data types by using the DBTYPE= option on the data set.

## ACCESS Procedure Data Conversions

Table 1.3 on page 24 shows the default SAS System variable formats that the ACCESS procedure assigns to SYBASE data types.

**Table 1.3** PROC ACCESS: Default SAS Formats for SYBASE Server Data Types

| SYBASE Column Type | SAS Data Type | Default SAS Format |
|---|---|---|
| CHAR(*n* ) | character | $*n*. (*n* <= 200)<br>$200. (*n* > 200) |
| VARCHAR(*n* ) | character | $*n*. (*n* <= 200)<br>$200. (*n* > 200) |
| BIT | numeric | 1.0 |
| TINYINT | numeric | 4.0 |
| SMALLINT | numeric | 6.0 |
| INT | numeric | 11.0 |
| FLOAT | numeric | BEST22. |
| REAL | numeric | BEST11. |
| SMALLMONEY | numeric | DOLLAR12.2 |
| MONEY | numeric | DOLLAR24.2 |

| SYBASE Column Type | SAS Data Type | Default SAS Format |
|---|---|---|
| SMALLDATETIME | numeric | DATETIME21.2 |
| DATETIME | numeric | DATETIME21.2 |

The ACCESS procedure also supports SYBASE user-defined data types. The ACCESS procedure uses the SYBASE data type on which a user-defined data type is based in order to assign a default SAS format for columns.

*Note:* The DECIMAL, NUMERIC, and TEXT data types are not supported in PROC ACCESS. The TIMESTAMP data type is not displayed in PROC ACCESS. △

## DBLOAD Procedure Data Conversions

Table 1.4 on page 25 shows the default SYBASE data types that the DBLOAD procedure assigns to SAS variable formats.

**Table 1.4** PROC DBLOAD: Default SYBASE Data Types for SAS Variable Formats

| SAS Variable Format | SYBASE Data Type |
|---|---|
| $*w*., $CHAR*w*., $VARYING*w*., $HEX*w*. | VARCHAR(*w*) |
| *w*. | TINYINT |
| *w*. | SMALLINT |
| *w*. | INT |
| *w*. | FLOAT |
| *w.d* | FLOAT |
| IB*w.d*, PIB*w.d* | INT |
| FRACT, E format, and other numeric formats | FLOAT |
| DOLLAR*w.d*, *w*<=12 | SMALLMONEY |
| DOLLAR*w.d*, *w*>12 | MONEY |
| any datetime, date, or time format | DATETIME |

The DBLOAD procedure also supports SYBASE user-defined data types. Use the TYPE= statement to specify a user-defined data type.

## Inserting TEXT into SYBASE from SAS

TEXT data can only be inserted into a SYBASE table by using the BULK= data set option, as in the following example:

```
data yourlib.newtable(bulk=yes);
   set work.sasbigtext;
run;
```

If the BULK option is not used, you will receive the following error message:

```
ERROR: Object not found in database. Error Code: -2782
An untyped variable in the PREPARE statement 'S401bcf78'
is being resolved to a TEXT or IMAGE type.
This is illegal in a dynamic PREPARE statement.
```

## National Language Support for SYBASE

To support output and update processing from SAS into SYBASE in languages other than English, special setup steps are required so that date, time, and datetime values can be processed correctly. In SAS, you must ensure that the DFLANG= system option is set to the correct language. This can be globally set by the system administrator or set by a user within a single SAS session. In SYBASE, the default client language, set in the *locales.dat* file, must match the language used in SAS.

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

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

ISBN 1-58025-540-X

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

1st printing, October 1999