CHAPTER

1

SAS/ACCESS for SYBASE

Introduction to the SAS/ACCESS Interface to SYBASE

This document includes details *only* about the SAS/ACCESS interface to SYBASE. It should be used as a supplement to the generic SAS/ACCESS documentation *SAS/ACCESS for Relational Databases: Reference*.

LIBNAME Statement Specifics for SYBASE

This section describes the LIBNAME statement as supported by the SAS/ACCESS interface to SYBASE. For a complete description of this feature, see the LIBNAME statement section in *SAS/ACCESS for Relational Databases: Reference*. The following is the SYBASE specific syntax for the LIBNAME statement:

LIBNAME *libref* **sybase** <*connection-options*> <*LIBNAME-options*>;

Arguments

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

sybase
is the SAS/ACCESS engine name for the interface to SYBASE.

connection-options
provide connection information and control how SAS manages the timing and concurrence of the connection to the DBMS. The following are the connection options for the interface to SYBASE:

USER=<'>*SYBASE-user-name*<'>
specifies the SYBASE user name (also called the login name) that you use to connect to your database. If the user name contains spaces or non-alphanumeric characters, you must enclose it in quotation marks.

PASSWORD=<'>*SYBASE-password*<'>
specifies the password that is associated with your SYBASE user name.

If you omit the password, a default password of NULL is used. If the password contains spaces or non-alphanumeric characters, you must enclose it 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 that you want to connect to. This server accesses the database that contains the tables and views that you want to access.

If the server name contains lowercase, spaces, or non-alphanumeric 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.

IP_CURSOR= YES | NO
specifies whether Implicit Proc SQL pass-through processes multiple result sets simultaneously.

IP_CURSOR is set to NO by default. Setting it to YES allows this type of extended processing, but will decrease performance, since cursors are being used, and not result sets. Do not set to YES unless needed.

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.

LIBNAME-options
define how DBMS objects are processed by SAS. Some LIBNAME options can enhance performance; others determine locking or naming behavior. The following table describes which LIBNAME options are supported for SYBASE, and presents default values where applicable. See the section about the SAS/ACCESS LIBNAME statement in *SAS/ACCESS for Relational Databases: Reference* for detailed information about these options.

Table 1.1 SAS/ACCESS LIBNAME Options for SYBASE

Option	Default Value
ACCESS=	none
AUTOCOMMIT=	YES
CONNECTION=	SHAREDREAD
CONNECTION_GROUP=	none
DBCOMMIT=	1000 (inserting) or 0 (updating)
DBCONINIT=	none
DBCONTERM=	none
DBCREATE_TABLE_OPTS=	none
DBGEN_NAME=	DBMS
DBINDEX=	NO
DBLIBINIT=	none
DBLIBTERM=	none
DBLINK=	the local database
DBMAX_TEXT=	1024
DBPROMPT=	NO
DBSASLABEL=	COMPAT
DBSLICEPARM=	THREADED_APPS,2 or 3
DEFER=	NO
DIRECT_EXE=	none
DIRECT_SQL=	YES
ENABLE_BULK=	YES

Option	Default Value
INTERFACE=	none
MAX_CONNECTS=	25
MULTI_DATASRC_OPT=	NONE
PACKETSIZE=	server setting
QUOTED_IDENTIFIER=	NO
READBUFF=	100
READ_ISOLATION_LEVEL=	1
READ_LOCK_TYPE=	NOLOCK
REREAD_EXPOSURE=	NO
SCHEMA=	none
SPOOL=	YES
UPDATE_ISOLATION_LEVEL=	1
UPDATE_LOCK_TYPE=	PAGE
UTILCONN_TRANSIENT=	NO

SYBASE LIBNAME Statement Example

In the following example, the libref MYDBLIB uses the SYBASE engine to connect to a SYBASE database. USER= and PASSWORD= are connection options.

```
libname mydblib sybase user=testuser password=testpass;
```

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.

Data Set Options for SYBASE

The following table describes the data set options that are supported for SYBASE, and provides default values where applicable. See the section about data set options in *SAS/ACCESS for Relational Databases: Reference* for detailed information about these options.

Table 1.2 SAS/ACCESS Data Set Options for SYBASE

Option	Default Value
AUTOCOMMIT=	LIBNAME option setting
BULK_BUFFER=	100
BULKLOAD=	NO
DBCOMMIT=	LIBNAME setting

Option	Default Value
DBCONDITION=	none
DBCREATE_TABLE_OPTS=	LIBNAME setting
DBFORCE=	NO
DBGEN_NAME=	LIBNAME option setting
DBINDEX=	LIBNAME option setting
DBKEY=	none
DBLABEL=	NO
DBLINK=	LIBNAME option setting
DBMASTER=	none
DBMAX_TEXT=	LIBNAME option setting
DBNULL=	_ALL_YES
DBPROMPT=	LIBNAME option setting
DBSASLABEL=	COMPAT
DBSLICE=	none
DBSLICEPARM=	THREADED_APPS,2 or 3
DBTYPE=	see “Data Types for SYBASE” on page 17
ERRLIMIT=	1
NULLCHAR=	SAS
NULLCHARVAL=	a blank character
READBUFF=	LIBNAME option setting
READ_ISOLATION_LEVEL=	LIBNAME option setting
READ_LOCK_TYPE=	LIBNAME option setting
SASDATEFMT=	DATETIME22.3
SCHEMA=	LIBNAME option setting
SEGMENT_NAME=	none
UPDATE_ISOLATION_LEVEL=	LIBNAME option setting
UPDATE_LOCK_TYPE=	LIBNAME option setting

Pass-Through Facility Specifics for SYBASE

See the section about the Pass-Through Facility in *SAS/ACCESS for Relational Databases: Reference* for general information about this feature.

The Pass-Through Facility specifics for SYBASE are as follows:

- The *dbms-name* is **SYBASE**.
- The CONNECT statement is optional. If you omit the CONNECT statement, an implicit connection is made using the default values for all of the connection options.
- The interface can connect multiple times to one or more servers.

□ The *database-connection-arguments* for the CONNECT statement are as follows:

USER=<'>*SYBASE-user-name*<'>
specifies the SYBASE user name (also called the login name) that you use to connect to your database. If the user name contains spaces or non-alphanumeric characters, you must enclose it 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 non-alphanumeric characters, you must enclose it in quotation marks.

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

Note: If you do not wish to enter your Sybase password in uncoded text, see PROC PWENCODE for a method to encode it. △

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 you want to connect to. This server accesses the database that contains the tables and views that you want to access.

If the server name contains lowercase, spaces, or non-alphanumeric 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.

INTERFACE=*filename*
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 database administrator to determine whether it applies to your operating environment.

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.

Note: Connection options for SYBASE are all case-sensitive. They are passed to SYBASE exactly as you type them. △

- The following LIBNAME options are available with the CONNECT statement:

  DBMAX_TEXT=

  MAX_CONNECTS=

  READBUFF=

  PACKETSIZE=

  See the section about the LIBNAME statement in *SAS/ACCESS for Relational Databases: Reference* for information about these options.

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 SAS.

```
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. △

Autopartitioning Scheme for SYBASE

See the section about threaded reads in *SAS/ACCESS for Relational Databases: Reference* for general information about this feature.

Overview

SYBASE autopartitioning utilizes the SYBASE MOD function (%) to create multiple SELECT statements with WHERE clauses, which, in the optimum scenario, divide the result set into equal chunks; one chunk per thread. For example, assume that your original SQL statement was **`SELECT * FROM DBTAB`**, and assume that DBTAB has a primary key column PKCOL of type integer and that you want it partitioned into three threads. The autopartitioning scheme would break up the table into three SQL statements as follows:

```
select * from DBTAB where (abs(PKCOL))%3=0
select * from DBTAB where (abs(PKCOL))%3=1
select * from DBTAB where (abs(PKCOL))%3=2
```

Since PKCOL is a primary key column, you should get a fairly even distribution among the three partitions, which is the primary goal.

Indexes

An index on a SAS partitioning column increases performance of the threaded read. If a primary key is not defined for the table, an index should be placed on the partitioning column in order to attain similar benefits. Understanding and following *Sybase ASE Performance and Tuning Guide* documentation recommendations with respect to index creation and usage is essential in order to achieve optimum database performance. The order of column selection for the partitioning column is as follows:

1. Identity column
2. Primary key column (integer or numeric)
3. integer, numeric, or bit; not nullable
4. integer, numeric, or bit; nullable

Note: If the column selected is a bit type, there will be only two partitions created since the only values are 0 and 1. △

Partitioning Criteria

The most efficient partitioning column is an Identity column, which is usually identified as a primary key column. Identity columns usually lead to evenly partitioned result sets because of the sequential values they store.

The least efficient partitioning column is a numeric, decimal, or float column that is NULLABLE, and does not have an index defined.

Given equivalent selection criteria, columns defined at the beginning of the table definition that meet the selection criteria will take precedence over columns defined toward the end of the table definition.

Data types

The following data types are supported in partitioning column selection:

integer

tinyint

smallint

numeric

decimal

float

bit.

Examples

The following are examples of generated SELECT statements involving various column data types:

COL1 is numeric, decimal, or float. This example uses three threads (the default) and COL1 is NOT NULL.

```
select * from DBTAB where (abs(convert(INTEGER, COL1)))%3=0
select * from DBTAB where (abs(convert(INTEGER, COL1)))%3=1
select * from DBTAB where (abs(convert(INTEGER, COL1)))%3=2
```

COL1 is bit, integer, smallint, or tinyint. This example uses two threads (the default) and COL1 is NOT NULL.

```
select * from DBTAB where (abs(COL1))%3=0
select * from DBTAB where (abs(COL1))%3=1
```

COL1 is and integer and is nullable.

```
select * from DBTAB where (abs(COL1))%3=0 OR COL1 IS NULL
select * from DBTAB where (abs(COL1))%3=1
```

Temporary Table Support for SYBASE

See the section on the temporary table support in *SAS/ACCESS for Relational Databases: Reference* for general information about this feature.

Establishing a Temporary Table

When you specify CONNECTION=GLOBAL, you can reference a temporary table throughout a SAS session, in both DATA steps and procedures. The name of the table MUST start with the character '#'. To reference it, use the SAS convention of an *n* literal, as in mylib.'#foo'n.

Terminating a Temporary Table

You can drop a temporary table at any time, or allow it to be implicitly dropped when the connection is terminated. Temporary tables do not persist beyond the scope of a singe connection.

Example

The following example demonstrates how to use temporary tables:

```
/* clear any connection */
libname x clear;

libname x sybase user=test pass=test connection=global;

/* create the temp table. You can even use bulk copy */
/* Notice how the name is specified: '#mytemp'n */
   data x.'#mytemp'n (bulk=yes);
   x=55;
   output;
   x=44;
   output;
run;

/* print it */
proc print data=x.'#mytemp'n;
run ;

/* The same temp table persists in PROC SQL, */
```

```
/* with the global connection specified */
proc sql;
   connect to sybase (user=austin pass=austin connection=global);
   select * from connection to sybase (select * from #mytemp);
quit;

/* use the temp table again in a procedure */
proc means data=x.'#mytemp'n;
run;

/* drop the connection, the temp table is automatically dropped */
libname x clear;

/* to convince yourself it's gone, try to access it */
libname x sybase user=austin password=austin connection=global;

/* it's not there */
proc print data=x.'#mytemp'n;
run;
```

ACCESS Procedure Specifics for SYBASE

See the section about the ACCESS procedure in *SAS/ACCESS for Relational Databases: Reference* for general information about this feature.

The SYBASE interface supports all of the ACCESS procedure statements. The SYBASE interface specifics for the ACCESS procedure are as follows:

- The DBMS= value for PROC ACCESS is **SYBASE**.
- The *database-description-statements* used by PROC ACCESS are identical to the Pass-Through Facility's CONNECT statement database-connection-arguments on page 6.
- The TABLE= statement for PROC ACCESS is:

  TABLE= <'>*table-name*<'>;
  specifies the name of the SYBASE table or SYBASE view on which the access descriptor is based.

Example

The following example creates 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.

```
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;
```

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.

DBLOAD Procedure Specifics for SYBASE

See the section about the DBLOAD procedure in *SAS/ACCESS for Relational Databases: Reference* for general information about this feature.

The SYBASE interface supports all of the DBLOAD procedure statements. The SYBASE interface specifics for the DBLOAD procedure are as follows:

- The DBMS= value for PROC DBLOAD is **SYBASE**.
- The TABLE= statement for PROC DBLOAD is:

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

□ PROC DBLOAD uses the following *database-description-statements*:

USER=<'>*SYBASE-user-name*<'>
specifies the SYBASE user name (also called the login name) that you use to connect to your database. If the user name contains spaces or non-alphanumeric characters, you must enclose it 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 non-alphanumeric characters, you must enclose it 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 that you want to connect to. This server accesses the database that contains the tables and views that you want to access.

If the server name contains lowercase, spaces, or non-alphanumeric 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.

INTERFACE=*filename*
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 database administrator to determine whether it applies to your operating environment.

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 positive 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=N, rows are inserted into the new table using Transact-SQL INSERT statements. Refer to your SYBASE documentation for more information about the bulk copy utility.

Example

The following example creates a new SYBASE table, EXCHANGE, from the DLIB.RATEOFEX data file. An access descriptor ADLIB.EXCHANGE is also created, and it 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;
```

Passing SAS Functions to SYBASE

The interface to SYBASE passes the following SAS functions to SYBASE for processing. See the section about optimizing SQL usage in *SAS/ACCESS for Relational Databases: Reference* for information.

ABS

ARCOS

ARSIN

ATAN

AVG

CEIL

COS

DATETIME

EXP

FLOOR

LOG

MAX

MIN

SIGN

SIN

SQRT

TAN

SUM

COUNT

Passing Joins to SYBASE

In order for a multiple libref join to pass to SYBASE, all of the following components of the LIBNAME statements must match exactly:

user ID

password

database

server.

See the section about performance considerations in *SAS/ACCESS for Relational Databases: Reference* for more information about when and how SAS/ACCESS passes joins to the DBMS.

Reading Multiple SYBASE Tables

SAS opens multiple SYBASE tables for simultaneous reading in the following situations:

- When you are using PROC COMPARE. For example:

```
proc compare base=syb.data1 compare=syb.data2;
```

- When you are running an SCL program that reads from more than one SYBASE table simultaneously.
- When you are joining SYBASE tables in SAS (when implicit pass through is not used (DIRECT_SQL=NO)). For example:

```
proc sql ;
   select * from syb.table1, syb.table2 where table1.x=table2.x;
```

or

```
proc sql;
   select * from syb.table1 where table1.x = (select x from syb.table2
   where y = 33);
```

or

```
proc sql;
   select empname from syb.employee where empyears > all (select empyears
   from syb.employee where emptitle = 'salesrep');
```

or

```
proc sql ;
   create view myview as
```

```
        select * from employee where empyears > all (select empyears from
        syb.employee where emptitle = 'salesrep');
    proc print data=myview ;
```

In order to read two or more SYBASE tables simultaneously, you must specify either the LIBNAME option CONNECTION=UNIQUE or the LIBNAME option READLOCK_TYPE=PAGE. Because READLOCK_TYPE=PAGE can degrade performance, it is generally recommended that you use CONNECTION=UNIQUE (unless there is a concern about the number of connections that are opened on the database).

Locking in the SYBASE Interface

The SAS/ACCESS interface to SYBASE supports the following LIBNAME and data set locking options. See the section about the LIBNAME statement in *SAS/ACCESS for Relational Databases: Reference* for additional information about these options.

READ_LOCK_TYPE= PAGE | NOLOCK
The default value for SYBASE is NOLOCK.

UPDATE_LOCK_TYPE= PAGE | NOLOCK

PAGE
SAS/ACCESS uses a cursor that can be updated. When you use this setting, it is recommended that the table have a defined primary key. PAGE is the default value for SYBASE.

NOLOCK
SAS/ACCESS uses SYBASE browse mode updating, in which the table that is being updated must have a primary key and timestamp.

READ_ISOLATION_LEVEL= 1 | 2 | 3
For reads, SYBASE supports isolation levels 1, 2, and 3, as defined in the following table. Refer to your SYBASE documentation for more information.

Table 1.3 Isolation Levels for SYBASE

Isolation Level	Definition
1	Prevents dirty reads. This is the default transaction isolation level.
2	Uses serialized reads.
3	Also uses serialized reads.

UPDATE_ISOLATION_LEVEL= 1 | 3
SYBASE uses a shared or update lock on base table pages that contain rows representing a current cursor position. 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.

For updates, SYBASE supports isolation levels 1 and 3, as defined in the preceding table. Refer to your SYBASE documentation for more information.

Understanding SYBASE Update Rules

To avoid data integrity problems when updating and deleting data in SYBASE tables, take the following precautionary measures:

- Always define a primary key.
- If the updates are not taking place via cursor processing, define a timestamp column as well.

It is not always obvious whether or not updates are utilizing cursor processing. Cursor processing is *never* used for LIBNAME statement updates if UPDATE_LOCK_TYPE=NOLOCK. Cursor processing is *always* used in the following situations:

- Updates using the LIBNAME statement with UPDATE_LOCK_TYPE=PAGE. *Note that this is the default setting for this option.*
- Updates using PROC SQL views.
- Updates using PROC ACCESS view descriptors.

Naming Conventions for SYBASE

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

- 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.
- A name must be from 1 to 30 characters long (or 28 characters if quoted).
- A name must start with an alphabetic character or an underscore (_), unless the name is enclosed in 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 (£).
- Embedded spaces are not permitted unless the name is enclosed in quotation marks.
- A name cannot be a SYBASE reserved word unless the name is enclosed in quotation marks. See your SYBASE documentation for more information about reserved words.
- Embedded quotation marks are not permitted.
- 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.

Note: By default, column and table names are not quoted 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 user name.

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 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 interface, objects are case-sensitive and require no quotation marks.

However, SYBASE 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 by the ACCESS or DBLOAD 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 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 "Naming Conventions for SYBASE" on page 16.

Data Types for SYBASE

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.

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 milliseconds 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 U.S. 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.

SYBASE Null Values

SYBASE has a special value that is called NULL. A SYBASE NULL value means an absence of information and is analogous to a SAS missing value. When SAS/ACCESS reads a SYBASE NULL value, it interprets it as a SAS 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 if it 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);
```

When creating a SYBASE table with SAS/ACCESS, you can use the DBNULL= data set option to indicate whether NULL is a valid value for specified columns.

For more information about how SAS handles NULL values, see in *SAS/ACCESS for Relational Databases: Reference*.

Note: To control how SAS missing character values are handled by SYBASE, use the NULLCHAR= and NULLCHARVAL= data set options. △

LIBNAME Statement Data Conversions

The following table shows the default SAS variable formats that SAS/ACCESS assigns to SYBASE data types during input operations when you use the LIBNAME statement.

Table 1.4 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) $255. (*n* > 255)
VARCHAR(*n*)	character	$*n*. (*n* <= 255) $255. (*n* > 255)
TEXT	character	$*n*. (where *n* is the value of the DBMAX_TEXT= option)
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
DATETIME	numeric	DATETIME22.3
TIMESTAMP	hex	$HEX*w*

The following table shows the default SYBASE data types that SAS/ACCESS assigns to SAS variable formats during output operations when you use the LIBNAME statement.

Table 1.5 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

SAS Variable Format	SYBASE Data Type
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= data set option.

ACCESS Procedure Data Conversions

The following table shows the default SAS variable formats that SAS/ACCESS assigns to SYBASE data types when you use the ACCESS procedure.

Table 1.6 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) $200. (*n* > 200)
VARCHAR(*n*)	character	$*n*. (*n* <= 200) $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
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

The following table shows the default SYBASE data types that SAS/ACCESS assigns to SAS variable formats when you use the DBLOAD procedure.

Table 1.7 PROC DBLOAD: Default SYBASE Data Types for SAS Variable Formats

SAS Variable Format	SYBASE Data Type
$*w*., $CHAR*w*., $VARYING*w*., $HEX*w*.	CHAR(*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.

Data Returned as SAS Binary Data with Default Format $HEX

BINARY
VARBINARY
IMAGE

Data Returned as SAS Character Data

NCHAR
NVARCHAR

Inserting TEXT into SYBASE from SAS

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

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

If the BULKLOAD= option is not used, you 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 that is used in SAS.