CHAPTER

1

ODBC Chapter, First Edition

Introduction

SAS/ACCESS software enables you to access data in a database management system (DBMS) and use that data in your SAS programs. The SAS/ACCESS interface to ODBC connects you to DBMS data by using the SAS LIBNAME statement, the SQL Procedure Pass-Through facility, and the DBLOAD procedure. The SQL procedure is a base SAS procedure that works with SAS/ACCESS software to send and receive data directly between a DBMS and SAS software. You can store Pass-Through code in a PROC SQL view for later use.

The CV2ODBC procedure converts Version 6 view descriptors for AS/400 or Microsoft SQL Server to Version 8 ODBC view descriptors for the SAS/ACCESS interface to ODBC, which runs under the Microsoft Windows and OS/2 platforms. The CV2ODBC procedure is described in detail in .

This chapter accompanies and should be used with *SAS/ACCESS Software for Relational Databases: Reference* (order #57204).*

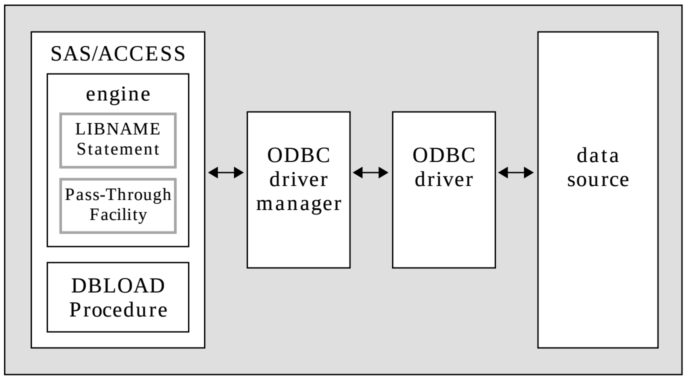
Overview of ODBC

Open database connectivity (ODBC) standards provide a common interface to a variety of databases, including AS/400, dBASE, Microsoft Access, ORACLE, Paradox, and SQL Server databases. Specifically, ODBC standards define application programming interfaces (APIs) that enable an application to access a database if both the application and the database adhere to the specification. ODBC also provides a mechanism to enable dynamic selection of a database that an application is accessing, so end users have the flexibility of selecting databases other than those that are specified by the application developer.

The basic components and features of ODBC include the following:

□ ODBC functionality is provided by three components: the client interface, the ODBC driver manager, and the ODBC driver. SAS Institute provides the SAS/ ACCESS interface to ODBC, which is the client interface. For PC platforms, Microsoft developed the ODBC Administrator, which is used from the Windows Control Panel to perform software administration and maintenance activities. The ODBC driver manager also manages the interaction between the client interface and the ODBC driver. Other software vendors provide the ODBC manager with their ODBC drivers, which process requests for external data. These drivers also either directly manipulate and retrieve the data or they pass the request to a native library for the specific DBMS. The ODBC interface to the SAS System is illustrated in Figure 1.1 on page 2.

Figure 1.1 The ODBC Interface to the SAS System



□ The ODBC administrator defines a *data source* as the data that is used in an application and the operating system and network that are used to access the data. You create a data source by using the ODBC administrator in the Windows Control Panel, selecting an ODBC driver, and providing the information (for example, data source name, userid, password, description, server name) required by the driver to make a connection to the desired data. The driver displays dialog boxes in which you enter this information. In operation, a client application usually requests a connection to a named data source, not just to a specific ODBC driver. In a UNIX environment such as HP-UX, AIX, or Solaris, no ODBC Administrator exists. During an install, the driver creates a generic **`.odbc.ini`** file that can be edited to create your own data source names.

For more information on customizing your SAS application, refer to your vendor-specific documentation.

- ODBC uses SQL syntax for queries and statement execution (or for statements that are executed as commands). However, all databases that support ODBC are not necessarily SQL databases. For example, many databases do not have system tables, and the term *tables* may be used to describe a variety of items, including a file, parts of files, groups of files, typical SQL tables, generated data, or any potential source of data. This distinction is important because although all ODBC data sources respond to a base set of SQL statements such as SELECT, INSERT, UPDATE, DELETE, CREATE, and DROP in their simplest forms, some databases do not support other statements and more complex forms of the SQL statements.
- The ODBC standard allows for various levels of conformance, generally categorized as low, medium, and high. As mentioned previously, the level of SQL syntax that is supported varies. There are also many programming interfaces that might not be supported by some drivers. The SAS/ACCESS Interface to ODBC is designed to work with API calls that conform to the lowest level of ODBC compliance, Level 1. However, the SAS/ACCESS Interface to ODBC does use some Level 2 API calls if they are available.

  However, it is the responsibility of the SAS programmer or end user to ensure that the SQL syntax that is used is supported by the particular driver that is being used. If the ODBC driver supports a higher level of API conformance, some of the advanced features are made available through the PROC SQL CONNECT statement and special queries supported by the SAS/ACCESS Interface to ODBC. For more information, see “Special ODBC Queries” on page 35.
- The ODBC manager and drivers return standard operation states and custom text for any warnings or errors. The state variables and their associated text are available through the SAS system macro variables SYSDBRC and SYSDBMSG.
- The SAS/ACCESS interface supports the ODBC 3.0 specifications that are part of the ODBC drivers that are provided with Microsoft Office 97. There are three types of data source names that can be specified. A User DSN is specific to an individual user and is available only to the user who creates it. A System DSN can be used by anyone who has permission to access the data source. A File DSN can be shared among users even though it is created locally. Since it is file based, it contains all the information that is required to connect to a data source.
- In addition to the information provided in this document, you need to refer to the documentation provided with your ODBC driver. Most ODBC drivers supply a help file that you can access online. In the Windows Control Panel, double click the ODBC icon to start the ODBC Administrator application. Within the ODBC Data Source Administrator, double click the data source name from the User DSN, System DSN, or File DSN tabbed dialog. This will bring up the ODBC driver setup dialog box for this specific ODBC driver. Clicking the Help button will provide information you will need to configure the ODBC data source for this driver.

SAS/ACCESS LIBNAME Statement

This section describes the LIBNAME statement and its options that are specific to ODBC. The LIBNAME statement and options that can be used in most databases are fully described in Chapter 3, "SAS/ACCESS LIBNAME Statement". This section describes the connection options for ODBC and any ODBC-specific LIBNAME options.

LIBNAME Statement: ODBC Specifics

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

Valid: in a DATA or PROC step

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.

SAS/ACCESS-engine-name
is a SAS/ACCESS engine name for your DBMS, in this case, ODBC. 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 in order to connect to a particular database; these options are different for each database. If the SAS/ACCESS engine 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 SAS/ACCESS engine connection options. See your DBMS-specific documentation for details.

SAS/ACCESS-LIBNAME-options
are options that apply to the objects in a DBMS, such as its tables or indexes. For example, the STRINGDATES= option specifies whether to read date and time values as character strings or as numeric date values. Support for many of these options is specific to ODBC.

Some SAS/ACCESS LIBNAME options have the same names as SAS/ACCESS engine data set options. When you specify an option in the LIBNAME statement, it applies to objects in the particular database (which is accessed 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 represents a DBMS table or view), the SAS System uses the value that is specified after the data set name. For more information, see Chapter 3, "SAS/ACCESS LIBNAME Statement".

Details The LIBNAME statement associates a libref with a SAS/ACCESS engine in order to access tables or views in a 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 MYLIB.EMPLOYEES_Q2, MYLIB is a SAS libref that points to a particular DBMS, and EMPLOYEES_Q2 is a DBMS table name. When you specify MYLIB.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.

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

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

USER= on page 5

PASSWORD= on page 5

DATASRC= on page 5

AUTOCOMMIT= on page 6

COMPLETE= on page 6

NOPROMPT = on page 6

PROMPT = on page 6

REQUIRED= on page 6

Note: Not all of these engine connection options are supported by all ODBC drivers. Refer to your vendor-supplied documentation for more information. △

There are multiple ways that you can connect to the DBMS when using the LIBNAME statement. Use only one of the following methods for each connection since they are mutually exclusive:

- specify USER=, PASSWORD=, and DATABASE=, or
- specify COMPLETE=, or
- specify NOPROMPT=, or
- specify PROMPT=, or
- specify REQUIRED=.

USER=<'>*username*<'>
enables you to connect to an ODBC database, such as SQL Server or AS/400, with a user ID that is different from the default ID.
The USER= and PASSWORD= connections are optional in ODBC. If you specify USER=, you must also specify PASSWORD=. If USER= is omitted, your default user ID is used.

Note: If you do not specify the data source name, but you do specify USER= and PASSWORD=, the default data source, default USER, and default PASSWORD are used. The USER and PASSWORD options specified in the connection will not be used. △

USER= can also be specified with the UID= alias.

PASSWORD=<'>*password*<'>
specifies the ODBC password that is associated with your user ID.
The USER= and PASSWORD= connection options are optional in ODBC because users have default user IDs. If you specify USER=, you must specify PASSWORD=.
PASSWORD= can also be specified with the PWD=, PW=, USING=, and PASS= aliases.

DATASRC=<'>*ODBC-data-source*<'>
specifies the ODBC data source or database to which you want to connect. There is no restriction on the length of the name.
DATASRC= is an optional connection option. If you omit it, you connect by using a default environment variable.
DATASRC= can also be specified with the DSN=, DS=, and DATABASE= aliases.

AUTOCOMMIT=YES | NO

indicates whether or not updates are committed immediately after they are submitted.

If AUTOCOMMIT=YES, no rollback is possible.

If AUTOCOMMIT=NO, the SAS/ACCESS engine automatically does the commit when it reaches the end of the file.

The default value for AUTOCOMMIT= under ODBC is NO if the ODBC driver supports transactions and the connection is used for updating. Otherwise, the default value is YES. The default value is always YES when the PROC SQL Pass-Through facility is used.

COMPLETE=<'>*connection-options*<'>

specifies connection options for your data source or database. If you specify enough correct connection options, the SAS/ACCESS engine connects to your data source or database. Otherwise, you are prompted for the connection options with a dialog box that displays the values from the COMPLETE= connection string. You can edit any field before you connect to the data source. You separate multiple options with a semicolon. When a successful connection is made, the complete connect string is returned in the SYSDBMSG macro variable.

COMPLETE= is similar to the PROMPT= option. However, if COMPLETE= attempts to connect and fails, then a dialog box is displayed and you can edit values or enter additional values.

COMPLETE= is optional.

See your driver documentation for more details.

NOPROMPT=<'>*connection-options*<'>

specifies connection options for your data source or database. You separate multiple options with a semicolon. If you specify enough correct connection options, the SAS/ACCESS engine connects to the data source or database. Otherwise, an error is returned and no dialog box is displayed. NOPROMPT= is optional. If connection options are not specified, the default settings are used.

PROMPT=<'> *connection-information*<'>

specifies connection options to the data source.

A dialog box is displayed, using the values from the PROMPT= connection string. You can edit any field before you connect to the data source. When a successful connection is made, the complete connect string is returned in the SYSDBMSG macro variable.

PROMPT= is similar to the COMPLETE= option. However, unlike COMPLETE=, PROMPT= does not attempt to connect to the DBMS first. It displays the dialog box where you can edit or enter additional values.

REQUIRED=<'>*connection-options*<'>

specifies connection options for your data source or database. You separate multiple options with a semicolon.

If you specify enough correct connection options, such as user ID, password, and data source name, the SAS/ACCESS engine connects to the data source or database. Otherwise, a dialog box is displayed to prompt you for the connection options. Options in the dialog box that are not related to the connection are disabled. REQUIRED= only allows you to modify required fields in the dialog box. When a successful connection is made, the complete connect string is returned in the SYSDBMSG macro variable.

REQUIRED= is similar to COMPLETE= because it attempts to connect to the DBMS first. However, if REQUIRED= attempts to connect and fails, then a dialog box is displayed and you can only edit values that are in the required fields.

REQUIRED= is optional.

SAS/ACCESS LIBNAME Options When you specify any of the following options on 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 ODBC supports all of the SAS/ACCESS LIBNAME options listed in Chapter 3, "SAS/ACCESS LIBNAME Statement" . In addition to the supported options, the following LIBNAME options are used only in the interface to ODBC or have ODBC–specific aspects to them:

BCP= on page 7

BCP_ERRORFILE= on page 8

CONNECTION= on page 8

CURSOR_TYPE= on page 8

DBINDEX= on page 8

DEFER= on page 8

DELETE_MULT_ROWS= on page 9

KEYSET_SIZE= on page 9

PRESERVE_COL_NAMES= on page 9

PRESERVE_TAB_NAMES= on page 9

QUALIFIER= on page 9

QUERY_TIMEOUT= on page 9

QUOTE_CHAR= on page 10

READ_ISOLATION_LEVEL= on page 10

READ_LOCK_TYPE= on page 11

ROWSET_SIZE= on page 11

SCHEMA= on page 12

SPOOL= on page 12

STRINGDATES= on page 12

TRACE= on page 12

TRACEFILE= on page 12

UPDATE_ISOLATION_LEVEL= on page 12

UPDATE_LOCK_TYPE = on page 14

UPDATE_MULT_ROWS= on page 14

USE_ODBC_CL= on page 14

BCP=

uses Microsoft's BCP interface to insert data into a Microsoft SQL Server database. The BCP= option is only valid in a LIBNAME statement that connects to Microsoft SQL Server.

Default value: NO.

BCP is Microsoft's bulk copy facility, a high performance method of inserting data into a DBMS table. As SAS sends each row of data to BCP, the data is buffered. After all insertions, the data is committed to the table. If errors occur, they are written to the file that you specify with the BCP_ERRORFILE= option. A generic error is printed in the SAS log.

Note that to use BCP, your installation of Microsoft SQL Server must include the ODBCBCP.DLL, which is currently only supported by Microsoft SQL Server 7.0. Alternatively, you can set the DBCOMMIT= option to commit rows after a specific number of insertions.

BCP_ERRORFILE=

specifies the name of the error file to which all errors are written when BCP=YES. The BCP_ERRORFILE= option is only valid in a LIBNAME statement that connects to Microsoft SQL Server.

Default value: No error file is specified.

If BCP_ERRORFILE= is not specified, errors are not recorded during BCP processing.

CONNECTION=SHAREDREAD | GLOBALREAD | UNIQUE

indicates whether multiple table opens in a DBMS can use the same connection.

Default value: If the data source supports only one active open cursor per connection, the default value is CONNECTION=UNIQUE; otherwise, the default value is CONNECTION=SHAREDREAD.

You may change the value of this option, which is fully described in Chapter 3, "SAS/ACCESS LIBNAME Statement".

CURSOR_TYPE=DYNAMIC | FORWARD_ONLY | KEYSET_DRIVEN | STATIC

specifies the cursor type for read-only and updatable cursors. Not all drivers support all cursor types. An error is returned if the specified cursor type is not supported.

By default, CURSOR_TYPE=DYNAMIC, but the driver is allowed to modify the default without an error.

If CURSOR_TYPE=DYNAMIC, then the cursor reflects all of the changes that are made to the rows in a result set as you scroll around the cursor. The data values and the membership of rows in the cursor can change dynamically on each fetch.

If CURSOR_TYPE=FORWARD_ONLY, then the cursor behaves like a DYNAMIC cursor except that it only supports fetching the rows sequentially.

If CURSOR_TYPE=KEYSET_DRIVEN, then the cursor determines which rows belong to the result set when the cursor is opened. However, changes that are made to these rows will be reflected as you scroll around the cursor.

If CURSOR_TYPE=STATIC, then the cursor builds the complete result set when the cursor is opened. No changes that are made to the rows in the result set after the cursor is opened will be reflected in the cursor. Static cursors are read-only.

CURSOR_TYPE= can also be specified with the CURSOR= alias.

See also: KEYSET_SIZE= on page 9.

DBINDEX=YES | NO

indicates whether or not SAS calls ODBC to find all indexes on the specified table.

Default value: YES

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

DEFER=NO | YES

determines when the connection to the DBMS occurs.

Default value: NO

If DEFER=YES, the connection to the DBMS occurs when a table in the DBMS is opened. If DEFER=NO, the connection to the DBMS occurs when the libref is assigned by a LIBNAME statement. The DEFER= option is ignored when CONNECTION=UNIQUE because a connection is performed for every open.

When setting DEFER=YES in the SAS/ACCESS Interface to ODBC, you must also set the PRESERVE_TAB_NAMES= and PRESERVE_COL_NAMES= options to their desired values. Normally, SAS queries the data source to default these values correctly during LIBNAME assignment, but setting DEFER=YES postpones the connection. Because these values must be set at the time of LIBNAME assignment, you must assign them explicitly when you set DEFER=YES.

DELETE_MULT_ROWS=YES | NO

indicates whether or not the ODBC driver can delete multiple rows from the DBMS table when the ODBC driver emulates the DELETE ... WHERE CURRENT OF CURSOR statement. Some drivers may delete more than one row even though only the current row was requested for deletion. This may produce unexpected results.

Default value: NO.

KEYSET_SIZE=*number-of-rows*

specifies the number of rows that are keyset driven.

Default value: 0

Alias: KEYSET=

This option is valid only when CURSOR_TYPE=KEYSET_DRIVEN. See CURSOR_TYPE= on page 8 for more information on KEYSET_DRIVEN cursors.

Valid values for KEYSET_SIZE= are 0 through the number of rows in the cursor. If KEYSET_SIZE=0, then the entire cursor is keyset driven. If a value greater than 0 is specified for KEYSET_SIZE=, then the value chosen indicates the number of rows within the cursor that will behave as a keyset driven cursor. When you scroll beyond the bounds that are specified by KEYSET_SIZE=, then the cursor becomes dynamic and new rows may be included in the cursor. This becomes the new keyset and the cursor behaves as a keyset driven cursor again. Whenever the value that is specified is between 1 and the number of rows in the cursor, the cursor is considered to be a mixed cursor since part of the cursor behaves as a keyset driven cursor and part of the cursor behaves as a dynamic cursor.

PRESERVE_COL_NAMES=YES | NO

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

Default value: YES for Microsoft Access, Microsoft Excel, and Microsoft SQL Server; NO for all others

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

PRESERVE_TAB_NAMES=YES | NO

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

Default value: YES for Microsoft Access, Microsoft Excel, and Microsoft SQL Server; NO for all others

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

QUALIFIER=*qualifier-name*

enables you to read database objects, such as tables and views, using the specified qualifier.

QUALIFIER= is optional. If it is omitted, you use the default DBMS qualifier name, if any. QUALIFIER= can be used for any DBMS that allows three part identifier names such as *qualifier.schema.object*. In the following example libname statement, the QUALIFIER= option causes any reference in SAS to **mydblib.employee** to be interpreted by ODBC as **mydept.scott.employee**.

```
libname mydblib odbc schema=scott qualifier=mydept;
```

QUERY_TIMEOUT=*number-of-seconds*

specifies the number of seconds of inactivity to wait before canceling a query.

Default value: 0

The default value of 0 indicates that there is no time limit for a query. This option is useful when you are testing a query, you suspect that a query might contain an endless loop, or you access a table that may be locked by other users.

QUERY_TIMEOUT= can also be specified with the TIMEOUT= alias.

QUOTE_CHAR=*character-value*

specifies which quotation mark character to use when delimiting identifiers. This option is mainly for the ODBC Interface to Sybase and should be used in conjunction with the DBCONINIT and DBLIBINIT LIBNAME options.

Default value: none

QUOTE_CHAR= overrides the ODBC default since some drivers return a blank for the identifier delimiter even though the DBMS uses a quote (for example, Intersolv to Sybase).

READ_ISOLATION_LEVEL= S | RR | RC | RU | V

defines the degree of isolation of the current application process from other concurrently running application processes. The isolation levels are as follows and are thoroughly described below:

S = Serializable

RR = Repeatable Read

RC = Read Committed

RU = Read Uncommitted

V = Versioning

Default value: RC

The degree of isolation identifies

- the degree with which rows that are read and updated by the current application are available to other concurrently executing applications
- the degree with which update activity of other concurrently executing application processes can affect the current application.

READ_ISOLATION_LEVEL= is ignored if READ_LOCK_TYPE= is not set to ROW.

The ODBC driver manager supports five isolation levels. The isolation levels are defined in terms of several possible occurrences:

- Dirty read — A transaction that exhibits this phenomenon has very minimal isolation from concurrent transactions. In fact, it will be able to see changes that are made by those concurrent transactions even before they commit.

  For example, suppose that transaction T1 performs an update on a row, transaction T2 then retrieves that row, and transaction T1 then terminates with rollback. Transaction T2 has then seen a row that no longer exists.

- Nonrepeatable read — If a transaction exhibits this phenomenon, it is possible that it may read a row once and, if it attempts to read that row again later in the course of the same transaction, the row might have been changed or even deleted by another concurrent transaction. Therefore, the read is not (necessarily) repeatable.

  For example, suppose that transaction T1 retrieves a row, transaction T2 then updates that row, and transaction T1 then retrieves the same row again. Transaction T1 has now retrieved the same row twice but has seen two different values for it.

- Phantom reads — When a transaction exhibits this phenomemon, a set of rows that it reads once might be a different set of rows if the transaction attempts to read them again.

  For example, suppose that transaction T1 retrieves the set of all rows that satisfy some condition. Suppose that transaction T2 then inserts a new row that satisfies that same condition. If transaction T1 now repeats its retrieval request, it will see a row that did not previously exist, a phantom.

The isolation levels for READ_ISOLATION_LEVEL= include the following:

- Serializable (S)
  - does not allow dirty reads
  - does not allow nonrepeatable reads
  - does not allow phantom reads
- Repeatable Read (RR)
  - does not allow dirty reads
  - does not allow nonrepeatable reads
  - allows phantom reads
- Read Committed (RC)
  - does not allow dirty reads
  - allows nonrepeatable reads
  - allows phantom reads

  This is the default value for ODBC.
- Read Uncommitted (RU)
  - allows dirty reads
  - allows nonrepeatable reads
  - allows phantom reads
- Versioning (V)
  - does not allow dirty reads
  - does not allow nonrepeatable reads
  - does not allow phantom reads

  These transactions are serializable, but higher concurrency is possible than with the Serializable isolation level. Typically a nonlocking protocol is used.

READ_ISOLATION_LEVEL= can also be specified with the following alias: RIL=.

See Also: UPDATE_ISOLATION_LEVEL= on page 12.

READ_LOCK_TYPE=ROW

specifies that a row or set of rows will be locked for ODBC tables during a READ operation. The READ_ISOLATION_LEVEL= option is used to determine which rows will be locked.

Default value: ROW

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

See also: UPDATE_LOCK_TYPE= on page 14.

ROWSET_SIZE=*number-of-rows*

specifies the number of rows to use when reading data from the DBMS.

Default value: 0

When ROWSET_SIZE=0, no internal SAS buffering is performed. Setting ROWSET_SIZE=0 causes the SQLFetch API call to be used.

When ROWSET_SIZE=1, only one row is retrieved at a time. The higher the value for ROWSET_SIZE=, the more rows the DB2 engine retrieves in one fetch operation. This option reduces the amount of I/O that is used and can help improve performance. However, because SAS software stores the rows in memory, higher values for ROWSET_SIZE= use more memory. In addition, if too many rows are selected at once, then the rows that are returned to the SAS application might be out of date. For example, if someone else modified the rows, you would not see the changes. Setting ROWSET_SIZE=1 or greater causes the SQLExtendedFetch API call to be used.

ROWSET_SIZE= can also be specified with the ROWSET= alias.

SCHEMA=*schema-name*

enables you to read database objects, such as tables and views, in the specified schema.

SCHEMA= is optional. If it is omitted, you connect to the default schema. In the following example LIBNAME statement, the SCHEMA= option causes any reference in SAS to **mydblib.employee** to be interpreted by ODBC as **scott.employee**.

```
libname mydblib odbc schema=scott;
```

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

SPOOL=YES | NO

specifies whether or not SAS creates a utility spool file during read operations that are performed with the specified LIBNAME.

Default value: YES

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

STRINGDATES=YES | NO

specifies whether to read date and time values from the database as character strings or as numeric date values.

Default value: NO

If STRINGDATES=YES, then the SAS application reads date-time values as character strings.

If STRINGDATES=NO, then the SAS application reads date-time values as numeric date values.

STRINGDATES= can also be specified with the STRDATES= alias.

STRINGDATES= is used for Version 6 compatibility.

TRACE=YES | NO

specifies whether or not to turn on tracing information that is used in debugging.

Default value: NO

If TRACE=YES, tracing is turned on, and the ODBC driver manager writes each function call to the trace file that is specified by TRACEFILE=.

If TRACE=NO, tracing is not turned on.

See also: TRACEFILE= on page 12.

TRACEFILE=*filename*

specifies the filename to which the ODBC driver manager writes trace information.

Default value: none

TRACEFILE= is used only when TRACE=YES.

See also: TRACE= on page 12.

UPDATE_ISOLATION_LEVEL=RC | S | RR | V

defines the degree of isolation of the current application process from other concurrently running application processes. The isolation levels are as follows and are thoroughly described below:

RC = Read Committed

S = Serializable

RR = Repeatable Read

V = Versioning

Default value: RC

The degree of isolation identifies

- the degree with which rows that are read and updated by the current application are available to other concurrently executing applications
- the degree with which update activity of other concurrently executing application processes can affect the current application.

UPDATE_ISOLATION_LEVEL= is ignored if UPDATE_LOCK_TYPE= is not set to ROW.

The ODBC driver manager supports four isolation levels. The isolation levels are defined in terms of several possible occurrences:

- Dirty read — A transaction that exhibits this phenomenon has very minimal isolation from concurrent transactions. In fact, it will be able to see changes that are made by those concurrent transactions even before they commit.

  For example, suppose that transaction T1 performs an update on a row, transaction T2 then retrieves that row, and transaction T1 then terminates with rollback. Transaction T2 has then seen a row that no longer exists.
- Nonrepeatable read — If a transaction exhibits this phenomenon, it is possible that it may read a row once and, if it attempts to read that row again later in the course of the same transaction, the row might have been changed or even deleted by another concurrent transaction. Therefore, the read is not (necessarily) repeatable.

  For example, suppose that transaction T1 retrieves a row, transaction T2 then updates that row, and transaction T1 then retrieves the same row again. Transaction T1 has now retrieved the same row twice but has seen two different values for it.
- Phantom reads — When a transaction exhibits this phenomemon, a set of rows that it reads once might be a different set of rows if the transaction attempts to read them again.

  For example, suppose that transaction T1 retrieves the set of all rows that satisfy some condition. Suppose that transaction T2 then inserts a new row that satisfies that same condition. If transaction T1 now repeats its retrieval request, it will see a row that did not previously exist, a phantom.

The isolation levels for UPDATE_ISOLATION_LEVEL= include the following:

- Serializable (S)
  - does not allow dirty reads
  - does not allow nonrepeatable reads
  - does not allow phantom reads
- Repeatable Read (RR)
  - does not allow dirty reads
  - does not allow nonrepeatable reads
  - allows phantom reads
- Read Committed (RC)
  - does not allow dirty reads
  - allows nonrepeatable reads
  - allows phantom reads

This is the default value for ODBC.

- Versioning (V)
  - does not allow dirty reads
  - does not allow nonrepeatable reads
  - does not allow phantom reads

  These transactions are serializable but higher concurrency is possible than with the Serializable isolation level. Typically a nonlocking protocol is used.

UPDATE_ISOLATION_LEVEL= can also be specified with the UIL= alias.
See Also: READ_ISOLATION_LEVEL= on page 10.

UPDATE_LOCK_TYPE =ROW | NOLOCK
: specifies how a ODBC table is locked during an UPDATE operation. The value ROW specifies that a row or set of rows will be locked. The UPDATE_ISOLATION_LEVEL= option is used to determine which rows will be locked.
Default value: ROW
The value NOLOCK specifies that there is no locking on the table when it is read for update. Although NOLOCK is allowed, not all ODBC drivers support this option. In this case, an error will be printed.
For a full description of this option, refer to Chapter 3, "SAS/ACCESS LIBNAME Statement".
See also: READ_LOCK_TYPE= on page 11.

UPDATE_MULT_ROWS=YES | NO
: indicates whether or not the ODBC driver can update multiple rows from the DBMS table when the ODBC driver emulates the UPDATE ... WHERE CURRENT OF CURSOR statement. Some drivers may update more than one row even though only the current row was requested for update. This may produce unexpected results.
Default value: NO
This option allows SAS to continue if multiple rows were updated.

USE_ODBC_CL= YES | NO
: indicates whether or not the Driver Manager uses the ODBC Cursor Library.
Default value: NO
If USE_ODBC_CL=YES, the Driver Manager uses the ODBC Cursor Library. The ODBC Cursor Library supports block scrollable cursors and positioned update and delete statements. For more information on the ODBC Cursor Library, see your vendor-specific documentation.
If USE_ODBC_CL=NO, the Driver Manager uses the scrolling capabilities of the driver.

Examples

Example 1: Specifying a LIBNAME Statement to Access ODBC Data on AS/400 In this example, the libref MYLIB uses the ODBC engine to connect to an AS/400 database. The SAS/ACCESS engine connection options are UID=, PWD=, and DSN=.

```
libname mydblib odbc dsn=as400 uid=testuser
   pwd=testpass;

proc print data=mydblib.customers;
   where state='CA';
run;
```

Example 2: Specifying a LIBNAME Statement to Access ODBC Data on SQL Server In this example, the libref MYDBLIB uses the ODBC engine to connect to a Microsoft SQL Server database. The SAS/ACCESS engine connection option is NOPROMPT=.

```
libname mydblib odbc
   noprompt="uid=testuser;pwd=testpass;dsn=sqlservr;"
   stringdates=yes;

proc print data=mydblib.customers;
   where state='CA';
run;
```

Data Set Options: ODBC Specifics

This section describes SAS/ACCESS data set options that can be applied to SAS data sets that access data in tables and views using ODBC. All of the data set options described in Chapter 4, "SAS/ACCESS Data Set Options" apply to ODBC. In some cases, an option has an ODBC-specific detail or an entire option is ODBC-specific. If so, these options are described in section.

When specified in a DATA step or SAS procedure, the following data set options can be used on a SAS data set that accesses data in a DBMS object, such as a table or view. A data set option applies only to the SAS data set on which it is specified.

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

“CURSOR_TYPE=” on page 15

“DBINDEX=” on page 16

“DBNULL=” on page 16

“DBSASTYPE=” on page 17

“DBTYPE=” on page 18

“KEYSET_SIZE=” on page 18

“QUALIFIER=” on page 19

“QUERY_TIMEOUT=” on page 19

“READ_ISOLATION_LEVEL=” on page 19

“READ_LOCK_TYPE=” on page 21

“ROWSET_SIZE=” on page 21

“SASDATEFMT=” on page 22

“SCHEMA=” on page 22

“UPDATE_ISOLATION_LEVEL=” on page 23

“UPDATE_LOCK_TYPE=” on page 25

CURSOR_TYPE=

Specifies the cursor type for read only and updatable cursors

Default value: DYNAMIC

Alias: CURSOR=

Syntax

CURSOR_TYPE=DYNAMIC | FORWARD_ONLY | KEYSET_DRIVEN | STATIC

Details Not all drivers support all cursor types. An error is returned if the specified cursor type is not supported.

By default, CURSOR_TYPE=DYNAMIC, but the driver is allowed to modify the default without an error.

If CURSOR_TYPE=DYNAMIC, then the cursor reflects all of the changes that are made to the rows in a result set as you scroll around the cursor. The data values and the membership of rows in the cursor can change dynamically on each fetch.

If CURSOR_TYPE=FORWARD_ONLY, then the cursor behaves like a DYNAMIC cursor except that it only supports fetching the rows sequentially.

If CURSOR_TYPE=KEYSET_DRIVEN, then the cursor determines which rows belong to the result set when the cursor is opened. However, changes that are made to these rows will be reflected as you scroll around the cursor.

If CURSOR_TYPE=STATIC, then the cursor builds the complete result set when the cursor is opened. No changes made to the rows in the result set after the cursor is opened will be reflected in the cursor. Static cursors are read only.

See Also

KEYSET_SIZE=

DBINDEX=

Indicates whether or not SAS calls the DBMS to find index(es) on the specified table

Default value: NO

Syntax

DBINDEX=YES | NO | <'>*index-name*<'>

Details

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

See Also

DBKEY=

DBNULL=

Indicates whether or not NULL is a valid value for the specified variables or columns

Default value: YES

Syntax

DBNULL=(*column-name-1*=YES | NO *column-name-n*=YES | NO)

Details

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

DBSASTYPE=

Specifies data type(s) to override the default SAS data type(s) during input processing of data through ODBC.

Default value: Varies by data type.

Syntax

DBSASTYPE=*(<column-name-1=<'>SAS-data-type<'>>*
<...<column-name-n=<'SAS-data-type<'>>>)

column-name
specifies a DBMS column name.

SAS-data-type
specifies one of the following SAS data types:

- CHAR(n)
- NUMERIC
- DATETIME
- DATE
- TIME

Details This option is valid only when you read data into SAS thorough ODBC.

By default, the SAS/ACCESS Interface to ODBC converts each ODBC data type to a predetermined SAS data type when processing data through ODBC. When you need a different data type, you can use DBSASTYPE= to override the default data type chosen by the SAS/ACCESS engine. SAS forces the ODBC driver to perform the data conversions. Some conversions might not be supported; if a conversion is not supported, SAS prints an error to the log.

In the following example, DBSASTYPE= specifies a data type to use for the column MYCOLUMN when printing the DBMS data in SAS. If the data in this DBMS column is stored in a format that SAS does not support, such as SQL_DOUBLE(20), this enables SAS to print the values.

```
proc print data=mylib.mytable
   (DBSASTYPE=(mycolumn='CHAR(20)'));
run;
```

See “ODBC Data Types” on page 40 for more details on the default data types for ODBC.

DBTYPE=

Specifies data types(s) to override the default ODBC data type(s) when SAS outputs data to DBMS tables through ODBC.

Default value: The default type for SAS character variables is SQL_VARCHAR(size) which corresponds to the DBMS data type for variable length character data. Size is derived from the length of the SAS variable. The default for SAS numeric variables is SQL_DOUBLE, which corresponds to the DBMS data type for a double numeric value.

Syntax

DBTYPE=*(column-name-1=DBMS-type <...> <column-name-n=DBMS-type>)*

Details

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

KEYSET_SIZE=

Specifies the number of rows in the cursor that are keyset driven.

Default value: 0

Alias: KEYSET=

Syntax

KEYSET_SIZE=*number-of-rows*

Details

This option is valid only when CURSOR_TYPE=KEYSET_DRIVEN. See “CURSOR_TYPE=” on page 15 for more information on KEYSET_DRIVEN cursors.

Valid values for KEYSET_SIZE= are 0 through the number of rows in the cursor. If KEYSET_SIZE=0, then the entire cursor is keyset driven. If a value greater than 0 is specified for KEYSET_SIZE=, then the value chosen indicates the number of rows within the cursor that will behave as a keyset driven cursor. When you scroll beyond the bounds that are specified by KEYSET_SIZE=, then the cursor becomes dynamic and new rows may be included in the cursor. This becomes the new keyset and the cursor behaves as a keyset driven cursor again. Whenever the value specified is between 1 and the number of rows in the cursor, the cursor is considered to be a mixed cursor since part of the cursor behaves as a keyset driven cursor and part of the cursor behaves as a dynamic cursor.

By default, KEYSET_SIZE=0.

See Also

CURSOR_TYPE=

QUALIFIER=

Specifies the qualifier to use when reading database objects such as tables and views

Default value: none

Syntax

QUALIFIER=*qualifier-name*

Details QUALIFIER= is optional. If it is omitted, you use the default DBMS qualifier name, if any. QUALIFIER= can be used for any DBMS that allows three part identifier names such as *qualifier.schema.object*.

QUERY_TIMEOUT=

Specifies the number of seconds of inactivity to wait before canceling a query

Default value: 0

Alias: TIMEOUT=

Syntax

QUERY_TIMEOUT=*number-of-seconds*

Details

The default value of 0 indicates that there is no time limit for a query. This option is useful when you are testing a query, you suspect that a query might contain an endless loop, or the data is locked by another user.

READ_ISOLATION_LEVEL=

Defines the degree of isolation of the current application process from other concurrently running application processes.

Default value: RC

Alias: RIL=

Syntax

READ_ISOLATION_LEVEL=S | RR | RC | RU | V

S = Serializable
RR = Repeatable Read
RC = Read Committed
RU = Read Uncommitted
V = Versioning

Details

The degree of isolation identifies

- the degree with which rows that are read and updated by the current application are available to other concurrently executing applications
- the degree with which update activity of other concurrently executing application processes can affect the current application.

READ_ISOLATION_LEVEL= is ignored if READ_LOCK_TYPE= is not set to ROW.

The ODBC driver manager supports five isolation levels. The isolation levels are defined in terms of several possible occurrences:

- Dirty read — A transaction that exhibits this phenomenon has very minimal isolation from concurrent transactions. In fact, it will be able to see changes made by those concurrent transactions even before they commit.

  For example, suppose that transaction T1 performs an update on a row, transaction T2 then retrieves that row, and transaction T1 then terminates with rollback. Transaction T2 has then seen a row that no longer exists.
- Nonrepeatable read — If a transaction exhibits this phenomenon, it is possible that it may read a row once and, if it attempts to read that row again later in the course of the same transaction, the row might have been changed or even deleted by another concurrent transaction. Therefore, the read is not (necessarily) repeatable.

  For example, suppose that transaction T1 retrieves a row, transaction T2 then updates that row, and transaction T1 then retrieves the same row again. Transaction T1 has now retrieved the same row twice but has seen two different values for it.
- Phantom reads — When a transaction exhibits this phenomemon, a set of rows that it reads once might be a different set of rows if the transaction attempts to read them again.

  For example, suppose that transaction T1 retrieves the set of all rows that satisfy some condition. Suppose that transaction T2 then inserts a new row that satisfies that same condition. If transaction T1 now repeats its retrieval request, it will see a row that did not previously exist, a phantom.

The isolation levels for READ_ISOLATION_LEVEL= include the following:

- Serializable (S)
  - does not allow dirty reads

    - does not allow nonrepeatable reads
    - does not allow phantom reads
- Repeatable Read (RR)
    - does not allow dirty reads
    - does not allow nonrepeatable reads
    - allows phantom reads
- Read Committed (RC)
    - does not allow dirty reads
    - allows nonrepeatable reads
    - allows phantom reads
- Read Uncommitted (RU)
    - allows dirty reads
    - allows nonrepeatable reads
    - allows phantom reads
- Versioning (V)
    - does not allow dirty reads
    - does not allow nonrepeatable reads
    - does not allow phantom reads

READ_LOCK_TYPE=

Specifies how a table is locked during a READ operation

Default value: ROW

Syntax

READ_LOCK_TYPE=ROW

Details

The value ROW specifies that a row or set of rows will be locked. The READ_ISOLATION_LEVEL= option is used to determine which rows will be locked.

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

ROWSET_SIZE=

Specifies the number of rows to use when reading data from the DBMS

Default value: 0

Alias: ROWSET=

Syntax

ROWSET_SIZE=*number-of-rows*

Details

By default, ROWSET_SIZE=0, so that no internal SAS buffering is performed. Setting ROWSET_SIZE=0 causes the SQLFetch API call to be used.

When ROWSET_SIZE=1, only one row is retrieved at a time. The higher the value for ROWSET_SIZE=, the more rows the DB2 engine retrieves in one fetch operation. This option reduces the amount of I/O that is used and can help improve performance. However, because SAS software stores the rows in memory, higher values for ROWSET_SIZE= use more memory. In addition, if too many rows are selected at once, then the rows that are returned to the SAS application might be out of date. For example, if someone else modified the rows, you would not see the changes. Setting ROWSET_SIZE=1 or greater causes the SQLExtendedFetch API call to be used.

SASDATEFMT=

Changes the SAS date format of a DBMS column

Default value: None

Syntax

SASDATEFMT=*(DBMS-date-col="SAS-date-format" ...)*

Details

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

Example: Using SASDATEFMT to convert a date column to a SAS date format

In this example, SAS changes the format of the HIRED column to the SAS DATE9. format for reading the data in SAS.

```
proc print data=mydblib.payroll
   (sasdatefmt=(hired='DATE9.'));
run;
```

SCHEMA=

Enables you to read database objects, such as tables and views, in the specified schema

Default value: None

Alias: OWNER=

Syntax

SCHEMA=*schema-name*

Details

A schema is a logical classification of objects in a database. You must have read privilege to the schema that is specified for this option to work. The SCHEMA= option is optional. If it is omitted, you connect to the default schema.

Example: Accessing a table using SCHEMA=

In this example, SAS sends any reference to **mydblib.employees** as **dbitest.employees**.

```
libname mydblib odbc user=testuser pwd=testpass
dsn=oracle;
proc print data=mydblib.employees (schema=dbitest);
run;
```

UPDATE_ISOLATION_LEVEL=

Defines the degree of isolation of the current application process from other concurrently running application processes

Default value: RC

Alias: UIL=

Syntax

UPDATE_ISOLATION_LEVEL=S | RC | RR | V

S = Serializable

RC = Read Committed

RR = Repeatable Read

V = Versioning

Details

The degree of isolation identifies

- the degree with which rows that are read and updated by the current application are available to other concurrently executing applications
- the degree with which update activity of other concurrently executing application processes can affect the current application.

UPDATE_ISOLATION_LEVEL= is ignored if UPDATE_LOCK_TYPE= is not set to ROW.

The ODBC driver manager supports four isolation levels. The isolation levels are defined in terms of several possible occurrences:

- Dirty read — A transaction that exhibits this phenomenon has very minimal isolation from concurrent transactions. In fact, it will be able to see changes that are made by those concurrent transactions even before they commit.

  For example, suppose that transaction T1 performs an update on a row, transaction T2 then retrieves that row, and transaction T1 then terminates with rollback. Transaction T2 has then seen a row that no longer exists.
- Nonrepeatable read — If a transaction exhibits this phenomenon, it is possible that it may read a row once and, if it attempts to read that row again later in the course of the same transaction, the row might have been changed or even deleted by another concurrent transaction. Therefore, the read is not (necessarily) repeatable.

  For example, suppose that transaction T1 retrieves a row, transaction T2 then updates that row, and transaction T1 then retrieves the same row again. Transaction T1 has now retrieved the same row twice but has seen two different values for it.
- Phantom reads — When a transaction exhibits this phenomemon, a set of rows that it reads once might be a different set of rows if the transaction attempts to read them again.

  For example, suppose that transaction T1 retrieves the set of all rows that satisfy some condition. Suppose that transaction T2 then inserts a new row that satisfies that same condition. If transaction T1 now repeats its retrieval request, it will see a row that did not previously exist, a phantom.

The isolation levels for UPDATE_ISOLATION_LEVEL= include the following:

- Serializable (S)
  - does not allow dirty reads
  - does not allow nonrepeatable reads
  - does not allow phantom reads
- Repeatable Read (RR)
  - does not allow dirty reads
  - does not allow nonrepeatable reads
  - allows phantom reads
- Read Committed (RC)
  - does not allow dirty reads
  - allows nonrepeatable reads
  - allows phantom reads
- Versioning (V)
  - does not allow dirty reads
  - does not allow nonrepeatable reads
  - does not allow phantom reads

  These transactions are serializable but higher concurrency is possible than with the Serializable isolation level. Typically, a nonlocking protocol is used.

UPDATE_LOCK_TYPE=

Specifies how an ODBC table is locked during an UPDATE operation

Default value: ROW

Syntax

UPDATE_LOCK_TYPE=ROW | NOLOCK

Details

The value ROW specifies that a row or set of rows will be locked. The UPDATE_ISOLATION_LEVEL= option is used to determine which rows will be locked.

The value NOLOCK specifies that there is no locking on the table when it is read for update. Although NOLOCK is allowed, not all ODBC drivers support this option. In this case, an error will be printed.

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

See Also

READ_LOCK_TYPE=

DBLOAD Procedure: ODBC Specifics

This section describes the statements that you use in the SAS/ACCESS interface to ODBC.

DBLOAD Procedure Statements for ODBC

To create and load an ODBC table, the SAS/ACCESS interface to ODBC uses the following statements in batch mode.

PROC DBLOAD DBMS=ODBC <DATA=<libref.>*SAS-data-set*>;

DSN | DATABASE | IN=<'>*database-name*<'>;

UID | USER=<'>*username*<'>;

PWD | PASSWORD | PW | PASS | USING=
<'>*password*<'>;

TABLE=<*authorization-id.*>*table-name*;

COMMIT=*commit-frequency*;

DELETE*variable-identifier-1*<*...variable-identifier-n*;>

ERRLIMIT=*error-limit*;

LABEL;

LIMIT=*load-limit*;

LIST<ALL | COLUMN | *variable-identifier*>;

NULLS *variable-identifier-1* = Y | N | D<*...variable-identifier-n= Y | N*>;

QUIT | EXIT;

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

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

SQL *DBMS-specific SQL-statement*;

TYPE *variable-identifier-1='column-type-1'*
<*...variable-identifier-n = 'column-type-n'*>;

WHERE*SAS-where-expression*;

LOAD;

RUN;

DSN | DATABASE | IN= <'>*database-name*<'>;

specifies the name of the database in which you want to store the new ODBC table. *Database-name* is limited to eight characters.

The database that you specify must already exist. If the database name contains the following special characters (_,$,@,#), you must enclose it in quotes. However, the ODBC standard recommends against using special characters in database names.

USER | UID= <'>*username* <'>;

enables you to connect to an ODBC database, such as SQL Server or AS/400, with a user ID that is different from the default ID.

The USER= and PASSWORD= statements are optional in ODBC. If you specify USER=, you must also specify PASSWORD=. If USER= is omitted, your default user ID is used.

PWD | PASSWORD | PW | PASS | USING=<'>*password*<'>;

specifies the ODBC password that is associated with your user ID.

The USER= and PASSWORD= statements are optional in ODBC because users have default user IDs. If you specify USER=, you must specify PASSWORD=.

Refer to Chapter 10, "DBLOAD Procedure Reference" for more information.

DBLOAD Procedure Examples

The following example creates a new ODBC table, TESTUSER.EXCHANGE, from the DLIB.RATEOFEX data file. You must be granted the appropriate privileges in order to create new ODBC tables or views.

```
proc dbload dbms=odbc data=dlib.rateofex;
   dsn=sample; user=testuser; password=testpass;
   table=exchange;
   rename fgnindol=fgnindollars
          4=dollarsinfgn;
   nulls updated=n fgnindollars=n
         dollarsinfgn=n country=n;
   load;
run;
```

The next example only sends an ODBC SQL GRANT statement to the SAMPLE database and does not create a new table. Therefore, the TABLE= and LOAD statements are omitted.

```
proc dbload dbms=odbc;
   user=testuser;
```

```
      password=testpass;
      dsn=sample;
      sql grant select on testuser.exchange
         to dbitest;
   run;
```

SQL Procedure Pass-Through Facility: ODBC Specifics

The SQL Procedure Pass-Through facility consists of three PROC SQL statements and one component. For details about the ODBC-specific information, see the CONNECT statement“CONNECT Statement” on page 27 and CONNECTION TO“CONNECTION TO Component” on page 30 component. For a complete description of the SQL Procedure Pass-Through facility, see Chapter 6, "SQL Procedure's Interaction with SAS/ACCESS Software" .

CONNECT Statement

Establishes a connection with the DBMS

Syntax

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

Arguments

You use the following arguments with the CONNECT statement:

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

(*ODBC-connection-arguments*)
specifies the DBMS-specific arguments that are needed by PROC SQL in order to connect to the DBMS. These arguments must be enclosed in parentheses. For some databases, these arguments have default values and therefore are optional. The arguments for ODBC are described in the following sections.

ODBC Connection Arguments PROC SQL supports multiple connections to ODBC. If you use multiple simultaneous connections, you must use the *alias* argument to identify the different connections. If you do not specify an alias, the default alias, **odbc**, is used. The functionality of multiple connections to the same ODBC data source may be limited by the particular data source's driver.

The CONNECT statement is required when connecting to ODBC data sources by way of the SQL Pass-Through Facility.

The following list describes the arguments that are used for ODBC in the CONNECT statement. You can use the arguments DSN=, UID=, and PWD= to connect to most data sources. Use the PROMPT, NOPROMPT, COMPLETE, REQUIRED, or AUTOCOMMIT arguments to provide additional information or to select and connect to the data source.

The arguments for the DBMS connection information arguments can be quoted by using either single or double quotes. Some values may include embedded spaces, semicolons, or quotes and, therefore, must be quoted.

Note: Not all of these engine connection options are supported by all ODBC drivers. Refer to your vendor-supplied documentation for more information. △

There are multiple ways that you can connect to the DBMS when using the CONNECT statement. Use only one of the following methods for each connection since they are mutually exclusive:

- □ specify USER=, PASSWORD=, and DATABASE=, or
- □ specify COMPLETE=, or
- □ specify NOPROMPT=, or
- □ specify PROMPT=, or
- □ specify REQUIRED=.

AUTOCOMMIT=YES | NO

indicates whether or not updates are committed immediately after they are submitted.

If AUTOCOMMIT=YES, no rollback is possible.

If AUTOCOMMIT=NO, the SAS/ACCESS engine automatically does the commit when it reaches the end of the file.

The default value for AUTOCOMMIT= under ODBC is NO if the ODBC driver supports transactions and the connection is used for updating. Otherwise, the default value is YES. The default value is always YES when the PROC SQL Pass-Through facility is used.

DSN | DS | DATASRC | DATABASE=<'>*data-source-name*<'>

specifies the ODBC data source to which you want to connect. For PC platforms, data sources must be configured by using the ODBC icon in the Windows Control Panel. For UNIX platforms, data sources must be configured by modifying the **`.odbc.ini`** file. DSN= indicates that the connection is attempted using the ODBC SQLConnect API, which requires a data source name. Optionally, a user ID and password (described below) can be used in conjunction with DSN=. This API is guaranteed to be present in all drivers.

Specify either DSN= or one (and only one) of the following arguments: PROMPT, NOPROMPT, COMPLETE, or REQUIRED. These arguments are all mutually exclusive of each other.

UID | USER=<'>*DBMS-user-id*<'>

specifies the DBMS user ID. The UID= argument can be used only in conjunction with the DSN= argument. Not all ODBC drivers accept user IDs. This argument is optional.

PWD | PASSWORD | PW | PASS | USING=<'>*DBMS-password*<'>

specifies the DBMS password. The PWD= argument can be used only in conjunction with the DSN= argument. Not all ODBC drivers accept passwords. This argument is optional.

COMPLETE=<'>*connection-options*<'>

specifies connection options for your data source or database. If you specify enough correct connection options, the SAS/ACCESS engine connects to your data source or database. Otherwise, you are prompted for the connection options with a dialog box. Separate multiple options with a semicolon. When a successful connection is made, the complete connect string is returned in the SYSDBMSG and SQLXMSG macro variables.

COMPLETE= is similar to the PROMPT= option. However, if COMPLETE= attempts to connect and fails, then a dialog box is displayed and you can edit values or enter additional values.

COMPLETE= is optional.

See your driver documentation for more details.

NOPROMPT=<'>*connection-options*<'>

specifies connection options for your data source or database. You separate multiple options with a semicolon. If you specify enough correct connection options, the SAS/ACCESS engine connects to the data source or database. Otherwise, an error is returned and no dialog box is displayed. NOPROMPT= is optional. If connection options are not specified, the default settings are used.

PROMPT=<'> *connection-information*<'>

specifies connection options to the data source.

A dialog box is displayed, using the values from the PROMPT= connection string. You can edit any field before you connect to the data source. When a successful connection is made, the complete connect string is returned in the SYSDBMSG macro variable.

PROMPT= is similar to the COMPLETE= option. However, unlike COMPLETE=, PROMPT= does not attempt to connect to the DBMS first. It displays the dialog box where you can edit or enter additional values.

REQUIRED=<'>*connection-options*<'>

specifies connection options for your data source or database. You separate multiple options with a semicolon.

If you specify enough correct connection options, such as user ID, password, and data source name, the SAS/ACCESS engine connects to the data source or database. Otherwise, a dialog box is displayed to prompt you for the connection options. Options in the dialog box that are not related to the connection are disabled. REQUIRED= only allows you to modify required fields in the dialog box. When a successful connection is made, the complete connect string is returned in the SYSDBMSG macro variable.

REQUIRED= is similar to COMPLETE= because it attempts to connect to the DBMS first. However, if REQUIRED= attempts to connect and fails, then a dialog box is displayed and you can only edit values that are in the required fields.

REQUIRED= is optional.

CONNECT Examples These examples use ODBC to connect to a data source that is configured under the data source name **`User's Data`** using the alias USER1. The first example uses the connect method that is guaranteed to be present at the lowest level of ODBC conformance. Note that DSN= names may contain quotes and spaces.

```
proc sql;
   connect to ODBC as user1
   (dsn="User's Data" uid=testuser pwd=testpass);
```

The next example uses the connect method that represents a more advanced level of ODBC conformance. It uses the input dialog box that is provided by the driver. The DSN= and UID= arguments are within the connect string and, therefore, are not parsed by the Pass-Through facility but instead are passed to the ODBC manager.

```
proc sql;
   connect to odbc as user1
   (required = "dsn=User's Data;uid=testuser");
```

The next ODBC example enables you to select any data source that is configured on your machine. The example uses the connect method that represents a more advanced level of ODBC conformance, Level 1. When a successful connection is made, the connect string is returned in the SQLXMSG and SYSDBMSG macro variables and can be stored if this method is used to configure a connection for later use.

```
proc sql;
   connect to odbc (required);
```

This last ODBC example prompts you to specify the information that is required to make a connection to the DBMS. You are prompted to supply the data source name, user ID, and password in the dialog boxes that are displayed.

```
proc sql;
   connect to odbc (prompt);
```

Tips for Connecting to a Microsoft Excel Data Source To connect to a Microsoft Excel (5.0, 7.0, and 8.0) data source, you must define a database as a named range within Excel. For this example, NEWSALES.XLS is the Excel file that contains sales data. Cell A1 contains “Region” and cell D5 contains the numeric value “399”. To create a named range within Excel, select the entire range including all column names. For this example, you would highlight the entire range between cells A1 and D5. From the Excel menu items, select Insert, select Name, and select Define. Type the name of the range, for example, SALES_Q1. To work with this data in SAS, you must be sure to save the .XLS file and close the file within Excel to prevent file locking errors when SAS uses ODBC to access the Excel file.

Multiple named ranges can exist within an Excel file. Each one would be treated as a separate table.

You must also create a data source name within the ODBC Administrator on your PC. Within the ODBC Administrator, select the Microsoft Excel ODBC driver and create a DSN. For this example, the DSN is named SALES_97 and is based on the Excel file NEWSALES.XLS. The following example will create a view and print the contents of the named range, SALES_Q1, from the Excel file NEWSALES.XLS.

```
proc sql;
   connect to odbc as mydb (dsn=sales_97);

create view regsales as
  select * from connection to mydb
    (select * from sales_q1);
quit;

proc print data=regsales;
run;
```

CONNECTION TO Component

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

Optional component

Syntax

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

Arguments

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

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

You must specify a *DBMS-SQL-query* argument in the CONNECTION TO component, and the query must be enclosed in parentheses. The query is passed to the DBMS exactly as you type it; therefore, if your DBMS is case sensitive, you must either use the correct case for DMBS object names or you must quote them. Quoted character strings are limited to 200 characters.

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

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

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

PROC SQL;

SELECT *column-list*

FROM CONNECTION TO *dbms-name (DBMS-SQL-query)*
    other-optional-PROC-SQL-clauses;

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

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

On many DBMSs, you can issue a CONNECTION TO component in a PROC SQL SELECT statement directly without first connecting to a DBMS (see “CONNECT Statement” on page 27). If you omit the CONNECT statement, an implicit connection is performed when the first PROC SQL SELECT statement that contains a CONNECTION TO component is passed to the DBMS. Default values are used for all connection arguments.

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

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

PROC SQL Pass-Through Views

A Version 6 PROC SQL Pass-Through view does not need to be updated to be used in Version 7 or Version 8. The conversion of PROC SQL Pass-Through views is automatic and does not require you to use the PROC CV2ODBC procedure, as described in . The ODBC interface and DBMS client must be available and ready to connect. In order for any truncated variable names to be correctly interpreted by the ODBC driver, you must specify the VALIDVARNAME=V6 option. See Chapter 5, "Macro Variables and System Options" for more information on the VALIDVARNAME= option.

Note: If the view cannot be processed, or if you want to see what the view is, use the DESCRIBE VIEW statement to see what the existing view is. Then you can use the PROC SQL statements to create a new view for the ODBC connection. △

IBM AS/400 Specifics

To run your SQL Pass-Through views for IBM AS/400, you must

- □ create a data source name first by using the ODBC administrator. Refer to “Data Source Configuration for the IBM AS/400” on page 34 for more information.
- □ set the environment variable AS400DSN to be ’*data-source-name*’. Quotation marks are required if the name includes blanks or special characters.

MS SQL Server Specifics

To run your SQL Pass-Through views for the MS SQL Server, you are encouraged, but not required, to create a data source name. You can use the ODBC administrator to create it. Refer to “Data Source Configuration for MS SQL Server” on page 35 for more information. If you do create a data source name, you must set the environment variable MSSQLDSN to be ’*data-source-name*’. Quotation marks are required if the name includes blanks or special characters.

Pass-Through View Examples

Example 1: An AS/400 Pass-Through View

In this example,

```
CONNECT TO AS400 AS market;
```

is converted to

```
CONNECT TO ODBC AS market
  (NOPROMPT="DSN=IBM AS/400 Database;
   UID=TESTUSER; PWD=TESTPASS;
   NAM=1"
  )
;
```

Example 2: An MS SQL Server Pass-Through View

In this example,

```
CONNECT TO SQLSERVR AS finance
  (user=testuser pass=testpass
   server='dbipc1.pc.sas.com'
   database='sample'
  )
;
```

is converted to

```
CONNECT TO ODBC AS finance
  (NOPROMPT="DSN=Microsoft SQL Server Database;
   SERVER=dbipc1.pc.sas.com;
   UID=testuser; PWD=testpass;
   DATABASE=sample"
  )
;
```

Example 3: Using the VALIDVARNAME=V6 Option

In this example, you must use the SAS option VALIDVARNAME=V6 in order to successfully process this Version 6 SQL Pass-Through view. See Chapter 5, "Macro Variables and System Options" for more information on this option.

```
options validvarname=v6;
proc sql;
  describe view as4sql.invoice4;
run;

/* NOTE: SQL view AS4SQL.INVOICE4 is defined as: */
select
 INVOICEN as INVOICE,
 AMTBILLE as AMOUNT format=DOLLAR20.2,
 BILLEDON
 from connection to AS400
 /* dbms=AS400, connect options=() */
 (select invoicenum, amtbilled, billedon
   from sasdemo/invoice
   where paidon ='18OCT1998');
```

In Version 6, the AS/400 column name INVOICENUM is mapped to the SAS variable INVOICEN, and AMTBILLED is mapped to AMTBILLE. If you do not specify option VALIDVARNAME=V6, you get the following error because the ODCB driver attempts to find the truncated column names in the DBMS table:

```
ERROR: The following columns were not found in the
       contributing  tables: AMTBILLE, INVOICEN.
```

Example 4: AS/400 Short Alias Names in SQL Pass-Through Views

This example demonstrates a problem in which AS/400 short alias names cannot be returned by the AS/400 ODBC driver. This problem causes you to get an error, for example, if you have specified the short alias names in your selection list before the CONNECTION TO component, but have not specified the short alias names in the selection list that defines the view. If you encounter this problem with your Version 6 SQL Pass-Through views, you need to re-create the views.

This example creates an AS/400 table named TEST5 with the columns CUSTOMER_FIRST_NAME and CUSTOMER_LAST_NAME. The short name alias for CUSTOMER_FIRST_NAME is FNAME and the short name alias for CUSTOMER_LAST_NAME is LNAME.

```
options validvarname=v6;
%let name=test5;
proc sql;
  describe view as4sql.&name;
  /* NOTE: SQL view AS4SQL.TEST5 is defined as: */
  select FNAME, LNAME from connection to AS400
  /* dbms=AS400, connect options=() */
  ( select * from sasdemo/test
      where lname = ’Ju’ );
quit;
proc print data=as4sql.&name;
run;
```

This example generates the following errors:

```
ERROR: The following columns were not found in
       the contributing tables: FNAME, LNAME.
ERROR: SQL View AS4SQL.TEST5 could not be processed.
```

The following two examples work successfully because the short alias names are specified in the SELECT statement that defines the view.

```
create view as4sql.&name as
  select FNAME, LNAME from connection to AS400
  /* dbms=AS400, connect options=() */
  (select FNAME, LNAME from sasdemo/test
    where lname = ’Ju’ );

create view as4sql.&name as
  select * from connection to AS400
  /* dbms=AS400, connect options=() */
  (select fname, lname from sasdemo/test
    where lname = ’Ju’ );
```

Data Source Configuration for the IBM AS/400

Note: The following instructions are specific to PC platforms. △

Use the following steps to configure your data source for AS/400.

1 Install the IBM AS/400 Client Access for Windows 95 or NT and Client Access ODBC Driver (32 bit).

   Note: You can install the software from ’IBM AS/400 Client Access Family’ CD-ROM. Refer to your IBM AS/400 documentation for more information. △

2 Double click on the 32–bit ODBC Administrator in the Control Panel.

3 To make the data source available to all users of your PC, click on the System DSN... button. If you do not want the data source to be available to all users of your PC, continue with step 4.

4 Select the Add button, and choose 'Client Access ODBC Driver(32 bit)'.

5 Type in a Data Source Name. 'AS400' is the standard, or you may choose another name.

6 Type in your user ID if you want it to be the default.

7 Make sure that the System Name is correct.

8 Select the Server tab, and clear the Default library.

9 Select the Format tab, and select 'System naming convention (*SYS)' as the naming convention.

10 Select the Performance tab, and uncheck 'Enable extended dynamic support'.

11 Select OK to save the data source configuration.

Data Source Configuration for MS SQL Server

Note: The following instructions are specific to PC platforms. △

Use the following steps to configure your data source for MS SQL Server.

1 Install the Microsoft SQL Server Client for Windows 95 or NT and Microsoft SQL Server ODBC Driver (32 bit). Refer to your MS SQL Server documentation for more information.

2 Double click on the 32–bit ODBC Administrator in the Control Panel.

3 To make the data source available to all users of your PC, click on the System DSN... button. If you do not want the data source to be available to all users of your PC, continue with step 4.

4 Select the Add button, and choose 'SQL Server ODBC Driver.'

5 Type in a Data Source Name. 'MSSQL' is the standard, or you may choose another name.

6 If you want to set default server and database names, type in the Server and Database Name fields.

7 Select OK to save the data source configuration.

Special ODBC Queries

The following special queries are supported by the SAS/ACCESS interface to ODBC. Many databases provide or use system tables that allow queries to return the list of available tables, columns, procedures, and other useful information. In ODBC, much of this functionality is provided through special APIs (application programming interfaces) in order to accommodate databases that do not follow the SQL table structure. You can use these special queries on non-SQL and on SQL databases. The general format of the special queries is:

ODBC::SQLAPI "*parameter 1*","*parameter n*"

where

ODBC::
: is required to distinguish special queries from regular queries

SQLAPI
: is the specific API that is being called. Both ODBC:: and SQLAPI are case sensitive.

"*parameter n*"
: is a quoted string that is delimited by commas.

Within the quoted string, two characters are universally recognized: the percent sign (%) and the underscore (_). The percent sign % matches any sequence of zero or more characters; the underscore represents any single character. Each driver also has an escape character that can be used to place characters within the string. Consult the driver's documentation to determine the valid escape character.

The values for the special query arguments are DBMS specific. For example, you supply the fully qualified table name for the "*Qualifier*" argument. In dBase, the value of "*Qualifier*" might be **`c:\dbase\tst.dbf`** and in SQL Server, the value might be **`test.customer`**. In addition, depending on the DBMS that you are using, valid values for "*Owner*" might be a user ID, a database name, or a library. All arguments are optional. If you specify some but not all parameters within an argument, use a comma to indicate the omitted parameters. If you do not specify any parameters, commas are not necessary. Note that these special queries may not be available for all ODBC drivers.

The following special queries are supported:

ODBC::SQLTables <"*Qualifier*", "*Owner*", "*Table-name*", "*Type*">
: returns a list of all the tables that match the specified arguments. If no arguments are specified, all accessible table names and information are returned.

ODBC::SQLColumns <"*Qualifier*", "*Owner*", "*Table-name*", "*Col-name*">
: returns a list of all the columns that match the specified arguments. If no arguments are specified, all accessible column names and information are returned.

ODBC::SQLColumnPrivileges <"*Qualifier*", "*Owner*", "*Table-name*", "*Col-name*">
: returns a list of all the column privileges that match the specified arguments. If no arguments are specified, all accessible column names and privilege information are returned.

ODBC::SQLForeignKeys <"*PK-qualifier*", "*PK-owner*", "*PK-table-name*", "*FK-qualifier*", "*FKOwner*", "*FKTableName*">
: returns a list of all the columns that comprise foreign keys that match the specified arguments. If no arguments are specified, all accessible foreign key columns and information are returned.

ODBC::SQLPrimaryKeys <"*Qualifier*", "*Owner*", "*Table-name*">
: returns a list of all the columns that compose the primary key that matches the specified table. A primary key can be composed of one or more columns. If no table name is specified, this special query will fail.

ODBC::SQLProcedureColumns <"*Qualifier*", "*Owner*", "*Proc-name*", "*Col-name*">
: returns a list of all the procedure columns that match the specified arguments. If no arguments are specified, all accessible procedure columns are returned.

ODBC::SQLProcedures <"*Qualifier*", "*Owner*", "*Proc-name*">
: returns a list of all the procedures that match the specified arguments. If no arguments are specified, all accessible procedures are returned.

ODBC::SQLStatistics <"*Qualifier*", "*Owner*", "*Table-name*">
: returns a list of the statistics for the specified table name, with options of SQL_INDEX_ALL and SQL_ENSURE set in the SQLStatistics API call. If the table name argument is not specified, this special query will fail.

ODBC::SQLTablePrivileges <"*Qualifier*", "*Owner*", "*Table-name*">
: returns a list of all the tables and associated privileges that match the specified arguments. If no arguments are specified, all accessible table names and associated privileges are returned.

ODBC::SQLGetTypeInfo
: returns information about the data types that are supported in the data source.

Examples

The following example sends an Oracle SQL query, presented in *italic* type, to the Oracle database for processing. The results from the Oracle SQL query serve as a virtual table for the PROC SQL FROM clause. In this example, MYCON is a connection alias.

```
proc sql;
connect to odbc as mycon
   (dsn=ora7 uid=testuser pwd=testpass);

select *
   from connection to mycon
      (select empid, lastname, firstname,
      hiredate, salary
          from sasdemo.employees
          where hiredate>='31.12.1988');

disconnect from mycon;
quit;
```

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

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

proc sql;
connect to odbc as mycon
   (dsn=ora7 uid=testuser pwd=testpass);

create view slib.hires88 as
 select *
    from connection to mycon
      (select empid, lastname, firstname,
       hiredate, salary from sasdemo.employees
       where hiredate>='31.12.1988');

disconnect from mycon;
quit;
```

The next example connects to Microsoft Access 7 and creates a view NEWORDERS from all the columns in the ORDERS table.

```
proc sql;
   connect to odbc as mydb
      (dsn=access7);
   create view neworders as
    select * from connection to mydb
      (select * from orders);
disconnect from mydb;
quit;
```

This ODBC example sends an SQL query to Microsoft SQL Server 6.5 configured under the data source name "SQL Server" for processing. The results from the query serve as a virtual table for the PROC SQL FROM clause. In this example, MYDB is the connection alias.

```
proc sql;
   connect to odbc as mydb
      (dsn="SQL Server" uid=testuser pwd=testpass);
   select * from connection to mydb
      (select CUSTOMER, NAME, COUNTRY
          from CUSTOMERS
          where COUNTRY <> 'USA');
quit;
```

The next ODBC example returns a list of the columns in the CUSTOMERS table.

```
proc sql;
   connect to odbc as mydb
      (dsn = "SQL Server" uid=testuser pwd=testpass);
   select * from connection to mydb
      (ODBC::SQLColumns (, , "CUSTOMERS"));
quit;
```

Date and Time Formats

Date, time, and datetime formats require the following syntax when used in ODBC.

For date:
{d '*yyyy-mm-dd*'}

For time:
{t '*hh:mm:ss*'}

For timestamp:
{ts '*yyyy-mm-dd hh:mm:ss[.ffff]*'}

This example uses the date format. It inserts three columns into a table named NEW_HIRES that is in a Microsoft Access 7 database named NORTHWIND. An SQL pass-through view is created for all new hires where CITY=SEATTLE.

```
proc sql;
connect to odbc (dsn=northwind);
execute (create table new_hires
   (lastname char(15), firstname char(10), hired date,
      city char(15))) by odbc;

execute (insert into new_hires
    values('Jones','Fred',{d '1998-01-11'}, 'Seattle'))
    by odbc;
execute (insert into new_hires
        values('Gomez','Maria',{d '1997-12-05'}, 'Tacoma'))
        by odbc;
    execute (insert into new_hires
        values('Smith','Mary',{d '1998-02-16'}, 'Seattle'))
        by odbc;

    create view region as
       select * from connection to odbc
         (select * from new_hires where city='Seattle');

    disconnect from odbc;
    quit;

    proc print data=region;
    title 'New Employees in Seattle Office';
    run;
```

ODBC Naming Conventions

Since ODBC is not a database but rather is an application programming interface, or API, table names and column names are determined at run time as described here. Beginning in Version 7 of SAS, table or column names can be up to 32 characters long. The ODBC engine supports table and column names up to 32 characters long. If the DBMS table names or column names are longer than 32 characters, they will be truncated to 32 characters. See Chapter 2, "SAS Names and Support for DBMS Names" for more information on SAS names.

PRESERVE_COL_NAMES= and PRESERVE_TAB_NAMES= are two LIBNAME options that specify whether to preserve spaces, special characters, and mixed case in DBMS column or table names. By setting these options to YES, SAS will guarantee to preserve the case sensitivity of the names. If PRESERVE_TAB_NAMES=NO and PRESERVE_COL_NAMES=NO, then column names and table names that are read from the DBMS are converted to SAS names by using the SAS name normalization rules.

This example specifies SYBASE as the DBMS. When interfacing to SYBASE through ODBC, you might also need to use the LIBNAME options QUOTE_CHAR=, DBCONINIT= and DBLIBINIT= for the engine to recognize embedded blanks in the table and column names. SYBASE is generally case sensitive. Therefore, this example would produce a SYBASE table named "a" and columns named "x" and "y".

```
    libname mydblib odbc user=TESTUSER password=testpass
            database=sybase;

    data mydblib.a;
         x=1;
         y=2;
    run;
```

If the DBMS was ORACLE, which is not case sensitive, the example would produce an ORACLE table named "A" and columns named "X" and "Y". The object names would be normalized to uppercase.

See “SAS/ACCESS LIBNAME Options” on page 7 for more information about the PRESERVE_COL_NAMES= and PRESERVE_TAB_NAMES= options.

ODBC Data Types

Every column in a table has a name and a data type. The data type tells the DBMS how much physical storage to set aside for the column and the form in which the data are stored.

Table 1.1 on page 40 shows all of the data types and default SAS formats that are supported by the ODBC engine. This table does not explicitly define the data types as they exist for each DBMS. It lists the SQL types that each DBMS data type would map to. For example, a CHAR data type under DB2 would map to an ODBC data type of SQL_CHAR. There are no unsupported data types.

Table 1.1 ODBC Data Types and Default SAS Formats

ODBC Data Type	Default SAS Format
SQL_CHAR	$*n*
SQL_VARCHAR	$*n*
SQL_LONGVARCHAR	$*n*
SQL_BINARY	$*n*.*
SQL_VARBINARY	$*n*.*
SQL_LONGVARBINARY	$*n*.*
SQL_DECIMAL	*m* or *m.n* or none if *m* and *n* are not specified
SQL_NUMERIC	*m* or *m.n* or none if *m* and *n* are not specified
SQL_INTEGER	11.
SQL_SMALLINT	6.
SQL_TINYINT	4.
SQL_BIT	1.
SQL_REAL	none
SQL_FLOAT	none
SQL_DOUBLE	none
SQL_BIGINT	none
SQL_DATE	DATE9.
SQL_TIME	TIME8. ODBC cannot support fractions of seconds for time values
SQL_TIMESTAMP	DATETIME*m.n* where *m* and *n* depend on precision

* Because the ODBC driver does the conversion, this field will be displayed as though the $HEX*n*. format were applied.

Table 1.2 on page 41 shows the default data types that the ODBC engine uses when creating tables.

Table 1.2 Default ODBC Output Data Types

SAS Variable Format	Default ODBC Data Type
m.n	SQL_DOUBLE or SQL_NUMERIC using *m.n* if the DBMS allows it
$*n*.	SQL_VARCHAR using *n*
datetime formats	SQL_TIMESTAMP
date formats	SQL_DATE
time formats	SQL_TIME

The ODBC engine allows nondefault data types to be specified with the DBTYPE= data set option. See "DBTYPE=" on page 18 for more information on this data set option.

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

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

ISBN 1–58025–543–4

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

1st printing, October 1999