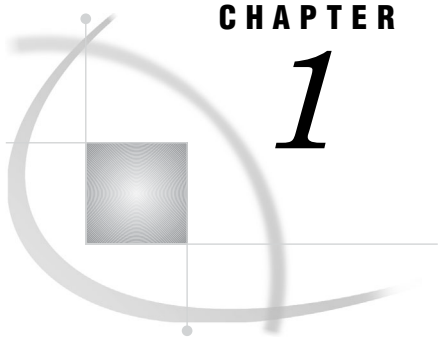


CHAPTER

1

SAS/ACCESS for ODBC

Introduction to the SAS/ACCESS Interface to ODBC

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

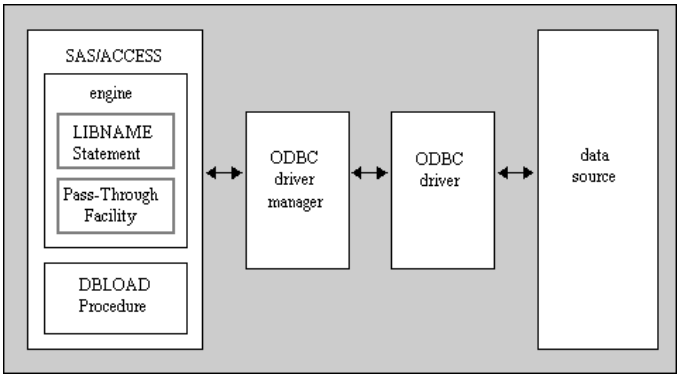
Overview of ODBC

Open database connectivity (ODBC) standards provide a common interface to a variety of databases, including AS/400, dBASE, Microsoft Access, Oracle, Paradox, and Microsoft 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 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 SAS is illustrated in the figure below.

Figure 1.1 The ODBC Interface to SAS



- 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, user ID, 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. During 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 about 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 *table* may be used to describe a variety of items, including a file, a part of a file, a group of files, a typical SQL table, 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 interface does use some Level 2 API calls if they are available.

  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 13.

- 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 macro variables SYSDBRC and SYSDBMSG.

- 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 documentation, 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 box. This brings up the ODBC driver setup dialog box for your specific ODBC driver. Clicking the Help button provides the information that you need to configure the ODBC data source for your driver.

LIBNAME Statement Specifics for ODBC

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

LIBNAME *libref* **odbc** <*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.

odbc
is the SAS/ACCESS engine name for the interface to ODBC.

connection-options
provide connection information and control how SAS manages the timing and concurrence of the connection to the DBMS. There are multiple ways that you can connect to ODBC when you use the LIBNAME statement. The methods are mutually exclusive, so you must use only one of the following for each connection:

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

These connection options are defined as follows:

USER=<'>*user-name*<'>
enables you to connect to an ODBC database, such as Microsoft SQL Server or AS/400, with a user ID that is different from the default ID.
USER= is optional. UID= is an alias for this option.

PASSWORD=<'>*password*<'>
specifies the ODBC password that is associated with your user ID.
PASSWORD= is optional. PWD is an alias for this option.

Note: If you do not wish to enter your ODBC password in clear text on this statement, see PROC PWENCODE for a method to encode it. △

DATASRC=<'>*ODBC-data-source*<'>
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= is an alias for this option that indicates that the connection is attempted using the ODBC SQLConnect API, which requires a data source name. Optionally, a user ID and password can be used in conjunction with DSN=.
If you want to use an ODBC file DSN, then instead of supplying DATASRC=<'>*ODBC-data-source*<'>, use the PROMPT= or NOPROMPT= option followed by "*filedsn*=(*name-of-your-file-dsn*);". For example:

```
libname mydblib odbc noprompt="filedsn=d:\share\msafiledsn.dsn;";
```

COMPLETE=<'>*ODBC-connection-options*<'>
specifies connection options for your data source or database. Separate multiple options with a semicolon. When a successful connection is made, the complete connection string is returned in the SYSDBMSG macro variable.
If you do not specify enough correct connection options, you are prompted 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.
This option is not supported on UNIX platforms. See your ODBC driver documentation for more details.

NOPROMPT=<'>*ODBC-connection-options*<'>
specifies connection options for your data source or database. Separate multiple options with a semicolon.
If you do not specify enough correct connection options, an error is returned. No dialog box is displayed to help you complete the connection string.

PROMPT=<'>*ODBC-connection-information*<'>
specifies connection options for your data source or database. Separate multiple options with a semicolon. When a successful connection is made, the complete connection string is returned in the SYSDBMSG macro variable.

PROMPT= does not immediately attempt to connect to the DBMS. Instead, it displays a dialog box that contains the values that you entered in the PROMPT= connection string. You can edit values or enter additional values in any field before you connect to the data source.

This option is not supported on UNIX platforms.

REQUIRED=<'>*ODBC-connection-options*<'>
specifies connection options for your data source or database. Separate multiple options with a semicolon. When a successful connection is made, the complete connection string is returned in the SYSDBMSG macro variable.

If you do not specify enough correct connection options, a dialog box prompts you for the connection options. REQUIRED= allows you to modify only required fields in the dialog box.

This option is not supported on UNIX platforms.

Note: See your ODBC driver documentation for a list of the ODBC connection options that your ODBC driver supports. △

The following ODBC connection options are not supported on UNIX:

REQUIRED=

BULKCOPY=

PROMPT=

COMPLETE=

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 the LIBNAME options that are supported for ODBC, 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 ODBC

Option	Default Value
ACCESS=	none
AUTOCOMMIT=	data source specific
BL_LOG=	none
BL_OPTIONS=	none
BULKLOAD=	NO
CONNECTION=	data source specific
CONNECTION_GROUP=	none
CURSOR_TYPE=	DYNAMIC
DBCOMMIT=	1000 (inserting) or 0 (updating)
DBCONINIT=	none
DBCONTERM=	none

Option	Default Value
DB_CREATE_TABLE_OPTS=	none
DBGEN_NAME=	DBMS
DBINDEX=	YES
DBLIBINIT=	none
DBLIBTERM=	none
DBMAX_TEXT=	1024
DBNULLKEYS=	YES
DBPROMPT=	NO
DBSLICEPARM=	THREADED_APPS,2 or 3
DEFER=	NO
DELETE_MULT_ROWS=	NO
DIRECT_EXE=	none
DIRECT_SQL=	YES
IGNORE_ READ_ONLY_COLUMNS=	NO
INSERT_SQL=	data source specific
INSERTBUFF=	1
KEYSET_SIZE=	0
MULTI_DATASRC_OPT=	NONE
PRESERVE_COL_NAMES=	see “Naming Conventions for ODBC” on page 25
PRESERVE_TAB_NAMES =	see “Naming Conventions for ODBC” on page 25
QUALIFIER=	none
QUERY_TIMEOUT=	0
QUOTE_CHAR=	none
READBUFF=	0
READ_ISOLATION_LEVEL=	RC (see “Locking in the ODBC Interface” on page 24)
READ_LOCK_TYPE=	ROW
REREAD_EXPOSURE=	NO
SCHEMA=	none
SPOOL=	YES
SQL_FUNCTIONS=	NONE
STRINGDATES=	NO
TRACE=	NO
TRACEFILE=	none
UPDATE_ISOLATION_LEVEL=	RC (see “Locking in the ODBC Interface” on page 24)
UPDATE_LOCK_TYPE=	ROW
UPDATE_MULT_ ROWS=	NO

Option	Default Value
UPDATE_SQL=	driver specific
USE_ODBC_CL =	NO
UTILCONN_TRANSIENT=	YES

ODBC LIBNAME Statement Examples

In the following example, USER=, PASSWORD=, and DATASRC= are connection options.

```
libname mydblib odbc user=testuser password=testpass datasrc=mydatasource;
```

In the following example, the libref MYLIB uses the ODBC engine to connect to an AS/400 database. The connection options are USER=, PASSWORD=, and DATASRC=.

```
libname mydblib odbc datasrc=as400 user=testuser
   password=testpass;

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

In the following example, the libref MYDBLIB uses the ODBC engine to connect to a Microsoft SQL Server database. The 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 for ODBC

The following table describes the data set options that are supported for ODBC, 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

Option	Default Value
CURSOR_TYPE=	LIBNAME option setting
DBCOMMIT=	LIBNAME option setting
DBCONDITION=	none
DBCREATE_TABLE_OPTS=	LIBNAME option setting
DBFORCE=	NO
DBGEN_NAME=	DBMS

Option	Default Value
DBINDEX=	LIBNAME option setting
DBKEY=	none
DBLABEL=	NO
DBMASTER=	none
DBMAX_TEXT=	1024
DBNULL=	YES
DBNULLKEYS=	LIBNAME option setting
DBPROMPT=	LIBNAME option setting
DBSASTYPE=	see “Data Types for ODBC” on page 26
DBSLICE=	none
DBSLICEPARM=	THREADED_APPS,2 or 3
DBTYPE=	see “Data Types for ODBC” on page 26
ERRLIMIT=	1
IGNORE_ READ_ONLY_COLUMNS=	NO
INSERT_SQL=	LIBNAME option setting
INSERTBUFF=	LIBNAME option setting
KEYSET_SIZE=	LIBNAME option setting
NULLCHAR=	SAS
NULLCHARVAL=	a blank character
PRESERVE_COL_NAMES=	LIBNAME option setting
QUALIFIER=	LIBNAME option setting
QUERY_TIMEOUT=	LIBNAME option setting
READBUFF=	LIBNAME option setting
READ_ISOLATION_LEVEL=	LIBNAME option setting
READ_LOCK_TYPE=	LIBNAME option setting
SASDATEFMT=	none
SCHEMA=	LIBNAME option setting
UPDATE_ISOLATION_LEVEL=	LIBNAME option setting
UPDATE_LOCK_TYPE=	LIBNAME option setting
UPDATE_SQL=	LIBNAME option setting

Pass-Through Facility Specifics for ODBC

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 the ODBC interface are as follows:

- The *dbms-name* is **ODBC**.
- The CONNECT statement is required.

- 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 might be limited by the particular data source's driver.
- The CONNECT statement *database-connection-arguments* are identical to its LIBNAME connection-options Not all of these arguments are supported by all ODBC drivers. Refer to your driver documentation for more information.
- 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 following LIBNAME options are available with the CONNECT statement:

  AUTOCOMMIT=
  CURSOR_TYPE=
  KEYSET_SIZE=
  QUERY_TIMEOUT=
  READBUFF=
  READ_ISOLATION_LEVEL=
  TRACE=
  TRACEFILE=
  USE_ODBC_CL=

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

CONNECT Statement Examples

The following 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 connection method that is guaranteed to be present at the lowest level of ODBC conformance. Note that DATASRC= names can contain quotation marks and spaces.

```
proc sql;
   connect to ODBC as user1
   (datasrc="User's Data" user=testuser password=testpass);
```

The following example uses the connection method that represents a more advanced level of ODBC conformance. It uses the input dialog box that is provided by the driver. The DATASRC= and USER= arguments are within the connection 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 following ODBC example enables you to select any data source that is configured on your machine. The example uses the connection method that represents a more advanced level of ODBC conformance, Level 1. When a successful connection is made, the connection 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);
```

The following 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);
```

Pass-Through Views

Version 6 SQL views do not need to be updated to be used in Version 7, Version 8, or SAS 9. 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.

The following example, must use the SAS option VALIDVARNAME=V6 in order to successfully process a Version 6 SQL view. See the section about the LIBNAME statement in *SAS/ACCESS for Relational Databases: Reference* for more information about 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');
```

Note: If a view cannot be processed, or if you want to see what a 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. △

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 ODBC 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.
```

IBM AS/400 Specifics

To run your SQL views for IBM AS/400, you must do the following:

- create a data source name first by using the ODBC administrator. Refer to the installation instructions for the SAS/ACCESS interface to ODBC for more information.
- set the environment variable AS400DSN (located in your SASV9.cfg) to the data source name that you assigned. Quotation marks are required if the name includes blanks or special characters.

In this example,

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

is converted to

```
CONNECT TO ODBC AS market
  (NOPROMPT="DATASRC=IBM AS/400 Database;
   USER=TESTUSER; PASSWORD=TESTPASS;
   NAM=1"
  )
;
```

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 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' );
```

Microsoft SQL Server Specifics

To run your SQL views for Microsoft 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 the installation instructions for this interface for more information. If you do create a data source name, you must set the environment variable MSSQLDSN to be the '*data-source-name*'. Quotation marks are required if the name includes blanks or special characters.

In this example,

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

is converted to

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

Connection To Component Examples

The following example sends an Oracle SQL query (presented in highlighted text) to the Oracle database for processing. The results from the 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
   (datasrc=ora7 user=testuser password=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 SQL view Samples.Hires88. The CREATE VIEW statement appears highlighted.

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

proc sql;
connect to odbc as mycon
   (datasrc=ora7 user=testuser password=testpass);
```

```
create view samples.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 following 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
      (datasrc=access7);
   create view neworders as
    select * from connection to mydb
      (select * from orders);
disconnect from mydb;
quit;
```

The following 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.

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

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

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

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 SQL databases. The general format of the special queries is as follows:

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 a "*Catalog*" argument. In dBase, the value of "*Catalog*" 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 a "*Schema*" argument might be a user ID, a database name, or a library. All arguments are optional. If you specify some but not all the arguments within a parameter, use a comma to indicate the omitted arguments. If you do not specify any parameters, commas are not necessary.

Note: These special queries might not be available for all ODBC drivers. △

The following special queries are supported:

ODBC::SQLTables <"*Catalog*", "*Schema*", "*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 <"*Catalog*", "*Schema*", "*Table-name*", "*Column-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 <"*Catalog*", "*Schema*", "*Table-name*", "*Column-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-catalog*", "*PK-schema*", "*PK-table-name*", "*FK-catalog*", "*FK-schema*", "*FK-table-name*">
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 <"*Catalog*", "*Schema*", "*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 fails.

ODBC::SQLProcedureColumns <"*Catalog*", "*Schema*", "*Procedure-name*", "*Column-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 <"*Catalog*", "*Schema*", "*Procedure-name*">
returns a list of all the procedures that match the specified arguments. If no arguments are specified, all accessible procedures are returned.

ODBC::SQLSpecialColumns <"*Identifier-type*", "*Catalog-name*", "*Schema-name*", "*Table-name*", "*Scope*", "*Nullable*">
returns a list of the optimal set of columns that uniquely identify a row in the specified table.

ODBC::SQLStatistics <"*Catalog*", "*Schema*", "*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 fails.

ODBC::SQLTablePrivileges <"*Catalog*", "*Schema*", "*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.

Autopartitioning Scheme for ODBC

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

Overview

The autopartitioning method available for SAS/ACCESS to ODBC is modeled after the MOD function method as described in the section about autopartitioning techniques in *SAS/ACCESS for Relational Databases: Reference*.

Autopartitioning Restrictions

SAS/ACCESS to ODBC places additional restrictions on which columns can be used for the partitioning column during the autopartitioning phase. Columns are partitioned as follows:

- SQL_INTEGER, SQL_BIT, SQL_SMALLINT, and SQL_TINYINT columns are given preference.
- SQL_DECIMAL, SQL_DOUBLE, SQL_FLOAT, SQL_NUMERIC, and SQL_REAL columns might be used for partitioning, provided the following conditions are met:
  - The ODBC driver supports converting these types to SQL_INTEGER via the INTEGER cast function.
  - The precision minus the scale of the column is greater than 0 but less than 10, that is, 0<(*precision-scale*)<10.

The exception to the above rule is for Oracle SQL_DECIMAL columns. As long as the scale of the SQL_DECIMAL column is 0, the column can be used as the partitioning column.

Nullable Columns

If a nullable column is selected for autopartitioning, then the SQL statement "OR<*column-name*>IS NULL" will be appended to the end of the SQL code that is

generated for the threaded read to ensure that any possible NULL values are returned in the result set. In addition, if the column to be used for the partitioning is SQL_BIT, then the number of threads will automatically be changed to two, regardless of the setting of the DBSLICEPARM= option.

Using WHERE Clauses

Autopartitioning does not select a column to be the partitioning column if it appears in the WHERE clause. For instance, the following data step would not be able to use a threaded read to retrieve the data since all of the numeric columns in the table are in the WHERE clause:

```
data work.locemp;
set trlib.MYEMPS;
where EMPNUM<=30 and ISTENURE=0 and
 SALARY<=35000 and NUMCLASS>2;
run;
```

Using DBSLICEPARM=

When using autopartitioning, and DBSLICEPARM= does not specify a maximum number of threads to use for the threaded read, SAS/ACCESS to ODBC defaults to three threads.

Using DBSLICE=

You might achieve the best possible performance when using threaded reads by specifying an ODBC-specific DBSLICE= option in your SAS operation. This is especially true if your DBMS supports multiple database partitions and provides a mechanism to allow connections to individual partitions. If your DBMS supports this concept, you can configure an ODBC datasource for each partition and use the DBSLICE= clause to specify both the datasource and the WHERE clause for each partition, as shown in the following example:

```
proc print data=trilib.MYEMPS(DBSLICE=(DSN1="EMPNUM BETWEEN 1 AND 33"
DSN2="EMPNUM BETWEEN 34 AND 66"
DSN3="EMPNUM BETWEEN 67 AND 100"));
run;
```

Consult your DBMS or ODBC driver documentation for more information about configuring for multiple partition access. You can also refer to “Configuring SQL Server Partitioned Views for Use with DBSLICE=” on page 17 for an example of configuring multiple partition access to a table.

Using the DATASOURCE= syntax is not required in order to use DBSLICE= with threaded reads for SAS/ACCESS to ODBC. The methods and examples described in DBSLICE= work well in instances where the table you want to read is not stored in multiple partitions in your DBMS. These methods also give you flexibility in column selection. For example, if you know that the STATE column in your employee table only contains a few distinct values, you can tailor your DBSLICE= clause accordingly:

```
datawork.locemp;
set trlib2.MYEMP(DBSLICE=("STATE='FL'" "STATE='GA'"
  "STATE='SC'" "STATE='VA'" "STATE='NC'"));
```

```
where EMPNUM<=30 and ISTENURE=0 and SALARY<=35000 and NUMCLASS>2;
run;
```

Configuring SQL Server Partitioned Views for Use with DBSLICE=

Microsoft SQL Server implements multiple partitioning by creating a global view across multiple instances of a Microsoft SQL Server database. For this example, assume that Microsoft SQL Server has been installed on three separate machines (SERVER1, SERVER2, SERVER3), and three ODBC datasources (SSPART1, SSPART2, SSPART3) have been configured against these servers. Also, a linked server definition for each of these servers has been defined. This example uses SAS to create the tables and associated views, but this can be accomplished outside of the SAS environment.

First create a local SAS table to build the Microsoft SQL Server tables:

```
data work.MYEMPS;
format HIREDATE mmddyy 0. SALARY 9.2
   NUMCLASS 6. GENDER $1. STATE $2. EMPNUM 10.;
do EMPNUM=1 to 100;
     morf=mod(EMPNUM,2)+1;
     if(morf eq 1) then
         GENDER='F';
     else
         GENDER='M';
     SALARY=(ranuni(0)*5000);
     HIREDATE=int(ranuni(13131)*3650);
     whatstate=int(EMPNUM/5);
     if(whatstate eq 1) then
         STATE='FL';
     if(whatstate eq 2) then
         STATE='GA';
     if(whatstate eq 3) then
         STATE='SC';
     if(whatstate eq 4) then
         STATE='VA';
     else
         state='NC';
     ISTENURE=mod(EMPNUM,2);
     NUMCLASS=int(EMPNUM/5)+2;
     output;
end;
run;
```

Next, create a table on each of the SQL server databases with the same table structure, and insert 1/3 of the overall data into each table. These table definitions also use CHECK constraints to enforce the distribution of the data on each of the subtables of the target view.

```
libname trlib odbc user=ssuser pw=sspwd dsn=sspart1;
proc delete data=trlib.MYEMPS1;
run;
data trlib.MYEMPS1(drop=morf whatstate
   DBTYPE=(HIREDATE="datetime" SALARY="numeric(8,2)"
   NUMCLASS="smallint" GENDER="char(1)" ISTENURE="bit" STATE="char(2)"
   EMPNUM="int NOT NULL Primary Key CHECK (EMPNUM BETWEEN 0 AND 33)"));
set work.MYEMPS;
```

```
where (EMPNUM BETWEEN 0 AND 33);
run;

libname trlib odbc user=ssuer pw=sspwd dsn=sspart2;
proc delete data=trlib.MYEMPS2;
run;
data trlib.MYEMPS2(drop=morf whatstate
   DBTYPE=(HIREDATE="datetime" SALARY="numeric(8,2)"
   NUMCLASS="smallint" GENDER="char(1)" ISTENURE="bit" STATE="char(2)"
   EMPNUM="int NOT NULL Primary Key CHECK (EMPNUM BETWEEN 34 AND 66)"));
set work.MYEMPS;
where (EMPNUM BETWEEN 34 AND 66);
run;

libname trlib odbc user=ssuer pw=sspwd dsn=sspart3;
proc delete data=trlib.MYEMPS3;
run;
data trlib.MYEMPS3(drop=morf whatstate
   DBTYPE=(HIREDATE="datetime" SALARY="numeric(8,2)"
   NUMCLASS="smallint" GENDER="char(1)" ISTENURE="bit" STATE="char(2)"
   EMPNUM="int NOT NULL Primary Key CHECK (EMPNUM BETWEEN 67 AND 100)"));
set work.MYEMPS;
where (EMPNUM BETWEEN 67 AND 100);
run;
```

Next, create a view using the UNION ALL construct on each Microsoft SQL Server instance which references the other two tables. This creates a global view that references the entire data set.

```
/*SERVER1,SSPART1*/
proc sql noerrorstop;
connect to odbc (UID=ssuser PWD=sspwd DSN=SSPART1);
execute (drop view MYEMPS) by odbc;
execute (create view MYEMPS AS
         SELECT * FROM users.ssuser.MYEMPS1
         UNION ALL
         SELECT * FROM SERVER2.users.ssuser.MYEMPS2
         UNION ALL
         SELECT * FROM SERVER3.users.ssuser.MYEMPS3) by odbc;
quit;

/*SERVER2,SSPART2*/
proc sql noerrorstop;
connect to odbc (UID=ssuser PWD=sspwd DSN=SSPART2);
execute (drop view MYEMPS) by odbc;
execute (create view MYEMPS AS
         SELECT * FROM users.ssuser.MYEMPS2
         UNION ALL
         SELECT * FROM SERVER1.users.ssuser.MYEMPS1
         UNION ALL
         SELECT * FROM SERVER3.users.ssuser.MYEMPS3) by odbc;
quit;

/*SERVER3,SSPART3*/
proc sql noerrorstop;
```

```
connect to odbc (UID=ssuser PWD=sspwd DSN=SSPART3);
execute (drop view MYEMPS) by odbc;
execute (create view MYEMPS AS
         SELECT * FROM users.ssuser.MYEMPS3
         UNION ALL
         SELECT * FROM SERVER2.users.ssuser.MYEMPS2
         UNION ALL
         SELECT * FROM SERVER1.users.ssuser.MYEMPS1) by odbc;
quit;
```

Finally, set up your SAS operation to perform the threaded read. The DBSLICE option contains the Microsoft SQL Server partitioning information.

```
proc print data=trlib.MYEMPS(DBLICE=(sspart1="EMPNUM BETWEEN 1 AND 33"
sspart2="EMPNUM BETWEEN 34 AND 66"
sspart3="EMPNUM BETWEEN 67 AND 100"));
run;
```

This configuration enables SAS/ACCESS to ODBC to access the data for the MYEMPS view directly from each subtable on the corresponding Microsoft SQL Server instance. The data is inserted directly into each subtable, but this process can also be accomplished by using the global view to divide up the data. For instance, you can create empty tables and then create the view as seen in the example with the UNION ALL construct. You can then insert the data into the view MYEMPS. The CHECK constraints will allow the Microsoft SQL Server query processor to determine which subtables should receive the data.

There are other tuning options available when configuring Microsoft SQL Server to use partitioned data. For more information, see the "Creating a Partitioned View" and "Using Views with Partitioned Data" sections in the *SQL Server On-line Guide*.

DBLOAD Procedure Specifics for ODBC

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

The ODBC interface supports all of the DBLOAD procedure statements (except ACCDESC=) in batch mode. The DBLOAD procedure specifics for ODBC are as follows:

- The DBLOAD step DBMS= value is **ODBC**.
- PROC DBLOAD uses the following database description statements:

  DSN= <'>*ODBC-data-source*<'>;
  specifies the name of the data source in which you want to store the new ODBC table. The *data-source* is limited to eight characters.

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

  USER= <'>*username*<'>;
  enables you to connect to an ODBC database, such as Microsoft SQL Server or AS/400, with a user ID that is different from the default ID.

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

  PASSWORD=<'>*password*<'>;
  specifies the ODBC password that is associated with your user ID.

PASSWORD= is optional in ODBC because users have default user IDs. If you specify USER=, you must specify PASSWORD=.

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

BULKCOPY= YES|NO;
determines whether SAS uses the Microsoft Bulk Copy facility to insert data into a DBMS table (Microsoft SQL Server only). The default value is NO.

BCP is Microsoft's Bulk Copy facility, and it enables you to efficiently insert rows of data into a DBMS table as a unit. As SAS/ACCESS sends each row of data to BCP, the data is written to an input buffer. When you have inserted all the rows, or the buffer reaches a certain size (as determined by the DBCOMMIT= data set option), all of the rows are inserted as a unit into the table, and the data is committed to the table.

Alternatively, you can set the DBCOMMIT=*n* option to commit rows after every *n* insertions.

If an error occurs, a message is written to the SAS log, and any rows that have been inserted in the table before the error are rolled back.

Note: To use BULKCOPY=, your installation of Microsoft SQL Server must include the ODBCBCP.DLL, which is currently only supported by Microsoft SQL Server 7.0. BULKCOPY= is not supported on UNIX. △

- The TABLE= statement is as follows:

  TABLE= <*authorization-id*.>*table-name*;
  identifies the table or view that you want to use to create an access descriptor. The TABLE= statement is required.

  The *authorization-id* is a user ID or group ID that is associated with the table.

- The NULLS statement is as follows:

  NULLS *variable-identifier-1* =Y|N|D < . . . *variable-identifier-n* =Y|N|D >;
  enables you to specify whether the columns that are associated with the listed SAS variables allow NULL values. By default, all columns accept NULL values.

  The NULLS statement accepts any one of these three values:

  Y – specifies that the column accepts NULL values. This is the default.

  N – specifies that the column does not accept NULL values.

  D – specifies that the column is defined as NOT NULL WITH DEFAULT

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 following 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;
```

Passing SAS Functions to ODBC

The interface to ODBC passes the following SAS functions to the data source for processing (if the DBMS server supports the function). See the section about optimizing SQL usage in *SAS/ACCESS for Relational Databases: Reference* for information.

ABS

ARCOS

ARSIN

ATAN

AVG

CEIL

COS

EXP

FLOOR

LOG

LOG10

LOWCASE

MAX

MIN

SIGN

SIN

SQRT

TAN

UPCASE

SUM

COUNT

Passing Joins to ODBC

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

user ID

password

datasource

catalog

UPDATE_ISOLATION_LEVEL=
(if specified)

READ_ISOLATION_LEVEL=
(if specified)

PROMPT=
must *not* be
specified

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.

Temporary Table Support for ODBC

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 want to use temporary tables that persist across SAS procedures and DATA steps with ODBC, you must use the CONNECTION=SHARED LIBNAME option. In doing so, the temporary table is available for processing until the libref is closed.

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 single connection.

Examples

Using the Internat sample table, the following example creates a temporary table, #LONDON, with Microsoft SQL Server that contains information about flights that flew to London. This table is then joined with a larger SQL Server table that lists all the flights, March, but matched only on flights that flew to London.

```
libname samples odbc dsn=lupinss uid=dbitest pwd=dbigrp1 connection=shared;
```

```
data samples.'#LONDON'n;
   set work.internat;
   where dest='LON';
run;

proc sql;
   select b.flight, b.dates, b.depart, b.orig
          from samples.'#LONDON'n a, samples.march b
          where a.dest=b.dest;
quit;
```

In the following example a temporary table called New is created with Microsoft SQL Server. The data from this table is then appended to an existing SQL Server table named Inventory.

```
libname samples odbc dsn=lupinss uid=dbitest pwd=dbigrp1 connection=shared;

data samples.inventory(DBTYPE=(itemnum='char(5)' item='varchar(30)'
                       quantity='numeric'));
   itemnum='12001';
      item='screwdriver';
      quantity=15;
      output;
   itemnum='12002';
      item='hammer';
      quantity=25:
      output;
   itemnum='12003';
      item='sledge hammer';
      quantity=10;
      output;
   itemnum='12004';
      item='saw';
      quantity=50;
      output;
   itemnum='12005';
      item='shovel';
      quantity=120;
      output;
run;

data samples.'#new'n(DBTYPE=(itemnum='char(5)' item='varchar(30)'
                     quantity='numeric'));
   itemnum='12006';
      item='snow shovel';
      quantity=5;
      output;
   itemnum='12007';
      item='nails';
      quantity=500;
      output;
run;

proc append base=samples.inventory data=samples.'#new'n;
run;
```

```
proc print data=samples.inventory;
run;
```

The following example demonstrates the use of a temporary table using the Pass-Through Facility.

```
proc sql;
   connect to odbc as test (dsn=lupinss uid=dbitest
                            pwd=dbigrp1 connection=shared);
   execute (create table #FRANCE (flight char(3), dates datetime,
            dest char(3))) by test;

   execute (insert #FRANCE select flight, dates, dest from internat
            where dest like '%FRA%') by test;
   select * from connection to test (select * from #FRANCE);
quit;
```

ODBC Bulk Loading

The LIBNAME option BULKLOAD= calls the Bulk Copy facility (BCP), which enables you to efficiently insert rows of data into a DBMS table as a unit. BCP= is an alias for this option.

Note: The Bulk Copy facility is available only when you are accessing Microsoft SQL Server data on Windows platforms. To use this facility, your installation of Microsoft SQL Server must include the ODBCBCP.DLL file. BULKCOPY= is not available on UNIX. △

As SAS/ACCESS sends rows of data to the Bulk Copy facility, the data is written to an input buffer. When you have sent all of the rows or when the buffer reaches a certain size (as determined by the DBCOMMIT= option), all of the rows are inserted as a unit into the table and the data is committed to the table. You can set the DBCOMMIT= option to commit rows after a specified number of rows are inserted.

If an error occurs, a message is written to the SAS log, and any rows that were inserted before the error are rolled back.

Locking in the ODBC Interface

The following LIBNAME and data set options enable you to control how the interface to ODBC handles locking. See the section about the LIBNAME statement in *SAS/ACCESS for Relational Databases: Reference* for additional information about these options.

READ_LOCK_TYPE= ROW | TABLE | NOLOCK

UPDATE_LOCK_TYPE= ROW | TABLE | NOLOCK

READ_ISOLATION_LEVEL= S | RR | RC | RU | V
The ODBC driver manager supports the S, RR, RC, RU, and V isolation levels defined in the following table.

Table 1.3 Isolation Levels for ODBC

Isolation Level	Definition
S (serializable)	Does not allow dirty reads, nonrepeatable reads, or phantom reads.
RR (repeatable read)	Does not allow dirty reads or nonrepeatable reads; does allow phantom reads.
RC (read committed)	Does not allow dirty reads or nonrepeatable reads; does allow phantom reads.
RU (read uncommitted)	Allows dirty reads, nonrepeatable reads and phantom reads.
V (versioning)	Does not allow dirty reads, nonrepeatable reads, or phantom reads. These transactions are serializable but higher concurrency is possible than with the serializable isolation level. Typically, a nonlocking protocol is used.

The terms in the table are defined as follows:

- *Dirty read* — A transaction that exhibits this phenomenon has very minimal isolation from concurrent transactions. In fact, it can 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 might 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 phenomenon, 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 sees a row that did not previously exist, a phantom.

UPDATE_ISOLATION_LEVEL= S | RR | RC | V
The ODBC driver manager supports the S, RR, RC, and V isolation levels defined in the preceding table.

Naming Conventions for ODBC

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. 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 column names

are longer than 32 characters, they are truncated to 32 characters. If truncating a columns name results in identical names, then SAS generates unique names by replacing the last character with a number. DBMS table names must be 32 characters or less, since SAS will *not* truncate a longer name. If you already have a table name greater than 32 characters, it is recommended that you create a table view.

The PRESERVE_COL_NAMES= and PRESERVE_TAB_NAMES= options determine how the interface to ODBC handles case sensitivity, spaces, and special characters. The default value for both options is YES for Microsoft Access, Microsoft Excel, and Microsoft SQL Server; and and NO for all others.

The following example specifies SYBASE as the DBMS.

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

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

SYBASE is generally case sensitive. Therefore, this example would produce a SYBASE table named **a** and columns named **x** and **y**.

If the DBMS being accessed 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.

Data Types for ODBC

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 is stored.

The following table shows all of the data types and default SAS formats that are supported by the SAS/ACCESS interface to ODBC. 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 1.4 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.

ODBC Data Type	Default SAS Format
SQL_TINYINT	4.
SQL_BIT	1.
SQL_REAL	none
SQL_FLOAT	none
SQL_DOUBLE	none
SQL_BIGINT	20.
SQL_INTERVAL	$n
SQL_GUID	$n
SQL_TYPE_DATE	DATE9.
SQL_TYPE_TIME	TIME8.
	ODBC cannot support fractions of seconds for time values
SQL_TYPE_TIMESTAMP	DATETIME*m.n* where *m* and *n* depend on precision

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

The following table shows the default data types that the SAS/ACCESS interface to ODBC uses when creating tables.

Table 1.5 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 interface to ODBC allows nondefault data types to be specified with the DBTYPE= data set option.

ODBC Null Values

Many relational database management systems have a special value called NULL. A DBMS NULL value means an absence of information and is analogous to a SAS missing value. When SAS/ACCESS reads a DBMS NULL value, it interprets it as a SAS missing value.

In most relational databases, columns can be defined as NOT NULL so that they require data (they cannot contain NULL values). When a column is defined as NOT NULL, the DBMS will not add a row to the table unless the row has a value for that column. When creating a DBMS table with SAS/ACCESS, you can use the DBNULL= data set option to indicate whether NULL is a valid value for specified columns.

ODBC mirrors the behavior of the underlying DBMS with regard to NULL values. Refer to the documentation for your DBMS for information about how it handles NULL values.

For more information about how SAS handles NULL values, see "Potential Result Set Differences When Processing Null Data" in *SAS/ACCESS for Relational Databases: Reference*.

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