CHAPTER

1

SAS/ACCESS for MySQL

Introduction to the SAS/ACCESS Interface to MySQL

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

LIBNAME Statement Specifics for MySQL

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

LIBNAME *libref* **mysql** <*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.

mysql
is the SAS/ACCESS engine name for the interface to MySQL.

connection-options
provide connection information for the connection to the DBMS. The connection options for the interface to MySQL are:

USER=<'>*username*<'>
specifies the MySQL user login ID. If this argument is not specified, the current user is assumed. If the user name contains spaces or non-alphanumeric characters, you must enclose the user name in quotation marks.

PASSWORD=<'>*password*<'>
specifies the MySQL password that is associated with the MySQL login ID. If the password contains spaces or non-alphanumeric characters, you must enclose the password in quotation marks.

DATABASE=<'>*database*<'>
specifies the MySQL database to which you want to connect. If the database name contains spaces or non-alphanumeric characters, you must enclose the database name in quotation marks.

SERVER=<'>*server*<'>
specifies the server name or IP address of the MySQL server. If the server name contains spaces or non-alphanumeric characters, you must enclose the server name in quotation marks.

PORT=*port*
specifies the port used to connect to the specified MySQL server.

LIBNAME-options
define how DBMS objects are processed by SAS. Some LIBNAME options can enhance performance; others determine naming behavior. The following table describes LIBNAME options that are supported for MySQL 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 MySQL

Option	Default Value
ACCESS=	NONE
AUTOCOMMIT=	YES
CONNECTION=	SHAREDREAD
CONNECTION_GROUP=	NONE
DBCOMMIT=	1000 when inserting rows; 0 when updating rows, deleting rows, or appending rows to an existing table

Option	Default Value
DBCONINIT=	NONE
DBCONTERM=	NONE
DBCREATE_TABLE_OPTS=	NONE
DBGEN_NAME=	DBMS
DBINDEX=	NO
DBLIBINIT=	NONE
DBLIBTERM=	NONE
DBMAX_TEXT=	1024
DBPROMPT=	NO
DBSASLABEL=	COMPAT
DEFER=	NO
DIRECT_EXE=	NONE
DIRECT_SQL=	YES
INSERTBUFF=	0
MULTI_DATASRC_OPT=	NONE
PRESERVE_COL_NAMES=	NO
PRESERVE_TAB_NAMES=	NO
QUALIFIER=	NONE
REREAD_EXPOSURE=	NO
SPOOL=	YES
SQL_FUNCTIONS=	NONE
UTILCONN_TRANSIENT=	NO

MySQL LIBNAME Statement Examples

In the following example, the libref MYSQLLIB uses the SAS/ACCESS interface to MySQL to connect to a MySQL database. The SAS/ACCESS connection options are USER=, PASSWORD=, DATABASE=, SERVER=, and PORT=.

```
libname mysqllib mysql user=testuser password=testpass database=mysqldb
server=mysqlserv port=9876;

proc print data=mysqllib.employees;
   where dept='CSR010';
run;
```

Data Set Options for MySQL

The following table describes all data set options that are supported for the MySQL interface. Default values are provided where applicable. See the section about data set

options in *SAS/ACCESS for Relational Databases: Reference* for general information about these options.

Table 1.2 Data Set Options for MySQL

Option	Default Value
AUTOCOMMIT=	the current LIBNAME option setting
DBCOMMIT=	the current LIBNAME option setting
DBCONDITION=	none
DBCREATE_TABLE_OPTS=	the current LIBNAME option setting
DBGEN_NAME=	DBMS
DBINDEX=	the current LIBNAME option setting
DBKEY=	none
DBLABEL=	NO
DBMASTER=	none
DBMAX_TEXT=	1024
DBNULL=	YES
DBPROMPT=	the current LIBNAME option setting
DBSASLABEL=	COMPAT
DBSASTYPE=	See "Data Types for MySQL Servers" on page 9.
DBTYPE=	See "LIBNAME Statement Data Conversions" on page 12.
INSERTBUFF=	0
NULLCHAR=	SAS
NULLCHARVAL=	a blank character
PRESERVE_COL_NAMES=	current LIBNAME option setting
QUALIFIER=	the current LIBNAME option setting
SASDATEFORMAT=	DATETIME20.0
UPDATE_ISOLATION_LEVEL=	the current LIBNAME option setting

Pass-Through Facility Specifics for MySQL

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 MySQL are as follows:

- The *dbms-name* is **mysql**.
- The *database-connection-arguments* for the CONNECT statement are as follows:

  USER=<'>*MySQL-login-ID*<'>
  specifies an optional MySQL login ID. If USER= is not specified, the current user is assumed. If you specify USER=, you also must specify PASSWORD=.

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

specifies the MySQL password that is associated with the MySQL login ID. If you specify PASSWORD=, you also must specify USER=.

DATABASE=<'>*database-name*<'>
specifies the MySQL database.

SERVER=<'>*server-name*<'>
specifies the name or IP address of the MySQL server to which to connect. If *server-name* is omitted or set to **`localhost`**, a connection to the local host is established.

PORT=*port*
specifies the port on the server that is used for the TCP/IP connection.

Examples

The following example uses the alias DBCON for the DBMS connection (the connection alias is optional):

```
proc sql;
   connect to mysql as dbcon
       (user=testuser password=testpass server=mysqlserv
        database=mysqldb port=9876);
quit;
```

The following example connects to MySQL and sends it two EXECUTE statements to process:

```
proc sql;
   connect to mysql (user=testuser password=testpass server=mysqlserv
        database=mysqldb port=9876);
   execute (create table whotookorders as
      select ordernum, takenby,
             firstname, lastname, phone
        from orders, employees
        where orders.takenby=employees.empid)
      by mysql;
   execute (grant select on whotookorders
            to testuser) by mysql;
   disconnect from mysql;
quit;
```

The following example performs a query, shown in highlighted text, on the MySQL table CUSTOMERS:

```
proc sql;
connect to mysql (user=testuser password=testpass server=mysqlserv
        database=mysqldb port=9876);
select *
   from connection to mysql
     (select * from customers
      where customer like '1%');
    disconnect from mysql;
quit;
```

Autocommit and Table Types

MySQL supports several table types, two of which are MyISAM (the default) and INNODB. A single database can contain tables of different types. The behavior of a table is determined by its table type. For example, by definition, a table created of MyISAM type does not support transactions. Consequently, all DML statements (updates, deletes, inserts) are automatically committed. If you need transactional support, specify a table type of INNODB in the DBCREATE_TABLE_OPTS LIBNAME option. This table type allows for updates, deletes, and inserts to be rolled back if an error occurs; or updates, deletes, and inserts to be committed if the SAS DATA step or procedure completes successfully.

By default, the MYSQL libname engine sets **AUTOCOMMIT=YES** regardless of the table type. If you are using tables of the type INNODB, set the LIBNAME option AUTOCOMMIT=NO to improve performance. To control how often COMMITS are executed, set the DBCOMMIT option.

Note: The DBCOMMIT option can affect SAS/ACCESS performance. Experiment with a value that best fits your table size and performance needs before using it for production jobs. Transactional tables require significantly more memory and disk space requirements. △

Understanding MySQL Update and Delete Rules

To avoid data integrity problems when updating or deleting data, you need a primary key defined on your table. Refer to the MySQL documentation for more information regarding table types and transactions.

The following example uses AUTOCOMMIT=NO and DBTYPE to create the primary key, and DBCREATE_TABLE_OPTS to determine the MySQL table type.

```
libname invty mysql user=dbitest server=d6687 database=test autocommit=no
reread_exposure=no;

proc sql;
drop table invty.STOCK23;
quit;

/* Create DBMS table with primary key and of type INNODB*/
data invty.STOCK23(drop=PARTNO DBTYPE=(RECDATE="date not null,
                    primary key(RECDATE)") DBCREATE_TABLE_OPTS="type = innodb");
  input PARTNO $ DESCX $ INSTOCK @17
        RECDATE date7. @25 PRICE;
  format RECDATE date7.;
  datalines;
K89R seal  34   27jul95 245.00
M447 sander 98  20jun95 45.88
LK43 filter 121 19may96 10.99
MN21 brace 43   10aug96 27.87
BC85 clamp 80   16aug96 9.55
KJ66 cutter 6   20mar96 24.50
UYN7 rod  211   18jun96 19.77
JD03 switch 383 09jan97 13.99
BV1I timer 26   03jan97 34.50
;
```

```
proc sql;
update invty.STOCK23 set price=price*1.1 where INSTOCK > 50;
quit;
```

Passing SAS Functions to MySQL

The interface to MySQL passes the following SAS functions to MySQL for processing when the is set to ALL. Where the MySQL function name differs from the SAS function name, the MySQL name appears in parentheses. See "Passing Functions to the DBMS using PROC SQL" in *SAS/ACCESS for Relational Databases: Reference* for information.

ABS

ARCOS (ACOS)

ARSIN (ASIN)

ATAN

BYTE

CEIL (CEILING)

COMPRESS

COS

COT

DATE

DATETIME

DAY

EXP

FLOOR

HOUR

INDEX

LOWCASE (LCASE)

LENGTH

LOG

LOG10

MINUTE

MOD

MONTH

QTR

REPEAT

SECOND

SIGN

SIN

SOUNDEX

SQRT

SUBSTR

TAN

TIME

TODAY

TRIM (TRIMN)

UPCASE (UCASE)

WEEKDAY

YEAR

Passing Joins to MySQL

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

user

password

database

server

See "Passing Joins to the DBMS" in *SAS/ACCESS for Relational Databases: Reference* for more information about when and how SAS/ACCESS passes joins to the DBMS.

Naming Conventions for MySQL

MySQL database identifiers that can be named include databases, tables, and columns. The MySQL documentation contains extensive on naming conventions. The following are some of the naming conventions that you must use.

- All identifier names must be from 1 to 64 characters long, except for aliases, which may be 255 characters.
- Database names must be unique. For each user within a database, names of database objects must be unique across all users (for example, if a database contains a department table created by user A, no other user can create a department table in the same database).

  Note: MySQL does not recognize the notion of schema. Consequently, tables are automatically visible to all users with appropriate privileges. Column names and index names must be unique within a table. △
- Database names can use any character that is allowed in a directory name except for periods and backward and forward slashes.
- Table names may use any character allowed in a filename except for periods and forward slashes.
- Column and alias names allow all characters.

- A name cannot be a MySQL reserved word unless the name is enclosed in quotation marks. See the MySQL documentation for more information about reserved words.
- Embedded spaces and other special characters are not permitted unless the name is enclosed in quotation marks.
- Embedded quotation marks are not permitted.
- Case sensitivity is set when a server is installed. By default, the names of database objects are case-sensitive on UNIX and not case-sensitive on Windows. For example, the names **`CUSTOMER`** and **`customer`** are different on a case-sensitive server.

Note: By default, column and table names are not quoted in the SAS/ACCESS interface to MySQL. To quote the table and column names, you must use the LIBNAME statement PRESERVE_TAB_NAMES=. △

Case Sensitivity for MySQL

In MySQL, databases and tables correspond to directories and files within those directories. Consequently, the case sensitivity of the underlying operating system determines the case sensitivity of database and table names. This means database and table names are case-insensitive in Windows, and case-sensitive in most varieties of UNIX.

In SAS, names can be entered in either uppercase or lowercase. MySQL recommends that you adopt a consistent convention of either all uppercase or all lowercase tablenames, especially on UNIX hosts. This can be easily implemented by starting your server with *-O lower_case_table_names=1*. Please see the MySQL documentation for more details.

If your server is on a case-sensitive platform, and you choose to allow case sensitivity, be aware that when you reference MYSQL objects through the SAS/ACCESS interface, objects are case-sensitive and require no quotation marks. Furthermore, in the pass-through facility, all MySQL object names are case-sensitive. The names are passed to MySQL exactly as they are typed.

For more information about case sensitivity and MySQL names, see Naming Conventions for MySQL.

Data Types for MySQL Servers

Overview

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

Character Data

BLOB
: contains binary data of variable length up to 64 kilobytes. Variables entered into columns of this type must be inserted as character strings.

CHAR *(n)*
: contains fixed-length character string data with a length of *n*, where *n* must be at least 1 and cannot exceed 255 characters.

ENUM (*"value1"*, *"value2"*, *"value3"*,...)
: contains a character value that can be chosen from the list of allowed values. You can specify up to 65535 ENUM values. If the column contains a string not specified in the value list, the column value is set to "0".

LONGBLOB
: contains binary data of variable length up to 4 gigabytes. Variables entered into columns of this type must be inserted as character strings. Available memory considerations might limit the size of a LONGBLOB data type.

LONGTEXT
: contains text data of variable length up to 4 gigabytes. Available memory considerations might limit the size of a LONGTEXT data type.

MEDIUMBLOB
: contains binary data of variable length up to 16 megabytes. Variables entered into columns of this type must be inserted as character strings.

MEDIUMTEXT
: contains text data of variable length up to 16 megabytes.

SET (*"value1"*, *"value2*", *"value3"*,...)
: contains zero or more character values that must be chosen from the list of allowed values. You can specify up to 64 SET values.

TEXT
: contains text data of variable length up to 64 kilobytes.

TINYBLOB
: contains binary data of variable length up to 256 bytes. Variables entered into columns of this type must be inserted as character strings.

TINYTEXT
: contains text data of variable length up to 256 bytes.

VARCHAR (*n*)
: contains character string data with a length of *n*, where *n* is a value from 1 to 255.

Numeric Data

BIGINT (*n*)
: specifies an integer value, where *n* indicates the display width for the data. You might experience problems with MySQL if the data column contains values that are larger than the value of *n*. Values for BIGINT can range from -9223372036854775808 to 9223372036854775808.

DECIMAL (*length*, *decimals*)
: specifies a fixed-point decimal number, where *length* is the total number of digits (precision), and *decimals* is the number of digits to the right of the decimal point (scale).

DOUBLE (*length*, *decimals*)
specifies a double-precision decimal number, where *length* is the total number of digits (precision), and *decimals* is the number of digits to the right of the decimal point (scale). Values can range from approximately –1.8E308 to –2.2E-308 and 2.2E-308 to 1.8E308 (if UNSIGNED is specified).

FLOAT (*length*, *decimals*)
specifies a floating-point decimal number, where *length* is the total number of digits (precision) and *decimals* is the number of digits to the right of the decimal point (scale). Values can range from approximately –3.4E38 to –1.17E-38 and 1.17E-38 to 3.4E38 (if UNSIGNED is specified).

INT (*n*)
specifies an integer value, where *n* indicates the display width for the data. You might experience problems with MySQL if the data column contains values that are larger than the value of *n*. Values for INT can range from –2147483648 to 2147483647.

MEDIUMINT (*n*)
specifies an integer value, where *n* indicates the display width for the data. You might experience problems with MySQL if the data column contains values that are larger than the value of *n*. Values for MEDIUMINT can range from –8388608 to 8388607.

SMALLINT (*n*)
specifies an integer value, where *n* indicates the display width for the data. You might experience problems with MySQL if the data column contains values that are larger than the value of *n*. Values for SMALLINT can range from –32768 to 32767.

TINYINT (*n*)
specifies an integer value, where *n* indicates the display width for the data. You might experience problems with MySQL if the data column contains values that are larger than the value of *n*. Values for TINYINT can range from -128 to 127.

Other Data Types

DATE
contains date values. Valid dates are from January 1, 1000, to December 31, 9999. The default format is YYYY-MM-DD, for example, 1961–06–13.

DATETIME
contains date and time values. Valid values are from 00:00:00 on January 1, 1000, to 23:59:59 on December 31, 9999. The default format is YYYY-MM-DD HH:MM:SS, for example, 1992–09–20 18:20:27.

TIME
contains time values. Valid times are –838 hours, 59 minutes, 59 seconds to 838 hours, 59 minutes, 59 seconds. The default format is HH:MM:SS, for example, 12:17:23.

TIMESTAMP
contains date and time values used to mark data operations. Valid values are from 00:00:00 on January 1, 1970, to 2037. The default format is YYYY-MM-DD HH:MM:SS, for example, 1995–08–09 15:12:27.

LIBNAME Statement Data Conversions

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

Table 1.3 LIBNAME Statement: Default SAS Formats for MySQL Data Types

MySQL Column Type	SAS Data Type	Default SAS Format
CHAR(*n*)	character	$*n*.
VARCHAR(*n*)	character	$*n*.
TINYTEXT	character	$*n*.
TEXT	character	$*n*. (where *n* is the value of the DBMAX_TEXT= option)
MEDIUMTEXT	character	$*n*. (where *n* is the value of the DBMAX_TEXT= option)
LONGTEXT	character	$*n*. (where *n* is the value of the DBMAX_TEXT= option)
TINYBLOB	character	$*n*. (where *n* is the value of the DBMAX_TEXT= option)
BLOB	character	$*n*. (where *n* is the value of the DBMAX_TEXT= option)
MEDIUMBLOB	character	$*n*. (where *n* is the value of the DBMAX_TEXT= option)
LONGBLOB	character	$*n*. (where *n* is the value of the DBMAX_TEXT= option)
ENUM	character	$*n*.
SET	character	$*n*.
TINYINT	numeric	4.0
SMALLINT	numeric	6.0
MEDIUMINT	numeric	8.0
INT	numeric	11.0
BIGINT	numeric	20.
DECIMAL	numeric	*m.n*
FLOAT	numeric	
DOUBLE	numeric	
DATE	numeric	DATE
TIME	numeric	TIME

MySQL Column Type	SAS Data Type	Default SAS Format
DATETIME	numeric	DATETIME
TIMESTAMP	numeric	DATETIME

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

Table 1.4 LIBNAME Statement: Default MySQL Data Types for SAS Variable Formats

SAS Variable Format	**MySQL Data Type**
*m.n**	DECIMAL ([*m*-1],*n*)**
n (where *n* <= 2)	TINYINT
n (where *n* <= 4)	SMALLINT
n (where *n* <=6)	MEDIUMINT
n (where *n* <= 17)	BIGINT
other numerics	DOUBLE
$*n* (where *n* <= 255)	VARCHAR(*n*)
$*n* (where *n* > 255)	TEXT
datetime formats	TIMESTAMP
date formats	DATE
time formats	TIME

* *n* in MySQL data types is equivalent to *w* in SAS formats.

** DECIMAL types are created as (m-1, n). SAS includes space to write the value, the decimal point, and a minus sign (if necessary) in its calculation for precision These must be removed when converting to MySQL.