1

DBD::mysql and DBD::mSQL

Version

Versions 1.20xx and 1.21_xx.

Version 1.20xx (even numbers) are the stable line, which is maintained for bug and portability fixes only. Version 1.21_xx (odd numbers) is used for development of the driver: All new features or interface modifications will be done in this line until it finally becomes 1.22xx.

Author and Contact Details

The driver author is Jochen Wiedmann. He can be contacted via the mailing list *Msql-Mysql-modules@tcx.se*.

The drivers include modules that emulate the old Msql and Mysql Perl extensions using the DBI and DBD modules.

Supported Database Versions and Options

MySQL and mSQL are freely available lightweight database servers. MySQL has a rich feature set while mSQL is very minimalist.

The `DBD::mysql` driver 1.20xx supports all MySQL versions since around 3.20. The `DBD::mysql` driver 1.21_xx supports MySQL 3.22 or later. Support for more recent versions may or may not be added at a later time.

The `DBD::mSQL` drivers 1.20xx and 1.21_xx support all mSQL versions upto and including mSQL 2.0.x.

Connect Syntax

The `DBI->connect()` Data Source Name, or *DSN*, can be one of the following:

```
DBI:mysql:database=$db
DBI:mysql:database=$db;<attrs>

DBI:mSQL:database=$db
DBI:mSQL:database=$db;<attrs>
```

The optional attributes are specified as a semicolon separated list of key/value pairs. Some significant attributes include:

host=$host

The host name you want to connect to, by default *localhost*.

msql_configfile=$file

Load driver specific settings from the given file, by default *InstDir/msql.conf*. This path is fixed at compile time.

mysql_compression=1

For slow connections, you may wish to compress the traffic between your client and the engine. If the MySQL engine supports it, this can be enabled by using this attribute. Default is off.

There are no driver specific attributes applicable to the `connect()` method.

Numeric Data Handling

MySQL has five sizes of integer data type, each of which can be signed (the default) or unsigned (by adding the word UNSIGNED after the type name).

```
Name      Bits      Signed Range          Unsigned Range
--------- ---- ----------------------   ----------------------
TINYINT     8         -128..127          0..255
SMALLINT   16       -32768..32767        0..65535
MEDIUMINT  24     -8388608..8388607      0..16777215
INTEGER    32  -2147483648..2147483647   0..4294967295
BIGINT     64      -(2*63)..(2**63-1)    0..(2**64)
```

The type INT can be used as an alias for INTEGER. Other aliases include FLOAT4=FLOAT, FLOAT8=DOUBLE, INT1=TINYINT, INT2=SMALLINT, INT3=MEDIUMINT, INT4=INT, INT8=BIGINT, MIDDLEINT=MEDIUMINT.

Note that all arithmetic is done using signed BIGINT or DOUBLE values, so you shouldn't use unsigned big integers larger than the largest signed big integer (except with bit functions). Note that -, +, and * will use BIGINT arithmetic when both arguments are INTEGER values. This means that if you multiply two big integers (or multiply the results from functions that return integers), you may get unexpected results if the result is bigger than 9223372036854775807.

MySQL has three main types of *non-integer* data type: FLOAT, DOUBLE, and DECIMAL.

In what follows, the letter `M` is used for the *maximum display size* or *PRECISION* in ODBC and DBI terminology. The letter `D` is used for the number of digits that may follow the decimal point. (*SCALE* in ODBC or DBI terminology).

Maximum display size (PRECISION) and number of fraction digits (SCALE) are typically not required. For example, if you use just ''DOUBLE'' then default values will be silently inserted.

DOUBLE(M,D)

A normal-size (double-precision) floating-point number. Allowable values are -1.7976931348623157E+308 to -2.2250738585072014E-308, 0 and 2.2250738585072014E-308 to 1.7976931348623157E+308.

REAL and DOUBLE PRECISION can be used as aliases for DOUBLE.

FLOAT(M,D)

A small (single-precision) floating-point number. Allowable values are -3.402823466E+38 to -1.175494351E-38, 0 and -1.175494351E-38 to 3.402823466E+38.

FLOAT(M)

A floating-point number. Precision (*M*) can be 4 or 8. `FLOAT(4)` is a single-precision number and `FLOAT(8)` is a double-precision number. These types are like the FLOAT and DOUBLE types described above. `FLOAT(4)` and `FLOAT(8)` have the same ranges as the corresponding FLOAT and DOUBLE types, but their display size and number of decimals is undefined. This syntax is provided for ODBC compatibility.

DECIMAL(M,D)

The DECIMAL type is an unpacked floating-point number type. NUMERIC is an alias for DECIMAL. It behaves like a CHAR column; ''unpacked'' means the number is stored as a string, using one character for each digit of the value, the decimal point, and, for negative numbers, the - sign). If *D* is 0, values will have no decimal point or fractional part. The maximum range of DECIMAL values is the same as for DOUBLE, but the actual range for a given DECIMAL column may be constrained by the choice of *M* and *D*.

NUMERIC can be used as an alias for DECIMAL.

The numeric data types supported by mSQL are much more restricted:

```
INTEGER - corresponds to MySQL's INTEGER type.
UINT    - corresponds to MySQL's INTEGER UNSIGNED type.
```

```
REAL    - corresponds to MySQL's REAL type.
```

The driver returns all data types, including numbers, as strings. It thus puts no restriction on size of PRECISION or SCALE.

String Data Handling

The following string types are supported by MySQL, quoted from *mysql.info* where *M* denotes the maximum display size or PRECISION:

CHAR(M)

A fixed-length string that is always right-padded with spaces to the specified length. The range of *M* is 1 to 255 characters.

VARCHAR(M)

A variable-length string. NOTE: Trailing spaces are removed by the database when the value is stored (this differs from the ANSI SQL specification). The range of *M* is 1 to 255 characters.

ENUM('value1','value2',...)

An enumeration. A string object that can have only one value, chosen from the specified list of values (or NULL). An ENUM can have a maxiumum of 65535 distinct values.

SET('value1','value2',...)

A set. A string object that can have zero or more values, each of which must be chosen from the specified list of values. A SET can have a maximum of 64 members.

CHAR and VARCHAR types have a limit of 255 bytes. Binary characters, including the NUL byte, are supported by all string types. (Use the `$dbh->quote()` method for literal strings).

These aliases are also supported:

```
BINARY(num)      CHAR(num) BINARY
CHAR VARYING     VARCHAR
LONG VARBINARY   BLOB
LONG VARCHAR     TEXT
VARBINARY(num)   VARCHAR(num) BINARY
```

With `DBD::mysql`, the *ChopBlanks* attribute is always on: The MySQL engine itself removes spaces from the strings right end. As far as I know, this ''feature'' cannot be turned off. Another ''feature'' is that CHAR and VARCHAR columns are always case-insensitive in comparisons and sort operations, unless you use the *BINARY* attribute, as in:

```
CREATE TABLE foo (A VARCHAR(10) BINARY)
```

With versions of MySQL after 3.23, you can perform a case-insensitve comparison of strings with the BINARY operator modifier:

```
SELECT * FROM table WHERE BINARY column = "A"
```

National language characters are handled in comparisons following the coding system that was specified at compile-time, by default ISO-8859-1. Non-ISO coding systems, and in particular UTF-16, are not supported.

Strings can be concatenated using the `CONCAT(s1, s2, . . . )` SQL function.

The mSQL engine (and hence the `DBD::mSQL` driver) only supports the `CHAR(`*M*`)` string type, which corresponds to the MySQL's `VARCHAR(`*M*`)` type, and a `TEXT(`*M*`)` type which is a cross between a CHAR and a BLOB. Also, mSQL has no way to concatenate strings.

Date Data Handling

The following date and time types are supported by MySQL, quoted from *mysql.info*:

DATE

A date. The supported range is `0000-01-01` to `9999-12-31`. MySQL displays *DATE* values in `YYYY-MM-DD` format, but allows you to assign values to *DATE* columns using these formats:

```
YYMMDD
YYYYMMDD
YY.MM.DD
YYYY.MM.DD
```

Where `.` may be any non-numerical separator and a two digit year is assumed to be 20*YY* if *YY* is less than 70.

DATETIME

A date and time combination. The supported range is `0000-01-01 00:00:00` to `9999-12-31 23:59:59`. MySQL displays *DATETIME* values in *YYYY-MM-DD HH:MM:SS* format, but allows you to assign values to *DATETIME* columns using the formats shown for DATE above but with '' HH:MM:SS'' appended.

TIMESTAMP(M)

A timestamp. The range is *1970-01-01 00:00:00* to sometime in the year 2032 (or 2106, depending on the OS specific type *time_t*). MySQL displays *TIMESTAMP* values in `YYYYMMDDHHMMSS`, `YYMMDDHHMMSS`, `YYYYMMDD`, or `YYMMDD` format, depending on whether *M* is 14 (or missing), 12, 8 or 6, but allows you to assign values to *TIMESTAMP* columns using either strings or numbers. This output format behavior disagrees with the manual, so check your version because the behavior may change.
A *TIMESTAMP* column is useful for recording the time of an *INSERT* or *UPDATE* operation because it is automatically set to the time of the last operation if you don't

give it an value yourself. You can also set it to the current time by giving it a *NULL* value.

TIME

A time. The range is -838:59:59 to 838:59:59. MySQL displays *TIME* values in `HH:MM:SS` format, but allows you to assign values to *TIME* columns using any of these formats: `HH:MM:SS`, `HHMMSS`, `HHMM`, or `'HH'`.

YEAR

A year. The allowable values are 1901 to 2155, and 0000. MySQL displays *YEAR* values in `YYYY` format. On input, 2 digits years in the range 00-69 are assumed to be 2000-2069. (*YEAR* is a new type for MySQL 3.22.)

If you are using two-digit-years as in `YY-MM-DD` (dates) or `YY` (years), then they are converted into 2000-2069 and 1970-1999, repectively. Thus, MySQL has no Y2K problem, but a Y2070 problem!

In MySQL 3.23 this will be changed to 2000-2068 and 1969-1999, following the X/Open Unix standard*.

```
* See:

    http://www.unix-systems.org/version2/whatsnew/year2000.html>
```

The `NOW()` function, and it's alias `SYSDATE`, allow you to refer to the current date and time in SQL.

The `DATE_FORMAT(date, format)` function can be used to format date and time values using `printf`-like format strings.

MySQL has a rich set of functions operating on dates and times, including `DAYOFWEEK(date)` (1 = Sunday, ..., 7 = Saturday), `WEEKDAY(date)` (0 = Monday, ..., 6 = Sunday), `DAYOFMONTH(date)`, `DAYOFYEAR(date)`, `MONTH(date)`, `DAYNAME(date)`, `MONTHNAME(date)`, `WEEK(date)`, `YEAR(date)`, `HOUR(time)`, `MINUTE(time)`, `SECOND(time)`, `DATE_ADD(date, interval)` (*interval* being something like ''2 HOURS''), and `DATE_SUB(date, interval)`.

The following SQL expression can be used to convert an integer ''seconds since 1-jan-1970 GMT'' value to the corresponding database date time:

```
FROM_UNIXTIME(seconds_since_epoch)
```

and the reverse:

```
UNIX_TIMESTAMP(timestamp)
```

MySQL does no automatic time zone adjustments.

The mSQL database supports these date/time types:

```
DATE  - corresponds to MySQL's DATE type
TIME  - corresponds to MySQL's TIME type
```

The only date format supported by mSQL is `DD-MMM-YYYY`, where `MMM` is the three character English abbreviation for the month name. The only time format supported by mSQL is `HH:MM:SS`.

LONG/BLOB Data Handling

These are MySQL's BLOB types, quoted from *mysql.info*:

```
TINYBLOB   / TINYTEXT    maximum length of 255 (2^8 - 1)
BLOB       / TEXT        maximum length of 65535 (2^16 - 1)
MEDIUMBLOB / MEDIUMTEXT  maximum length of 16777215 (2^24 - 1)
LONGBLOB   / LONGTEXT    maximum length of 4294967295 (2^32 - 1)
```

Binary characters in all BLOB types are allowed. None of the types are passed as strings of hex digits. The *LongReadLen* and *LongTruncOk* types are not supported.

The maximum length of `bind_param()` parameter values is only limited by the maximum length of an SQL statement. By default that's 1MB but can be extended to just under 24MB by changing the *mysqld* variable `max_allowed_packet`.

No TYPE or other attributes need to be given to `bind_param()` when binding these types.

The only blob type supported by mSQL is TEXT. This is a cross between a traditional VARCHAR type and a BLOB. An *average* width is specified, and data longer than *average* is automatically stored in an overflow area in the table.

Other Data Handling issues

The driver versions 1.21_xx do support the `type_info()` methods. Version 1.20xx doesn't.

MySQL supports automatic conversions between data types wherever it's reasonable. mSQL, on the other hand, supports none.

Transactions, Isolation and Locking

Both engines do not support transactions.

Since both mSQL and MySQL currently execute statements from multiple clients one at a time (atomic), and don't support transactions, there's no need for a default locking behavior to protect transaction isolation.

With MySQL locks can be explicitly obtained on tables. For example:

```
LOCK TABLES $table1 READ, $table2 WRITE
```

Locks are released with any subsequent LOCK TABLES statement, by dropping a connection or with an explicit

```
UNLOCK TABLES
```

There are also user defined locks that can be manipulated with the `GET_LOCK()` and `RELEASE_LOCK()` SQL functions. You can't automatically lock rows or tables during select statements; you have to do it explicitly.

And, as you might guess, mSQL doesn't support any kind of locking at the moment.

No-Table Expression Select Syntax

With MySQL you can select constant expressions without naming a table. For example:

```
SELECT NOW()
```

You can't do so with mSQL.

Table Join Syntax

Joins are supported with the usual syntax:

```
SELECT * FROM a,b WHERE a.field = b.field
```

or, alternatively:

```
SELECT * FROM a JOIN b USING field
```

Outer joins are supported by MySQL, not mSQL, with

```
SELECT * FROM a LEFT OUTER JOIN b ON condition
```

Outer joins in MySQL are always left outer joins.

Table and Column Names

MySQL table and column names may have at most 64 characters. mSQL table and column names are limited to 35 characters.

MySQL can use all alphanumeric characters defined in the current character set as table and column names. MySQL also allows you to use the _ and $ characters in names. A table name can't start with a number. It is unknown whether non-alphanumeric table and column names can be used with mSQL.

Neither mSQL nor MySQL support putting quotes around table or column names.

Table names are limited by the fact that tables are stored in files and the table names are really file names. In particular, the case sensitivity of table names depends on the underlying file system.

Column names are case insensitive with MySQL and sensitive with mSQL, but both engines store them without case conversions.

Names can include national character set characters (with the 8th bit set) in MySQL but not mSQL.

Case Sensitivity of LIKE Operator

With MySQL, case sensitivity of *all* character comparison operators, including LIKE, requires on the presence of the BINARY attribute on at least one of the fields—either on the field type in the CREATE TABLE statement or on the field name in the comparison operator. However, you can always force case insensitivity using the `TOLOWER` function.

mSQL has three LIKE operators: LIKE is case sensitive, CLIKE is case insensitive, and RLIKE uses UNIX style regular expressions.

Row ID

MySQL doesn't have row ID's. mSQL has a pseudo column `_rowid`.

The mSQL `_rowid` column value is numeric and, since mSQL doesn't automatically convert strings to numbers, you must take care not to quote the value when using it in later select statements.

Note that there's a risk that the row identified by a `_rowid` value you just fetched may have been deleted and possibly replaced by a different row by the time you use the row ID value moments later.

Automatic Key or Sequence Generation

All MySQL integers can have an AUTO_INCREMENT attribute. That is, given a table:

```
CREATE TABLE a (
  id INTEGER AUTO_INCREMENT NOT NULL PRIMARY KEY,
  ...)
```

and a statement:

```
INSERT INTO a (id, ...) VALUES (NULL, ...)
```

a unique ID will be generated automatically (similarly if the ID field had not been mentioned in the insert statement at all). The generated ID can later be retrieved with

```
$sth->{mysql_insertid}              (1.21_xx)
$sth->{insertid}                    (1.20xx)
```

or, if you've used `$dbh->do` and not prepare/execute, then use:

```
$dbh->{mysql_insertid}              (1.21_xx)
$dbh->do("SELECT LAST_INSERT_ID()"); (1.20_xx)
```

MySQL does not support sequence generators directly, but they can be emulated with a little care using `UPDATE seq SET id=last_insert_id(id+1)`. Refer to the MySQL manual for details.

The mSQL database supports sequence generators, but just one per table. After executing:

```
CREATE SEQUENCE on A
```

you can later do:

```
SELECT _seq FROM A
```

to fetch the value. You can't refer directly to the sequence from an insert statement; instead, you have to fetch the sequence value and then execute an insert with that value. There seems to be no protection against someone inserting rows without using the sequence.

Automatic Row Numbering and Row Count Limiting

Neither engine supports automatic row numbering of select statement results.

MySQL does support row count limitations with:

```
SELECT * FROM A LIMIT 10
```

to retrieve the first 10 rows only, and:

```
SELECT * FROM A LIMIT 20, 10
```

to retrieve rows 20-29, with the count starting at 0.

Parameter Binding

Neither engine supports placeholders, but the `DBD::mysql` and `DBD::mSQL` drivers provide full emulation. Question marks are used as placeholders, as in:

```
$dbh->do("INSERT INTO A VALUES (?, ?)", undef, $id, $name);
```

The `:1` placeholder style is not supported.

In the above example, the driver attempts to guess the inserted columns datatype by looking at Perl's datatype. This is fine with MySQL, because MySQL can deal with expressions like:

```
INSERT INTO A (id) VALUES ('2')
```

if *id* is a numeric column. But this doesn't apply to mSQL so you sometimes need to force a datatype, either by using:

```
$dbh->do("INSERT INTO A VALUES (?, ?)", undef, int($id), $name);
```

or by using the TYPE attribute of the `bind_param()` method:

```
use DBI qw(:sql_types);
$sth = $dbh->prepare("INSERT INTO A VALUES (?, ?)");
$sth->bind_param(1, $id,   SQL_INTEGER);
$sth->bind_param(2, $name, SQL_VARCHAR);
$sth->execute;
```

Unsupported values of the TYPE attribute do not currently generate a warning.

Stored Procedures

Neither mSQL nor MySQL have a concept of stored procedures.

Table Metadata

The 1.21_xx version of the drivers to support the *table_info()* method. The 1.20xx versions don't.

To obtain information on a generic table, you can use the query

```
LISTFIELDS $table
```

This will return a statement handle without result rows. The *TYPE*, *NAME*, ... attributes are describing the table.

With MySQL you can use:

```
SHOW INDEX FROM $table
```

to retrieve information on a table's indexes, in particular a primary key. The information will be returned in rows. The `DBD::mSQL` driver does support a similar thing *via*

```
LISTINDEX I<$table> I<$index>
```

with *$index* being the name of a given index.

Driver-specific Attributes and Methods

The following driver specific database handle attributes are supported:

mysql_info

mysql_thread_id

mysql_insertid

These are corresponding to the C calls `mysql_info()`, `mysql_thread_id()`, and `mysql_insertid()`, repectively.

The following driver specific statement handle attributes are supported:

mysql_use_result

mysql_store_result

With `DBD::mysql` there are two different ways the driver fetches results from the server. With *mysql_store_result* enabled, it fetches all rows at once, creating a result table in memory and returns it to the caller (a 100% row cache).

With *mysql_use_result*, it returns rows to the application as they are fetched. This is less memory consuming on the client side, but should not be used in situations where multiple people can query the database, because it can block other applications. (Don't confuse that with locking!)

mysql_insertid

A previously generated *auto_increment* column value, if any.

mysql_is_blob

mysql_is_key

msql_is_num

mysql_is_num

msql_is_pri_key

mysql_is_pri_key

These are returning a array ref with the given flags set for any column of the result set. Note you may use these with the LISTFIELDS query to obtain information about the columns of a table.

mysql_max_length

Unlike the *PRECISION* attribute, this returns the true actual maximum length of the particular data in the current result set. This can be helpful, for example, when displaying ASCII tables.

This attribute doesn't work with *mysql_use_result* enabled, since it needs to look at all the data.

msql_table

mysql_table

Similar to *NAME*, but the table names and not the column names are returned.

msql_type

mysql_type

These are similar to the *TYPE* attribute, but they return the respective engines native type, like `DBD::mysql`*::FIELD_TYPE_ENUM()* or `DBD::mSQL`*::IDX_TYPE()*.

msql_type_name

mysql_type_name

Similar to *msql_type* and *mysql_type*, but column names are returned, that you can use in a CREATE TABLE statement.

A single private method called `admin()` is supported. It provides a range of administration functions:

```
$rc = $drh->func('createdb', $db, $host, $user, $password, 'admin');
$rc = $drh->func('dropdb',   $db, $host, $user, $password, 'admin');
$rc = $drh->func('shutdown',      $host, $user, $password, 'admin');
$rc = $drh->func('reload',        $host, $user, $password, 'admin');

$rc = $dbh->func('createdb', $database, 'admin');
$rc = $dbh->func('dropdb',   $database, 'admin');
$rc = $dbh->func('shutdown',            'admin');
$rc = $dbh->func('reload',              'admin');
```

These correspond to the respective commands of *mysqladmin* and *msqladmin*.

Positioned updates and deletes

Neither positioned updates nor deletes are supported by MySQL or mSQL.

Differences from the DBI Specification

The `DBD::mysql` driver cannot turn off the *ChopBlanks* attribute, because the database server always chops trailing blanks.

Both `DBD::mysql` and `DBD::mSQL` do not fully parse the statement until it's executed. Thus attributes like *$sth*-{NUM_OF_FIELDS}> are not available until after `$sth`-execute> has been called. This is valid behaviour but is important to note when porting applications written originaly for other drivers.

Also note that many statement attributes cease to be available after fetching all the result rows or calling the `finish()` method.

URLs to More Database/Driver Specific Information

For MySQL:

```
http://www.mysql.com/
```

For mSQL:

```
http://www.hughes.com.au/
```

Concurrent use of Multiple Handles

The number of database and statement handles is limited by memory only. There are no restrictions on their concurrent use.