NAME

DBD::mSQL / DBD::mysql − mSQL and mysql drivers for the Perl5 Database Interface (DBI)

SYNOPSIS

```
    use DBI;

    $dbh = DBI->connect("DBI:mSQL:$database:$hostname:$port",
                        undef, undef);

        or

    $dbh = DBI->connect("DBI:mysql:$database:$hostname:$port",
                        $user, $password);

    @databases = DBD::mysql::dr->func( $hostname, '_ListDBs' );
    @tables = $dbh->func( '_ListTables' );

    $sth = $dbh->prepare("LISTFIELDS $table");
    $sth->execute;
    $sth->finish;

    $sth = $dbh->prepare("SELECT * FROM foo WHERE bla");
    $sth->execute;
    $numRows = $sth->rows;
    $numFields = $sth->{'NUM_OF_FIELDS'};
    $sth->finish;

    $rc = $drh->func( $database, '_CreateDB' );
    $rc = $drh->func( $host, $database, '_CreateDB' );
    $rc = $drh->func( $database, '_DropDB' );
    $rc = $drh->func( $host, $database, '_DropDB' );
```

DESCRIPTION

<DBD::mysql> and <DBD::mSQL> are the Perl5 Database Interface drivers for the mysql, mSQL 1.*x* and mSQL 2.*x* databases. The drivers are part of the *mysql-modules* and *Msql-modules* packages, respectively.

Class Methods

connect

```
    use DBI;

    $dbh = DBI->connect("DBI:mSQL:$database", undef, undef);
    $dbh = DBI->connect("DBI:mSQL:$database:$hostname", undef, undef);
    $dbh = DBI->connect("DBI:mSQL:$database:$hostname:$port",
                        undef, undef);

        or

    use DBI;
    $dbh = DBI->connect("DBI:mysql:$database", $user, $password);
    $dbh = DBI->connect("DBI:mysql:$database:$hostname",
                        $user, $password);
    $dbh = DBI->connect("DBI:mysql:$database:$hostname:$port",
                        $user, $password);
```

A `database` must always be specified.

The hostname, if not specified or specified as ‘’, will default to an mysql or mSQL daemon running on the local machine on the default port for the UNIX socket.

Should the mysql or mSQL daemon be running on a non-standard port number, you may explicitly state the port number to connect to in the `hostname` argument, by concatenating the *hostname* and *port number* together separated by a colon (`:`) character.

Private MetaData Methods

ListDBs

```
    @dbs = $dbh->func("$hostname:$port", '_ListDBs');
```

Returns a list of all databases managed by the mysql daemon or mSQL daemon running on `$hostname`, port `$port`. This method is rarely needed for databases running on `localhost`: You should use the portable method

```
    @dbs = DBI->data_sources("mysql");

        or

    @dbs = DBI->data_sources("mSQL");
```

whenever possible. It is a design problem of this method, that there's no way of supplying a host name or port number to `data_sources`, that's the only reason why we still support `ListDBs`. :−(

ListTables

```
    @tables = $dbh->func('_ListTables');
```

Once connected to the desired database on the desired mysql or mSQL mSQL daemon with the `DBI-`*connect()*> method, we may extract a list of the tables that have been created within that database.

`ListTables` returns an array containing the names of all the tables present within the selected database. If no tables have been created, an empty list is returned.

```
    @tables = $dbh->func( '_ListTables' );
    foreach $table ( @tables ) {
        print "Table: $table\n";
      }
```

ListFields
Deprecated, see the section on */COMPATIBILITY ALERT* below.

ListSelectedFields
Deprecated, see the section on */COMPATIBILITY ALERT* below.

Database Manipulation

CreateDB

DropDB

```
$rc = $drh->func( $database, '_CreateDB' );
$rc = $drh->func( $database, '_DropDB' );

  or

$rc = $drh->func( $host, $database, '_CreateDB' );
$rc = $drh->func( $host, $database, '_DropDB' );
```

These two methods allow programmers to create and drop databases from DBI scripts. Since mSQL disallows the creation and deletion of databases over the network, these methods explicitly connect to the mSQL daemon running on the machine `localhost` and execute these operations there.

It should be noted that database deletion is *not prompted for* in any way. Nor is it undo-able from DBI.

```
Once you issue the dropDB() method, the database will be gone!
```

These methods should be used at your own risk.

DATABASE HANDLES

The DBD::mysql driver supports the following attributes of database handles (read only):

```
$infoString = $dbh->{'info'};
$threadId = $dbh->{'thread_id'};
```

These correspond to *mysql_info()* and *mysql_tread_id()*, respectively.

STATEMENT HANDLES

The statement handles of DBD::mysql and DBD::mSQL support a number of attributes. You access these by using, for example,

```
my $numFields = $sth->{'NUM_OF_FIELDS'};
```

Note, that most attributes are valid only after a successfull *execute*. An `undef` value will returned in that case. The most important exception is the `mysql_use_result` attribute: This forces the driver to use mysql_use_result rather than mysql_store_result. The former is faster and less memory consuming, but tends to block other processes. (That's why mysql_store_result is the default.)

To set the `mysql_use_result` attribute, use either of the following:

```
my $sth = $dbh->prepare("QUERY", { "mysql_use_result" => 1});
```

or

```
my $sth = $dbh->prepare("QUERY");
$sth->{"mysql_use_result"} = 1;
```

Of course it doesn't make sense to set this attribute before calling the `execute` method.

Column dependent attributes, for example *NAME*, the column names, are returned as a reference to an array. The array indices are corresponding to the indices of the arrays returned by *fetchrow* and similar methods. For example the following code will print a header of table names together with all rows:

```
my $sth = $dbh->prepare("SELECT * FROM $table");
if (!$sth) {
    die "Error:" . $dbh->errstr . "\n";
}
if (!$sth->execute) {
    die "Error:" . $sth->errstr . "\n";
}
my $names = $sth->{'NAME'};
my $numFields = $sth->{'NUM_OF_FIELDS'};
for (my $i = 0;  $i < $numFields;  $i++) {
    printf("%s%s", $$names[$i], $i ? "," : "");
}
print "\n";
while (my $ref = $sth->fetchrow_arrayref) {
    for (my $i = 0;  $i < $numFields;  $i++) {
        printf("%s%s", $$ref[$i], $i ? "," : "");
    }
    print "\n";
}
```

For portable applications you should restrict yourself to attributes with capitalized or mixed case names. Lower case attribute names are private to DBD::mSQL and DBD::mysql. The attribute list includes:

ChopBlanks
: this attribute determines whether a *fetchrow* will chop preceding and trailing blanks off the column values. Chopping blanks does not have impact on the *max_length* attribute.

insertid
: MySQL has the ability to choose unique key values automatically. If this happened, the new ID will be stored in this attribute. This attribute is not valid for DBD::mSQL.

is_blob
: Reference to an array of boolean values; TRUE indicates, that the respective column is a blob. This attribute is valid for MySQL only.

is_key
: Reference to an array of boolean values; TRUE indicates, that the respective column is a key. This is valid for MySQL only.

is_num
: Reference to an array of boolean values; TRUE indicates, that the respective column contains numeric values.

is_pri_key
: Reference to an array of boolean values; TRUE indicates, that the respective column is a primary key. This is only valid for MySQL and mSQL 1.0.x: mSQL 2.x uses indices.

is_not_null
: A reference to an array of boolean values; FALSE indicates that this column may contain NULL's. You should better use the *NULLABLE* attribute above which is a DBI standard.

length

max_length
: A reference to an array of maximum column sizes. The *max_length* is the maximum physically present in the result table, *length* gives the theoretically possible maximum. *max_length* is valid for MySQL only.

NAME
A reference to an array of column names.

NULLABLE
A reference to an array of boolean values; TRUE indicates that this column may contain NULL's.

NUM_OF_FIELDS
Number of fields returned by a *SELECT* or *LISTFIELDS* statement. You may use this for checking whether a statement returned a result: A zero value indicates a non-SELECT statement like *INSERT*, *DELETE* or *UPDATE*.

table
A reference to an array of table names, useful in a *JOIN* result.

type
A reference to an array of column types. It depends on the DBMS, which values are returned, even for identical types. mSQL will return types like &DBD::mSQL::INT_TYPE, &DBD::msql::TEXT_TYPE etc., MySQL uses &DBD::mysql::FIELD_TYPE_SHORT, &DBD::mysql::FIELD_TYPE_STRING etc.

COMPATIBILITY ALERT

As of version 0.70 DBD::mSQL has a new maintainer. Even more, the sources have been completely rewritten in August 1997, so it seemed apropriate to bump the version number: Incompatibilities are more than likely.

Recent changes:

New connect method
DBD::mSQL and DBD::mysql now use the new *connect* method as introduced with DBI 0.83 or so. For compatibility reasons the old method still works, but the driver issues a warning when he detects use of the old version. There's no workaround, you must update your sources. (Sorry, but the change was in DBI, not in DBD::mysql and DBD::mSQL.)

_ListFields returning statement handle
As of Msql-modules 1.1805, the private functions

```
    $dbh->func($table, "_ListFields");
```

and

```
    $sth->func("_ListSelectedFields");
```

no longer return a simple hash, but a statement handle. (*_ListSelectedFields* is a stub now which just returns `$self`.) This should usually not be visible, when your statement handle gets out of scope. However, if your database handle (`$dbh` in the above example) disconnects, either because you explicitly disconnect or because he gets out of scope, and the statement handle is still active, DBI will issue a warning for active cursors being destroyed.

The simple workaround is to execute `$sth->finish` or to ensure that `$sth` gets out of scope before `$dbh`. Sorry, but it was obvious nonsense to support two different things for accessing the basically same thing: A *M* (y)SQL result.

The drivers do not conform to the current DBI specification in some minor points. For example, the private attributes *is_num* or *is_blob* have been written *IS_NUM* and *IS_BLOB*. For historical reasons we continue supporting the capitalized names, although the DBI specification now reserves capitalized names for standard names, mixed case for DBI and lower case for private attributes and methods.

We currently consider anything not conforming to the DBI as deprecated. It is quite possible that we remove support of these deprecated names and methods in the future. In particular these includes:

`$sth->func($table, '_ListSelectedFields')`

highly deprecated, all attributes are directly accessible via the statement handle. For example instead of

```
  $ref = $sth->func($table, '_ListSelectedFields')
  my @names = $ref->{'NAME'}
```

you just do a

```
  my @names = @{$sth->{'NAME'}};
```

Capitalized attribute names

Deprecated, should be replaced by the respective lower case names.

BUGS

The *port* part of the first argument to the connect call is implemented in an unsafe way when using mSQL. In fact it is just stting the environment variable MSQL_TCP_PORT during the connect call. If another connect call uses another port and the handles are used simultaneously, they will interfere. I doubt that this will ever change.

Please speak up now (June 1997) if you encounter additional bugs. I'm still learning about the DBI API and can neither judge the quality of the code presented here nor the DBI compliancy. But I'm intending to resolve things quickly as I'd really like to get rid of the multitude of implementations ASAP.

When running ''make test'', you will notice that some test scripts fail. This is due to bugs in the respective databases, not in the DBI drivers:

Nulls

mSQL seems to have problems with NULL's: The following fails with mSQL 2.0.1 running on a Linux 2.0.30 machine:

```
    [joe@laptop Msql-modules-1.18]$ msql test
    Welcome to the miniSQL monitor.  Type \h for help.
    mSQL > CREATE TABLE foo (id INTEGER, name CHAR(6))\g
    Query OK.  1 row(s) modified or retrieved.
    mSQL > INSERT INTO foo VALUES (NULL, 'joe')\g
    Query OK.  1 row(s) modified or retrieved.
    mSQL > SELECT * FROM foo WHERE id = NULL\g
    Query OK.  0 row(s) modified or retrieved.
    +----------+------+
    | id       | name |
    +----------+------+
    +----------+------+
    mSQL >
```

Blanks

mysql has problems with Blanks on the right side of string fields: They get chopped of. (Tested with mysql 3.20.25 on a Linux 2.0.30 machine.)

```
[joe@laptop Msql-modules-1.18]$ mysql test
Welcome to the mysql monitor.  Commands ends with ; or \g.
Type 'help' for help.
mysql> CREATE TABLE foo (id INTEGER, bar CHAR(8));
Query OK, 0 rows affected (0.10 sec)
mysql> INSERT INTO foo VALUES (1, ' a b c ');
Query OK, 1 rows affected (0.00 sec)
mysql> SELECT * FROM foo;
1 rows in set (0.19 sec)
+------+--------+
| id   | bar    |
+------+--------+
|    1 |  a b c |
+------+--------+
mysql> quit;
[joe@laptop Msql-modules-1.18]$ mysqldump test foo

[deleted]

INSERT INTO foo VALUES (1,' a b c');
```

AUTHOR

DBD::mSQL has been primarily written by Alligator Descartes (*descarte@hermetica.com*), who has been aided and abetted by Gary Shea, Andreas Koenig and Tim Bunce amongst others. Apologies if your name isn't listed, it probably is in the file called 'Acknowledgments'. As of version 0.80 the maintainer is Andreas König. Version 2.00 is an almost complete rewrite by Jochen Wiedmann.

COPYRIGHT

This module is Copyright (c)1997 Jochen Wiedmann, with code portions Copyright (c)1994-1997 their original authors. This module is released under the 'Artistic' license which you can find in the perl distribution.

This document is Copyright (c)1997 Alligator Descartes. All rights reserved. Permission to distribute this document, in full or in part, via email, Usenet, ftp archives or http is granted providing that no charges are involved, reasonable attempt is made to use the most current version and all credits and copyright notices are retained (the *AUTHOR* and *COPYRIGHT* sections). Requests for other distribution rights, including incorporation into commercial products, such as books, magazine articles or CD−ROMs should be made to Alligator Descartes <*descarte@hermetica.com*>.

ADDITIONAL DBI INFORMATION

Additional information on the DBI project can be found on the World Wide Web at the following URL:

```
http://www.hermetica.com/technologia/perl/DBI
```

where documentation, pointers to the mailing lists and mailing list archives and pointers to the most current versions of the modules can be used.

Information on the DBI interface itself can be gained by typing:

```
perldoc DBI
```

right now!

Interface (DBI)"