**MQLMySQLClass.mqh**

**“CLASSic version”**

V3.0

Interface Library Reference

Rev.2019-12-17

**Introduction**

The interface library MQLMySQLClass.mqh consists of functions set can be used for MySQL database connectivity. Any MQL program can include interface library to make possible of using MySQL database. Simple schema of interface listed below:

Metatrader 5

MQL 5

Library

MQLMySQLClass.mqh

MQLMySQL.dll

libmysql.dll

**MySQL**

**database**

The MQL5 program make calls to interface library, then interface library calls special functions from standard **libmysql.dll** through the wrapper **MQLMySQL.dll**. The **libmysql.dll** dynamic link library can be found in any MySQL related software or in MySQL distribution package. It is prepare connection to the MySQL database and send queries to.

To make possible executing SELECT statements and fetching data from database, the **MQLMySQL.dll** library was developed. It has number of functions to handle database cursors and retrieve data using string type (**char\*/wchar\_t\*** type). Maximal number of cursors opened concurrently is set to 256. This value can be changed by recompiling **MQLMySQL.DLL** library. Highly recommended to do not use so complicated SELECT statements. This can make data retrieving easy. If you need to use complex SELECT statement, you may create *database view* based on your query and build your SELECT statement based on view.

The functionality of **MQLMySQL** can be extended easily for your other needs; in this case you may study API functions of **libmysql.dll** (<http://dev.mysql.com/doc/refman/5.0/en/c-api-functions.html>) and implement the functions you need.

Project consists of:

|  |  |
| --- | --- |
| Filename | Description |
| MQL5\Libraries\libmysql.dll | MySQL standard library with C++ API. |
| MQL5\Libraries\libssl-1\_1-x64.dll | \*Required for libmysql.dll |
| MQL5\Libraries\libcrypto-1\_1-x64.dll | \*Required for libmysql.dll |
| MQL5\Libraries\MQLMySql.dll | Developed library to extend **libmysql.dll** functionality for MQL programs. |
| MQL5\Libraries\MQLMySql.def | Definition file of **MQLMySql.dll** library, should be located in the same directory with DLL. |
| MQL5\Include\MQLMySqlClass.mqh | Interface library which provides access to MySQL database for MQL programs. |
| MQL5\Scripts\MySQL-XXXc.mq5 | Examples of using MQLMySQL.mqh interface library |
| MQLMySQL Technical Reference.docx | The document you are reading. |

\*Note: For MQL5 x64 Terminal the version of MySQL library is 8.0.18 and it is required libcrypto-1\_1-x64.dll and libssl-1\_1-x64.dll. If you will not include them, you’ll receive error 126 on MQL side (DLL cannot be loaded).

**Important:** To enable interface between expert advisor and MySQL database you need to allow DLL imports (Expert Advisor’s properties → “Common” tab → Allow DLL imports).

**Development specification**

|  |  |
| --- | --- |
| Name | Description |
| DLL Development | MS Visual Studio 2017 (MSVC) |
| MQL Development | MetaTrader v5.0 build 2280 x64 |
| Operating System | Windows 7 x64 |
| Database | MySQL v8.0.18 x64 |

**Interface functions**

You may create connections (up to 32) to the MySQL database server by using interface functions. The **Execute** function can be used to send SQL queries or special commands of MySQL database (such as USE, SET and so on) and can be called after connection was created by **Connect**. To close any connection created you may use **Disconnect** function.

**Class CMySQL**

| Return type | Name | Parameters | Description |
| --- | --- | --- | --- |
| bool | Connect | This function can be used to establish connection to MySQL database server. The return value is **true** on success or **false** on fail, you need to call ***LastError()*** of ***LastErrorMessage()*** functions to see the problem details.  This function can be called from ***OnInit()*** function of MQL program. | |
| **string** pHost | DNS name or IP-address of MySQL server |
| **string** pUser | Database user (f.e. root) |
| **string** pPassword | Password of user (f.e. Zok1LmVdx) |
| **string** pDatabase | Database name (f.e. metatrader) |
| **int** pPort | TCP/IP port of database listener (f.e. 3306) |
| **string** pSocket | UNIX socket (for sockets or named pipes) |
| **int** pClientFlag | Combination of the flags for features (usual 0) |
| bool | Connect | This function can be used to establish connection to MySQL by using predefined database credentials. You can use ***SetCredentials()*** function to define database credentials or you can use ***LoadCredentials()*** to load them from INI file. | |
| void | Disconnect | This function can be called to close database connection.  This function can be called from ***OnDeinit()*** function of MQL program. | |
| bool | Execute | This function can be used for sending non-SELECT SQL queries to the MySQL database server when connection was established by ***Connect()***. When execution of SQL command succeded – this function will return ***true***, otherwise – ***false***. To see error details please call ***LastError()*** of ***LastErrorMessage()*** | |
| **string** pQuery | An SQL query |
| string | DllVersion | Function can be used to get information about version of **MQLMySql.dll** | |
| void | SetTrace | Enable or disable debug tracing. For testing purposes tracing can be set to ***true***, for release – to ***false***. | |
| **bool** pState | State (***true*** – tracing enabled, ***false*** - disabled) |
| void | SetCredentials | This function sets database credentials for further ***Connect().*** | |
| **string** pHost | DNS name or IP-address of MySQL server |
| **string** pUser | Database user (f.e. root) |
| **string** pPassword | Password of user (f.e. Zok1LmVdx) |
| **string** pDatabase | Database name (f.e. metatrader) |
| **int** pPort | TCP/IP port of database listener (f.e. 3306) |
| **string** pSocket | UNIX socket (for sockets or named pipes) |
| **int** pClientFlag | Combination of the flags for features (usual 0) |
| void | LoadCredentials | This function loads database credentials from INI file for further ***Connect()***. | |
| **string** pINI | INI file name with specific keys and database credentials. |
| int | RowsAffected | This function returns the number of rows affected by last DML operation (INSERT/UPDATE/DELETE). Also it can be used to retrieve the number of rows was returned by last SELECT statement, but it’s not preferable. | |
| int | GetConnectID | Return internal identifier of connection. Required by ***CMySQLCursor*** class only. | |
| bool | GetTrace | Return tracing state (true/false). Required by ***CMySQLCursor*** class only. | |
| int | LastError | Return error code of last error raised by database connection/execution. | |
| string | LastErrorMessage | Return error message of last error raised by database connection/execution. | |

**Class CMySQLCursor**

| Return type | Name | Parameters | Description |
| --- | --- | --- | --- |
| bool | Open | This function can be used to select data from database (execution of SELECT statement). The return value is **true** on success or **false** on fail, you need to call ***LastError()*** of ***LastErrorMessage()*** functions to see the problem details.  This function can be called from ***OnInit()*** function of MQL program. | |
| **CMySQL** pConnection | Database connection object |
| **string** pQuery | SELECT statement |
| void | Close | This function can be used to close previously opened cursor. | |
| int | Rows | This function return the number of rows was selected by SELECT query when cursor was opened. | |
| bool | Fetch | Function ***Fetch()*** loads single row from database to the internal buffer where fields data can be reached by next functions: ***FieldAsString(), FieldAsInt(), FieldAsDouble(), FieldAsDatetime().***  Return value is true on success, or false when no more rows to fetch are available (reached the end of row set). | |
| string | FieldAsString | Return string value of field. | |
| **int** pField | Field number, starting from 0. |
| int | FieldAsInt | Return integer value of field. | |
| **int** pField | Field number, starting from 0. |
| double | FieldAsDouble | Return double value of field. | |
| **int** pField | Field number, starting from 0. |
| datetime | FieldAsDatetime | Return datetime value of field. | |
| **int** pField | Field number, starting from 0. |
| int | LastError | Return error code of last error raised by cursor execution. | |
| string | LastErrorMessage | Return error message of last error raised by cursor execution. | |

**Additions**

1. Reading .ini files

Sometimes it is better to keep database credentials outside the MQL program. For this reason the function ReadIni was integrated into **MQLMySQL.dll**:

| Return type | Name | Parameters | Description |
| --- | --- | --- | --- |
| String | ReadIni | Read the data from .ini file and return the value of key. | |
| string pFileName | The name of .ini file |
| string pSection | The name of section |
| string pKey | The name of key |

**Example:** Your database credentials stored in file “C:\Metatrader5\MQL5\Experts\MyConnection.ini”

[MYSQL]

Server = 127.0.0.1

User = root

Password = Adm1n1str@t0r

Database = mysql

Port = 3306

The reading data from this .ini file into MQL variable can be done like:

string vServer = ReadIni("C:\\Metatrader5\\MQL5\\Experts\\MyConnection.ini", "MYSQL", "Server");

1. Using multi-statements query

For transferring big arrays of data from Metatrader to database and reduce the number of calls and network traffic, you may use multi-statements queries. It looks like usual queries separated by semicolon “;”:

string Query = "INSERT INTO my\_table(field1) VALUES (1); UPDATE my\_table SET field1 = 2;";

To execute such query you can use **MySqlExecute** function. But you have to open the database connection with ***pClientFlag*** = CLIENT\_MULTI\_STATEMENTS (decimal value 65536). For example:

CMQLMySQL\* DB = new CMQLMySQL();

bool res = DB.Connect(vHost, vUser, vPass, vDatabase, 3306, "", CLIENT\_MULTI\_STATEMENTS);

Here is a list of possible pClientFlag constants:

#define CLIENT\_LONG\_PASSWORD 1 /\* new more secure passwords \*/

#define CLIENT\_FOUND\_ROWS 2 /\* Found instead of affected rows \*/

#define CLIENT\_LONG\_FLAG 4 /\* Get all column flags \*/

#define CLIENT\_CONNECT\_WITH\_DB 8 /\* One can specify db on connect \*/

#define CLIENT\_NO\_SCHEMA 16 /\* Don't allow database.table.column \*/

#define CLIENT\_COMPRESS 32 /\* Can use compression protocol \*/

#define CLIENT\_ODBC 64 /\* Odbc client \*/

#define CLIENT\_LOCAL\_FILES 128 /\* Can use LOAD DATA LOCAL \*/

#define CLIENT\_IGNORE\_SPACE 256 /\* Ignore spaces before '(' \*/

#define CLIENT\_PROTOCOL\_41 512 /\* New 4.1 protocol \*/

#define CLIENT\_INTERACTIVE 1024 /\* This is an interactive client \*/

#define CLIENT\_SSL 2048 /\* Switch to SSL after handshake \*/

#define CLIENT\_IGNORE\_SIGPIPE 4096 /\* IGNORE sigpipes \*/

#define CLIENT\_TRANSACTIONS 8192 /\* Client knows about transactions \*/

#define CLIENT\_RESERVED 16384 /\* Old flag for 4.1 protocol \*/

#define CLIENT\_SECURE\_CONNECTION 32768 /\* New 4.1 authentication \*/

#define CLIENT\_MULTI\_STATEMENTS 65536 /\* Enable/disable multi-stmt support \*/

#define CLIENT\_MULTI\_RESULTS 131072 /\* Enable/disable multi-results \*/

#define CLIENT\_PS\_MULTI\_RESULTS 262144 /\* Multi-results in PS-protocol \*/

**Examples**

**Include MQLMySQL into your MQL project:**

#include <MqlMySqlClass.mqh>

**Connection to MySQL v1:**

CMQLMySQL\* DB = new CMQLMySQL();

if (!DB.Connect("localhost", "root", "ioctrl", "metatrader", 3306, "", 0))

{

Print("Error #", DB.LastError(), ": ", DB.LastErrorMessage());

return (1);

}

else Print ("Connected!");

… your working code …

DB.Disconnect();

delete DB;

**Connection to MySQL v2 (set database credentials before connect):**

CMQLMySQL\* DB = new CMQLMySQL();

DB.SetCredentials("localhost", "root", "ioctrl", "metatrader", 3306, "", 0);

if (!DB.Connect())

{

Print("Error #", DB.LastError(), ": ", DB.LastErrorMessage());

return (1);

}

else Print ("Connected!");

… your working code …

DB.Disconnect();

delete DB;

**Connection to MySQL v3 (load database credentials from .ini file before connect):**

CMQLMySQL\* DB = new CMQLMySQL();

DB.LoadCredentials("C:\\Metatrader5\\MQL5\\Experts\\MyConnection.ini");

if (!DB.Connect())

{

Print("Error #", DB.LastError(), ": ", DB.LastErrorMessage());

return (1);

}

else Print("Connected!");

… your working code …

DB.Disconnect();

delete DB;

**Execution of non-SELECT statements:**

string query;

query = "INSERT INTO Metatrader.Ticks\_" + Symbol() + " (price\_ask, price\_bid, spread) " +

"VALUES (" + DoubleToString(SymbolInfoDouble(Symbol(), SYMBOL\_ASK), \_Digits) + ", " +

DoubleToString(SymbolInfoDouble(Symbol(), SYMBOL\_BID), \_Digits) + ", " +

DoubleToString(((double)SymbolInfoInteger(Symbol(), SYMBOL\_SPREAD) / \_Point), 0) + ")";

if (!DB.Execute(query))

{

Print("Error #", DB.LastError(), ": ", DB.LastErrorMessage(),

" Problem with query: ", query);

}

**Disconnection from MySQL:**

DB.Disconnect();

Or

delete DB; -- deletion of object also will call disconnection routine

**Selecting data from MySQL table:**

int Rows, i;

string Query;

string time;

CMQLCursor \*Cursor = new CMQLCursor();

int vId;

string vSubSymbol;

datetime vStartDate;

double vBid;

time = "\'" + TimeToString(TimeCurrent(), TIME\_DATE|TIME\_MINUTES) + "\'";

Query = "SELECT id, sub\_symbol, start\_date, bid FROM stored\_symbols " +

"WHERE broker\_id = " + IntegerToString(gBrokerID) +

" AND symbol = \'TGH4\'" +

" AND start\_date <= " + time + " LIMIT 10";

if (Cursor.Open(DB, Query)) // cursor opened within connection created as DB object

{

Rows = Cursor.Rows();

Print (Rows, " row(s) selected.");

for (i=0; i<Rows; i++)

if (Cursor.Fetch())

{

vId = Cursor.FieldAsInt(0); // id as integer

vSubSymbol = Cursor.FieldAsString(1); // sub\_symbol as string

vStartDate = Cursor.FieldAsDatetime(2); // start\_date as datetime

vBid = Cursor.FieldAsDouble(3); // bid as double

Print ("ROW[", i, "]: id = ", vId,

", sub\_symbol = ", vCode,

", start\_date = ", TimeToString(vStartDate, TIME\_DATE|TIME\_MINUTES),

", bid = ", vBid);

}

Cursor.Close(); // NEVER FORGET TO CLOSE CURSOR !!!

}

else

{

Print ("Cursor opening failed. Error: ", Cursor.LastErrorMessage());

}

delete Cursor;