**MQLMySQL.mqh**

**“Old School version”**

V3.0

Interface Library Reference

Rev.2019-12-14

**Introduction**

The interface library MQLMySQL.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

MQLMySQL.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 currently opened cursors 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 make selection from view.

The functionality of **MQLMySQL** can be extended easily for 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\MQLMySql.mqh | Interface library which provides access to MySQL database for MQL programs. |
| MQL5\Scripts\MySQL-XXX.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).

**Interface variables**

There are some interface’s variables can be used for error handling.

|  |  |  |
| --- | --- | --- |
| Type | Name | Description |
| int | MySqlErrorNumber | The number of last MySQL error |
| string | MySqlErrorDescription | The description of last MySQL error |

**Interface functions**

You may create connections (up to 32) to the MySQL database server by using interface functions. The **MySqlExecute** 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 **MySqlConnect**. To close any created connection you may use **MySqlDisconnect** function.

| Return type | Name | Parameters | Description |
| --- | --- | --- | --- |
| int | MySqlConnect | This function can be used to establish connection to MySQL database server. The return value is database connection identifier. If **MySqlConnect** returns “-1”, this means that an error was raised, you need to check ***MySQLErrorNumber*** of ***MySqlErrorDescription*** 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) |
| void | MySqlDisconnect | This function can be called to close database connection. It has only one parameter – database connection identifier (which can be obtained from **MySqlConnect** function).  This function can be called from ***OnDeinit()*** function of MQL program. | |
| **int** pConnection | Database connection identifier |
| bool | MySqlExecute | This function can be used for sending non-SELECT SQL queries to the MySQL database server when connection was established by **MySqlConnect**. When execution of SQL command succeded – this function will return “true”, otherwise – “false”. To see error details please check ***MySQLErrorNumber*** or ***MySqlErrorDescription*** variables. | |
| **int** pConnection | Database connection identifier |
| **string** pQuery | An SQL query |
| string | MySqlVersion | Function can be used to get information about version of **MQLMySql.dll** | |
| string | MySqlGetRowField | This function retrieves one string value from fetched row. After cursor opening, you should fetch row from database, and after that it would be possible to get value of any row’s field. | |
| **int** pCursorID | Cursor identifier, returned by function **MySqlCursorOpen** |
| **int** pField | Field number in SELECT clause (started from 0) |
| int | MySqlCursorOpen | This function opens cursor for SELECT statement and returns cursor identifier. You may open up to 256 concurrent cursors (this restriction was made just in **MQLMySql.dll**, but this value can be changed). In case when any error raised during SELECT execution, this function returns “-1”. | |
| **int** pConnection | Database connection identifier |
| **string** pQuery | SQL query (SELECT command) |
| void | MySqlCursorClose | This function used to close any opened cursor and free memory.  **Important:** Do not forget to close cursor after using. | |
| **int** pCursorID | Cursor identifier, returned by function **MySqlCursorOpen** |
| int | MySqlCursorRows | This function counts the number of rows was selected by cursor. It can be used for fetching all rows in cycle. | |
| **int** pCursorID | Cursor identifier, returned by function **MySqlCursorOpen** |
| bool | MySqlCursorFetchRow | This function should be used to fetch one row from cursor’s record set into temporary buffer. After this operation it would be possible to get values from row’s fields. | |
| **int** pCursorID | Cursor identifier, returned by function **MySqlCursorOpen** |
| int | MySqlGetFieldAsInt | You may use this function after row fetching to get field value, represented as INTEGER. All fetched values are stored using STRING data type, conversion of type based on MQL functions. | |
| **int** pCursorID | Cursor identifier, returned by function **MySqlCursorOpen** |
| **int** pField | Field number in SELECT clause (started from 0) |
| double | MySqlGetFieldAsDouble | This function returns representation of field’s value using DOUBLE data type. | |
| **int** pCursorID | Cursor identifier, returned by function **MySqlCursorOpen** |
| **int** pField | Field number in SELECT clause (started from 0) |
| datetime | MySqlGetFieldAsDatetime | This function returns representation of field’s value using DATETIME data type. | |
| **int** pCursorID | Cursor identifier, returned by function **MySqlCursorOpen** |
| **int** pField | Field number in SELECT clause (started from 0) |
| string | MySqlGetFieldAsString | This function returns representation of field’s value using STRING data type. Synonym to MySqlGetRowField function | |
| **int** pCursorID | Cursor identifier, returned by function **MySqlCursorOpen** |
| **int** pField | Field number in SELECT clause (started from 0) |
| int | MySqlRowsAffected | 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. | |
| **int** pConnection | Database connection identifier |

**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:

int DB = MySqlConnect(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 <MqlMySql.mqh>

**Connection to MySQL:**

int DBConnection = MySqlConnect("localhost", "root", "ioctrl", "metatrader", 3306, "", 0);

if (DBConnection==-1)

{

Print("Error #", MySqlErrorNumber, ": ", MySqlErrorDescription);

return (1);

}

else Print ("Connected!");

**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 (!MySqlExecute(DBConnection, query))

{

Comment("Error #", MySqlErrorNumber, ": ", MySqlErrorDescription,

"\nProblem with query: ", query);

}

**Disconnection from MySQL:**

MySqlDisconnect(DBConnection);

**Selecting data from MySQL table:**

int Cursor,Rows;

string Query, sub\_symbol;

string time;

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

Query = "SELECT sub\_symbol FROM stored\_symbols " +

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

" AND symbol = \'TGH4\'" +

" AND start\_date <= " + time +

" AND end\_date > " + time + " LIMIT 1";

Cursor = MySqlCursorOpen(DBConnection, Query);

if (Cursor >= 0) // cursor opened

{

Rows = MySqlCursorRows(Cursor);

if (Rows > 0) // record exists

{

if (MySqlCursorFetchRow(Cursor))

{

// retrieve sub-symbol

sub\_symbol = MySqlGetFieldAsString(Cursor, 0);

}

}

MySqlCursorClose(Cursor);

}