MySQL Library

MySQL Library for use with CoDeSys Control v3

Short Description:

This Library allows your CODESYS v3.5 application, to connect with a MySQL Server database. It gives you the opportunity to read and store process data to an external or internal MySQL database, without any 3rd party software or driver. Whether you want to store data for long-term logging, exchange data with 3rd party applications or collecting mass of machine data for your next IoT project, it's all possible with our MySQL-Library for CODESYS.

Revision:

0.1	02.08.2016	Kevin Rohn	Created first revision with Quick-Start G.
0.2	06.08.2016	Kevin Rohn	Added FB Desc. and example Desc,
1.0	08.09.2016	Kevin Rohn	Release

Pfänder GmbH
Germany
74585 Hausen am Bach
Bachstrasse 15

Internet: www.pfaender.de
Phone: +49 7958 9800 0
E-Mail: support@pfaender.de

Table of Contents

Product Description

This Library allows your CODESYS v3.5 application, to connect with a MySQL-Server database. It gives you the opportunity to read and store process data to an external or internal MySQL database, without any 3rd party software or driver. Whether you want to store data for long-term logging, exchange data with 3rd party applications or collecting mass of machine data for your next IoT project, it's all possible with our MySQL-Library for CODESYS.

You will be able to connect to a MySQL-Server, which is directly running on the same system, as your PLC (local) *or* you can also connect to a remote hosted MySQL-Server, anywhere in the world.

MySQL is a fast, easy-to-handle, and lightweight database system and compatible with most of the well-known operating systems.

Functional Description

With the Library you can perform most of the common MySQL Commands.
For Example: *INSERT*, *UPDATE*, *SELECT*, *DELETE*, *ALTER, DROP,* etc.

The following topic will describe the different function boxes and functions, which you can use within your application.

All function boxes are built on best practice coding guidelines and are very easy to use. We've tried to design this library as simple as possible. For "worry free" troubleshooting we've defined meaningful error categories and output messages.

Function Box “MySQL_Open”

Short Description:

This function block opens the MySQL connection.

Detailed Description:

The opening process starts on a raising edge of “xStart”. The “xStart” var will be resetted at the end of the process.

If an error occurs during the closing process, you will see an ERROR code and the current execute state, this helps you to identify the problem.

InOut:

Scope	Name	Type	Initial	Comment
Input	sHost	STRING		MySQL-Server hostname or IP-Address
	uiPort	UINT	3306	MySQL-Server port (Default: 3306)
	sDatabase	STRING		MySQL database name
	sUsername	STRING		MySQL database user
	sPassword	STRING		MySQL user password
Inout	xStart	BOOL		Start execution on a rising edge
	MySQL_Connec-tion	*MySQL_ConnectionS-tring*		MySQL connection string which holds the server and login information
Output	xConnected	BOOL		TRUE when connected
	eError	*ERROR*		Error type
	sExeute_State	STRING(200)		Current execute state

Function Box “MySQL_Close”

Short Description:

This function block closes the MySQL connection.

Detailed Description:

The closing process starts on a raising edge of “xStart”. The “xStart” var will be resetted at the end of the process.
If an error occurs during the closing process, you will see an ERROR code and the current execute state, which helps you to identify the problem.

InOut:

Scope	Name	Type	Comment
Inout	xStart	BOOL	Start execution on a rising edge
	MySQL_Connec-tion	*MySQL_ConnectionS-tring*	MySQL connection string which holds the server and login information
Output	eError	*ERROR*	Error type
	sExeute_State	STRING(200)	Current execute state

Function Box “MySQL_Exec”

Short Description:

This function block executes a MySQL command without no result like INSERT, UPDATE, DELETE, ALTER, DROP, etc.

Detailed Description:

The MySQL commands needs to be built with an array. The length of the array is defined by the GLOBAL CONSTANTS. The FBox is built on the common behavior model “Etrig” so, you are able to handle and see the current process state within your application.

Info about MySQL_Command:

Non-Numeric SQL parameter values must be marked with a quote (apostrophe = ‘). To use an apostrophe inside a CODESYS String you need do declare it with a dollar char + apostrophe’ ($) or with $27.

e.g.:

MySQL_Command[0] := ‘SELECT * FROM table WHERE name =’

MySQL_Command[1] := ‘$27nonNumberValue$27 ‘;

Caution: Maximum length for identifiers are 250 bytes.

InOut:

Scope	Name	Type	Comment	Inherited from
Input	xExecute	BOOL	Rising edge: Action start Falling edge: Resets outputs If a falling edge occurs before the function block has completed its action, the outputs operate in the usual manner and are only reset if either the action is completed or in the event of an error. In this case, the corresponding output values (xDone, xError, iError) are present at the outputs for exactly one cycle.	ETrig
Output	xDone	BOOL	Action successfully completed	ETrig
	xBusy	BOOL	Function block active	ETrig

Scope	**Name**	**Type**	**Comment**	**Inherited from**
	xError	BOOL	TRUE: error occurred, function block aborts action FALSE: no error	ETrig
Inout	MySQL_Con-nection	*MySQL_ConnectionString*	MySQL connection string which holds the server and login information	
Input	MySQL_Com-mand	ARRAY [0.. gc_MySQL_iStatementMax] OF STRING(gc_MySQL_iState-mentLength)	MySQL command	
Output	eError	*ERROR*	Error type	
	sExe-cute_State	STRING(200)	Current execute state	

Function Box “MySQL_Query”

Short Description:

This function block executes a MySQL command with results like SELECT, etc.

Detailed Description:

The MySQL commands needs to be built with an array. The length of the array is defined by the GLOBAL CONSTANTS. The FBox is built on the common behavior model “Etrig” so, you are able to handle and see the current process state within your application.

Info about MySQL_Command:

Non-Numeric SQL parameter values must be marked with a quote (apostrophe = ‘). To use an apostrophe inside a CODESYS String you need do declare it with an dollar char + apostrophe’ ($) or with $27.

e.g.:

MySQL_Command[0] := ‘SELECT * FROM table WHERE name =’

MySQL_Command[1] := ‘$27nonNumberValue$27 ‘;

Caution: Maximum length for identifiers are 250 bytes and for field data 500 bytes,

InOut:

Scope	**Name**	**Type**	**Comment**	**Inherited from**
Input	xExecute	BOOL	Rising edge: Action start Falling edge: Resets outputs If a falling edge occurs before the function block has completed its action, the outputs operate in the usual manner and are only reset if either the action is completed or in the event of an error. In this case, the corresponding output values (xDone, xError, iError) are present at the outputs for exactly one cycle.	ETrig
Output	xDone	BOOL	Action successfully completed	ETrig
	xBusy	BOOL	Function block active	ETrig
	xError	BOOL	TRUE: error occurred, function block aborts action	ETrig

			FALSE: no error	
Input	MySQL_Com-mand	ARRAY [0.. gc_MySQL_iState-mentMax] OF STRING(gc_MySQL_iState-mentLength)	MySQL command	
Inout	MySQL_Connec-tion	*MySQL_Connecti-onString*	MySQL connection string which holds the server and login information	
	stResultSet	*MySQL_DataSet*	Query result	
Output	eError	*ERROR*	Error type	
	sExecute_State	STRING(200)	Current execute state	

Function Box “MySQL_ConnectionString”

Short Description:

This function block holds the login and server information.

Detailed Description:

The function block needs to be defined one time in the PLC Program. It holds all the parameters from the other MySQL-FBoxes instances.

Caution: For stable functionality, please do not “call” this functionblock inside the PLC Program

InOut:

Scope	Name	Type	Comment
Input	xConnected	BOOL	TRUE: Successfully connected and logged in.
	sHost	STRING	MySQL-Server host address as hostname or IP-address
	uiPort	UINT	MySQL-Server port (Default: 3306)
	sUsername	STRING	MySQL database username
	sPassword	STRING	MySQL database user password
	sDatabase	STRING	MySQL database schema name
	stServerInfo	*MySql_ServerInfo*	MySQL-Server information which are stored in the MySQL_ServerInfo structure
	internal_Socket-Handle	DWORD	Socket handler
	internal_Se-maBuffer	DWORD	sema buffer
	abTxBuffer	ARRAY [0.. gc_MySQL_dwTxBufferSize] OF BYTE	Transmit buffer
	abRxBuffer	ARRAY [0..gc_ gc_MySQL_dwRxBufferSize] OF BYTE	Receive buffer

Function “MySQL_GetStringValue”

Short Description:

This function converts the requested field data and returns it as readable value in “sValue”.

Detailed Description:

To select the data from the response DataSet, you have to declare the iRow number and the iColumn number. If you specified an existing row and column, you will get the result as string in “sValue”.

Caution: Fields are limited to 500 bytes

InOut:

Scope	**Name**	**Type**	**Comment**
Return	MySql_GetStringValue	DWORD	
Input	iRow	INT	Row index (First index starts at 1)
	iColumn	INT	Column index (First index starts at 1)
Inout	stQueryResult	*MySQL_DataSet*	MySql query set
	sValue	STRING(500)	Query string value (Limited to 500 Bytes)

Global Parameter list

Detailed Description:

This parameters can be changed for configure optional settings. Please only change the parameters, if you know what you do. The default parameters are tested and are for best practise use.

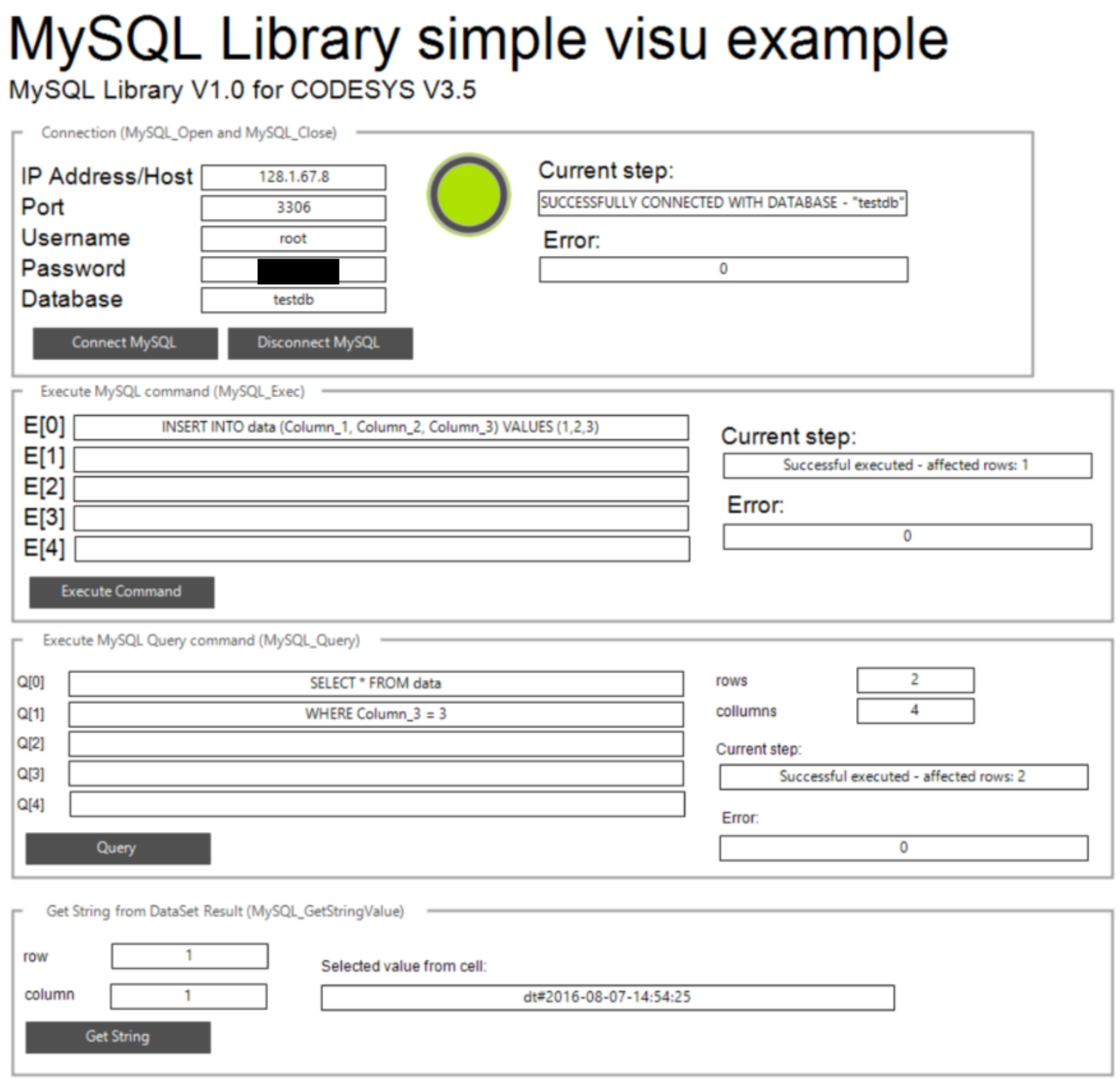
Scope	Name	Type	Initial	Comment
Constant	gc_MySQL_wClientFlag	WORD	16#8601	Used in: MySQL_Open | Desc: Client Flag | Default: 16#8601
	gc_MySQL_wExtClientFlag	WORD	16#3	Used in: MySQL_Open | Desc: Extended Client Flag (For More Information see documentation | Default:
	gc_MySQL_dwMaxPackets	DWORD	1024	Used in: MySQL_Open | Desc: Number of maximum bytes in a client packet | Default: 1024
	gc_MySQL_tConnTimeOut	TIME	TIME#10s0ms	Used in: MySQL_Open, MySQL_Close | Desc: Timeout for Open and Close Process | Default: 10 Sec
	gc_MySQL_iStatementMax	INT	10	Used in: MySQL_Exec, MySQL_Query | Desc: Upper Bound of the arrSQLStatement | Default: 10
	gc_MySQL_iStatementLength	INT	100	Used in: MySQL_Exec, MySQL_Query | Desc: Size in Bytes of the arrSQLStatement | Default: 100
	gc_MySQL_xClearDataSet	BOOL	FALSE	Used in: MySQL_Query | Desc: Clear Result Set before Use (Caution: needs a lot of cycles) | Default: FALSE
	gc_MySQL_tReceiveResponseTime	TIME	TIME#50ms	Used in: MySQL_Query | Desc: Wait time before processing received response data (Info: expand this value for slow Servers) | Default: 50MS
	gc_MySQL_dwMaxColumns	DWORD	25	Used in: MySQL_DataSet | Desc: Number of maximum columns which can be processed | Default: 25 columns
	gc_MySQL_dwMaxRows	DWORD	40	Used in: MySQL_DataSet | Desc: Number of maximum rows which can be processed | Default: 40 rows
	gc_MySQL_dwMaxRowSize	DWORD	500	Used in: MySQL_DataSet | Desc: Maximum lenght in bytes of one row which can be processed | Default: 500 bytes
	gc_MySQL_dwTxBufferSize	DWORD	8000	Used in: MySQL_ConnectionString | Desc: Transmit Buffer Size | Default: 8000
	gc_MySQL_dwRxBufferSize	DWORD	40000	Used in: MySQL_ConnectionString | Desc: Receive Buffer Size | Default 40000
	gc_MySQL_bCharsetNumber	BYTE	16#8	Used in: MySQL_GetStringValue | Desc: Character Set | Default: 16#08 for Latin1
	gc_MySQL_dwMaxColumnNameLength	DWORD	80	Used in: <PRIVATE> | Desc: Maximum lenght of a column name in bytes | Default: 80

Example applications

This topic describes the demo project. It’s made for you, to get directly started with the MySQL-Library.

Example application with CFC and web visualisation

This example brings a visualisation, which shows you how easy it is to use the MySQL-Library. You can open and close a MySQL connection directly from the visu. You are also able to perform MySQL execute commands and MySQL execute commands with data results, like SELECT.



MySQL Library simple visu example

MySQL Library V1.0 for CODESYS V3.5

Connection (MySQL_Open and MySQL_Close)

IP Address/Host	128.1.67.8
Port	3306
Username	root
Password	[redacted]
Database	testdb

Current step: SUCCESSFULLY CONNECTED WITH DATABASE - "testdb"

Error: 0

Execute MySQL command (MySQL_Exec)

E[0]	INSERT INTO data (Column_1, Column_2, Column_3) VALUES (1,2,3)
E[1]	
E[2]	
E[3]	
E[4]	

Current step: Successful executed - affected rows: 1

Error: 0

Execute MySQL Query command (MySQL_Query)

Q[0]	SELECT * FROM data
Q[1]	WHERE Column_3 = 3
Q[2]	
Q[3]	
Q[4]	

rows: 2

collumns: 4

Current step: Successful executed - affected rows: 2

Error: 0

Get String from DataSet Result (MySQL_GetStringValue)

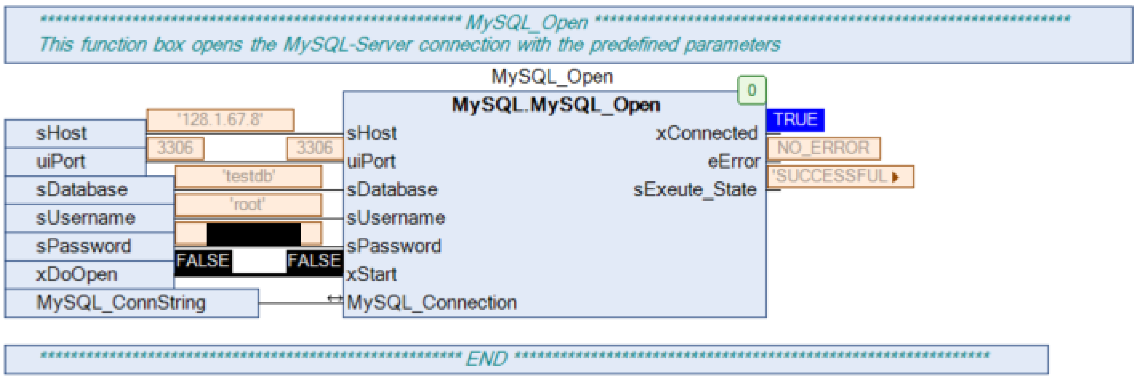
row: 1

column: 1

Selected value from cell: dt#2016-08-07-14:54:25

The first group box holds the parameters for the opening process. By clicking on “Connect MySQL” it tries to open the MySQL connection. If it was successfully, the current step will show the connection state and the lamp will switch on.

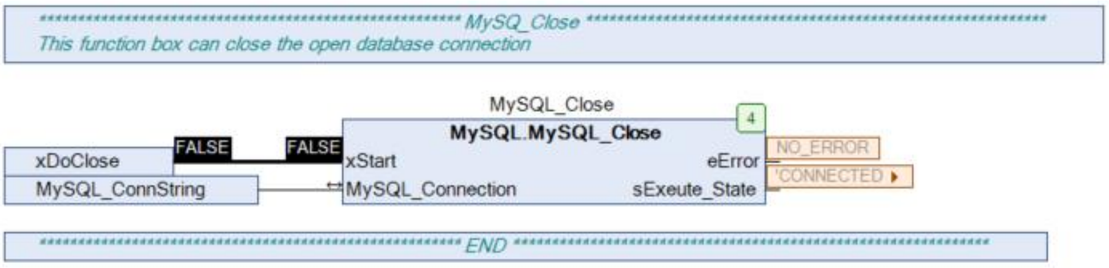
To perform this process, the following FBox is used in the program:



```
************************************************** MySQL_Open **************************************************
This function box opens the MySQL-Server connection with the predefined parameters
```

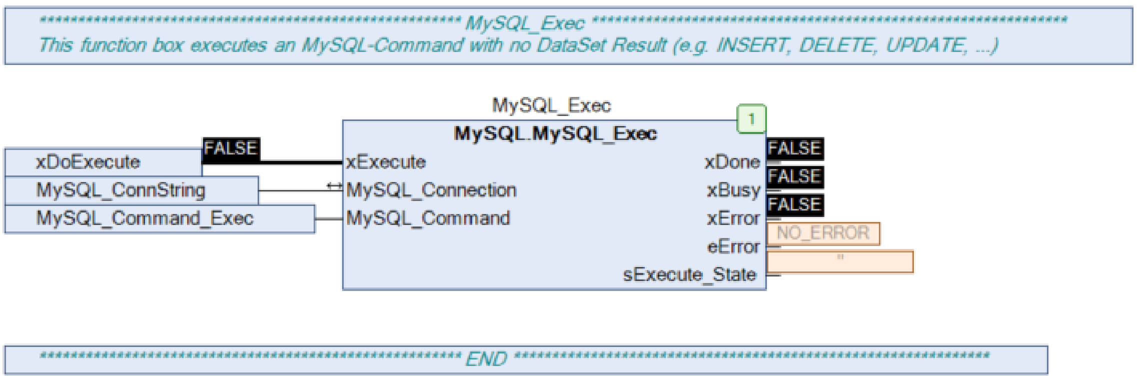
By clicking on “Disconnect MySQL” the MySQL session will be destroyed, the MySQL connection will be closed and the lamp will switch off.

For this step, the MySQL_Close Fbox is used:



```
************************************************** MySQ_Close **************************************************
This function box can close the open database connection
```

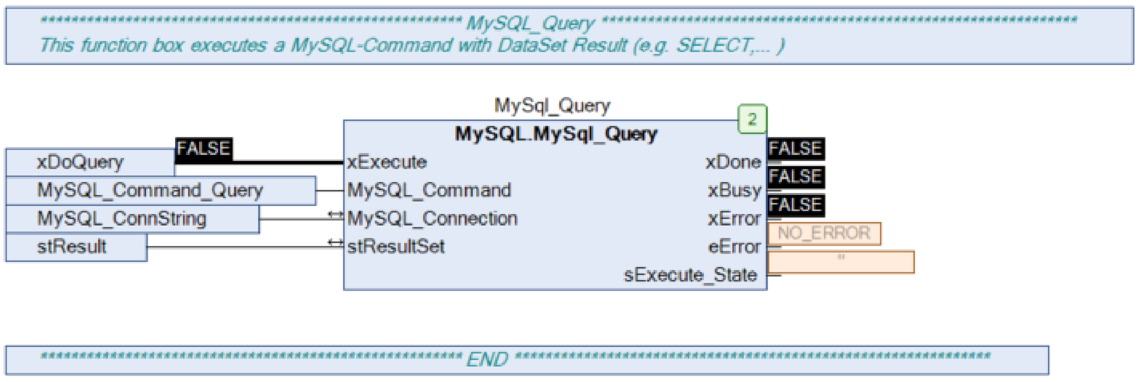
To perform a MySQL execute command, we first have to create the MySQL command string, and this can be done, by creating an array. We are using the textboxes E[0] to E[4] to do this. After the MySQL command was created, we can perform the execute command, by clicking on “Execute Command”.



```
************************************************** MySQL_Exec **************************************************
This function box executes an MySQL-Command with no DataSet Result (e.g. INSERT, DELETE, UPDATE, ...)
```

It’s also possible to query data from the database. To do this, we are using the FBox “MySQL_Query”. But first at all we also need to create the MySQL command string, by creating an array. This is the same step as creating an execute command.

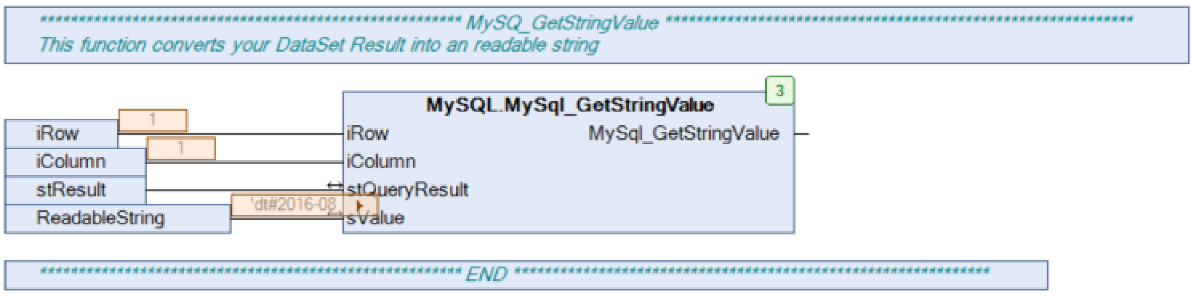
If the “Query” button is pressed in the visu, than it will execute the SELECT command.



```
*********************************************************** MySQL_Query ***********************************************************************
This function box executes a MySQL-Command with DataSet Result (e.g. SELECT,... )
```

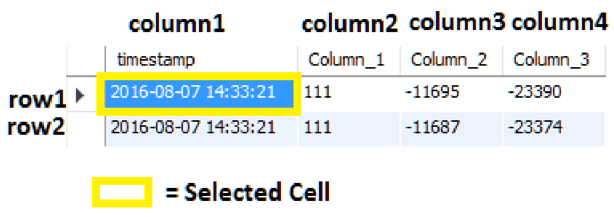
```
*********************************************************** END *********************************************************************
```

Now we want to read the queried DataSet. To do this, use the function MySQL_GetStringValue. To select the data from a cell in a readable string, just define the row and the column.



```
*********************************************************** MySQ_GetStringValue *******************************************************************
This function converts your DataSet Result into an readable string
```

```
*********************************************************** END *********************************************************************
```



	column1	column2	column3	column4
	timestamp	Column_1	Column_2	Column_3
row1	2016-08-07 14:33:21	111	-11695	-23390
row2	2016-08-07 14:33:21	111	-11687	-23374

🟨 = Selected Cell

Example application with Structured Text (ST)

The example application with structured text is programmed for demo purposes. With the demo application, you can do quick function tests. Here is a list of tests, which are described in this topic:

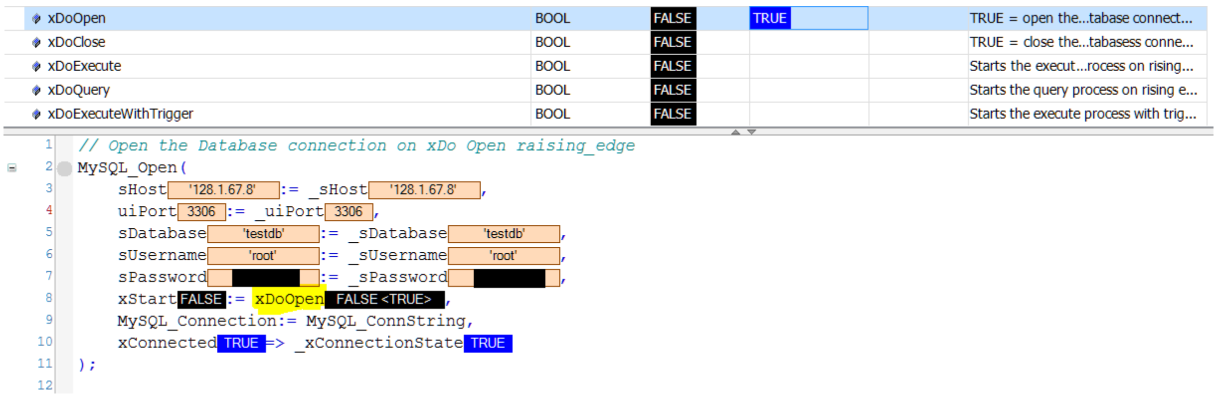
- Open the MySQL-Server connection
- Close the MySQL-Server connection
- Execute a MySQL command with no response result
- Execute a MySQL command with response result

Open the MySQL-Server connection

First at all, you need to configure the host parameters. To do this, configure the parameters under Section “//** Connection Parameters”.

```
// ** Connection Parameters
_sHost                      : STRING := 'hostaddress';         // Hostname or IP-adress
_uiPort                     : UINT   := 3306;                  // MySQL Server Port
_sDatabase                  : STRING := 'testdb';              // Database Name
_sUsername                  : STRING := 'root';                // Database Username
_sPassword                  : STRING := 'password';            // User password
```

To open the database connection with the specified parameters, you need to change “xDoOpen” to TRUE. If the FB finished the opening process, the “xStart” var will be resetted.



xDoOpen	BOOL	FALSE	TRUE	TRUE = open the...tabase connect...
xDoClose	BOOL	FALSE		TRUE = close the...tabasess conne...
xDoExecute	BOOL	FALSE		Starts the execut...rocess on rising...
xDoQuery	BOOL	FALSE		Starts the query process on rising e...
xDoExecuteWithTrigger	BOOL	FALSE		Starts the execute process with trig...

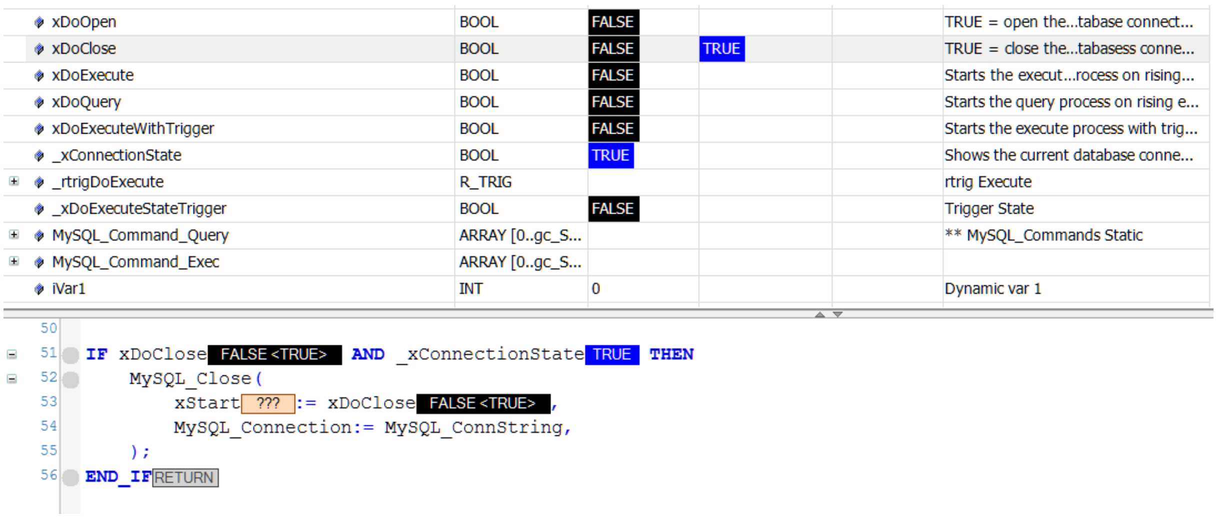
```
// Open the Database connection on xDo Open raising_edge
MySQL_Open(
     sHost '128.1.67.8' := _sHost '128.1.67.8',
     uiPort 3306 := _uiPort 3306,
     sDatabase 'testdb' := _sDatabase 'testdb',
     sUsername 'root' := _sUsername 'root',
     sPassword [redacted] := _sPassword [redacted],
     xStart FALSE := xDoOpen FALSE <TRUE>,
     MySQL_Connection:= MySQL_ConnString,
     xConnected TRUE => _xConnectionState TRUE
);
```

If the output “xConnected” changes his state to TRUE, than your PLC is successful connected with the database Server. Otherwise, please check the output eError, to identify the problem.

Close the MySQL-Server Connection

Closing a currently opened MySQL-Server session is a very easy task. There is only one thing you need to do – use the MySQL_Close function box as described now:

Before closing the connection, you need to check, if the connection is already open. If yes, than you can close it, otherwise it will have no effect.
To start the closing process, just set “xDoClose” to TRUE. If the process is finished, than the “xStart” var will be resetted.



xDoOpen	BOOL	FALSE			TRUE = open the...tabase connect...
xDoClose	BOOL	FALSE	TRUE		TRUE = close the...tabasess conne...
xDoExecute	BOOL	FALSE			Starts the execut...rocess on rising...
xDoQuery	BOOL	FALSE			Starts the query process on rising e...
xDoExecuteWithTrigger	BOOL	FALSE			Starts the execute process with trig...
_xConnectionState	BOOL	TRUE			Shows the current database conne...
_rtrigDoExecute	R_TRIG				rtrig Execute
_xDoExecuteStateTrigger	BOOL	FALSE			Trigger State
MySQL_Command_Query	ARRAY [0..gc_S...				** MySQL_Commands Static
MySQL_Command_Exec	ARRAY [0..gc_S...				
iVar1	INT	0			Dynamic var 1

```
IF xDoClose FALSE <TRUE>  AND _xConnectionState TRUE  THEN
    MySQL_Close(
        xStart  ???  := xDoClose  FALSE <TRUE> ,
        MySQL_Connection:= MySQL_ConnString,
    );
END_IF RETURN
```

Execute a MySQL command with no response result

You can perform execute commands with static or with dynamically created MySQL Command strings.

To declare a static MySQL command, type in the SQL Command under the VAR declaration section.

```
['INSERT INTO data (Column_1, Column_2, Column_3) VALUES (1, 5, 10)'];
```

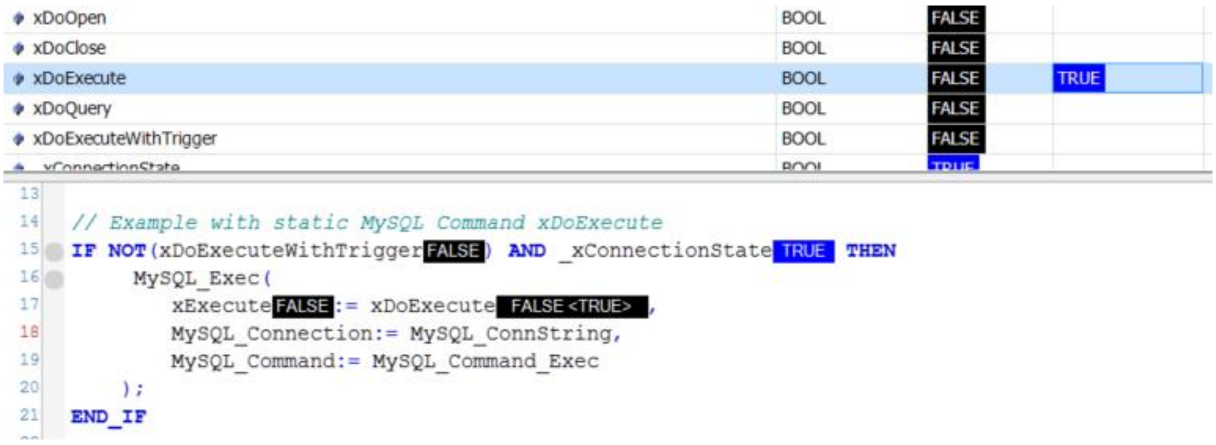
For dynamic MySQL commands you are completely free to configure the string with an array:

```
iCounterVar1 := iCounterVar1 +1;
iCounterVar2 := iCounterVar2 +2;

MySQL_Command_Exec_Dyn[0] := 'INSERT INTO data (Column_1, Column_2, Column_3) VALUES (';
MySQL_Command_Exec_Dyn[1] := '111,'; // --> Allways 111
MySQL_Command_Exec_Dyn[2] := CONCAT(INT_TO_STRING(iCounterVar1), ',');
MySQL_Command_Exec_Dyn[3] := CONCAT(INT_TO_STRING(iCounterVar2), ')');
```

Restriction:

- Array length: gc_StatemantUpBound = Default: 10
- String length: gc_iSqlLength = Default: 100

To execute a MySQL Command manually, change the “xDoExecute” var to TRUE.

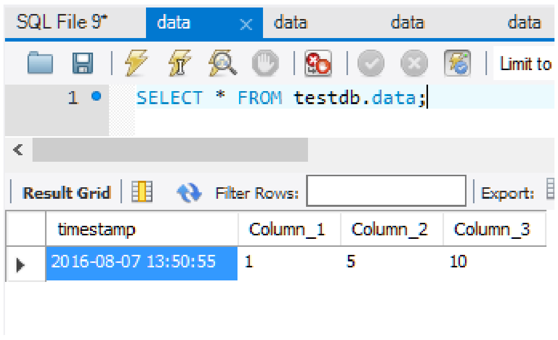


xDoOpen	BOOL	FALSE	
xDoClose	BOOL	FALSE	
xDoExecute	BOOL	FALSE	TRUE
xDoQuery	BOOL	FALSE	
xDoExecuteWithTrigger	BOOL	FALSE	
xConnectionState	BOOL	TRUE	

```
// Example with static MySQL Command xDoExecute
IF NOT(xDoExecuteWithTrigger FALSE ) AND _xConnectionState TRUE THEN
    MySQL_Exec(
        xExecute FALSE := xDoExecute FALSE <TRUE> ,
        MySQL_Connection:= MySQL_ConnString,
        MySQL_Command:= MySQL_Command_Exec
    );
END_IF
```

The executing process will start on raising edge. To handle the process, there is used an ETrig behavior model. To check the current process, you can also monitor the outputs “xBusy”, “xError” and “xDone”.

After executing the program, you can see that the entry is successfully stored in your db.



```
SELECT * FROM testdb.data;
```

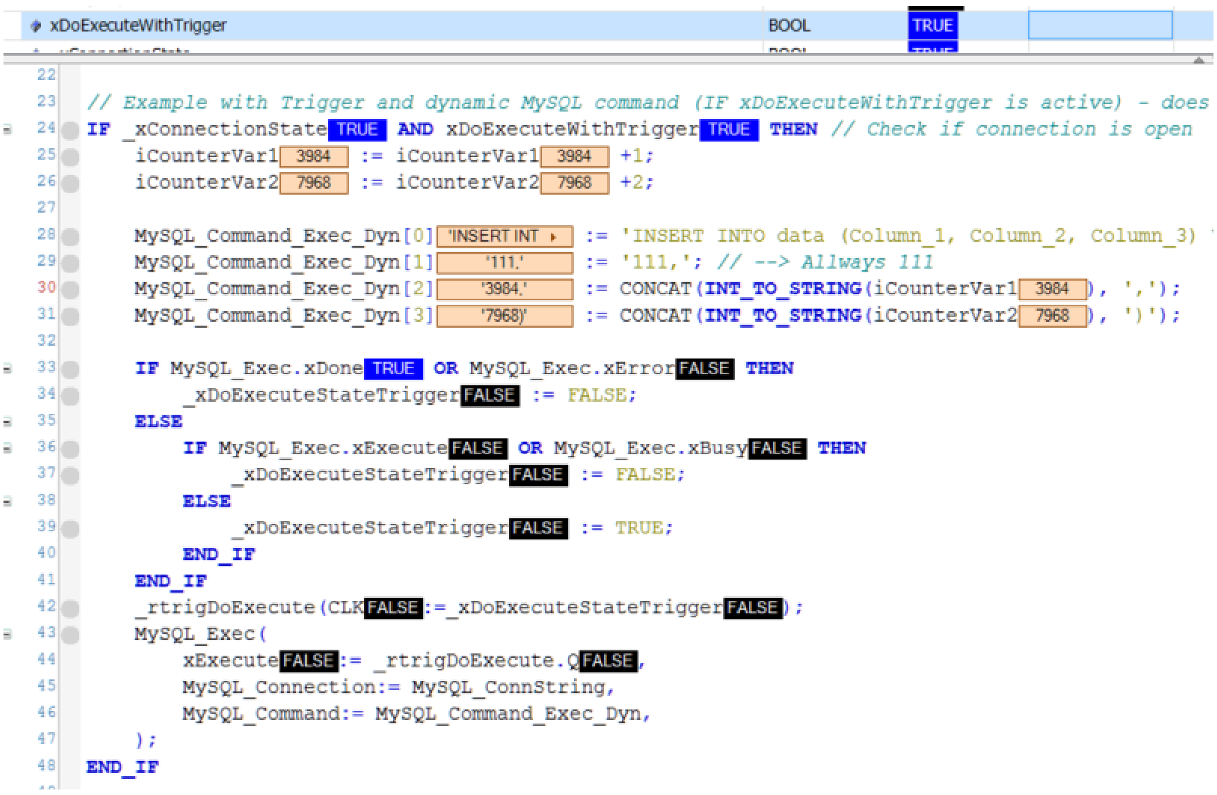
timestamp	Column_1	Column_2	Column_3
2016-08-07 13:50:55	1	5	10

If you are new to MySQL, than it’s useful to use tools with a graphic interface instead of the MySQL command CLI. There are a lot of tools available to use with MySQL.

For Windows, we are using MySQL Workbench from Oracle, which is free to use.
For Linux, we are using phpMyAdmin, its open source and can be directly installed on the MySQL Server machine.

The installation process for phpMyAdmin is also described in this manual.
There is also an SQL Query for you, which creates the sample database.

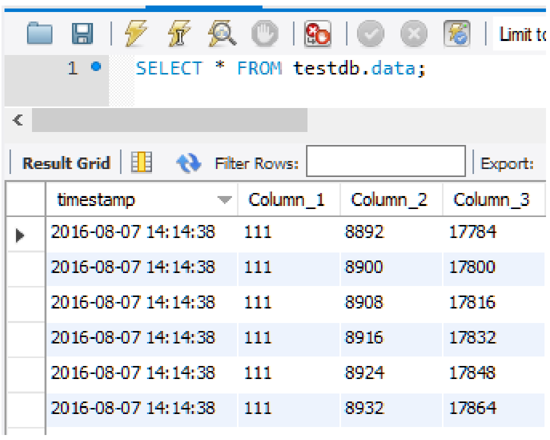
As we described before, you can also do execute commands, with dynamically created SQL commands. You can test it by changing “xDoExecuteWithTrigger” to TRUE.



```
xDoExecuteWithTrigger                                   BOOL    TRUE

// Example with Trigger and dynamic MySQL command (IF xDoExecuteWithTrigger is active) - does
IF _xConnectionState TRUE AND xDoExecuteWithTrigger TRUE THEN // Check if connection is open
    iCounterVar1 3984 := iCounterVar1 3984 +1;
    iCounterVar2 7968 := iCounterVar2 7968 +2;

    MySQL_Command_Exec_Dyn[0] 'INSERT INT := 'INSERT INTO data (Column_1, Column_2, Column_3) '
    MySQL_Command_Exec_Dyn[1] '111,' := '111,'; // --> Allways 111
    MySQL_Command_Exec_Dyn[2] '3984,' := CONCAT(INT_TO_STRING(iCounterVar1 3984), ',');
    MySQL_Command_Exec_Dyn[3] '7968)' := CONCAT(INT_TO_STRING(iCounterVar2 7968), ')');

    IF MySQL_Exec.xDone TRUE OR MySQL_Exec.xError FALSE THEN
        _xDoExecuteStateTrigger FALSE := FALSE;
    ELSE
        IF MySQL_Exec.xExecute FALSE OR MySQL_Exec.xBusy FALSE THEN
            _xDoExecuteStateTrigger FALSE := FALSE;
        ELSE
            _xDoExecuteStateTrigger FALSE := TRUE;
        END_IF
    END_IF
    _rtrigDoExecute(CLK FALSE := _xDoExecuteStateTrigger FALSE);
    MySQL_Exec(
        xExecute FALSE := _rtrigDoExecute.Q FALSE,
        MySQL_Connection:= MySQL_ConnString,
        MySQL_Command:= MySQL_Command_Exec_Dyn,
    );
END_IF
```

This example shows you, how to perform a MySQL Command with dynamic MySQL Commands. The execute process is active as long you deactivate the “xDoExecuteWithTrigger”.

So check your database after a few seconds, and you will see a lot of entries. This also shows you, that the execute process is very fast (depending on your task speed, MySQL-Server and network speed).

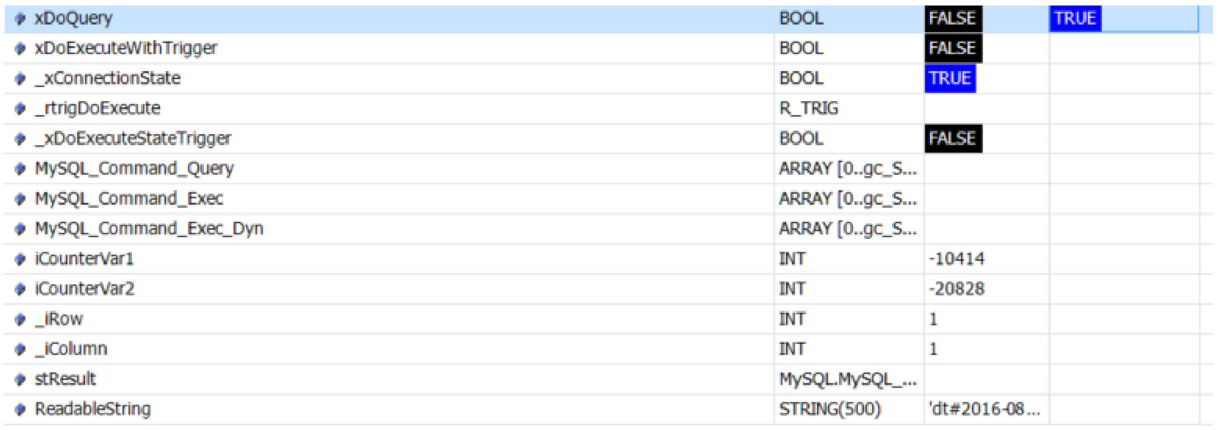


```
SELECT * FROM testdb.data;
```

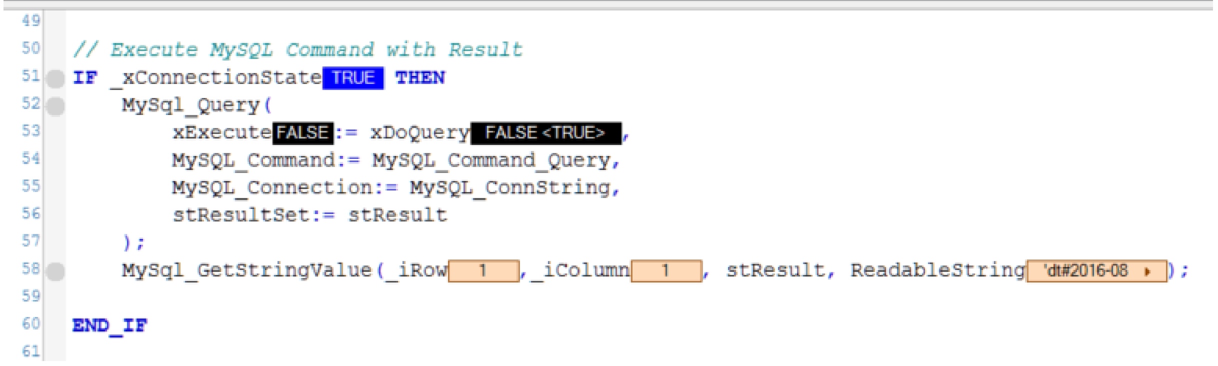
timestamp	Column_1	Column_2	Column_3
2016-08-07 14:14:38	111	8892	17784
2016-08-07 14:14:38	111	8900	17800
2016-08-07 14:14:38	111	8908	17816
2016-08-07 14:14:38	111	8916	17832
2016-08-07 14:14:38	111	8924	17848
2016-08-07 14:14:38	111	8932	17864

Execute a MySQL command with response result

To execute a command with result, like an SELECT Statement, you have to switch “xDoQuery” to True.



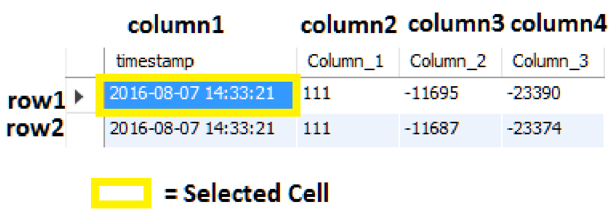
xDoQuery	BOOL	FALSE	TRUE
xDoExecuteWithTrigger	BOOL	FALSE	
_xConnectionState	BOOL	TRUE	
_rtrigDoExecute	R_TRIG		
_xDoExecuteStateTrigger	BOOL	FALSE	
MySQL_Command_Query	ARRAY [0..gc_S...		
MySQL_Command_Exec	ARRAY [0..gc_S...		
MySQL_Command_Exec_Dyn	ARRAY [0..gc_S...		
iCounterVar1	INT	-10414	
iCounterVar2	INT	-20828	
_iRow	INT	1	
_iColumn	INT	1	
stResult	MySQL.MySQL_...		
ReadableString	STRING(500)	'dt#2016-08...	



```
// Execute MySQL Command with Result
IF _xConnectionState TRUE THEN
    MySql_Query(
        xExecute FALSE := xDoQuery  FALSE <TRUE> ,
        MySQL_Command:= MySQL_Command_Query,
        MySQL_Connection:= MySQL_ConnString,
        stResultSet:= stResult
    );
    MySql_GetStringValue(_iRow 1 ,_iColumn 1 , stResult, ReadableString 'dt#2016-08 );

END_IF
```

The Default Query in this project is ‘SELECT * FROM data’ please verify, that the database is not too big, because the result buffer is limited. You can also define dynamic SELECT query here. To do this, check out the FBox description and Info about MySQL_Command.

To display the selected data in a readable String, please use the function MySQL_GetStringValue. As you can see in the picture, we select the data from the cell on row position 1 and column position 1 (index starts at 1).



	column1	column2	column3	column4
	timestamp	Column_1	Column_2	Column_3
row1	2016-08-07 14:33:21	111	-11695	-23390
row2	2016-08-07 14:33:21	111	-11687	-23374

MySQL-Library compatibility

The MySQL-Library was successfully tested on the following Platforms.

Platform	Runtime Version
CODESYS Control Win V3 (32bit)	3.5.9.0
Raspberry Pi Runtime	3.5.8.0 and 3.5.9.1
BeagleBone Black	3.5.9.0
Wago PFC200	3.5.8.10

For the compatibility test, we've used a lot of different mechanisms.
We've executed a lot of SELECT query's and INSERT, UPDATE, DROP, etc. commands. All tests scenarios were successful. But we cannot guarantee for 100% functionality of all MySQL commands, because our test scenario was limited.

Quick Start - MySQL-Server installation

This Quick Start Tutorial explains you how to install MySQL-Server on raspberry pi, which are supported by the CODESYS v3.5 runtime. It doesn't describe any optimization or security configuration options, which may be required for productive use.

This document describes the installing process, with the last "stable-release" of MySQL-Server on documentation creation date.

Platform	Used Version
Rasbian Jessie (Debian)	5.5.50-0+deb8u1

The Quick Start section describes also the installation of "phpMyAdmin" for Raspberry pi. phpMyAdmin is a program, which allows you to manage MySQL-Server database via an user friendly web frontend.

Install MySQL-Server on Raspberry Pi

Step 1: Preparation for the MySQL-Server installation

Before installing the MySQL-Server, be sure that your System is up-to-date.

So the first step you have to do is, updating the System with the command bellow.

```
pi@raspberrypi:~ $ sudo apt-get update && sudo apt-get upgrade
```

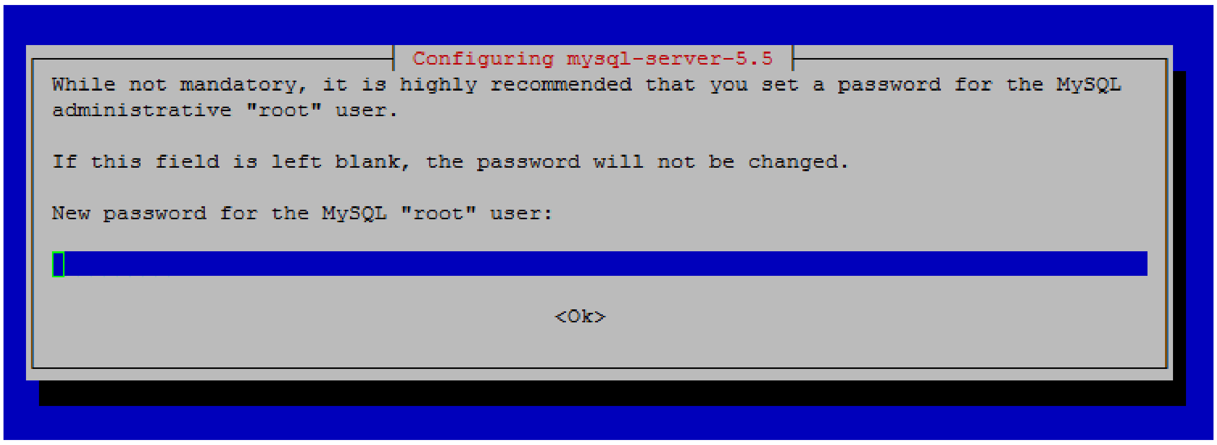
Step 2: Install MySQL-Server package

After updateing the System, you can start to install the MySQL-Server. To do this, just type in the following command:

```
pi@raspberrypi:~ $ sudo apt-get install mysql-server --fix-missing
```

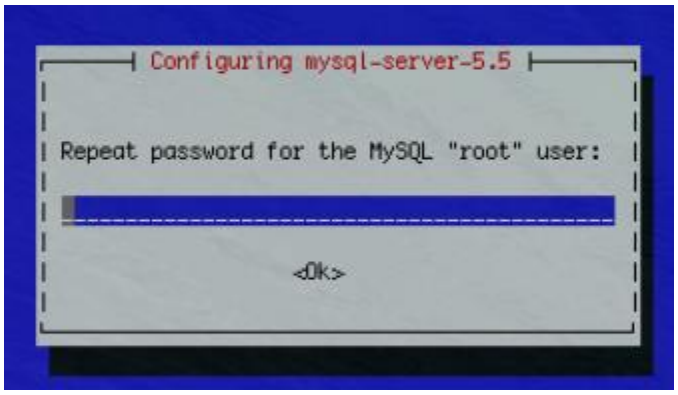
Step 3: MySQL-Server configuration wizard

During the MySQL-Server installation it will show the configuration wizard. The first configuration step asks you to define a password for the MySQL “root” user.



```
Configuring mysql-server-5.5

While not mandatory, it is highly recommended that you set a password for the MySQL
administrative "root" user.

If this field is left blank, the password will not be changed.

New password for the MySQL "root" user:

<Ok>
```

To avoid typing errors it prompts you to type in the “root” password again.



```
Configuring mysql-server-5.5

Repeat password for the MySQL "root" user:

<Ok>
```

Install phpMyAdmin on Raspberry Pi

Step 1: Preparation for the phpMyAdmin installation

Before installing phpMyAdmin, be sure that your System is up-to-date.

So the first step you have to do is, updating the System with the command bellow.

```
pi@raspberrypi:~ $ sudo apt-get update && sudo apt-get upgrade
```

Step 2: Install phpMyAdmin

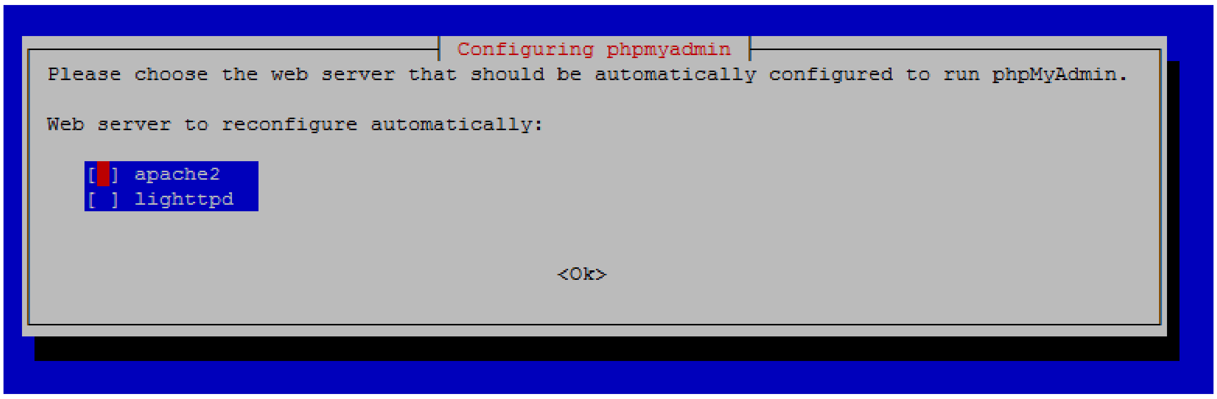
Start the installation process with the following command:

```
pi@raspberrypi:~ $ sudo apt-get install phpmyadmin
```

Step 3: phpMyAdmin configuration

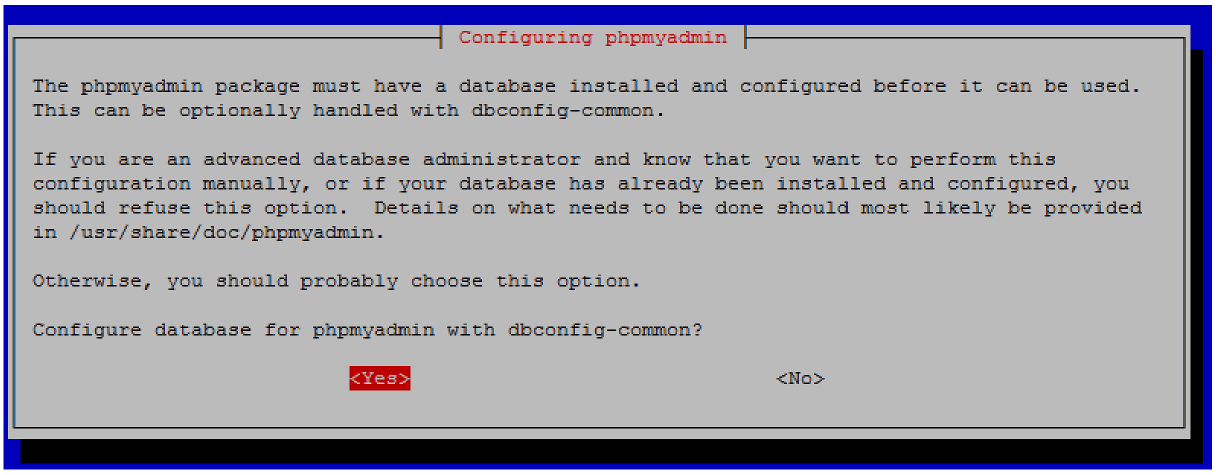
Select a webserver of your choice to install for phpMyAdmin and continue with “Ok”.

Recommended: apache2



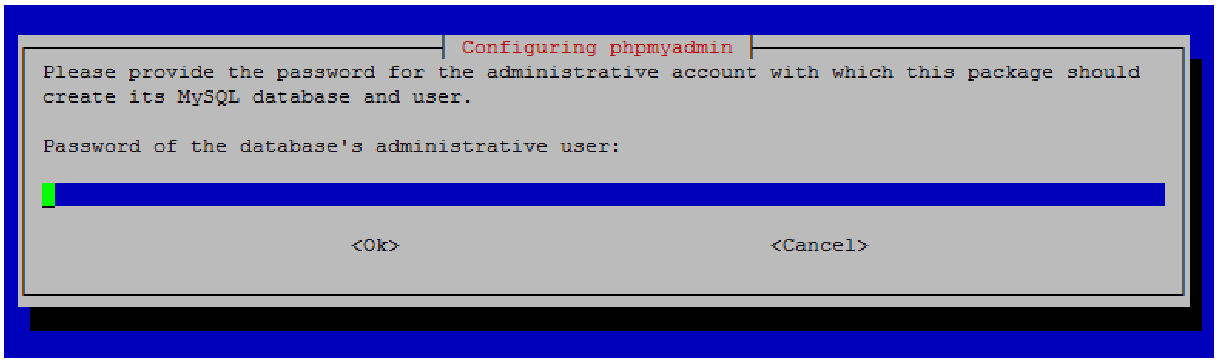
```
Configuring phpmyadmin
Please choose the web server that should be automatically configured to run phpMyAdmin.

Web server to reconfigure automatically:

    [ ] apache2
    [ ] lighttpd

                                 <Ok>
```

Confirm the next step to create the default database for use phpMyAdmin



```
Configuring phpmyadmin

The phpmyadmin package must have a database installed and configured before it can be used.
This can be optionally handled with dbconfig-common.

If you are an advanced database administrator and know that you want to perform this
configuration manually, or if your database has already been installed and configured, you
should refuse this option.  Details on what needs to be done should most likely be provided
in /usr/share/doc/phpmyadmin.

Otherwise, you should probably choose this option.

Configure database for phpmyadmin with dbconfig-common?

                  <Yes>                                  <No>
```

The next step prompts you to type in the MySQL-Server password. Type in the password, which you have defined for your MySQL root user.



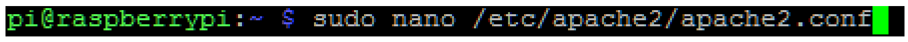
```
Configuring phpmyadmin
Please provide the password for the administrative account with which this package should
create its MySQL database and user.

Password of the database's administrative user:


          <Ok>                                  <Cancel>
```

Step 4: Configure Apache to use phpMyAdmin

Before you can use phpMyAdmin the apache configuration needs to be adjusted. Adding the text “Include /etc/phpmyadmin/apache.conf” (without quations marks) to the end of the “/etc/apache2/apache2.conf” file.

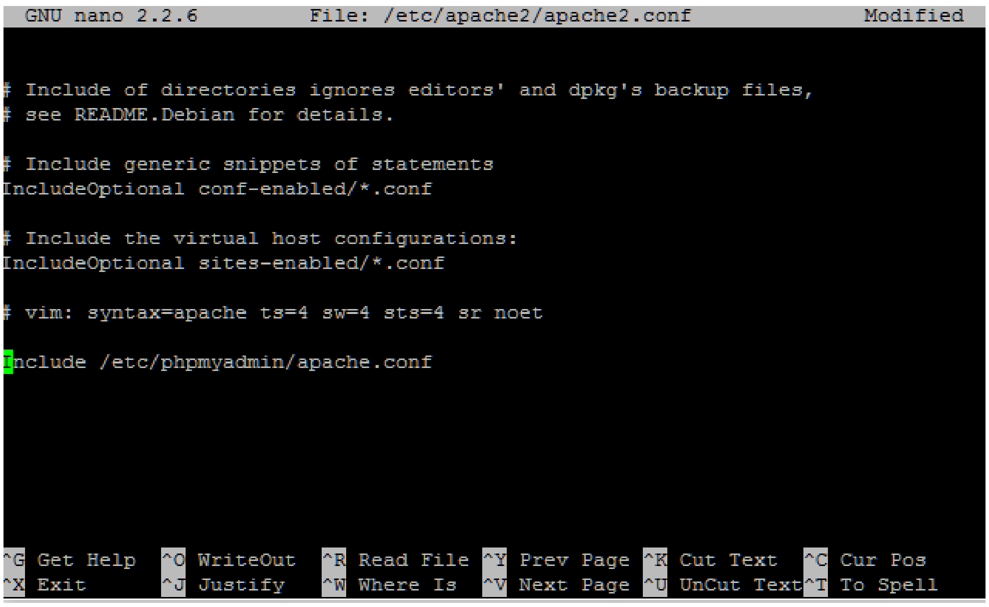
To edit the configuration file, choose an editor of your choice, for example nano.

Open the file:



```
pi@raspberrypi:~ $ sudo nano /etc/apache2/apache2.conf
```

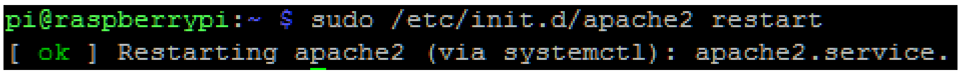
Edit the File:



```
  GNU nano 2.2.6           File: /etc/apache2/apache2.conf                Modified


# Include of directories ignores editors' and dpkg's backup files,
# see README.Debian for details.

# Include generic snippets of statements
IncludeOptional conf-enabled/*.conf

# Include the virtual host configurations:
IncludeOptional sites-enabled/*.conf

# vim: syntax=apache ts=4 sw=4 sts=4 sr noet

Include /etc/phpmyadmin/apache.conf





^G Get Help   ^O WriteOut   ^R Read File  ^Y Prev Page  ^K Cut Text   ^C Cur Pos
^X Exit       ^J Justify    ^W Where Is   ^V Next Page  ^U UnCut Text^T To Spell
```

To save the edited file, press CTRL+X to leave the editor and then press CTRL+Y to override the modified file.

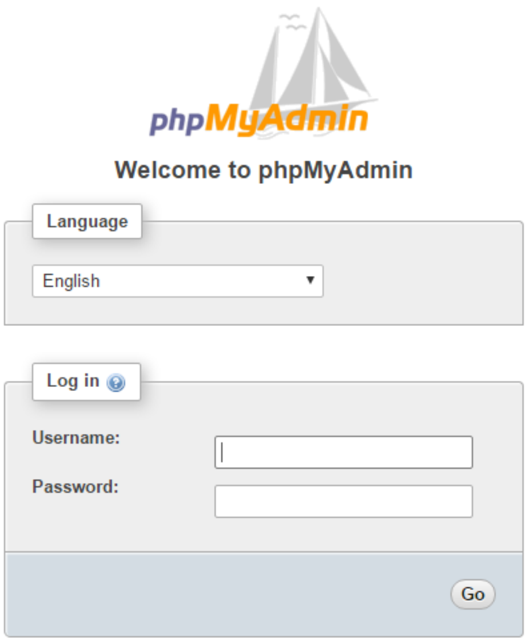
After editing the file, the apache2 service needs to be restarted. To do this, type in the following command:



```
pi@raspberrypi:~ $ sudo /etc/init.d/apache2 restart
[ ok ] Restarting apache2 (via systemctl): apache2.service.
```

Step 5: Accessing phpMyAdmin

To access phpMyAdmin, open the webpage: http://<your-device-ip/phpmyadmin



MySQL database SQL dump:

MySQL database dump for example project.

```
CREATE TABLE `data` (
  `timestamp` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  `Column_1` int(16) NOT NULL,
  `Column_2` int(255) NOT NULL,
  `Column_3` int(32) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```