MySQL Protocol Tutorial

Stéphane Legrand <stephleg@free.fr>

October 6, 2013

Abstract

This tutorial illustrates the use of the MySQL Protocol library, a native OCaml implementation of the MySQL client protocol.

You can download this library from the MySQL Protocol home page.

Contents

1 Introduction

The library has to be installed before using the code below. Please read the INSTALL file in the library archive. The OCaml source code of this tutorial is available in the *examples* directory.

2 Modules

First, some convenient alias for the modules used.

```
module Mp_client = Mysql_protocol.Mp_client;;
module Mp_data = Mysql_protocol.Mp_data;;
module Mp_execute = Mysql_protocol.Mp_execute;;
module Mp_result_set_packet =
    Mysql_protocol.Mp_result_set_packet;;
```

3 Helper functions

We define one function to print the result of SQL statements like INSERT, UPDATE, GRANT...

```
let print_result sql r =
  print_endline ("Result of the SQL statement \"" ^ sql ^ "\": \n
       " ^ (Mp_client.dml_dcl_result_to_string r) ^ "\n")
;;
```

And two others to print the result of SELECT SQL statements.

```
let print_row fields row =
  let print_data f =
    let (field_name, field_pos) = f in
    let data = List.nth row field_pos in
    print_endline ("  " ^ field_name ^ ": " ^ (Mp_data.to_string
        data))
  in
  let () = List.iter print_data fields in
  print_endline "  -- --  "
;;
let print_set sql r =
  let (fields, rows) = r.Mp_result_set_packet.rows in
  let () = print_endline ("Result set for the SQL statement \"" ^
      sql ^ "\": \n") in
  let print_rows =
    let () = List.iter (print_row fields) rows in
    print_newline ()
  in
  print_rows
;;
```

4 Configuration

To be able to connect to the MySQL server, we have first to configure the client.

```
let addr = Unix.inet_addr_loopback;;
let port = 3306;;
let sockaddr = Unix.ADDR_INET(addr, port);;

let db_user = "user_ocaml_ocmp";;
let db_password = "ocmp";;
let db_name = "test_ocaml_ocmp_utf8";;
```

The MySQL server is listening on the loopback interface and it uses the standard MySQL server port. The login to authenticate to the server is "user_ocaml_ocmp" with the password "ocmp". And we will use the "test_ocaml_ocmp_utf8" database.

Now, we can create a configuration.

```
let config = Mp_client.configuration  ~user:db_user
    ~password:db_password ~sockaddr:sockaddr
    ~databasename:db_name () in
```

Here, the default charset and collation is used. The default value for charset is `Utf8` and the default value for collation is `Utf8_general_ci`. If you want to specify an other value, you can use the `charset` parameter. For instance:

```
   let config = Mp_client.configuration  ~user:db_user
       ~password:db_password ~sockaddr:sockaddr
       ~databasename:db_name ~charset:(Mp_charset.Latin1,
       Mp_charset.Latin1_swedish_ci) () in
```

To have the complete list of available charset and collation, you can read the documentation of the `Mp_Charset` module.

5 Connection

Now, we can connect our client to the MySQL server.

```
let connection = Mp_client.connect ~configuration:config () in
```

By default, the connection is not initialized right after the call to the `connect()` function. It's delayed until necessary (ie until the first real request). You can immediately force the connection by using the `force` parameter:

```
let connection = Mp_client.connect ~configuration:config
    ~force:true () in
```

6 Select database

To specify the current database, use the following function:

```
let () = Mp_client.use_database ~connection:connection
    ~databasename:db_name in
```

7 Non prepared statement

A non prepared statement is the simplest way to send a statement to the server. If your statement doesn't have any parameter (ie is a static string) and is used only a few times, it's usually sufficient.

WARNING: You SHOULD NOT use a non prepared statement if it contains a parameter with non trusted value.

7.1 Create

The first step is to create the statement from the SQL string.

```
let sql = "INSERT INTO ocmp_table (col1, col2) VALUES ('col1',
    123.45)" in
let stmt = Mp_client.create_statement_from_string sql in
```

7.2 Execute

Next, we send the statement to the server to execute it.

```
let r = Mp_client.execute ~connection:connection ~statement:stmt
    () in
```

7.3 Get result

After being executed, the statement result can be retrieved.

7.3.1 Result without record

For statement which returns only a simple result without any record (INSERT, UPDATE, DELETE, GRANT... statement), you can use the `get_result_ok()` function.

```
let r = Mp_client.get_result_ok r in
```

To print this result, use the `print_result()` helper function.

```
let () = print_result sql r in
```

7.3.2 Result with records

For statement which returns records (typically SELECT statement), you can use the `get_result_set()` function.

```
let r = Mp_client.get_result_set r in
```

To print this result, use the `print_set()` helper function.

```
let () = print_set sql r in
```

8 Prepared statement

Especially when the statement includes some parameters, you should use a prepared statement. The parameters values will then be correctly enclosed in the statement by the MySQL server and all special characters will be automatically escaped. You can of course also use a prepared statement even if the statement doesn't have any parameter.

8.1 Create

The first step is the same as for a non prepared statement, you have to create the statement from the SQL string with the same function.

```
let sql = "SELECT * FROM ocmp_table WHERE col1=?" in
let stmt = Mp_client.create_statement_from_string sql in
```

8.2 Prepare

Then, you prepare the statement.

```
let prep = Mp_client.prepare ~connection:connection
    ~statement:stmt in
```

Once a statement has been prepared, you can execute it several times without calling the `prepare()` function again.

8.3 Execute

You execute a prepared statement with the same function as for a non prepared one. Nonetheless, for a prepared statement, the `execute()` function may accept more parameters.

8.3.1 Simple execute

For the simplest use case (no parameters in the statement, no cursor), you execute the statement as for a non prepared one.

```
let r = Mp_client.execute ~connection:connection ~statement:prep
    () in
```

8.3.2 With parameters

If you have some parameters in the statement, you first need to create the list of these parameters in the same order of appearance as in the statement. Please see the documentation for the `Mp_data` module to have a complete list of data constructor and learn which one to use for each MySQL column types.

```
let params = [Mp_data.Varstring "col2"; Mp_data.Decimal
    (Num.num_of_string "98765/100")] in
```

And you add the `params` function parameter for the execution.

```
let r = Mp_client.execute ~connection:connection ~statement:prep
    ~params:params () in
```

8.3.3 With cursor

By default, no cursor is used when a prepared statement is executed. So the server will always return all the corresponding records. If you want to be able to fetch the result by parts (record by record for instance), you need to specify the cursor option in the `execute()` function.

```
let stmt = Mp_client.execute ~connection:connection
    ~statement:prep ~params:params
    ~flag:Mp_execute.Cursor_type_read_only () in
```

WARNING: For now, only `Cursor_type_read_only` type is supported.

8.4 Get result

After being executed, the statement result can be retrieved.

8.4.1 Result without record

For statement which returns only a simple result without any record (INSERT, UPDATE, DELETE, GRANT... statement), there is no difference compared with the non prepared statements. You can also use the `get_result_ok()` function.

```
let r = Mp_client.get_result_ok r in
```

To print this result, use the `print_result()` helper function.

```
let () = print_result sql r in
```

8.4.2 Result with records

For statement which returns records (typically SELECT statement), if you haven't used a cursor, you cannot use fetch. So you will retrieve all the rows with the same method as a non prepared statement.

```
let r = Mp_client.get_result_set r in
```

To print this result, use the `print_set()` helper function.

```
let () = print_set sql r in
```

If you have used a cursor, you have to use fetch to retrieve the records. By default, the `fetch()` function get one record at each call. To specify an other number, use the `nb_rows` function parameter.

```
let stmt = Mp_client.execute ~connection:connection
    ~statement:prep ~params:params
    ~flag:Mp_execute.Cursor_type_read_only () in
let () =
  try
    while true do
      let rows = Mp_client.fetch ~connection:connection
          ~statement:stmt () in
      let rows = Mp_client.get_fetch_result_set rows in
      print_set sql rows
    done
  with
  | Mp_client.Fetch_no_more_rows -> () (* no more rows in the
      result *)
in
```

8.5 Close statement

When a prepared statement has become useless (ie you don't need to execute it again), you can and should destroy it.

```
let () = Mp_client.close_statement ~connection:connection
    ~statement:prep in
```

9 Ping server

To avoid a timeout or to test the connection, you can send a ping to the server. No result is returned but an `Mp_client.Error` exception can be raised.

```
let () = Mp_client.ping ~connection:connection in
```

10 Reset the session

This is useful if you need to destroy the session context (temporary tables, session variables, etc.) in the MySQL server.

```
let () = Mp_client.reset_session ~connection:connection in
```

11 Catching errors

Whenever the MySQL server returns an error, an `Mp_client.Error` exception is raised.

```
let stmt = Mp_client.create_statement_from_string ("BAD SQL
    QUERY") in
let () =
  try
    let stmt = Mp_client.execute ~connection:connection
        ~statement:stmt () in
    ()
  with
  | Mp_client.Error error ->
      print_newline ();
      print_endline ("This is a test to show how to catch a MySQL
          error, the exception is: " ^
          (Mp_client.error_exception_to_string error))
in
```

12 Disconnect

To close the connection to the server, use the `disconnect()` function.

```
let () = Mp_client.disconnect ~connection:connection in
```