

## Computing: Free Pascal Programming

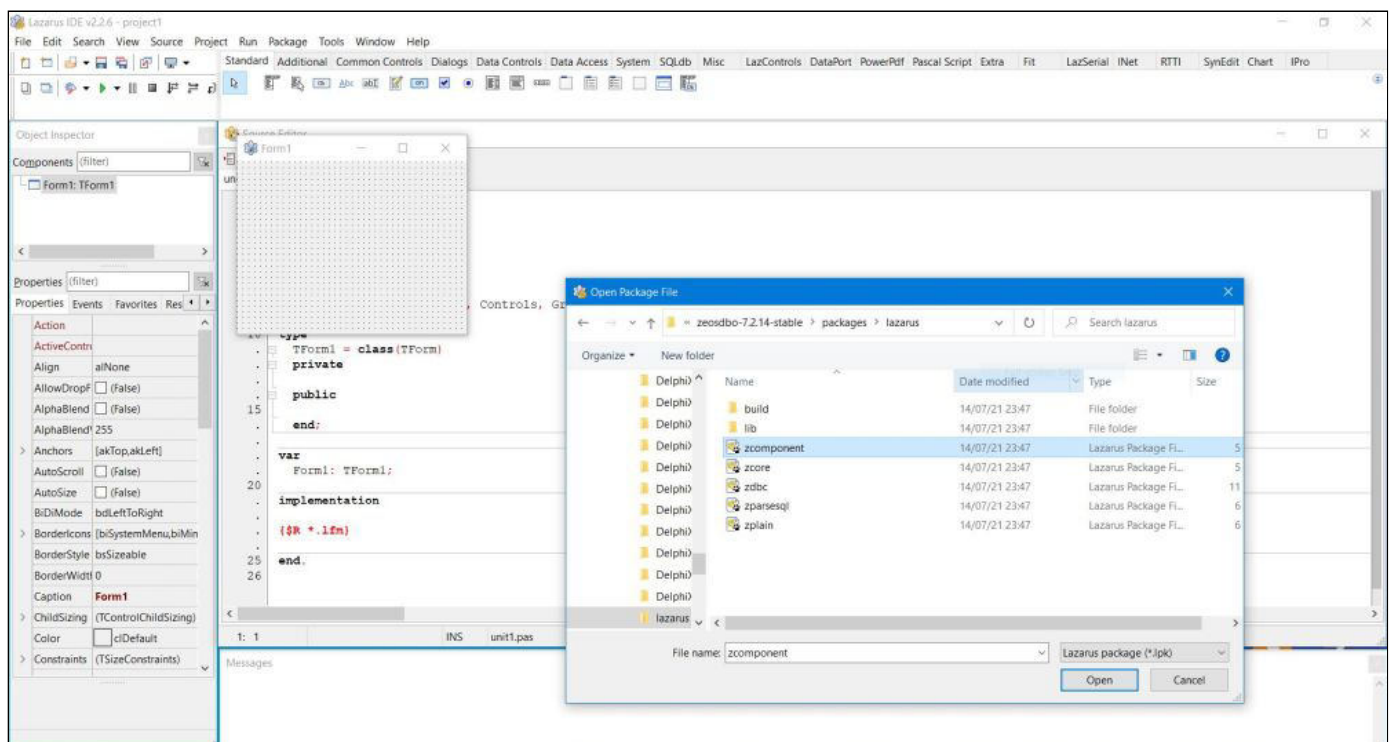
### Connecting from Lazarus/Free Pascal to MySQL/MariaDB using Zeos components.

Beside using the standard SQLdb components shipped with Lazarus, there are other ways to connect to a database from a Free Pascal application. One of them, freeware and open source, is the Zeos library. This is a set of database components for MySQL, MariaDB, PostgreSQL, Interbase, Firebird, MS SQL Server, SAP Adaptive Server Enterprise and Adaptive Server Anywhere (previously Sybase), Oracle and SQLite for Delphi, Lazarus/Free Pascal and C++ Builder. It can be downloaded from the ZeosLib page at the [Sourceforge](https://sourceforge.net/projects/zeoslib/) website.

This text is not a Zeos tutorial. It just shows how to install the library into the Lazarus IDE, and how to proceed to connect to a database (and perform a simple select operation) using Zeos components. The databases used in the tutorial are MySQL and MariaDB; the sample applications should also work with other databases. The samples have been tested on Windows 10, using Lazarus 2.2.6 (with FPC 3.2.2) and ZeosLib 7.2.14. The databases used were MySQL 8.0 and MariaDB 10.6. No idea, if ZeosLib works on macOS or Linux... Click the following link to download the [Lazarus/Free Pascal source code](#) of the tutorial sample applications.

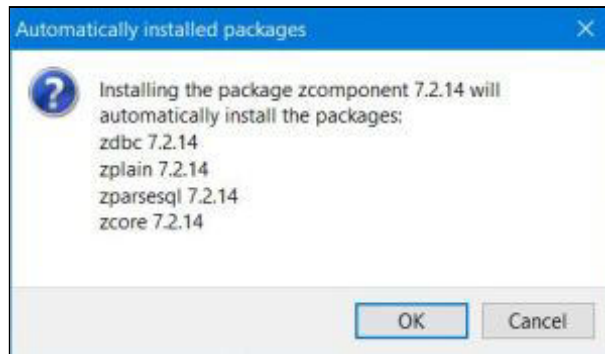
Move the content of the download archive to some folder on your PC. Please, note that you must keep the files in this folder, after ZeosLib has been installed.

To install the library, use the command Package > Open package file (\*.lpk)... in the IDE menu. Browse to the folder containing your download files and navigate to the packages\lazarus folder. In this folder, select the file zcomponent.lpk.



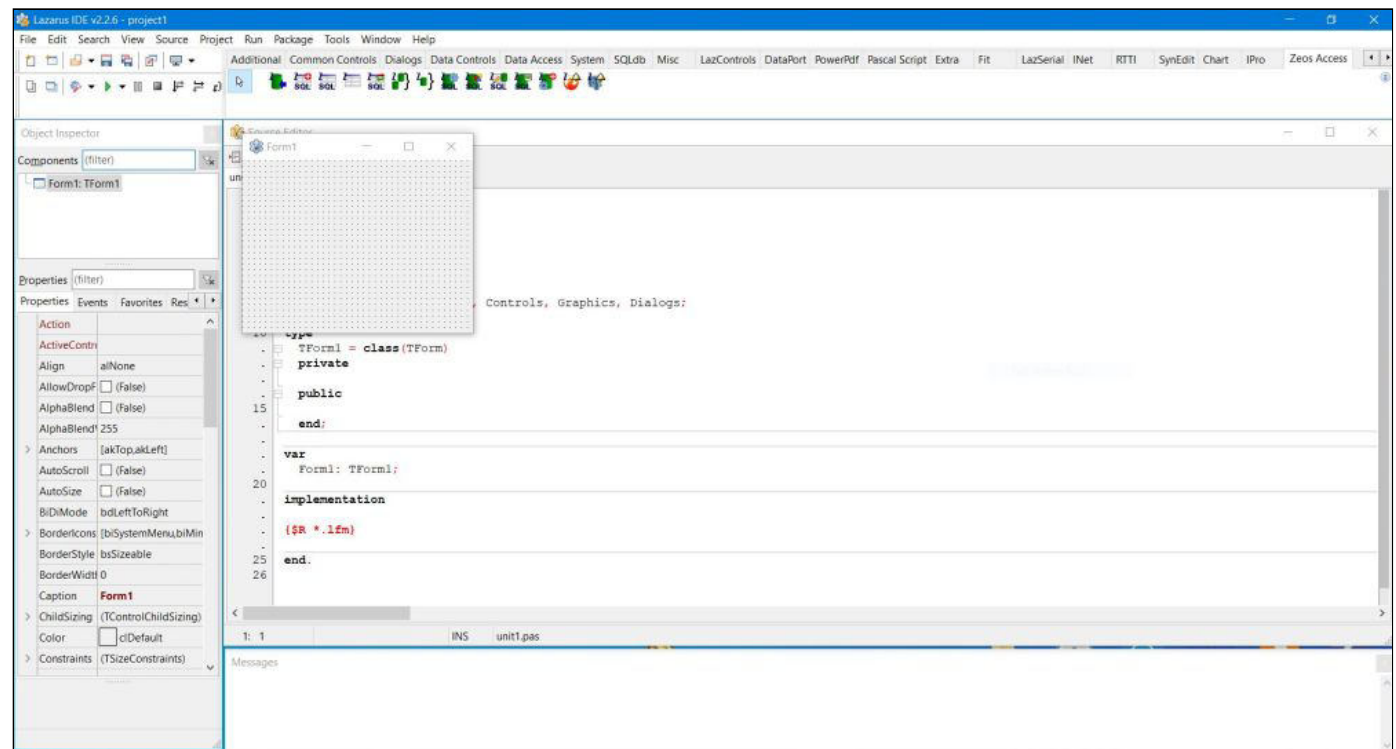
In the Package zcomponent window, choose the command Use > Install. A dialog box opens, showing the other ZeosLib packages that were automatically selected for installation. Push the

OK button to confirm.



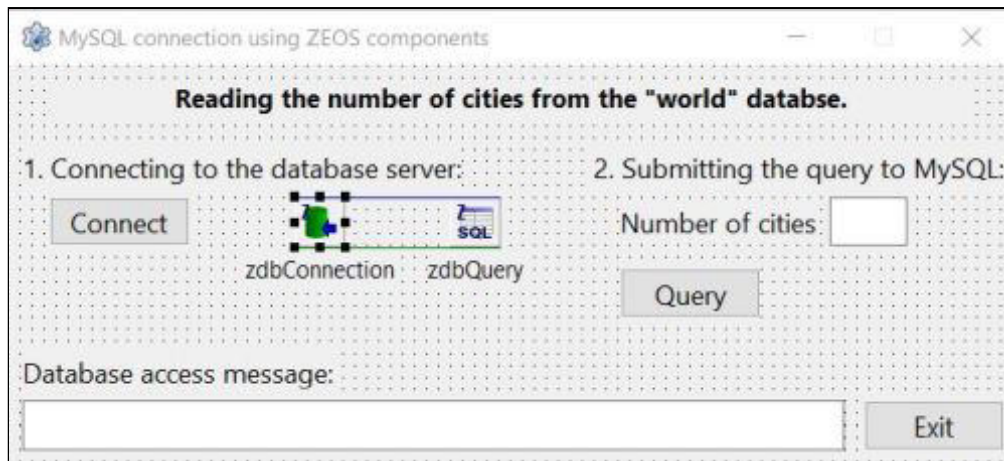
Lazarus actually only supports static linked packages. This means that you have to rebuild the IDE. In the corresponding dialog box, push the OK button to do so. When done, the IDE closes and restarts automatically. You can verify that the package has been correctly installed by having a look at the installed packages list (use the menu command Package > Install/Uninstall packages); the package is listed as zcomponent.

Having a look at the component bar, you'll notice that a new entry, called Zeos Access has been created. Among the 13 new components available, we'll choose two to be used in our sample applications.



### Sample MySQL application.

Create a new Lazarus application, as shown on the screenshot below. The program should allow to connect to a local MySQL server and reading the number of records in the "city" table of the "world" database (with MySQL 8.0 Community Edition, the "world" database may be installed, using the MySQL "all-in-one" installer). There are two ZeosLib components, that we'll use here: TZConnection (I called my object "zdbConnection"), that allows to configure the connection to any of the databases supported by the library (so not only MySQL and MariaDB, but also SQL Server, PostgreSQL, etc.), and TZQuery (I called my object "zdbQuery"), that allows to send the SQL queries to the database server and retrieve the dataset returned.



The TZConnection object has several important properties, that you might have to or want to adapt when using the component with a more complex application, even though I think that the default values should be adequate in most cases. In particular, `AutoCommit = True` automatically commits the transactions, so immediately updates data changes in the physical database. `AutoEncodeStrings = True` tells the server to try to convert your data between the client codepage and the settings of the `ControlsCodePage`. This property set to "True" may result in a loss of data if the conversion isn't possible. I suppose that if you have created your tables using UTF-8, and `ControlsCodePage = cCP_UTF8`, everything should work well. `TransactIsolationLevel = tiNone` is the default and ok here (note that this setting results in an error with some databases, as for example Firebird). In a "real world" application, you would probably choose one of the other options: `tiRepeatableRead`: the available data corresponds to a snapshot taken when the transaction is started (changes by other transactions are not recognized, an error occurs when two transactions try to change the same record); `tiReadCommitted`: the available data corresponds to the actual data (changes by other transactions are recognized, waiting for blocked records may be switched on or off); `tiSerializable`: the records that the transaction works with are exclusive for this transaction (no multi-user access possible). Concerning the connection parameters (host, database, user...), we will set the corresponding properties within the code.

Using a TZQuery object is the simplest way to access a database using the ZeosLib components. Just setting the SQL property to the SQL statement, opening the query and retrieving the dataset returned (we'll do all this within the code). It is obvious that the TZQuery object must be associated with a TZConnection object. This is done by setting its Connection property to the name of that object. In our case: `zdbQuery.Connection = 'zdbConnection'`.

The connection to the database should be opened when the "Connect" button is pushed, so we have to put the corresponding code within the `TfMySQLZ.btConnectClick` method ("fMySQLZ" being the name of my form, "btConnect" the name of my button):

```
procedure TfMySQLZ.btConnectClick(Sender: TObject);
begin
  if zdbConnection.Connected then
    zdbConnection.Disconnect;
  // Set the connection parameters
  zdbConnection.Protocol := 'mysql'; // connect to a MySQL database
  zdbConnection.HostName := 'localhost';
  zdbConnection.Port := 3306;
  zdbConnection.User := 'nemo';
  zdbConnection.Password := 'nemo';
  zdbConnection.Database := 'world';
  // Connect to the "world" database
  try
    zdbConnection.Connect;
    edMess.Text := 'Connection to MySQL database "world" = OK!';
  except
    on E: Exception do
```

```

        edMess.Text := E.Message;
    end;
end;

```

The important setting here is the ZConnection.Protocol property: It allows to specify the database server that we want to access. The property value is a string; you can view the options available by clicking the property in the object's property sheet and then extending the list by clicking the arrow in the values field. I chose the first of the two "mysql" values; it seems to work fine with MySQL 8.0.

Setting the MySQL connection properties and connecting to the MySQL server is essentially the same as for a TMySQLxxxConnection object. Just note that the property and method names may be somewhat different (examples: TMySQLxxxConnection.UserName vs TZConnection.User; TMySQLxxxConnection.Open vs TZConnection.Connect).

ZeosLib being just a try-out for me, I did not look up the documentation if there are specialized exception conditions available. Instead, I used the generic condition on E: Exception. Perhaps not really the best choice, but a choice that always works!

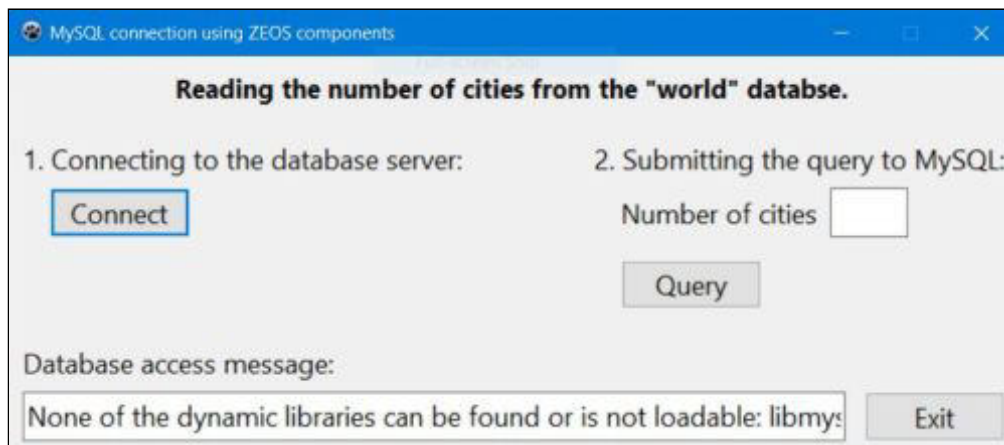
We should disconnect from the server before quitting the application (using the "Exit" button). Here is the code of the TfMySQLZ.btExitClick method:

```

procedure TfMySQLZ.btExitClick(Sender: TObject);
begin
    if zdbConnection.Connected then
        zdbConnection.Disconnect;
    Close;
end;

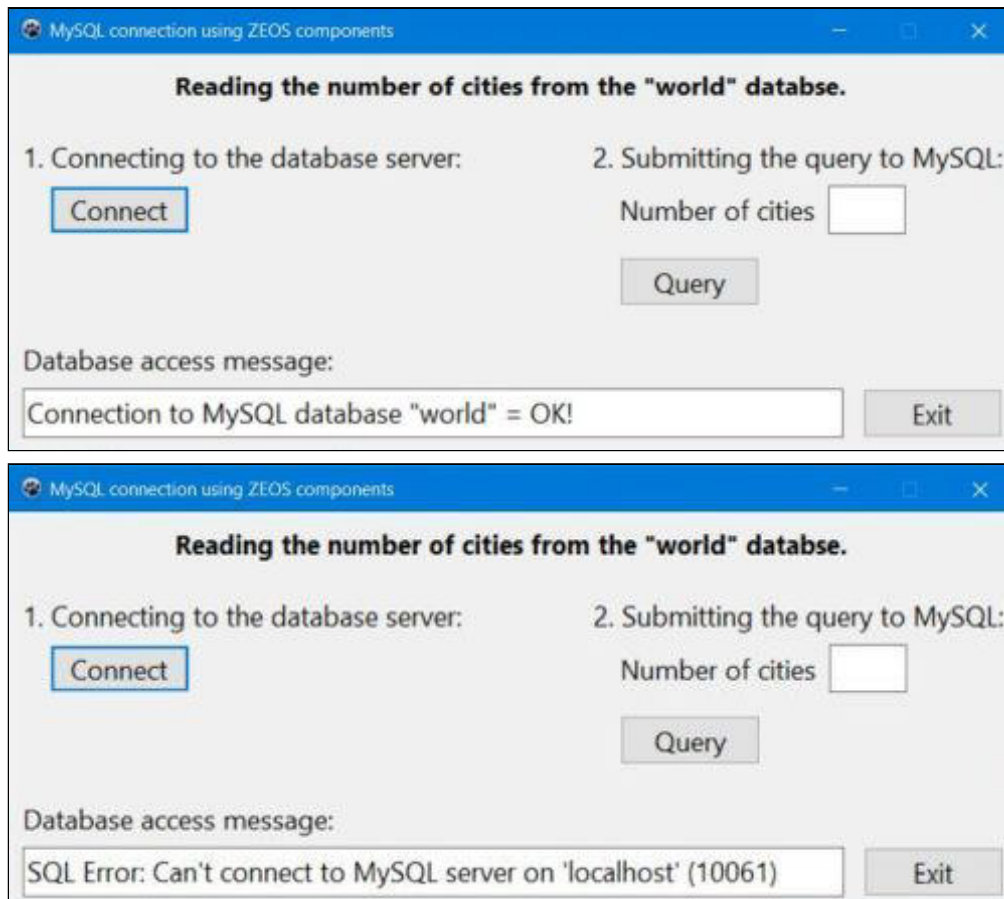
```

The screenshot below shows my first execution of the sample application. Pushing the "Connect" button results in the error message None of the dynamic libraries can be found or is loadable. That again is the same as when using a TMySQLxxxConnection object. The MySQL client DLL, called libmysql.dll has to be copied to the project output folder (or later, together with the executable). With MySQL 8.0 having been installed to the default directory, you can find it in C:\Program Files\MySQL\MySQL Server 8.0\lib.



The DLL having been copied, the connection to the MySQL server succeeds (screenshot on the left), or not (e.g. because the server is offline; screenshot on the right).



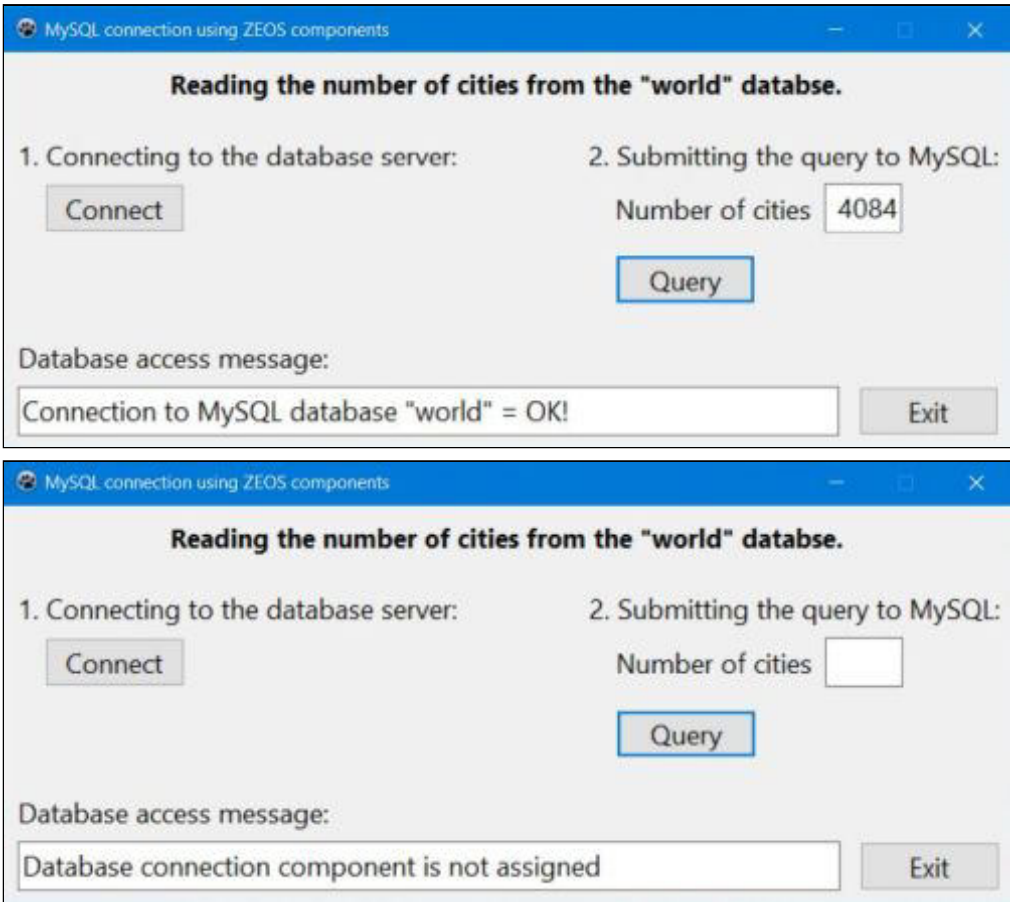


When we push the "Query" button, the application should read the number of cities in the city table and display it in the corresponding edit field. Here the code of my TfMySQLZ.btQueryClick method:

```
procedure TfMySQLZ.btQueryClick(Sender: TObject);
var
  Count: Integer;
begin
  if zdbConnection.Connected then begin
    // Query the database
    zdbQuery.SQL.Clear;
    zdbQuery.SQL.Add('SELECT count(*) FROM city');
    try
      zdbQuery.Open;
      if zdbQuery.EOF then
        Count := 0
      else
        Count := zdbQuery.Fields[0].AsInteger;
      zdbQuery.Close;
      // Display the query result
      edCities.Text := IntToStr(Count);
    except
      on E: Exception do
        edMess.Text := E.Message;
    end;
  end;
end;
```

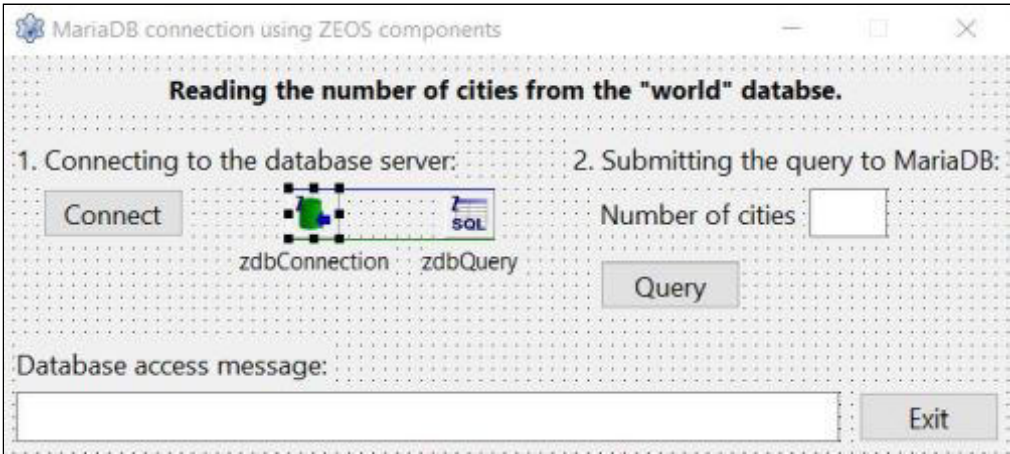
The SQL statements are passed as strings to the TStrings property zdbQuery.SQL, then the SELECT query can be executed using the zdbQuery.Open method. In the case where the returned dataset is a single row (in our case, a row with just one element), the result value(s) can be retrieved using the zdbQuery.Fields array.

The screenshot on the left shows a successful query. The screenshot on the right shows the kind of error that you get if you forgot to associate the TZQuery object with a TZConnection object.



Sample MariaDB application.

So, lets create the MariaDB application, as shown on the screenshot below. In fact, besides some label captions, it's the same form with the same components as for MySQL, in particular the two ZeosLib components TZConnection (that I called "zdbConnection"), and TZQuery (that I called "zdbQuery"). Be sure to set the Connection property of the TZQuery object to the name of the TZConnection object; in our case: `zdbQuery.Connection = 'zdbConnection'`.



As the connection to the database should be opened when the "Connect" button is pushed, we have to put the corresponding code within the `TfMariaDBZ.btConnectClick` method ("`fMariaDBZ`" being the name of my form, "`btConnect`" the name of my button). If you compare the code below with the one for MySQL above, you can see that it's essentially the same (and this remains true for other databases, too). One important difference however: the `TZConnection.Protocol` property has to set depending on the database used. For my MariaDB 10.6 database server, quite obvious to choose "`MariaDB-10`". The code to be executed when the user pushes the "Exit" button, coded within the `TfMariaDBZ.btExitClick` method, is exactly the same as for MySQL.

Here is the code of my TfMariaDBZ.btConnectClick and TfMariaDBZ.btExitClick methods:

```
procedure TfMariaDBZ.btConnectClick(Sender: TObject);
begin
    if zdbConnection.Connected then
        zdbConnection.Disconnect;
    // Set the connection parameters
    zdbConnection.Protocol := 'MariaDB-10'; // connect to a MariaDB 10 database
    zdbConnection.HostName := 'localhost';
    zdbConnection.Port := 3307;
    zdbConnection.User := 'nemo';
    zdbConnection.Password := 'nemo';
    zdbConnection.Database := 'world';
    // Connect to the "world" database
    try
        zdbConnection.Connect;
        edMess.Text := 'Connection to MariaDB database "world" = OK!';
    except
        on E: Exception do
            edMess.Text := E.Message;
    end;
end;

procedure TfMariaDBZ.btExitClick(Sender: TObject);
begin
    if zdbConnection.Connected then
        zdbConnection.Disconnect;
    Close;
end;
```

Note: If you wonder why in the code above the non-standard port 3307 is used, the reason simply is that on my system, MariaDB listens to that port (port 3306 being used by my MySQL server).

As for MySQL, the client library (DLL) must be present in the project output folder (later, in the same directory as the executable). The MariaDB library is called libmariadb.dll and if you use MariaDB 10.6, installed into the default directory, you find the DLL in C:\Program Files\MariaDB 10.6\lib. To note that you can also use libmysql.dll (the one shipped with MySQL 8.0 works fine with my sample application).

Note: The "world" database is not included with MariaDB. To use the sample application as is, you'll have to create this database yourself; cf. my tutorial [Web development environment setup on MS Windows: MariaDB database server](#).

The code of the TfMariaDBZ.btQueryClick method (executed when the "Query" button is pushed) is exactly the same as for MySQL:

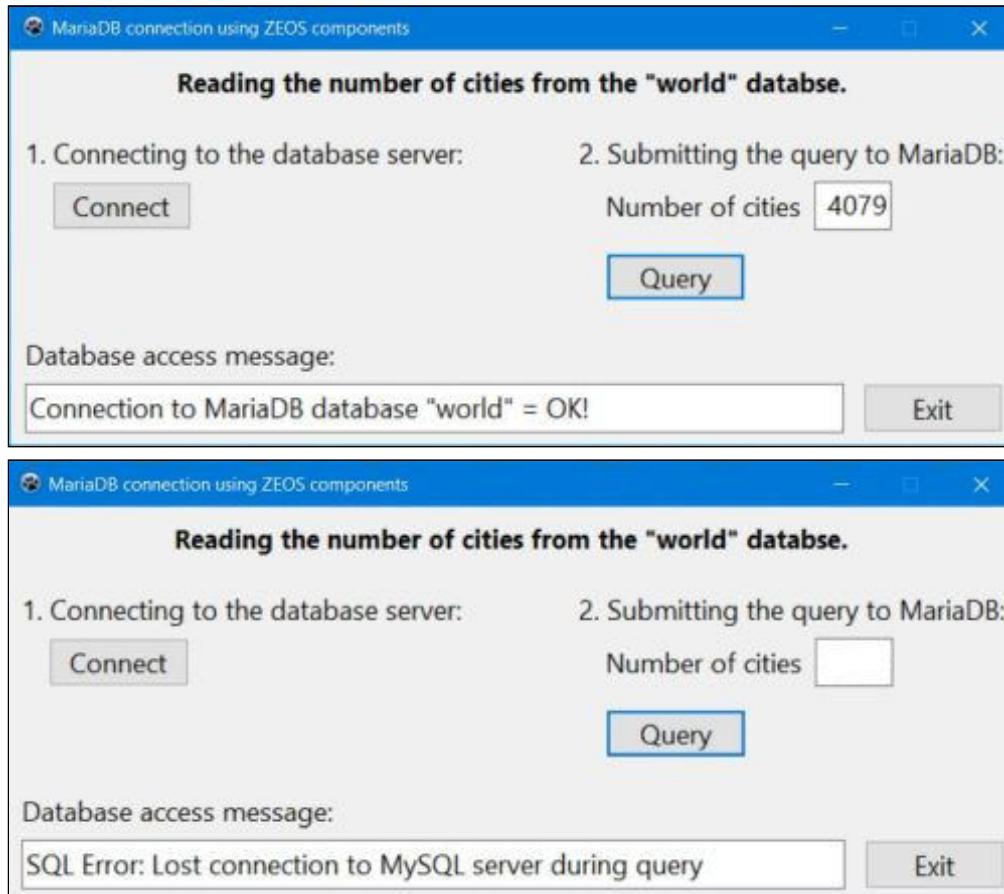
```
procedure TfMariaDBZ.btQueryClick(Sender: TObject);
var
    Count: Integer;
begin
    if zdbConnection.Connected then begin
        // Query the database
        zdbQuery.SQL.Clear;
        zdbQuery.SQL.Add('SELECT count(*) FROM city');
        try
            zdbQuery.Open;
            if zdbQuery.EOF then
                Count := 0
            else
                Count := zdbQuery.Fields[0].AsInteger;
            zdbQuery.Close;
```

```

    // Display the query result
    edCities.Text := IntToStr(Count);
except
    on E: Exception do
        edMess.Text := E.Message;
end;
end;
end;
end;

```

The screenshot on the left shows a successful query. The one on the right shows the error that you get if after having connected to the server, this one disconnects and you try to run a query (sent to server that is no more online, without that the client knows about this).



If you find this text helpful, please, support me and this website by [signing my guestbook](#).