

**C07215/C07515**

## **Advanced Web Technologies**

### **Lab 2**

Dr Stephan Reiff-Marganiec  
Department of Computer Science

September 2013  
Updated October 2015



### Lab 2: Database Programming in C# with MySQL

The main task of this lab session is to develop a simple C# database program that interacts with a MySQL database. We decided to use MySQL rather than SQLServer as being able to work with an independent external database is very useful and many of the issues discussed work for SQLServer, some in easier ways as better support for SQLServer is built into VS Studio.

In this lab you will learn about many object oriented features as well as database access.

- You will learn how to import SQL from terminal to create tables and how to query/add/delete records by executing SQL query in C#.

I would expect you to be able to finish the work within two hours. This time there are no special “practice” bits, you just have to follow all the instructions and complete anything where you are asked to do something. Note also that there is less “handholding” here in that there are no detailed step by step instructions for everything – tasks are described and you should be able to use the experience gained in earlier labs to complete the tasks. Also, feel free to explore and try other things!

### MySQL Database Setup

The password associated with your MySQL account on **mysql.mcscw3.le.ac.uk** can be found in a hidden text file in your Linux home directory. You will need to remote login to a Linux host from a Windows machine (or perform the database tasks while being logged into Linux).

**Download Putty.exe and login to Linux.** (you can skip this step if you are working directly in Linux). In order to login to Linux from Windows, you need PuTTY. PuTTY is a free implementation of Telnet and SSH for Win32, which you can download from

<http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html>,

(Please select the corresponding version: Windows 95, 98, ME, NT, 2000, XP and Vista on Intel x86). For campus based students: PuTTY is available in your software, but probably needs to be installed. Once installed, start `putty.exe` and you will see the window below:

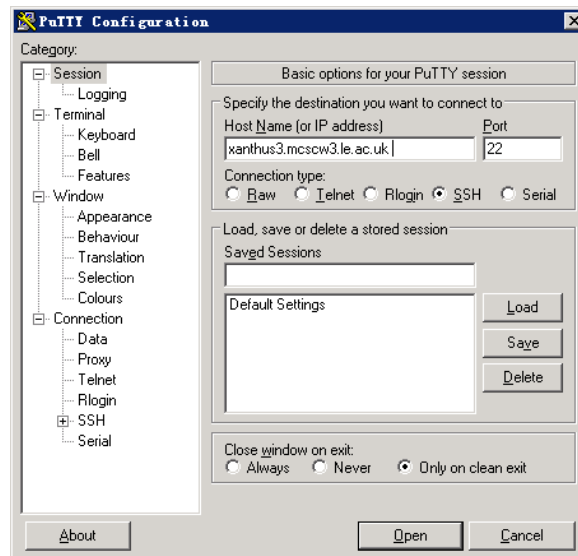


Figure 1

Fill in the “Host Name (or IP address)” field with `xanthus3.mcscw3.le.ac.uk`, click open and then type your Linux account username (e.g. `yh37`) and password. After that you should be able to see a terminal window as follow:

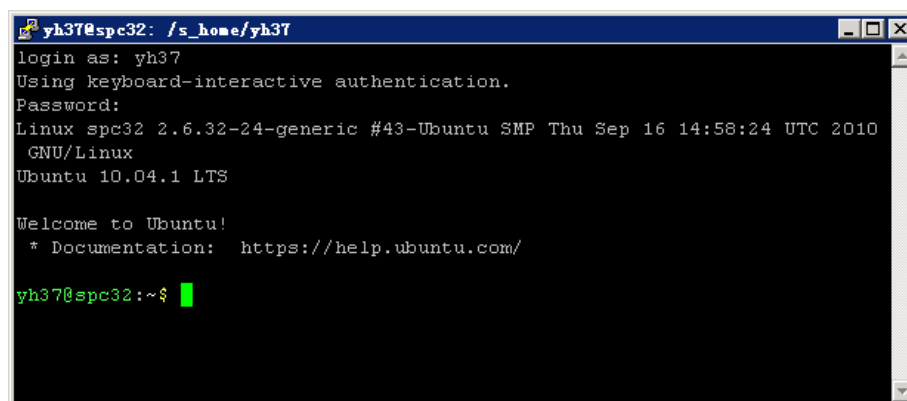


Figure 2

**Check your MySQL account password.** To find your password, type the command below in your home directory:

```
more .my.cnf
```

You should see the output like that:


```
[client]
```

```
    Password=xxxxxx
```

“xxxxx” is your MySQL password, you’ll need it in the next step.

### Create Tables and Import Testing Data

You will need to create a new table called “*employee*” which contains following fields:

Name	Type	Length	Decimals	Allow Null	
id	int	11	0	<input type="checkbox"/>	
firstname	varchar	255	0	<input checked="" type="checkbox"/>	
lastname	varchar	255	0	<input checked="" type="checkbox"/>	
address	varchar	255	0	<input checked="" type="checkbox"/>	
salary	double	0	0	<input checked="" type="checkbox"/>	

And simply add some data for testing:

id	firstname	lastname	address	salary
2	James	Gordon Brown	white house	200000
3	Bush	George W	downing street	300000

Figure 3 MySQL table structure and testing data

As we don’t use a graphical user interface to access the MySQL server, type the command line below in the Linux terminal window to download the SQL file : `employee.sql`

```
wget http://tyche.mcscw3.le.ac.uk:8080/mysql/employee.sql
```

`employee.sql` will be saved in your home directory, the next step is to import this sql file into your database – (!!don’t forget to replace both occurrences of **user** with your account name e.g. **yh37!!**)

```
mysql -u user -p -h achilles.mcscw3.le.ac.uk user<employee.sql
```

The database tables for this assignment have now been created. You can always login with **mysql -u user -p -h achilles.mcscw3.le.ac.uk** and type `show tables` to see the tables in the database and of course run any additional sql commands that you feel like. MySQL is well documented, SQL is something that you should be familiar with, so we will not go into details here.

### Task 2: Create .NET Bean Classes

In the previous lab sessions, we have learnt how to protect a field in a class by reading and writing to it through a property, in this step you’ll need to create an employee class for the database structure. Firstly, launch Visual Studio.NET and follow the instructions below:

**Create a new Project** (From File->New->Project->Visual C#->ConsoleApplication) and name it **EmployeeDBApplication**.

**Create new Class** (Right click “EmployeeDBApplication” in the Solution Explorer and select Add->Class) giving a filename **Employee.cs**.

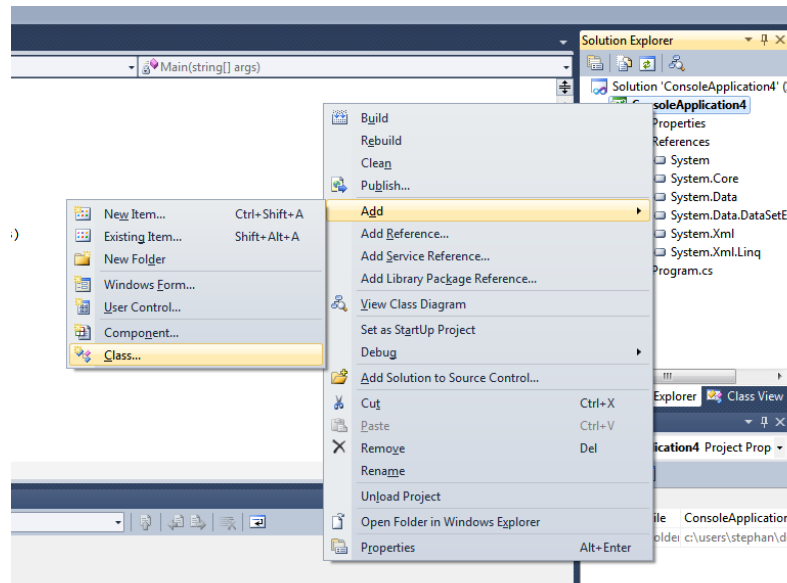


Figure 4

**Add properties to Employee class.** Class *Employee* should contain several properties (or attributes), as we learnt in Lab 1. Encapsulation provides a way to protect data from accidental corruption. In this case, add fields `id`, `firstname`, `lastname`, `address`, `salary` to the *Employee* class. The fields should all be private, `id` is of type `int`, `salary` of type `double` and the remaining three fields are of type `string`. Leave the main method empty for now.

Getters and setters for a class are probably the most common methods in C#. It is easy and trivial to write these methods manually but Visual Studio provides a nice feature to generate them automatically: Simply move the cursor to a private variable, right click it to open the context menu and select

“*Refactor->Encapsulate Field*”.

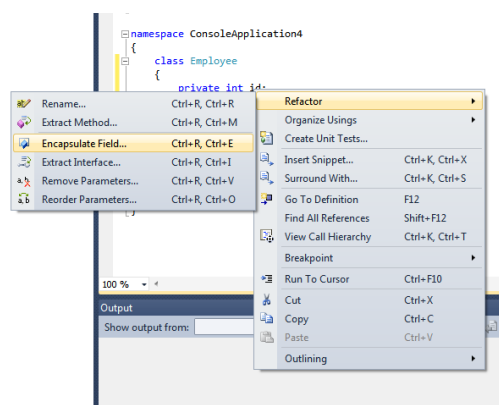


Figure 5

Repeat the same steps for the other private fields to generate getter and setter methods for them, too.

**Override the ToString() method.** In order to print an employee directly with `Console.WriteLine()`, you will need to override the default `ToString()` method inherited from `System.Object` (every `Object` has a `ToString` method by default, but the default behavior might not be what you want); remember to use the keyword `override` in the method declaration [refer to the last lab]:

We want the method to return the following: `"Employee ID:" + this.Id + "\tName:" + this.firstname + "\t" + this.lastname + "\n\tSalary:" + this.salary + "\t Address:" + this.address;`

Your `Employee` class is complete now.

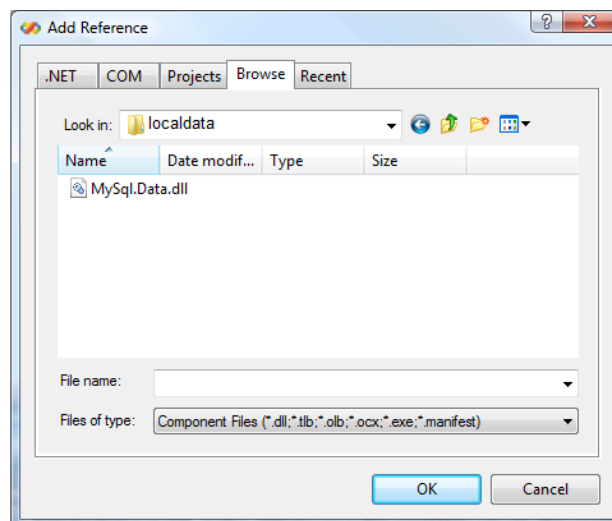
### List All Employees from the Database

Let us just recap what we have: we have a datatable in our database with some data and we have a C# object that allows for data of the same schema to be stored. Now, let's get some data from the database.

Of course we will work with our *EmployeeDBApplication* from earlier on, look at it in the solution explorer *Add Reference->Browser*; choose *MySQLData.dll* (This can be downloaded from:

`http://tyche.mcscw3.le.ac.uk:8080/mysql/MySQL.Data.dll,`

and you should then save it to the Z: drive, from where you can then locate it inside VS)



## Query Database with SQL

What is connection string? A connection string is a string that specifies information about a data source and the means of connecting to it. It is passed in code to an underlying driver or provider in order to initiate the connection. Here is an example:

```
//      Connection string description
//      Database   : database name
//      Data Source: MySQL server address
//      User id    : MySQL user id
//      Password   : MySQL password
string myConnectionString = "Database=your_id;Data
Source=achilles.mcscw3.le.ac.uk ;User Id=your_id;Password=your_password;
```

Switch to the *Program.cs* tab and type the following source code, make sure to replace the database username and password with your account name and MySQL password. A static method `getConnection()` is defined in order to get an object of type `Connection` – your way to access the database. Another method `getAllEmployees()` will open the connection, initialise a `MySqlCommand` with an SQL query and execute it. The Query result will be stored in a `MySqlDataReader`, which allows you to iterate over the result set by checking the value of `HasRows`. We initialise every employee in the result set and add them to a list, which will finally be returned by the method.

```
using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using MySql.Data.MySqlClient;

namespace EmployeeDBApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            Program app=new Program();

            // The loop will print every employee in the list

            foreach (Employee e in app.getAllEmployee()) {

                Console.WriteLine(e); // Employee overrided ToString() method

            }
        }
    }
}
```

```
string s = Console.ReadLine();

//Pause

}

static MySqlConnection getConnection(){

//this static method will return a connection constructed by connection string

    string myConnectionString = "Database=your_id;Data
Source=achilles.mcscw3.le.ac.uk ;User Id=your_id;Password=your_password";
    MySqlConnection connection = new MySqlConnection(myConnectionString);

    return connection;

}

public List<Employee> getAllEmployees() {

//a list of employees

    List<Employee> employees = new List<Employee>();

    try
    {

        //Open connection

        MySqlConnection connection = getConnection();

        connection.Open();

        // SQL query assignment

        MySqlCommand mycm = new MySqlCommand("select * from employee",
connection);

        //execute query

        MySqlDataReader msdr = mycm.ExecuteReader();

        while (msdr.Read())
        {

            if (msdr.HasRows)
            {

                Employee e = new Employee();

//read each column and assign value to corresponding property of Employee

                e.Id = msdr.GetInt32("id");

                e.Firstname = msdr.GetString("firstname");
```



replace *your\_id* with  
your account name.

replace *your\_password*  
with your MySQL  
password



```

        e.Lastname = msdr.GetString("lastname");

        e.Salary = msdr.GetDouble("salary");

        e.Address = msdr.GetString("address");

        //add this employee to the list

        employees.Add(e);

    }

}

msdr.Close();

connection.Close();

}

catch (Exception ex) {

    Console.WriteLine(ex.ToString());

}

//return employee list

return employees;

}

}

}

```



Initialise instance of *Employee*, assign property value and add every employee to the return list

You should see output as follows with 2 records being displayed in the window (the actual data might vary, and definitely will if you added further employees to the data table):

### Query for a Particular Employee

In the previous task we list all employees from the MySQL database; similarly we can add another method to obtain an instance of *Employee* by its *id*. Write a method `public Employee getEmployeeById(int id)`.

You will need similar code to the `getAllEmployees` method, but you will have to do a touch more work on the `SQLQuery`.

MySQL server variables are also denoted using '?' symbol. When encountered in query text, such fragments are treated as parameters only if they are found in `MySQLCommand.Parameters` collection.

```
// before this ... open connection

MySQLCommand mycm = new MySQLCommand("", connection);

mycm.Prepare();

mycm.CommandText = String.Format("select * from employee where id=?id_para");

//replace parameter "?id_para" with value of id

mycm.Parameters.AddWithValue("?id_para", id);

//after this ... execute query
```



Parameterise query,  
similar to  
*PrepareStatement* in  
Java. SQL parameter  
can be specified

Adding `Console.WriteLine(app.getEmployeeById(2));` to the Main method to display the detail of the employee whose id is 2, assuming you implemented the above method correctly.

### Add an Employee to the database

Adding an employee to the database is similar to the example for querying, but there are some important differences:

- Insert statements do not return result set
- Insert queries are executed using `ExecuteNonQuery()` instead of `ExecuteReader()`

Write a new method `public bool addEmployee(Employee e)`, which takes an *Employee* as parameter and adds a new record to the database. You should be able to do most of this already, but you will need to have a few new lines for the `MySQLCommand`:

```
//e is you employee object

//as before open a connection before getting here

MySQLCommand mycm = new MySQLCommand("", connection);

mycm.Prepare();

mycm.CommandText = String.Format("insert into
employee(id,firstname,lastname,address,salary) values
(?id_para,?firstname_para,?lastname_para,?address_para,?salary_para)");

mycm.Parameters.AddWithValue("?id_para", e.Id);

mycm.Parameters.AddWithValue("?firstname_para", e.Firstname);

mycm.Parameters.AddWithValue("?lastname_para", e.Lastname);

mycm.Parameters.AddWithValue("?address_para", e.Address);

mycm.Parameters.AddWithValue("?salary_para", e.Salary);
```



`ExecuteNonQuery()`  
instead of

```
//insert statement do not return any rows

mycm.ExecuteNonQuery();

//close connection after this
```

Add code to the Main method to create a new employee object and call the addEmployee method to see if things work. After calling this a new employee should be in your database (you can check with Select on the MySQL command console or by listing all employees using your C# programme).

### Delete an Employee from database [sample answer will be provided]

Write a delete method that will delete an employee from the database; like add it will take an id as argument and should return true or false depending on its success. SQL syntax for deleting a record is:

```
delete from employee where id=4      //(delete empolyee with id 4)
```

### Advanced Search with Wild Card Support [sample answer will be provided]

For those who manage to finish all these tasks above -- the final challenge is to develop a slightly complicated search method, which provides following interface:

```
public List<Employee> advancedSeaerch(string keyword) {

    //complete your method here

}

// You might need to use SQL like condition
// tutorial can be found ats http://www.techonthenet.com/sql/like.php
```



This method takes a string as input parameter and returns a list of Employees; it should list the employees whose address field contains a particular keyword (Non case-sensitive). The method should be able to support wild cards in the string (a wild card is a "\*", which essentially matches any character).

**Consider these examples for extra explanation.** Suppose we have these addresses:

*83A London Road, Leicester, UK*

*Market Harborough, Leicestershire*

*Kiev, Ukraine*

*Leicester Square, London*

*Birmingham New Street, Birmingham UK*

(1) Keyword: **Leicester**

Result:

83A London Road, Leicester, UK  
Market Harborough, Leicestershire  
Leicester Square London

(2) Keyword: **Leicester\***

Result:

Leicester Square London

(3) Keyword: **\*UK**

Result:

83A London Road, Leicester, UK  
Birmingham New Street, Birmingham UK

**Note** that for the last two tasks sample answers will be provided on Monday 22<sup>nd</sup> October.