

Spike Outcomes

Set: 1b

Languages C#

Names: Reuben Wilson and Yan Chernikov

Goals:

The overall goal of this spike was to implement the classes required to connect to a MySql database and manipulate the data contained within using C# programming language. Data manipulation was achieved using basic SQL queries. There was a special requirement, where the data returned from executing a select query was used to instantiate a number of classes, each class representing a single row from the result set. In this case, the class instantiated is called archer due to the fact that the table being worked with is called archer where each unique row represents a single archer value.

Some secondary goals include establishing a clear understanding of which objects are required to initialise such a connection and how they interact with each other. As an example, the team developed an understanding of how a MySqlCommand object needs to be initialised using a valid MySqlConnection object in order to execute queries.

Personnel:

Primary - Reuben Wilson 9988289

Secondary - Yan Chernikov 9991379

Technologies, Tools, and Resources used:

The technologies used to develop the overall result for this spike are defined below:

C#

.NET Framework 4.5

Mono Framework

MySql

ADO .NET

The tools used to complete this spike are defined below:

Xamarin IDE

Terminal

Mac OSX

The resources that were used to gain the knowledge required to complete this spike are defined below:

MSDN C# Documentation Portal:

<http://msdn.microsoft.com/en-AU/library/618ayhy6.aspx>

MSDN ADO .NET Documentation Portal:

[http://msdn.microsoft.com/en-us/library/e80y5yhx\(v=vs.110\).aspx](http://msdn.microsoft.com/en-us/library/e80y5yhx(v=vs.110).aspx)

Stack Overflow Community Discussion Board:

<http://stackoverflow.com>

Tasks undertaken:

1. The first task undertaken was to ensure that a MySqlConnection object could be instantiated with the appropriate connection string. The figure below offers a detailed insight into the code that was used to achieve this as well as a detailed explanation as to what the code is doing.

```
public DBHandler ()
{
    _serverName = "localhost";
    _databaseName = "archery_db";
    _user = "root";
    _password = "821190";
    string connectionString = "Server=" + _serverName + ";Database=" + _databaseName + ";User ID=" + _user + ";Pwd=" + _password + ";";
    _connection = new MySqlConnection (connectionString);
}
```

Figure 1b.1 - Constructor for DBHandler Class

The class constructor, as represented in Figure 1b.1 above, is responsible for instantiating an instance of the class that manages the program's interaction with the database. The class is called DBHandler. There are a number of fields, which are used to store the appropriate connection variables, which are then concatenated into a single string (see connectionString variable above). The field _connection is responsible for storing a single instance of the class MySqlConnection, which is initialised using the appropriate connection string.

2. After an instance of MySqlConnection was successfully instantiated, it was time to ensure that the connection could be opened and closed properly. The two figures below demonstrate the code that was implemented in order to appropriately open and close connections to the database.

```
private void OpenConnection()
{
    try
    {
        if (_connection != null && _connection.State == System.Data.ConnectionState.Closed)
        {
            _connection.Open ();
        }
    }
    catch (MySqlException e)
    {
        Console.WriteLine(e.Message);
    }
}
```

Figure 1b.2 - Open method for class, DBHandler

The method OpenConnection utilises a try/catch block in order to handle any exceptions that may occur when attempting to open a connection to the database. There is a simple condition, which is checked before the connection is opened, and that is, ensure that the connection is currently closed and that the MySqlConnection object exists before opening.

```
private void CloseConnection()
{
    try
    {
        if (_connection != null && _connection.State == System.Data.ConnectionState.Open)
        {
            _connection.Close ();
        }
    }
    catch (MySqlException e)
    {
        Console.WriteLine(e.Message);
    }
}
```

Figure 1b.3 - Close method for class, DBHandler

The method CloseConnection utilises a try/catch block in order to handle any exceptions that may occur when attempting to close a connection to the database. There is a simple condition, which is checked before the connection is closed, and that is, ensure that the connection is currently open and that the MySqlConnection object exists before closing.

3. After connectivity was achieved, the next task was to ensure that both types of SqlCommand could be executed successfully on the database. The two types are classed as query and non query. An SqlCommand of type query retrieves data from the database (consider a select query) and an SqlCommand of type non query modifies data within the database (consider the sql queries delete, update and insert). The code method depicted in the figure below offers an insight into how a SqlCommand of query type is handled.

```
public void ExecuteNonQuery(string nonQueryString)
{
    OpenConnection();
    Console.WriteLine(
        "The query: " + nonQueryString);
    MySqlCommand command = new MySqlCommand (nonQueryString, _connection);
    try
    {
        int numberOfRecords = command.ExecuteNonQuery();
        Console.WriteLine(
            numberOfRecords + " row(s) affected.");
        CloseConnection();
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
        CloseConnection();
    }
}
```

Figure 1b.4 - ExecuteNonQuery method for class, DBHandler

The figure above demonstrates the method used to execute an SqlCommand of type non query. The query is passed as a string to the method. The method implements a try/catch block in order to handle any errors that may occur in an appropriate manner. Before the try block, the connection to the database is opened. Within the try block, the query is executed, the number of rows manipulated in the database is output to the user and the connection is finally closed.

```

public List<Archer> Select(string queryString)
{
    OpenConnection ();
    MySqlCommand command = new MySqlCommand (queryString, _connection);
    List<Archer> results = new List<Archer> ();
    MySqlDataReader reader = command.ExecuteReader();

    while (reader.Read())
    {
        if (reader.FieldCount != 17)
        {
            Console.WriteLine ("Error! Incorrect data!");
            return null;
        }
        Archer archer = new Archer ();

        archer.id = int.Parse(reader[0].ToString());
        archer.last_updated = reader[1].ToString();
        archer.archived = bool.Parse(reader[2].ToString());
        archer.fin_year = int.Parse(reader[3].ToString());
        archer.gender = char.Parse(reader[4].ToString());
        archer.surname = reader[5].ToString();
        archer.given_name = reader[6].ToString();
        archer.initial = reader[7].ToString().Length > 0 ? char.Parse(reader[7].ToString()) : ' ';
        archer.join_date = reader[8].ToString();
        archer.birthyear = int.Parse(reader[9].ToString());
        archer.status = char.Parse(reader[10].ToString());
        archer.club = reader[11].ToString();
        archer.notes = reader[12].ToString();
        archer.title = reader[13].ToString();
        archer.id_number = int.Parse(reader[14].ToString());
        archer.default_equipment_id = int.Parse(reader[15].ToString());
        archer.default_discipline_id = int.Parse(reader[16].ToString());

        results.Add (archer);
    }

    Console.WriteLine(results.Capacity + " results returned.\n");

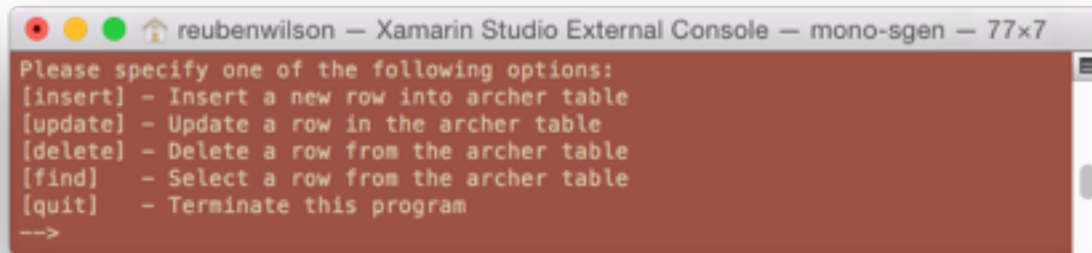
    reader.Close();
    CloseConnection();
    return results;
}

```

Figure 1b.5 - Select method for class, DBHandler

The figure above demonstrates the method used to execute an SqlCommand of type query. The query is passed as a string to the method. The method implements a try/catch block in order to handle any errors that may occur in an appropriate manner. Before the try block, the connection to the database is opened. Within the try block, the query is executed, and a List of instantiated archer objects is returned. There is one archer initialised for each row returned within the result set.

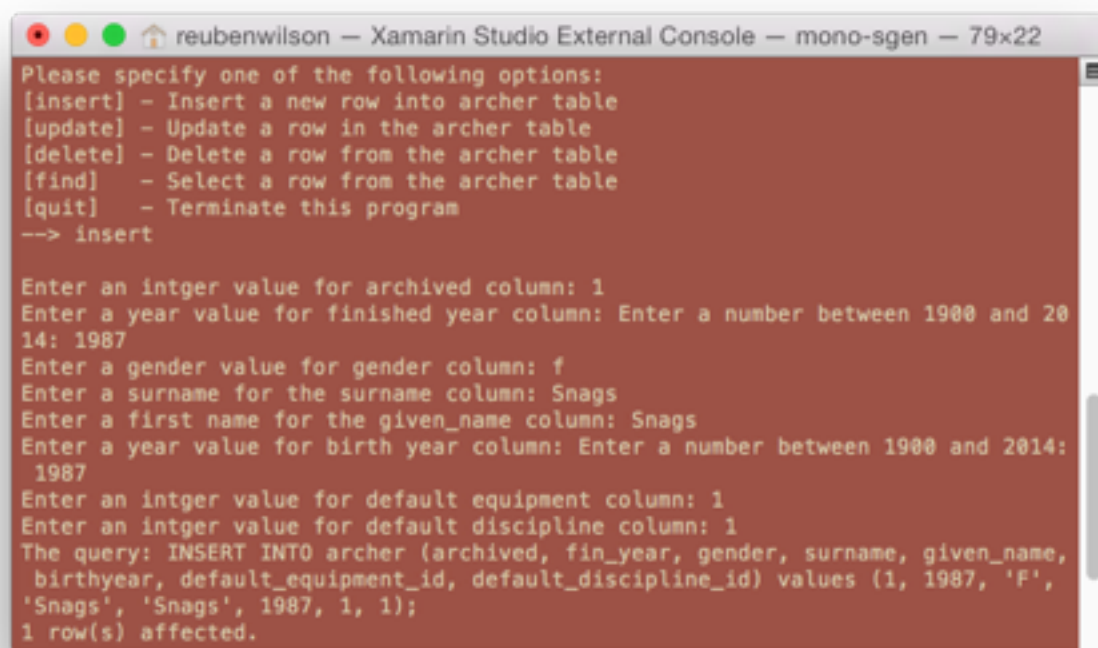
4. The next task was to piece all of the functionality together in order to offer a small, functional program that can be interacted with by a user at the terminal. The application offers the user the ability to view and manipulate the data within the database.

What I found out:A screenshot of a console window titled 'reubenwilson - Xamarin Studio External Console - mono-sgen - 77x7'. The window has a dark red background with white text. It displays a menu with the following options: [insert] - Insert a new row into archer table, [update] - Update a row in the archer table, [delete] - Delete a row from the archer table, [find] - Select a row from the archer table, and [quit] - Terminate this program. A prompt '-->' is at the bottom.

```
reubenwilson - Xamarin Studio External Console - mono-sgen - 77x7
Please specify one of the following options:
[insert] - Insert a new row into archer table
[update] - Update a row in the archer table
[delete] - Delete a row from the archer table
[find]   - Select a row from the archer table
[quit]   - Terminate this program
-->
```

Figure 1b.6: Basic console application with menu

The menu output of the console program developed to interact with the database.

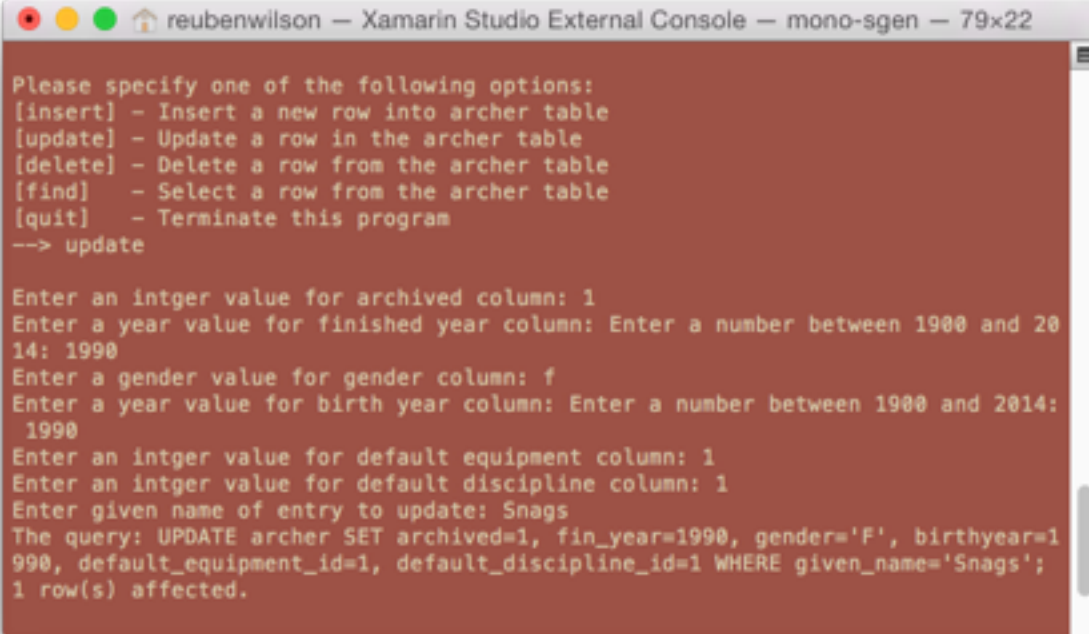
A screenshot of a console window titled 'reubenwilson - Xamarin Studio External Console - mono-sgen - 79x22'. The window has a dark red background with white text. It shows the same menu as Figure 1b.6, but with 'insert' entered. Below the menu, it prompts for values for 'archived', 'finished year', 'gender', 'surname', 'given_name', 'birth year', 'default equipment', and 'default discipline' columns. It then displays the SQL query: 'INSERT INTO archer (archived, fin_year, gender, surname, given_name, birthyear, default_equipment_id, default_discipline_id) values (1, 1987, 'F', 'Snags', 'Snags', 1987, 1, 1);' and the result '1 row(s) affected.'.

```
reubenwilson - Xamarin Studio External Console - mono-sgen - 79x22
Please specify one of the following options:
[insert] - Insert a new row into archer table
[update] - Update a row in the archer table
[delete] - Delete a row from the archer table
[find]   - Select a row from the archer table
[quit]   - Terminate this program
--> insert

Enter an intger value for archived column: 1
Enter a year value for finished year column: Enter a number between 1900 and 2014: 1987
Enter a gender value for gender column: f
Enter a surname for the surname column: Snags
Enter a first name for the given_name column: Snags
Enter a year value for birth year column: Enter a number between 1900 and 2014: 1987
Enter an intger value for default equipment column: 1
Enter an intger value for default discipline column: 1
The query: INSERT INTO archer (archived, fin_year, gender, surname, given_name,
birthyear, default_equipment_id, default_discipline_id) values (1, 1987, 'F',
'Snags', 'Snags', 1987, 1, 1);
1 row(s) affected.
```

Figure 1b.7: Results from running 'insert' option

The output represented above demonstrates the program's output following a successful insert query being executed on the database. To insert a new record, the 'insert' menu option is invoked.



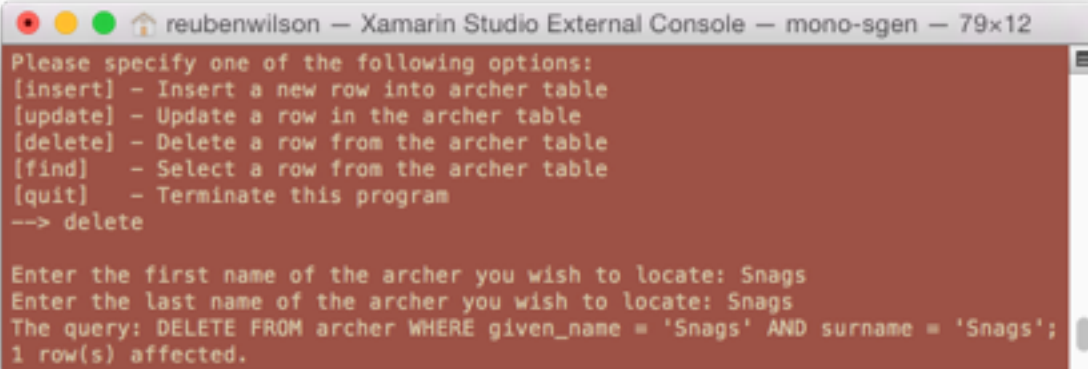
```
reubenwilson — Xamarin Studio External Console — mono-sgen — 79x22

Please specify one of the following options:
[insert] - Insert a new row into archer table
[update] - Update a row in the archer table
[delete] - Delete a row from the archer table
[find]   - Select a row from the archer table
[quit]  - Terminate this program
--> update

Enter an integer value for archived column: 1
Enter a year value for finished year column: Enter a number between 1900 and 2014: 1990
Enter a gender value for gender column: f
Enter a year value for birth year column: Enter a number between 1900 and 2014: 1990
Enter an integer value for default equipment column: 1
Enter an integer value for default discipline column: 1
Enter given name of entry to update: Snags
The query: UPDATE archer SET archived=1, fin_year=1990, gender='F', birthyear=1990, default_equipment_id=1, default_discipline_id=1 WHERE given_name='Snags';
1 row(s) affected.
```

Figure 1b.8. Results from running 'update' option

The output represented above demonstrates the program's output following a successful update query being executed on the database. To update a record, the 'update' menu option is invoked.



```
reubenwilson — Xamarin Studio External Console — mono-sgen — 79x12

Please specify one of the following options:
[insert] - Insert a new row into archer table
[update] - Update a row in the archer table
[delete] - Delete a row from the archer table
[find]   - Select a row from the archer table
[quit]  - Terminate this program
--> delete

Enter the first name of the archer you wish to locate: Snags
Enter the last name of the archer you wish to locate: Snags
The query: DELETE FROM archer WHERE given_name = 'Snags' AND surname = 'Snags';
1 row(s) affected.
```

Figure 1b.9. Results from running 'delete' option

The output represented above demonstrates the program's output following a successful delete query being executed on the database. To delete a record, the 'delete' menu option is invoked.


```

Please specify one of the following options:
[insert] - Insert a new row into archer table
[update] - Update a row in the archer table
[delete] - Delete a row from the archer table
[find] - Select a row from the archer table
[quit] - Terminate this program
--> find

Enter the first name of the archer you wish to locate: Sarah
Enter the last name of the archer you wish to locate: Walters
4 results returned.

4 29/04/2007 12:00:00 AM True 2006 F Walters Sarah 1/01/2005 12:00:00 AM 1989 J BHCA Ms 537 8 1

```

Figure 1b.10. Results from running 'find' option

The output represented above demonstrates the program's output following a successful select query being executed on the database. To find a single record, the 'find' menu option is invoked.

Open issues/risks:

The major issue that the team uncovered was the fact that Sql objects, as referenced in the lectures (i.e. SqlConnection etc) are not valid within ADO .NET. You have to use MySql objects (i.e. MySqlConnection). I faced a big issue when trying to use SqlCommand objects, though. We're not certain as to whether or not it was because of our object oriented design, but for some reason, the query string within a SqlCommand object is not executing the way it should be. I had to implement a work around in order to accommodate for this obstacle. That work around involved extracting the query string from the command object and then instantiating a new command object with the extracted string. It makes no sense why this issue is apparent. The figure below demonstrates the code that I used to extract the query string from a MySqlCommand object.

```

private static string GetQueryStringFromCommand(MySqlCommand theCommand)
{
    string query = theCommand.CommandText;

    foreach (MySqlParameter p in theCommand.Parameters)
    {
        query = query.Replace(p.ParameterName, p.Value.ToString());
    }
    return query;
}

```

Figure 1b.11. GetQueryStringFromCommand method

Recommendations:

In order to resolve the above stated issue, I will seek professional guidance from other peers and academic staff in order to try and better understand what the issue is and what the root cause is.