Lab 8 Due next lab period – week of June 14-16

Purpose

1. Introduction to the Visual Studio .NET framework with C#
2. Querying information in a Microsoft SQL Server database using Microsoft ADO.NET
3. Define entity classes for holding data retrieved from a database
4. Use LINQ to SQL to query a database and populate instances of entity classes
5. Create a custom *DataContext* class for accessing a database in a typesafe manner

What to Submit

1. Upload your project, compressed into a .zip file to the Dropbox on D2L.
2. Submit the answers to the questions below to D2L.

Evaluation (12 marks):

1. Answers to questions from part 1 (3 marks)
2. Your Solution will be assessed based on (3 marks each):
    - Successful connection to database using integrated security and no embedded passwords
    - Correctly outputting the salesperson information
    - C#.NET code style – variables named appropriately, comments used in the code where needed, code is readable and understandable. Header comment included.
3. **All work must be individual.** Any plagiarism will result in a mark of 0 and possibly additional penalties.

Description

The Microsoft ADO.NET technology is a library of objects specifically designed to make it easy to write applications that use databases. The model defined by ADO.NET is based on the notion of data providers. Each database management system (such as SQL Server, Oracle, IBM DB2, and so on) has its own data provider that implements an abstraction of the mechanisms for connecting to a database, issuing queries, and updating data. By using these abstractions, you can write portable code that is independent of underlying database management system. In this lab, you will connect to a database managed by a SQL Server 2008, but the techniques that you will learn are equally applicable when using a different database management system.

The AdventureWorks company is a fictitious commercial entity that manufactures and sells bicycles. Microsoft built the sample databases for training and educational purposes. There are several parts to the data organization of AdventureWorks. The AdventureWorks OLTP database supports standard online transaction processing scenarios such as Manufacturing, Sales, Purchasing, Product Management, Contact Management, and Human Resources. The AdventureWorks DW is a data warehouse. The SQL Azure OLTP database is a SQL Azure version of the database. If you are interested, the link for the information is at http://msftdbprodsamples.codeplex.com/ and the downloads are at http://msftdbprodsamples.codeplex.com/releases/view/55330.

The data dictionary of AdventureWorks is found at http://msdn.microsoft.com/en-us/library/ms124438(v=sql.100).aspx (the Microsoft Word version is at http://hal.cs.camosun.bc.ca/~langs/comp157-12/adventureworks.docx -- do not print – it's over 100 pages).

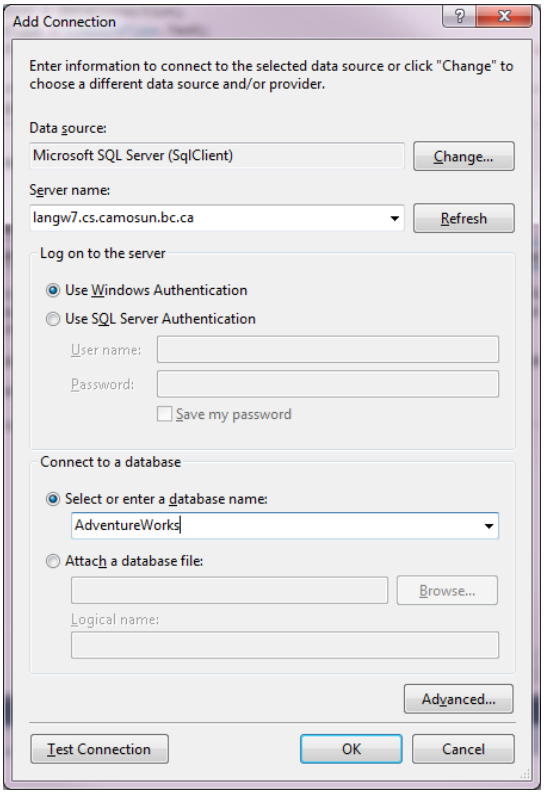
And the SQL Server objects list is http://msdn.microsoft.com/en-us/library/ms124425(v=sql.100)

The AdventureWorks schema is viewable at
http://hal.cs.camosun.bc.ca/~langs/comp157-12/AdvWorks2005.htm or you can download and install the files at
https://www.microsoft.com/en-ca/server-cloud/products/sql-server-editions/sql-server-express.aspx

Process

1. View the OLTP schema (online transaction processing) of AdventureWorks database using one of the methods described above in the Description. The line with the arrow indicates a relationship between two objects (tables).
    a. Questions: (to be submitted to D2L) How many different schemas are there?
    b. How many different schemas make use of the Product object?
    c. Which two objects share a 1:1 relationship?
    d. Which two objects share a 1:M relationship?
    e. Which two share a M:M relationship through an intermediate object? -- hints: look at the PK (primary key) definitions in the objects.
2. Create a new C# console application project called ReportOrders.
3. a) From the menu select Project then Add New Data Source.
   b) Select Database in the "Choose a Data Source Type" panel. Click Next.
   c) Select Dataset in the "Choose a Database Model" panel. Click Next.
   d) Click New Connection... button in the "Choose Your Data Connection" panel. Select the "No, exclude sensitive data from the connection string." Option.

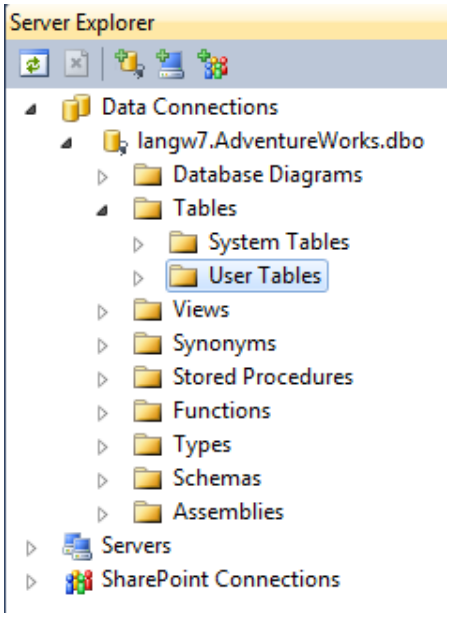
   Make sure the Data Source and Server name are set as shown (Microsoft SQL Server and langw7.cs.camosun.bc.ca). The database name is AdventureWorks. Click Test Connection to confirm. This Test will work only if you are on campus. If you are doing this process from home or outside the Camosun student domain, it will not work.



The connection should show as “langw7.AdventureWorks.dbo”. Click Finish. This should work while you are logged on with your Camosun student domain C999999 account.

When asked to choose your database objects: selected all.

Click on View in the menu and select Server Explorer. Expand the Data Connections to reveal the newly connected langw7 SQL Server connection.



4. In Solution Explorer, right click the file Program.cs and rename it to Report.cs.
5. In the Code and Text Editor window, add the following *using* statements to the list at the top of the Report.cs. file:

```
using System.Data;
using System.Data.Sqlclient;
```

6. The System.Data namespace contains many of the types used by ADO.NET. The System.Data.SqlClient namespace contains the SQL Server data provider classes for ADO.NET. These classes are specialized versions of the ADO.NET classes, optimized for working with SQL Server.
7. In the Main method of the Report class, add the following statement, which creates a SqlConnection object:

```
static void Main(string[] args)
{
   SqlConnection dataConnection = new SqlConnection();
}
```

SqlConnection is a subclass of an ADO.NET class called Connection. It is designed to handle connections to SQL Server databases

8. After the variable declaration, add a try/catch block to the Main method as shown next. All the code that you will write for gaining access to the database goes inside the try part of this block. In the catch block, add a simple handler that catches SqlException exceptions

```
static void Main(string[] args)
{  ...

   try
   {
          // You will add your code here in a moment
   }
   catch (SqlException e)
    {
         Console.WriteLine("Error accessing the database: {0}", e.Message);
         string exceptresp = Console.ReadLine();
    }
}
```

A SqlException is thrown if an error occurs when accessing a SQL Server database

9. Replace the comment in the try block with the code shown here:

```
SqlConnectionStringBuilder builder = new SqlConnectionStringBuilder();
builder.DataSource = "langw7.cs.camosun.bc.ca";
builder.InitialCatalog = "AdventureWorks";
builder.UserID = "student01";
builder.Password = "Winter2012$";  // oops !
dataConnection.ConnectionString = builder.ConnectionString;
```

To connect to a SQL Server database, you must construct a connection string that specifies the database to connect to, the instance of SQL Server holding this database, and how the application will identify itself as a valid user of the database to SQLServer. The simplest way to do this is to use a SqlConnectionStringBuilder object. The SqlConnectionStringBuilder class exposes properties for each of the elements of a connection string. You can then read a complete connection string that combines all of these elements in the correct format from the ConnectionString property. This code uses a SqlConnectionStringBuilder object to build a connection string for accessing the

AdventureWorks database running on the instance of SQL Server. The code specifies that the connection will not use Windows Authentication to connect to the database. (Setting `builder.IntegratedSecurity = true;` means use Windows Authentication ...e.g. your student domain account C99999 will be your authentication so the sample student01 account and password definition can be eliminated from the code). This is not the preferred method of access because you have to prompt the user for a user name or password, and you may be tempted to hard-code user names and passwords into your application. The connection string is stored in the ConnectionString property of the SqlConnection object, which you will use in the next step. You can also encode many other elements in the connection string by using the SqlConnectionStringBuilder class—the properties shown in this example are a minimal but sufficient set.

10. Add the following bold statement to the try block:

```
try
{
    ...
   dataConnection.Open();
}
```

This statement uses the connection string specified by the ConnectionString property of the dataConnection object to open a connection to the database. If the connection is successful, you can use the dataConnection object to perform database commands and queries. If the connection is unsuccessful, the statement throws a SQLException exception.

Windows Authentication is useful for authenticating users who are all members of a Windows domain (e.g. your accounts are in the Camosun student domain). However, there might be occassions when the user accessing the database does not have Windows account – for example, if you are building an application designed to be accessed by remote users over the internet. In these cases, you can use the User ID and Password parameters like this:

```
string userName = ...;
string password = ...;
// Prompt the user for name and password, and fill these variables

string ConnString = String.Format(
    "User ID={0};Password={1};Initial Catalog=AdventureWorks;" +
   "Data Source=langw7.cs.camosun.bc.ca", userName, password);

myConnection.ConnectionString = connString;
```

Never, never hard-code user names and passwords into your production applications. Anyone who obtains the source code or reverse engineers the executable can see this information.

11. Add the statements shown here to the try block after the dataConnection.Open();

```
try
{
   ...
```

```
Console.Write("Please enter a customer ID number:" );
string customerId = Console.ReadLine();
}
```

These statements prompt the user for a customer ID and read the user's response in the string variable customerId.

12. Add in the statements shown here after the previous code:

```
try
{
  ...
      SqlCommand dataCommand = new SqlCommand();
      dataCommand.Connection = dataConnection;
      dataCommand.CommandType = CommandType.Text;
      dataCommand.CommandText = "SELECT AccountNumber, CustomerType " +
            " FROM Sales.Customer where CustomerID=@CustomerIdParam";
   }
```

The first statement creates a SqlCommand object. Like SqlConnection, this is a specialized version of an ADO.NET class, Command, that has been designed for performing queries against a SQL Server database. An ADO.NET Command object is used to execute a command against a data source. In the case of a relational database, the text of the command is a SQL statement.

The second line of code sets the Connection property of the SqlCommand object to the database connection you opened in the preceding section. The next two statements specify that the SqlCommand object contains the text of a SQL statement (you can also specify the name of a stored procedure or the name of a single table in the database) and populate the CommandText property with a SQL SELECT statement that retrieves information from the Sales.Customer table for all customers that have a specified CustomerID. The text @CustomerIdParam is a placeholder for a SQL parameter (the @ symbol indicates to the data provider that this is a parameter and not the name of a column in the database). The value for the CustomerID will be passed as a SqlParameter object in the next step.

13. Add the following statements shown to the try block, after the code you entered in the previous step:

```
try
{
  ...
    SqlParameter param = new SqlParameter("@CustomerIdParam", SqlDbType.Char, 5);
    param.Value = customerID;
    dataCommand.Parameters.Add(param);
}
```

These statements create a SqlParameter object that can be substituted for the @CustomerIdParam when the SqlCommand object is executed. The parameter is marked as a database Char type (the SQL Server equivalent of a fixed-length string), and the length of this string is specified as 5 characters. The SqlParameter is populated with the string entered by the user in the customerId variable and then added to the Parameter collection of the SqlCommand. When SQL Server runs this command, it will examine the Parameters collection of the command for a parameter named @CustomerIdParam and then substitute the value of this parameter into the text of the SQL statement

14. Add the following statements to the try block, after the code you entered:

```
try
{
  ...
Console.WriteLine("About to find data on customer {0}\n\n", customerId);
SqlDataReader dataReader = dataCommand.ExecuteReader();

while (dataReader.Read())
      {
          // Code to display the current row
      }
}
```

The ExecuteReader method of a SqlCommand object constructs a SqlDataReader object that you can use to fetch the rows identified by the SQL statement. The SqlDataReader class provides the fastest mechanism available (as fast as the network permits) for retrieving data from a SQL Server.

The Read method of the SqlDataReader class fetches the next row from the database. It returns true if another row was retrieved successfully; otherwise, it returns false, usually because there are no more rows. The while loop keeps reading rows from the dataReader variable and finishes when there are no more rows available.

15. Add the statements shown to the body of the while loop you created: (replace the comment)

```
while (dataReader.Read())
{
   string accountNum = dataReader.GetString(0);
   string custType = dataReader.GetString(1);

   Console.WriteLine("Oder: {0}\n accountNum: {1} - custType {2}\n",
                        customerID, accountNum, custType);
   string resp = Console.ReadLine();

 }
```

This block of code shows how you read the data from the database by using a SqlDataReader object. A SqlDataReader object contains the most recent row retrieved from the database. There are GetXXX methods to extract the information from each column in the row—e.g. getDateTime, getInt32. The GetXXX methods take a parameter indicating which column to read: 0 is the first column, 1 is the second column, and so on. SqlDataReader fetches rows one at a time and does not retain any locks on a row after it has been retrieved. It is vital for improving concurrency in database applications. Sometimes the SqlDataReader class is referred to as the "firehose cursor" because it pours data out as quickly as possible.

16. When you have finished using a database, it is good practice to close your connection and release any resources you are using. After the while loop:

```
dataReader.Close();
```

This statement closes the SqlDataReader object. You should always close a SqlDataReader object when you have finished with it because you will not be able to use the current SqlConnection object to run any more commands until you do.

17. After the catch block, add the finally block:

```
finally
```

```
{
    dataConnection.Close();
}
```

Putting this statement in a finally block guarantees that the SqlConnection will be closed, even if an exception occurs.

An alternative approach to using a finally block is to wrap the code that creates the SqlDataConnection object in a *using* statement:

```
using (SqlConnection dataConnection = new SqlConnection())
{
    try
        {
        SqlConnectionStringBuilder builder = ...
      }
    catch (SqlException e)
     {
     ....
    }
}
```

18. Run the application and enter a few customer numbers to test (#55 shown).

```
Please enter a customer ID : 55
About to find orders for customer 55

Customer: 55
 accountNum: AW00000055 - custType S

-
```

19. Modify your C# code so that the following data from the Sales.vSalesPerson view is shown:
    - FirstName
    - LastName
    - JobTitle
    - City
    - StateProvinceName
    - SalesQuota
    - SalesYTD

    when you enter a SalesPersonID (e.g. 277). Valid SalesPersonID column values you can find on the Server Explorer (expand Data Connections, then expand langw7.AdventureWorks.dbo, then Views, then Sales.vSalesPerson). Console output is sufficient.