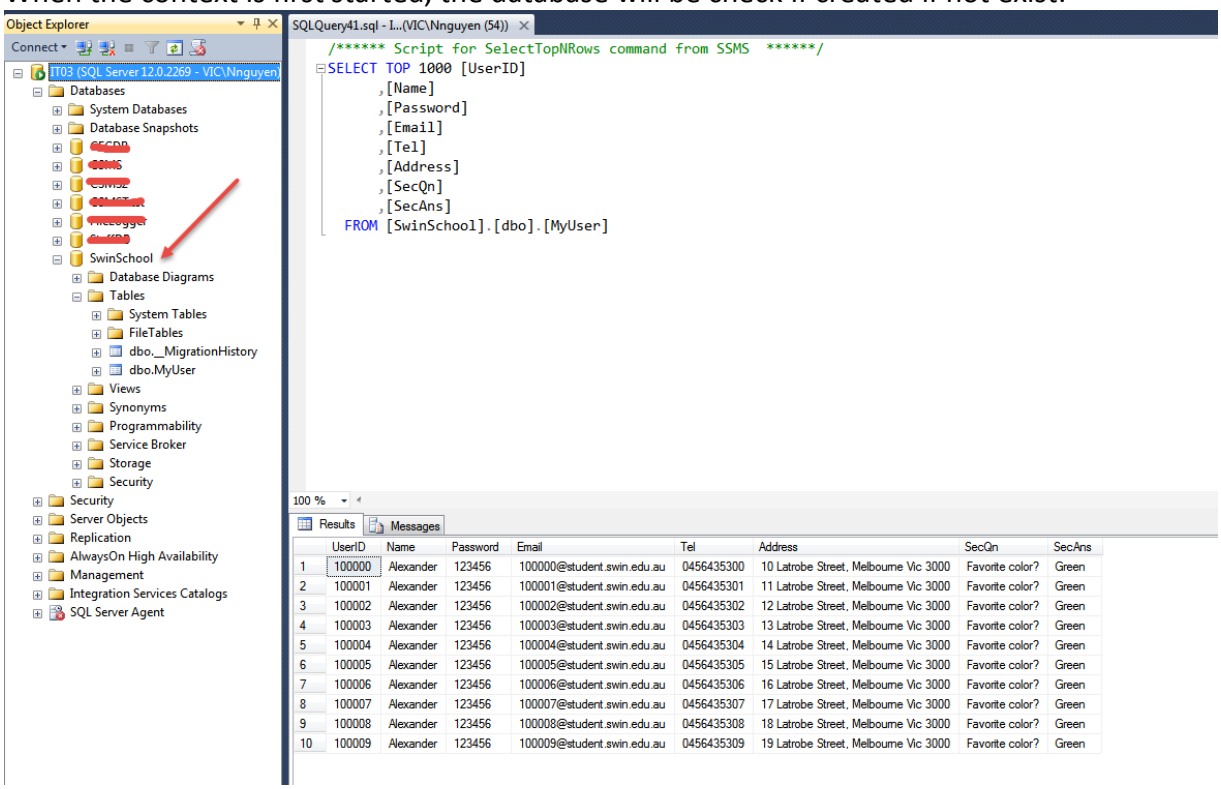
Pass Task 1.2 Database Connectivity (C#)

Task 1. Create database table MyUser and populate sample data:
This is done by using the EntityFramework CodeFirst strategy:

```
public class SchoolInitializer : DropCreateDatabaseIfModelChanges<SchoolContext>
{
    protected override void Seed(SchoolContext context)
    {
        var sampleUserList = new List<MyUser>();
        for (int i = 0; i < 10; i++)
        {
            sampleUserList.Add(
                new MyUser
                {
                    UserID = "10000"+i,
                    Name = "Alexander",
                    Email = "10000"+i+"@student.swin.edu.au",
                    Address = "1"+i+" Latrobe Street, Melbourne Vic 3000",
                    Tel = "04564353"+i.ToString("00"),
                    Password = "123456",
                    SecQn = "Favorite color?",
                    SecAns = "Green"
                });
        }

        sampleUserList.ForEach(s => context.MyUsers.Add(s));
        context.SaveChanges();
    }
}
```

When the context is first started, the database will be check if created if not exist:



Task 2. Data Access Object using ADO:

First, I define an interface as a structure of the MyUserDao:

```
    public interface IMyUserDao
    {
        List<MyUser> GetAll();
        MyUser GetById(string userId);
        int Create(MyUser myUser);
        int Update(MyUser myUser);
        int Delete(MyUser myUser);
    }
```

The ADO.NET implementation of this interface will be as follow:

```
public class MyUserAdoDao : IMyUserDao
    {
        DaoConnection _daoConn;
        public MyUserAdoDao(string connectionString)
        {
            _daoConn = new DaoConnection(connectionString);
        }
        public int Create(MyUser myUser)
        {
            string query = @"INSERT INTO MyUser(UserID, Name, Password, Email, Tel,
Address, SecQn, SecAns)
                VALUES(@UserID, @Name, @Password, @Email, @Tel, @Address, @SecQn,
@SecAns)";
            SqlParameter[] parameters = new SqlParameter[]
            {
                new SqlParameter("@UserID", myUser.UserID),
                new SqlParameter("@Name", myUser.Name),
                new SqlParameter("@Password", myUser.Password),
                new SqlParameter("@Email", myUser.Email),
                new SqlParameter("@Tel", myUser.Tel),
                new SqlParameter("@Address", myUser.Address),
                new SqlParameter("@SecQn", myUser.SecQn),
                new SqlParameter("@SecAns", myUser.SecAns)
            };
            return _daoConn.ExecuteNonQuery(query, parameters);
        }

        public int Update(MyUser myUser)
        {
            string query = @"UPDATE MyUser SET Name=@Name
                                , Password=@Password
                                , Email=@Email
                                , Tel=@Tel
                                , Address=@Address
                                , SecQn=@SecQn
                                , SecAns=@SecAns
                            WHERE UserID=@UserID";
            SqlParameter[] parameters = new SqlParameter[]
            {
                new SqlParameter("@UserID", myUser.UserID),
                new SqlParameter("@Name", myUser.Name),
                new SqlParameter("@Password", myUser.Password),
                new SqlParameter("@Email", myUser.Email),
                new SqlParameter("@Tel", myUser.Tel),
                new SqlParameter("@Address", myUser.Address),
                new SqlParameter("@SecQn", myUser.SecQn),
                new SqlParameter("@SecAns", myUser.SecAns)
            };
            return _daoConn.ExecuteNonQuery(query, parameters);
```

```
        }

        public int Delete(MyUser myUser)
        {
            string query = @"DELETE FROM MyUser
                             WHERE UserID=@UserID";
            SqlParameter[] parameters = new SqlParameter[]
            {
                new SqlParameter("@UserID", myUser.UserID)
            };
            return _daoConn.ExecuteNonQuery(query, parameters);
        }

        public List<MyUser> GetAll()
        {
            var query = "SELECT * FROM MyUser";
            var resultTable = _daoConn.ExecuteForTableResult(query);
            var resultList = resultTable.ToObjects<MyUser>();
            return resultList;
        }

        public MyUser GetById(string userId)
        {
            var query = "SELECT * FROM MyUser WHERE UserID = @UserID";
            var parameters = new SqlParameter[]
            {
                new SqlParameter("@UserID", userId)
            };
            var resultTable = _daoConn.ExecuteForTableResult(query, parameters);
            if (resultTable.Rows.Count > 0)
            {
                return resultTable.ToObjects<MyUser>()[0];
            }
            else return null;
        }
    }
```

In this implementation, I use a DaoConnection to encapsulate the interaction to and from the database. Basically, it contains a SqlConnection object which will be created in the constructor by the connection string passed in. Some of the interesting functions of this object are:

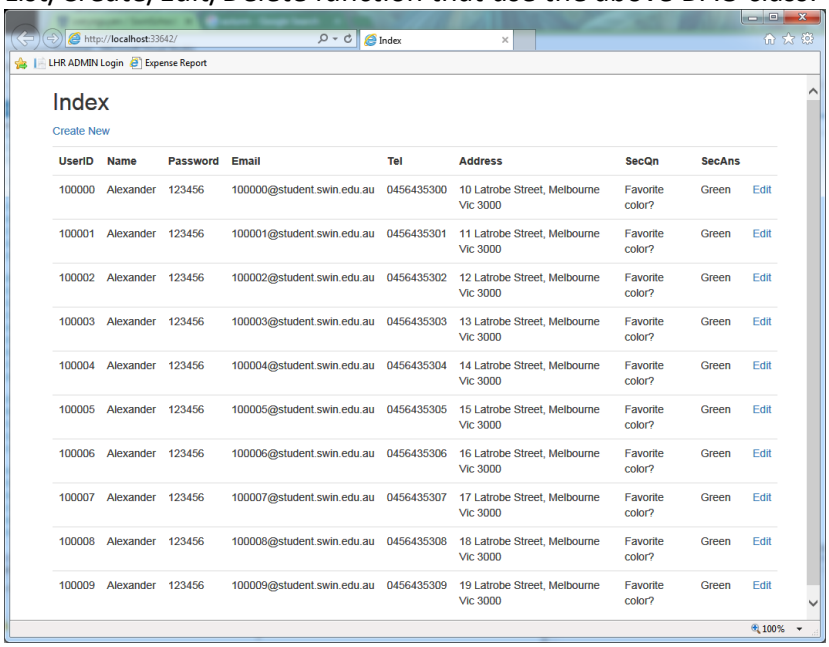
```
public DataTable ExecuteForTableResult(string statement, SqlParameter[] parameters)
        {
            try
            {
                OpenConnection();
                DataTable tblResult = new DataTable();
                SqlCommand cmd = _conn.CreateCommand();
                cmd.CommandText = statement;
                cmd.Parameters.AddRange(parameters);

                SqlDataAdapter adapt = new SqlDataAdapter(cmd);
                adapt.Fill(tblResult);
                return tblResult;
            }
            finally
            {
                CloseConnection();
            }
        }
```

```
public int ExecuteNonQuery(string statement, SqlParameter[] parameters)
        {
            try
            {
                OpenConnection();
                var cmd = _conn.CreateCommand();
                cmd.CommandText = statement;
                cmd.Parameters.AddRange(parameters);
                return cmd.ExecuteNonQuery();
            }
            finally
            {
                CloseConnection();
            }
        }
```

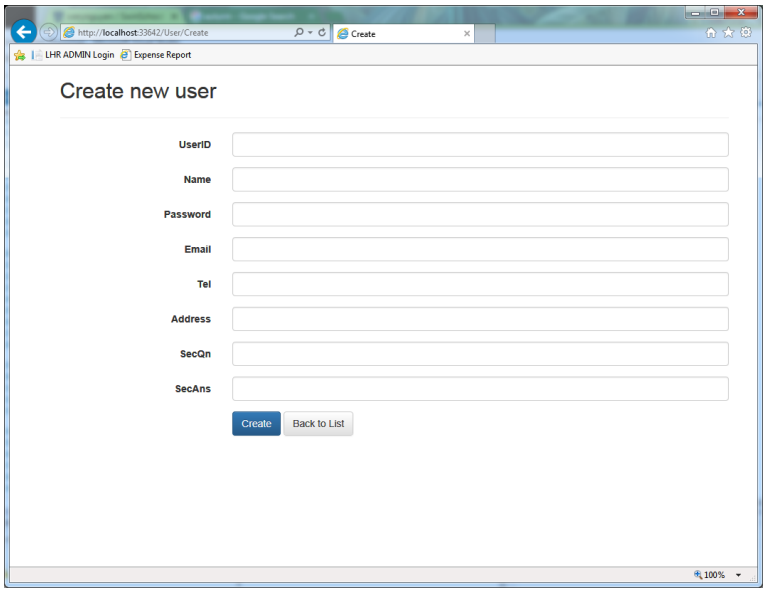
```
public T ExecuteScalar<T>(string statement, SqlParameter[] parameters)
        {
            try
            {
                OpenConnection();
                var cmd = _conn.CreateCommand();
                cmd.CommandText = statement;
                cmd.Parameters.AddRange(parameters);
                return (T)Convert.ChangeType(cmd.ExecuteScalar(), typeof(T));
            }
            finally
            {
                CloseConnection();
            }
        }
```

Task 3. Develop a test program using this DAO:

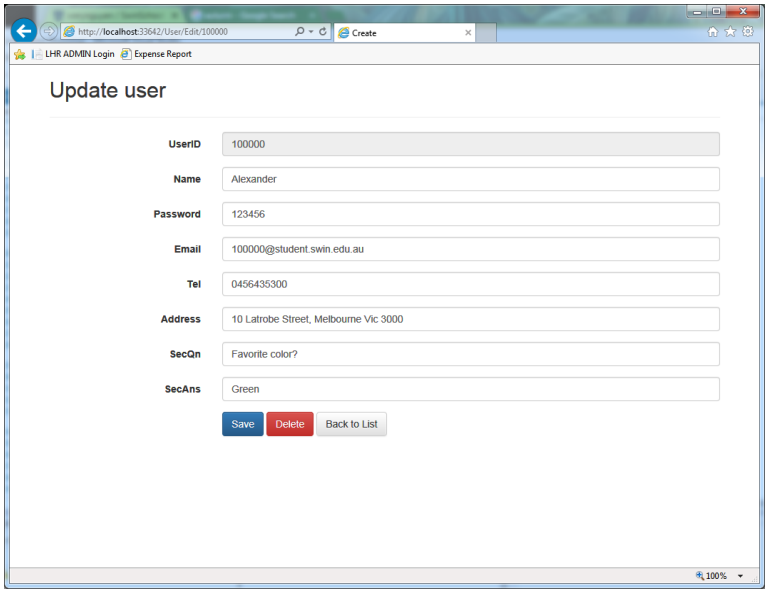
I developed a web application using ASP.NET MVC which have a basic List/Create/Edit/Delete function that use the above DAO class:



User Listing page.



Create new user page



Edit existing user page

Task 4. Test cases:

The following test cases have been taken out to make sure that the DAO works as expected:

Test Case Name	Step	Expected Output	Result
Get all users	View the page /User	A list of 10 users is displayed on the screen	Pass
Get one user	Click to the Edit link of one user in the list	Display the detailed information of that user in a form	Pass
Create an user	- Click to the Create New link. - Enter new user information - Click Create	The new user is added to the list. The list is now showing 11 users	Pass

Update existing user	- Click to the Edit link of one user in the list - Change one or more detail of the user using the form - Click Save	The updated information of that user should be displayed on the list	Pass
Delete existing user	- Click to the Edit link of one user in the list - Click Delete	The user is removed from the list	Pass

Full Code Reference:
For the full code reference, please check out the source tag **PT12** in this git repository: https://bitbucket.org/werynguyen/swinschool/src/19e22f56bf0836ed28ef3267cbb9e3e23f94be34/?at=PT12