TECNICATURA UNIVERSITARIA EN PROGRAMACIÓN UTN-FRC

TUP

UTN

Facultad Regional Córdoba

TECNICATURA UNIVERSITARIA EN PROGRAMACIÓN

PROGRAMACIÓN III

Unidad Temática 4: Acceso a Datos con .Net

Material de Estudio

2do Año - 3er Cuatrimestre

2020

V.0.1

Índice

Introducción 2
ADO.net (Connection, Command, DataReader) 2
SqlConnection 2
DbCommand 3
Métodos principales del objeto Command 5
ExecuteNonQuery (Inserts, Delete, Update) 5
ExecuteScalar (Insert, Delete, Update) 6
ExecuteReader (Select) 7
Seguridad (SQL Injection). 8
ABM simple 9
ABM con tabla asociada 23
Bibliografía 40

Introducción

En esta unidad vamos a ver cómo es el acceso a datos en C# a través de la librería ADO.net. Vamos a analizar cómo son las instrucciones para interactuar con una base de datos.

Además veremos la importancia de prevenir los ataques de sql injection.

Por último analizaremos dos ejemplos importantes de aplicaciones que nos ayudarán a entender cómo es la interacción con la BD

Con esta unidad concluimos este curso de programación web.

ADO.net (Connection, Command, DataReader)

Según Microsoft .NET docs (2017) “Ado .Net Es un conjunto de clases que exponen el acceso a datos para el framework .NET. (...)”.

Para conectar a Microsoft sql server, se deberá usar el objeto SqlConnection del proveedor de datos del framework .Net para Sql Server. Para conectar a una fuente OLE DB se deberá utilizar OleDbConnection. Para cada uno de los tipos de base de datos se deberá utilizar un tipo distinto de conector.

Nosotros utilizaremos SqlConnection con SQL server.

SqlConnection

Hay que recordar cerrar las conexiones una vez abiertas. Es recomendable hacer uso de la palabra clave using con recursos para que automáticamente se cierre la conexión al terminar el bloque.

```
// Assumes connectionString is a valid connection string.
using (SqlConnection connection = new SqlConnection(connectionString))
{
    connection.Open();
    // Do work here.
}
```

Otra alternativa es la de cerrar a mano la conexión con connection.Close()

DbCommand

Después de haber establecido la conexión con la fuente de datos, se pueden ejecutar comandos y devolver los resultados de la base de datos usando el objeto DbCommand

El objeto DbCommand, Expone métodos para ejecutar los comandos basado en el tipo de comando y en el valor de retorno deseado.

Por ejemplo, Si queremos retornar un objeto DataReader hay que utilizar el comando ExecuteReader.

Por otro lado si queremos devolver un valor escalar único debemos utilizar ExecuteScalar.

Finalmente si queremos ejecutar un comando que no devuelve ninguna fila debemos utilizar ExecuteNonQuery

Por ejemplo

```
static void GetSalesByCategory(string connectionString,
    string categoryName)
{
    using (SqlConnection connection = new
SqlConnection(connectionString))
    {
        // Create the command and set its properties.
        SqlCommand command = new SqlCommand();
        command.Connection = connection;
        command.CommandText = "SalesByCategory";
        command.CommandType =
CommandType.StoredProcedure;

        // Add the input parameter and set its
properties.
        SqlParameter parameter = new SqlParameter();
        parameter.ParameterName = "@CategoryName";
        parameter.SqlDbType = SqlDbType.NVarChar;
        parameter.Direction = ParameterDirection.Input;
```

```
                parameter.Value = categoryName;

                // Add the parameter to the Parameters
collection.
                command.Parameters.Add(parameter);

                // Open the connection and execute the reader.
                connection.Open();
                using (SqlDataReader reader =
command.ExecuteReader())
                {
                    if (reader.HasRows)
                    {
                        while (reader.Read())
                        {
                            Console.WriteLine("{0}: {1:C}",
reader[0], reader[1]);
                        }
                    }
                    else
                    {
                        Console.WriteLine("No rows found.");
                    }
                    reader.Close();
                }
            }
        }
```

Si bien en el ejemplo podemos especificar el tipo del parámetro hay una manera más sencilla en donde no se declara el tipo. Para ver la otra manera, remitirse al ejemplo ABM simple al final de la unidad.

Sin embargo, los tipos de Parameter son:

Tipo de .NET Framework	SqlDbType
Boolean	Bit
DateTime	DateTime
Double	Float
Single	Real
Int32	Int
String	NVarChar

Tabla 1: Elaboración propia

Métodos principales del objeto Command

ExecuteNonQuery (Inserts, Delete, Update)

SqlCommand.ExecuteNonQuery ejecuta una sentencia de Transact-SQL (para nosotros de sql) contra la conexión y devuelve el número de filas afectadas.

```
private static void CreateCommand(string queryString,
    string connectionString)
{
    using (SqlConnection connection = new SqlConnection(
            connectionString))
    {
        SqlCommand command = new SqlCommand(queryString,
connection);
        command.Connection.Open();
        command.ExecuteNonQuery();
    }
}
```

Sirve para ejecutar consultas que modifican la base de datos por ejemplo para insertar, borrar o modificar.

ExecuteScalar (Insert, Delete, Update)

SqlCommand.ExecuteScalar ejecuta una consulta y devuelve la primera columna de la primera fila como resultado de la consulta. Las demás columnas y filas son ignoradas.

Funciona igual que ExecuteNonQuery.

```
static public int AddProductCategory(string newName,
string connString)
    {
        Int32 newProdID = 0;
        string sql =
            "INSERT INTO Production.ProductCategory (Name)
VALUES (@Name); "
            + "SELECT CAST(scope_identity() AS int)";
        using (SqlConnection conn = new
SqlConnection(connString))
        {
            SqlCommand cmd = new SqlCommand(sql, conn);
            cmd.Parameters.Add("@Name", SqlDbType.VarChar);
            cmd.Parameters["@name"].Value = newName;
            try
            {
                conn.Open();
                newProdID = (Int32)cmd.ExecuteScalar();
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
        }
        return (int)newProdID;
    }
```

ExecuteReader (Select)

Para obtener datos usando un DataReader hay que crear una instancia del objeto Command y después crear un DataReader llamando a Command.ExecuteReader.

El DataReader es una buena elección cuando está por traer de la base de datos una gran cantidad de datos porque los datos no son almacenados en memoria caché.

```
reader = command.ExecuteReader();
```

El método DataReader.Read() obtiene una fila de los resultados de la consulta.

```
static void HasRows(SqlConnection connection)
{
    using (connection)
    {
        SqlCommand command = new SqlCommand(
          "SELECT CategoryID, CategoryName FROM
Categories;",
          connection);
        connection.Open();

        SqlDataReader reader = command.ExecuteReader();

        if (reader.HasRows)
        {
            while (reader.Read())
            {
                Console.WriteLine("{0}\t{1}",
reader.GetInt32(0),
                    reader.GetString(1));
            }
        }
        else
        {
```

```
                Console.WriteLine("No rows found.");
            }
            reader.Close();
        }
    }
```

Siempre hay que recordar cerrar el recurso con DataReader.Close().

Seguridad (SQL Injection)

Según OWASP, “un ataque SQL Injection (inyección de SQL) consiste en la inserción de una consulta sql a través de un campo de texto desde el cliente de la aplicación. Un ataque exitoso puede hacer lectura y datos sensibles desde la base de datos, modificar los datos de la base de datos, ejecutar operaciones de administración en la BD y en algunos casos hasta puede ejecutar comandos del sistema operativo (...)”

Para evitar que esto suceda en nuestras aplicaciones siempre debemos ejecutar consultas con parámetros. Nunca debemos ejecutar una consulta concatenando variables.

Al hacer esto, el motor de base de datos asegura que no podrán insertarse cadenas extrañas fuera de lo esperado.

Es más, no hace falta ser un hacker para efectuar este tipo de ataques. nosotros mismos podemos probarlo (con responsabilidad), en nuestro código.

Si armamos la consulta sql concatenando dos variables podríamos concatenar una variable que tenga un contenido parecido al siguiente: “; drop table…”. El punto y coma inicial hace que se trunque la consulta y empiece otra con lo que queramos poner. Por ejemplo podemos borrar una tabla de la base de datos, entre otras cosas. De ahí deviene la importancia de resguardar nuestros datos.

Para insertar parámetro en la consulta, debemos hacer lo siguiente:

```
//Primero creamos el comando
 SqlCommand comm = conn.CreateCommand();
//Luego le asignamos la consulta. Notar que los
parámetros se ingresan con “@”
                comm.CommandText = "insert into
Personas(nombre, apellido, edad) values (@Nombre, @Apellido,
@Edad)";
```

```
      //Ya que tenemos los lugares para los parámetros, los
reemplazamos por

      //los valores correspondientes.

                  comm.Parameters.Add(new
SqlParameter("@Nombre",nueva.Nombre));

                  comm.Parameters.Add(new
SqlParameter("@Apellido",nueva.Apellido));

                  comm.Parameters.Add(new
SqlParameter("@Edad",nueva.Edad));
```

De esta forma prevenimos el ataque. Sin embargo podemos hacer más segura nuestra aplicación con lo siguiente.

Otra forma de prevenir es la de tener dos usuarios para la base de datos, uno con permisos de lectura y escritura y otro con permisos de lectura solamente. Así, podremos restringir el acceso dependiendo del tipo de consulta que debamos realizar.

Lo del párrafo anterior se hace desde el motor de base de datos y no desde el framework MVC ASP.Net

ABM simple

Se denomina ABM (alta baja modificación) a la funcionalidad de una página o ventana que permite el alta, la baja y modificación de alguna entidad del modelo.

Primero vamos a ver un ejemplo de un ABM en donde sólo haremos el alta de una sola tabla de la base de datos.

Este ejemplo tiene como entidad de modelo a una persona con algunos atributos, nombre, apellido y edad.

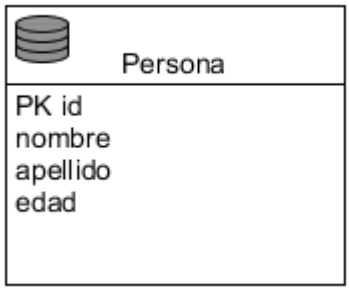

Imagen 1: autoría propia

Empezaremos por el controller:

- **Controllers>PersonasController.cs**

```
using ABMPersonas.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace ABMPersonas.Controllers
{
    public class PersonasController : Controller
    {
        // GET: Personas
        public ActionResult Index()
        {
            return View();
        }

        public ActionResult Agregar()
        {
            Persona p = new Persona();
            return View(p);
        }

        [HttpPost]
        public ActionResult Agregar(Persona p)
        {
            GestorPersonas g = new GestorPersonas();
            g.AgregarPersona(p);
```

```
            List<Persona> lista = g.ObtenerPersonas();
            return View("Listar", "", lista);
        }

        public ActionResult Eliminar(int id)
        {
            GestorPersonas g = new GestorPersonas();
            g.Eliminar(id);

            List<Persona> lista = g.ObtenerPersonas();
            return View("Listar", "", lista);
        }

        public ActionResult Listar()
        {
            GestorPersonas gestorPersonas = new
GestorPersonas();
            List<Persona> lista =
gestorPersonas.ObtenerPersonas();

            return View("Listar","",lista);
        }

        public ActionResult Editar(int id)
        {
            GestorPersonas g = new GestorPersonas();
            Persona p = g.ObtenerPersona(id);

            return View(p);
        }
```

```
        [HttpPost]
        public ActionResult Editar(Persona p)
        {
            GestorPersonas g = new GestorPersonas();
            g.Modificar(p);

            List<Persona> lista = g.ObtenerPersonas();
            return View("Listar", "", lista);
        }
    }
}
```

Primero vemos la acción Index(). Esta función es llamada al ingresar por la URL al controlador con (/personas). Siempre en todos los controladores tendremos esta acción que en invocada por defecto. El controlador llama entonces a la vista con el mismo nombre, o sea, a index.cshtml que está en la carpeta Personas mediante la sentencia return View().

Nosotros no tuvimos que configurar nada para que sepa que vista llamar. Es por eso que es fundamental seguir la convención de MVC.Net respecto a los nombres. Es decir, el controlador deberá terminar con el sufijo Controller, se creará una vista en la carpeta dentro de Views que tiene el nombre del controlador (sin el sufijo Controller).

Por otro lado, la función Agregar() está sobrecargada, es decir, tiene en una versión parámetros y en otra no.

La función sin parámetro, o acción, es la que se muestra al llamar en la URL a agregar (/agregar). Al ingresar la dirección en la URL, el enrutador llama automáticamente a esta función/acción por el método HTTP de get(). O sea, esta función responde a Get.

Al ser invocada, se crea en memoria el objeto persona y es pasado a la vista. La última línea invoca a esa vista.

La acción Agregar() con parámetro es invocada mediante la petición HTTP Post. Para diferenciar cual es llamada por get o por post, la función se “adorna” con la propiedad de [HttpPost]. O sea, para cambiar la invocación de get a post agregamos dicho “adorno”.

La función eliminar tiene un parámetro id que es un entero. Como vimos al no estar adornada entra por Get. Pero cómo hacemos para pasarle ese parámetro? Pues bien, lo hacemos mediante la URL según la convención dispuesta en el archivo **App_Start>RouteConfig.cs.**

La línea que nos interesa de ese archivo es la siguiente:

```
url: "{controller}/{action}/{id}",
```

Como podemos apreciar, allí se especifica el formato de la URL. Primero va el controller, luego la acción y por último un número que identifica a algún recurso, el id. Ese id es el que se pasa como parámetro a la función Eliminar(int id).

O sea, para eliminar la persona con id=42 pasaremos la siguiente URL /personas/eliminar/42.

Observación:

la línea,

```
defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
```

le indica a la aplicación web que comience por el controlador, en este caso, “Home” y por la acción “Index” al ejecutar la app, o sea, al navegar a la URL <nuestro_dominio>/. Si queremos que comience por el index de PersonasController, basta con cambiar a “Home” por “Personas”.

Finalmente, las otras acciones no presentan ninguna particularidad que no hayamos mencionado.

- **Models>Persona.cs**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace ABMPersonas.Models
{
    public class Persona
    {
        public Persona()
```

```
        {

        }

        public Persona(int id, string nombre, string
apellido, int edad)
        {
            Id = id;
            Nombre = nombre;
            Apellido = apellido;
            Edad = edad;
        }

        private int id;

        public int Id {
            get { return id; }
            set { id = value; }
        }

        private string nombre;

        public string Nombre {
            get { return nombre; }
            set { nombre = value; }
        }

        private string apellido;

        public string Apellido {
            get { return apellido; }
```

```
            set { apellido = value; }
        }

        private int edad;

        public int Edad {
            get { return edad; }
            set { edad = value; }
        }

        public override string ToString()
        {
            return id + ": " + nombre + " " + apellido +
" - " + edad;
        }
    }
}
```

Persona es un objeto común de C# con sus getters y setter. A este tipo de clases se la denomina Plain Old CLR Objects (POCO) que significa un objeto común del lenguaje que no depende del framework MVC.Net.

- **Models>GestorPersonas.cs**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Data;
using System.Data.SqlClient;

namespace ABMPersonas.Models
{
```

```
public class GestorPersonas
{

    public void AgregarPersona(Persona nueva) {
        // 1: Crear la conexion
        SqlConnection conn = new
SqlConnection(@"Server=172.16.140.13;Database=Personas;User
Id=alumno1w1;Password=alumno1w1;");
        conn.Open();
        // 2: Crear un objeto command y
        // establecer la sentencia SQL
        SqlCommand comm = conn.CreateCommand();
        comm.CommandText = "insert into
Personas(nombre, apellido, edad) values (@Nombre, @Apellido,
@Edad)";
        // 2: Asignar los parámetros de la sentencia
        comm.Parameters.Add(new
SqlParameter("@Nombre",nueva.Nombre));
        comm.Parameters.Add(new
SqlParameter("@Apellido",nueva.Apellido));
        comm.Parameters.Add(new
SqlParameter("@Edad",nueva.Edad));
        // 3: Ejecutar la sentencia
        comm.ExecuteNonQuery();
        // 4: Cerrar todo
        conn.Close();
    }

    public List<Persona> ObtenerPersonas() {
        List<Persona> lista = new List<Persona>();
        // 1: Crear la conexion
```

```
                SqlConnection conn = new
SqlConnection(@"Server=172.16.140.13;Database=Personas;User
Id=alumno1w1;Password=alumno1w1;");
                conn.Open();
                // 2: Crear un objeto command y
                // establecer la sentencia SQL
                SqlCommand comm = conn.CreateCommand();
                comm.CommandText = "select * from Personas";
                // 3: Ejecutar la sentencia
                SqlDataReader dr = comm.ExecuteReader();
                // 4: Recorrer el conjunto de filas
                while(dr.Read())
                {
                    // Da una vuelta por cada fila
                    int id = dr.GetInt32(0);
                    string nombre = dr.GetString(1);
                    string apellido = dr.GetString(2);
                    int edad = dr.GetInt32(3);

                    Persona p = new Persona(id, nombre,
apellido, edad);
                    lista.Add(p);
                }
                // 5: Cerrar todo
                dr.Close();
                conn.Close();
                // 6: Retornar la lista
                return lista;
            }

            public void Eliminar(int id)
            {
```

```
                SqlConnection conn = new
SqlConnection(@"Server=172.16.140.13;Database=Personas;User
Id=alumno1w1;Password=alumno1w1;");
                conn.Open();

                SqlCommand comm = new SqlCommand("DELETE from
Personas WHERE IdPersona=@id;", conn);
                comm.Parameters.Add(new SqlParameter("@id",
id));

                comm.ExecuteNonQuery();
                conn.Close();
            }

            public Persona ObtenerPersona(int id)
            {
                Persona p = null;

                SqlConnection conn = new
SqlConnection(@"Server=172.16.140.13;Database=Personas;User
Id=alumno1w1;Password=alumno1w1;");
                conn.Open();

                SqlCommand comm = conn.CreateCommand();
                comm.CommandText = "select * from Personas
where IdPersona=@id;";
                comm.Parameters.Add(new SqlParameter("@id",
id));

                SqlDataReader dr = comm.ExecuteReader();
                if (dr.Read())
                {
                    int idp = dr.GetInt32(0);
```

```
                    string nombre = dr.GetString(1);
                    string apellido = dr.GetString(2);
                    int edad = dr.GetInt32(3);

                    p = new Persona(idp, nombre, apellido,
edad);
                }
                // 5: Cerrar todo
                dr.Close();
                conn.Close();
                // 6: Retornar la lista
                return p;
            }

            public void Modificar(Persona p)
            {
                SqlConnection conn = new
SqlConnection(@"Server=172.16.140.13;Database=Personas;User
Id=alumno1w1;Password=alumno1w1;");
                conn.Open();

                SqlCommand comm = new SqlCommand("update
Personas set nombre=@nombre, apellido=@apellido, edad=@edad
WHERE IdPersona=@id;", conn);
                comm.Parameters.Add(new
SqlParameter("@nombre", p.Nombre));
                comm.Parameters.Add(new
SqlParameter("@apellido", p.Apellido));
                comm.Parameters.Add(new SqlParameter("@edad",
p.Edad));
                comm.Parameters.Add(new SqlParameter("@id",
p.Id));
```

```
                comm.ExecuteNonQuery();
                conn.Close();
            }
        }
    }
```

El gestor persona es la clase encargada de interactuar con la Base de Datos. Es una buena práctica generar un archivo especializado a esta función. Podemos hacer un gestor por cada clase del modelo o, si el sistema lo amerita, poner toda la gestión de la BD en una única clase.

En el código podemos apreciar la explicación de cada conjunto de líneas mediante los comentarios escritos.

Siempre realizamos la misma secuencia:

1. Abrimos la conexión con la cadena de conexión.
2. Ejercitamos la BD con una consulta
3. Si devuelve resultados, creamos los objetos correspondientes
4. Cerramos los recursos abiertos.

El resto de las funciones no presenta ninguna dificultad.

- **Views>Personas>Listar.cshtml**

```
@using ABMPersonas.Models;
@model List<Persona>

@{
    ViewBag.Title = "Listar";
}

<h2>Listado de personas</h2>

<table>
@foreach (Persona p in Model)
{
<tr>
```

```
<td>@p.Nombre</td>
<td>@p.Apellido</td>
<td>@p.Edad</td>

<td><a href="/Personas/Editar/@p.Id">Editar</a></td>
<td><a href="/Personas/Eliminar/@p.Id">Eliminar</a></td>
</tr>
}
</table>

<a href="/Personas/Agregar">Agregar persona</a>
```

Con respecto a las vistas, nunca hay que olvidar la línea en donde se declara @model

Esa línea especifica el parámetro que recibirá la vista y será el recurso a consultar. Para el caso de la vista anterior, se recibirá una lista de personas (List<Persona>).

Entonces podremos iterar para mostrar cada uno de los ítems de la lista mediante la directiva @foreach. No olvidarse de las arrobas “@”.

- **Views>Personas>Agregar.cshtml**

```
@model ABMPersonas.Models.Persona

@{
    ViewBag.Title = "Agregar";
}

<h2>Agregar</h2>

@using (Html.BeginForm())
{
```

```
<p>Nombre:</p> <br>
@Html.TextBoxFor(p => p.Nombre)
<p>Apellido:</p> <br>
@Html.TextBoxFor(p => p.Apellido)
<p>Edad:</p> <br>
@Html.TextBoxFor(p => p.Edad)

<input type="submit" value="Agregar">
}
```

Lo importante de esta vista es el parámetro que será un objeto persona vacío. Lo hemos pasado desde el controlador con un new Persona().

ViewBag.title es en donde establecemos una variable “global” para especificar el nombre de la página.

@using (Html.BeginForm()) se transformará en un formulario de HTML con el method seteado y action también. No nos tenemos que ocupar de ello. Al enviarse el formulario con un Submit, automáticamente irá por post a la acción correspondiente del controlador correspondiente.

@Html.TextBoxFor(p => p.Nombre) se convertirá en un HTML input y se seteará automáticamente el objeto p con el nombre.

Las demás líneas son iguales a lo anterior.

- **Views>Personas>Editar.cshtml**

```
@model ABMPersonas.Models.Persona

@{
    ViewBag.Title = "Editar";
}

<h2>Editar</h2>

@using (Html.BeginForm())
{
```

```
<p>Nombre:</p> <br>
@Html.TextBoxFor(p => p.Nombre)
<p>Apellido:</p> <br>
@Html.TextBoxFor(p => p.Apellido)
<p>Edad:</p> <br>
@Html.TextBoxFor(p => p.Edad)

<input type="submit" value="Modificar">
}
```

La particularidad de @Html.TextBoxFor(p => p.Nombre) es que automáticamente nos llena el contenido del input con el valor del objeto correspondiente enviado por el @model.

ABM con tabla asociada

El siguiente ejemplo es similar al anterior con la salvedad de que ahora tenemos una relación uno-a-muchos que une la tabla Persona con EstadoCivil.

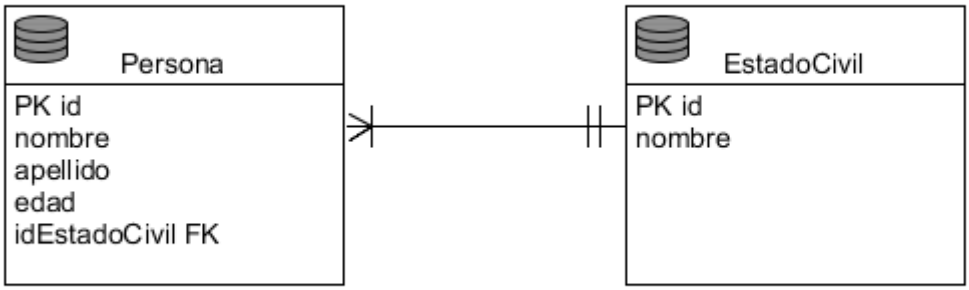

Imagen 2: autoría propia

- **Controllers>PersonasController.cs**

```
using ABMPersonas.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
```

```
using System.Web.Mvc;

namespace ABMPersonas.Controllers
{
    public class PersonasController : Controller
    {
        public ActionResult Agregar()
        {
            /*Persona p = new Persona(0, "Juan", "Perez",
30);
            GestorPersonas gestorPersonas = new
GestorPersonas();
            gestorPersonas.AgregarPersona(p);
            */
            GestorPersonas g = new GestorPersonas();

            Persona p = new Persona();
            PersonaViewModel pvm = new
PersonaViewModel();
            pvm.persona = p;
            pvm.listaec = g.ObtenerEstadosCiviles();

            return View(pvm);
        }

        [HttpPost]
        public ActionResult Agregar(PersonaViewModel pvm)
        {
            GestorPersonas g = new GestorPersonas();
            g.AgregarPersona(pvm.persona);
```

```
                List<PersonaConEstadoCivil> lista =
g.ObtenerPersonas();
                return View("Listar", "", lista);
            }

            public ActionResult Eliminar(int id)
            {
                GestorPersonas g = new GestorPersonas();
                g.Eliminar(id);

                List<PersonaConEstadoCivil> lista =
g.ObtenerPersonas();
                return View("Listar", "", lista);
            }

            public ActionResult Listar()
            {
                GestorPersonas gestorPersonas = new
GestorPersonas();
                List<PersonaConEstadoCivil> lista =
gestorPersonas.ObtenerPersonas();

                return View("Listar","",lista);
            }

            public ActionResult Editar(int id)
            {
                GestorPersonas g = new GestorPersonas();
                Persona p = g.ObtenerPersona(id);

                //List<EstadoCivil> le =
g.ObtenerEstadosCiviles();
```

```
                return View(p);
            }

            [HttpPost]
            public ActionResult Editar(Persona p)
            {
                GestorPersonas g = new GestorPersonas();
                g.Modificar(p);

                List<PersonaConEstadoCivil> lista =
g.ObtenerPersonas();
                return View("Listar", "", lista);
            }
        }
    }
```

A diferencia de la sección anterior (ABM simple) tenemos el concepto de ViewModel. Por convención se declara como ViewModel a un objeto común de C# especificamente creado para ser pasado a una vista.

Éstos objetos se encargan de agrupar todo lo que necesita la vista para su funcionamiento. En el caso anterior se pasa por el ViewModel no sólo la persona a agregar (estamos hablando de la acción Agregar()) sino que también se envía la lista de Estados Civiles.

Hacemos esto porque las vistas reciben un único parámetro. Es por ello que usamos una clase para “agrupar” a varias otras.

- **Models>Persona.cs**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
```

```
namespace ABMPersonas.Models
{
    public class Persona
    {
        public Persona()
        {

        }

        public Persona(int id, string nombre, string
apellido, int edad)
        {
            Id = id;
            Nombre = nombre;
            Apellido = apellido;
            Edad = edad;
        }

        private int id;

        public int Id {
            get { return id; }
            set { id = value; }
        }

        private string nombre;

        public string Nombre {
            get { return nombre; }
            set { nombre = value; }
```

```
            }

            private string apellido;

            public string Apellido {
                get { return apellido; }
                set { apellido = value; }
            }

            private int edad;

            public int Edad {
                get { return edad; }
                set { edad = value; }
            }

            public int IdEstadoCivil { get => idEstadoCivil;
set => idEstadoCivil = value; }

            private int idEstadoCivil;

            public override string ToString()
            {
                return id + ": " + nombre + " " + apellido +
" - " + edad;
            }
        }
    }
```

- **Models>EstadoCivil.cs**

```
    using System;
```

```
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace ABMPersonas.Models
{
    public class EstadoCivil
    {
        private int id;
        private string nombre;

        public EstadoCivil(int id, string nombre)
        {
            this.id = id;
            this.nombre = nombre;
        }

        public int Id { get => id; set => id = value; }
        public string Nombre { get => nombre; set =>
nombre = value; }
    }
}
```

- **Models>PersonaConEstadoCivil.cs**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace ABMPersonas.Models
{
```

```
        public class PersonaConEstadoCivil
        {
            public PersonaConEstadoCivil(int id, string
nombre, string apellido, int edad, string nombreEstadoCivil)
            {
                this.id = id;
                this.nombre = nombre;
                this.apellido = apellido;
                this.edad = edad;
                this.nombreEstadoCivil = nombreEstadoCivil;
            }

            public int id { get; set; }
            public string nombre { get; set; }
            public string apellido { get; set; }
            public int edad { get; set; }
            public string nombreEstadoCivil { get; set; }
        }
    }
```

La razón de ser de esta clase, es la de armar un objeto parecido a persona pero con un atributo más para el estado civil. Nos servirá para llenar el resultado de algún JOIN de la BD para luego poder listar con ese atributo de más.

- **Models>PersonaViewModel.cs**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace ABMPersonas.Models
{
    public class PersonaViewModel
    {
        public Persona persona { get; set;  }
        public List<EstadoCivil> listaec { get; set; }
    }
}
```

- **Models>GestorPersonas.cs**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Data;
using System.Data.SqlClient;

namespace ABMPersonas.Models
{
    public class GestorPersonas
    {

        public void AgregarPersona(Persona nueva) {
```

```
                // 1: Crear la conexion
                SqlConnection conn = new
SqlConnection(@"Server=172.16.140.13;Database=Personas;User
Id=alumno1w1;Password=alumno1w1;");
                conn.Open();
                // 2: Crear un objeto command y
                // establecer la sentencia SQL
                SqlCommand comm = conn.CreateCommand();
                comm.CommandText = "insert into
Personas(nombre, apellido, edad, estadoCivilId) values
(@Nombre, @Apellido, @Edad, @estadoCivilId)";
                // 2: Asignar los parámetros de la sentencia
                comm.Parameters.Add(new
SqlParameter("@Nombre",nueva.Nombre));
                comm.Parameters.Add(new
SqlParameter("@Apellido",nueva.Apellido));
                comm.Parameters.Add(new SqlParameter("@Edad",
nueva.Edad));
                comm.Parameters.Add(new
SqlParameter("@EstadoCivilId", nueva.IdEstadoCivil));
                // 3: Ejecutar la sentencia
                comm.ExecuteNonQuery();
                // 4: Cerrar todo
                conn.Close();
            }

            public List<PersonaConEstadoCivil>
ObtenerPersonas() {
                List<PersonaConEstadoCivil> lista = new
List<PersonaConEstadoCivil>();
                // 1: Crear la conexion
                SqlConnection conn = new
SqlConnection(@"Server=172.16.140.13;Database=Personas;User
Id=alumno1w1;Password=alumno1w1;");
```

```
                conn.Open();
                // 2: Crear un objeto command y
                // establecer la sentencia SQL
                SqlCommand comm = conn.CreateCommand();
                comm.CommandText = "select * from Personas p
JOIN EstadoCivil e ON p.estadoCivilId = e.id";
                // 3: Ejecutar la sentencia
                SqlDataReader dr = comm.ExecuteReader();
                // 4: Recorrer el conjunto de filas
                while(dr.Read())
                {
                    // Da una vuelta por cada fila
                    int id = dr.GetInt32(0);
                    string nombre = dr.GetString(1);
                    string apellido = dr.GetString(2);
                    int edad = dr.GetInt32(3);
                    string nombreEstadoCivil =
dr.GetString(6);

                    //Persona p = new Persona(id, nombre,
apellido, edad);
                    //lista.Add(p);
                    PersonaConEstadoCivil p = new
PersonaConEstadoCivil(id, nombre, apellido, edad,
nombreEstadoCivil);
                    lista.Add(p);
                }
                // 5: Cerrar todo
                dr.Close();
                conn.Close();
                // 6: Retornar la lista
                return lista;
```

```
        }

        public void Eliminar(int id)
        {
            SqlConnection conn = new
SqlConnection(@"Server=172.16.140.13;Database=Personas;User
Id=alumno1w1;Password=alumno1w1;");
            conn.Open();

            SqlCommand comm = new SqlCommand("DELETE from
Personas WHERE IdPersona=@id;", conn);
            comm.Parameters.Add(new SqlParameter("@id",
id));

            comm.ExecuteNonQuery();
            conn.Close();
        }

        public Persona ObtenerPersona(int id)
        {
            Persona p = null;

            SqlConnection conn = new
SqlConnection(@"Server=172.16.140.13;Database=Personas;User
Id=alumno1w1;Password=alumno1w1;");
            conn.Open();

            SqlCommand comm = conn.CreateCommand();
            comm.CommandText = "select * from Personas
where IdPersona=@id;";
            comm.Parameters.Add(new SqlParameter("@id",
id));
```

```
                SqlDataReader dr = comm.ExecuteReader();
                if (dr.Read())
                {
                    int idp = dr.GetInt32(0);
                    string nombre = dr.GetString(1);
                    string apellido = dr.GetString(2);
                    int edad = dr.GetInt32(3);

                    p = new Persona(idp, nombre, apellido,
edad);
                }
                // 5: Cerrar todo
                dr.Close();
                conn.Close();
                // 6: Retornar la lista
                return p;
            }

            public void Modificar(Persona p)
            {
                SqlConnection conn = new
SqlConnection(@"Server=172.16.140.13;Database=Personas;User
Id=alumno1w1;Password=alumno1w1;");
                conn.Open();

                SqlCommand comm = new SqlCommand("update
Personas set nombre=@nombre, apellido=@apellido, edad=@edad
WHERE IdPersona=@id;", conn);
                comm.Parameters.Add(new
SqlParameter("@nombre", p.Nombre));
                comm.Parameters.Add(new
SqlParameter("@apellido", p.Apellido));
```

```
                comm.Parameters.Add(new SqlParameter("@edad",
p.Edad));
                comm.Parameters.Add(new SqlParameter("@id",
p.Id));

                comm.ExecuteNonQuery();
                conn.Close();
            }

            public List<EstadoCivil> ObtenerEstadosCiviles()
            {
                List<EstadoCivil> lista = new
List<EstadoCivil>();
                // 1: Crear la conexion
                SqlConnection conn = new
SqlConnection(@"Server=172.16.140.13;Database=Personas;User
Id=alumno1w1;Password=alumno1w1;");
                conn.Open();
                // 2: Crear un objeto command y
                // establecer la sentencia SQL
                SqlCommand comm = conn.CreateCommand();
                comm.CommandText = "select * from
EstadoCivil";
                // 3: Ejecutar la sentencia
                SqlDataReader dr = comm.ExecuteReader();
                // 4: Recorrer el conjunto de filas
                while (dr.Read())
                {
                    // Da una vuelta por cada fila
                    int id = dr.GetInt32(0);
                    string nombre = dr.GetString(1);
```

```
                    EstadoCivil ec = new EstadoCivil(id,
nombre);
                    lista.Add(ec);
                }
                // 5: Cerrar todo
                dr.Close();
                conn.Close();
                // 6: Retornar la lista
                return lista;
            }
        }
    }
```

- **Views>Listar.cshtml**

```
@using ABMPersonas.Models;
@model List<PersonaConEstadoCivil>

@{
    ViewBag.Title = "Listar";
}

<h2>Listado de personas</h2>

<table>
@foreach (PersonaConEstadoCivil p in Model)
{
<tr>
<td>@p.nombre</td>
<td>@p.apellido</td>
<td>@p.edad</td>
```

```
<td>@p.nombreEstadoCivil</td>

<td><a href="/Personas/Editar/@p.id">Editar</a></td>
<td><a href="/Personas/Eliminar/@p.id">Eliminar</a></td>
</tr>
}
</table>

<a href="/Personas/Agregar">Agregar persona</a>
```

- **Views>Agregar.cshtml**

```
@model ABMPersonas.Models.PersonaViewModel

@{
    ViewBag.Title = "Agregar";
}

<h2>Agregar</h2>

@using (Html.BeginForm())
{
<p>Nombre:</p> <br>
@Html.TextBoxFor(p => p.persona.Nombre)
<p>Apellido:</p> <br>
@Html.TextBoxFor(p => p.persona.Apellido)
<p>Edad:</p> <br>
@Html.TextBoxFor(p => p.persona.Edad)
<p>Estado Civil</p> <br>
@Html.DropDownListFor(p => p.persona.IdEstadoCivil,
new SelectList(@Model.listaec, "Id", "Nombre")
```

```
)

<input type="submit" value="Agregar">
}
```

- **Views>Editar.cshtml**

```
@model ABMPersonas.Models.Persona

@{
    ViewBag.Title = "Editar";
}

<h2>Editar</h2>

@using (Html.BeginForm())
{
<p>Nombre:</p> <br>
@Html.TextBoxFor(p => p.Nombre)
<p>Apellido:</p> <br>
@Html.TextBoxFor(p => p.Apellido)
<p>Edad:</p> <br>
@Html.TextBoxFor(p => p.Edad)

<input type="submit" value="Modificar">
}
```

Bibliografía

Adam Freeman (2013), Pro ASP.NET MVC 5, ISBN 978-1-4302-6530-6, Apress

Microsoft .NET Docs, Ado.Net (2017) Consultado el (28/03/2020) Recuperado de https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/

Nick Harrison (2015), ASP.NET MVC Succinctly, Syncfusion, USA. Recuperado de: https://www.syncfusion.com/ebooks/aspnet_mvc_succinctly

Open Web Application Security Project (OWASP) Consultado el (22/03/2020) Recuperado de: https://owasp.org/www-community/attacks/SQL_Injection

Atribución-NoComercial-SinDerivadas

Se permite descargar esta obra y compartirla, siempre y cuando no sea modificado y/o alterarse su contenido, ni se comercializarse. Referenciarlo de la siguiente manera:

Universidad Tecnológica Nacional Regional Córdoba (2020). Material para la Tecnicatura en Programación Semipresencial de Córdoba. Argentina.