

Tema 9. Acceso a Datos

1. Introducción.....	2
2. Conectar a una Base de Datos de SQL Server.....	5
3. Utilización de DataSet y DataAdapter.....	9
4. Mostrar datos del Dataset.....	11
5. Navegación por la Base de Datos.....	13
6. Añadir un nuevo registro a la Base de Datos.	15
7. Actualizar y Eliminar Registros.	17
8. Acceso a datos utilizando POO.....	18

1. Introducción.

En los siguientes temas se tratará el acceso a datos desde VC#.NET haciendo uso modelo de acceso a datos incluido en la plataforma .NET Framework: **ADO .NET**.

Se mostrarán las tareas básicas para el acceso a datos desde aplicaciones Windows empleando la tecnología proporcionada por ADO .NET.

ADO .NET es la nueva versión del modelo de objetos ADO (ActiveX Data Objects), es decir, la tecnología que ofrece Microsoft para el acceso a datos. ADO .NET utiliza XML como formato universal de transmisión de datos.

COMPONENTES DE ADO .NET

Los componentes de ADO.NET han sido diseñados para separar el acceso a datos de la manipulación de los datos.

Existen dos componentes principales de ADO.NET que lo cumplen: el componente *DataSet* y los proveedores de datos .NET.

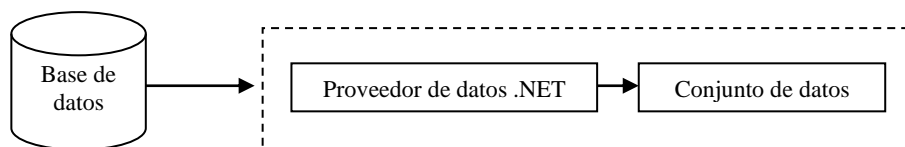
ADO.NET y Windows Forms proporcionan componentes que podemos utilizar para mostrar nuestros datos. Incluyen controles como *DataGridView*, que pueden ser enlazados a datos, y propiedades de enlace a datos en la mayoría de los controles estándar de Windows, como los controles *TextBox*, *Label*, *ComboBox* y *ListBox*.

El .NET Framework incluye numerosos proveedores de datos .NET, incluyendo el proveedor de datos de .NET para SQL Server, el proveedor de datos de .NET OLE DB para SQL, y el proveedor OLE DB para Microsoft Jet.

ADO .NET utiliza dos componentes para recuperar y modificar la información contenida en una base de datos:

- Proveedor de datos .NET
- Conjunto de datos

El proveedor de datos .NET se utiliza para acceder a los datos y el conjunto de datos para manipularlos. En la siguiente figura podemos ver los componentes de ADO .NET que se describirán más adelante.



Componentes de ADO .NET

PROVEEDOR DE DATOS .NET

El proveedor de datos .NET vincula una aplicación a una fuente de datos y permite a la aplicación acceder a los datos de la misma. Por medio del proveedor de datos .NET podemos llevar a cabo las siguientes tareas:

- Conectarnos a una base de datos
- Ejecutar comandos
- Recuperar resultados

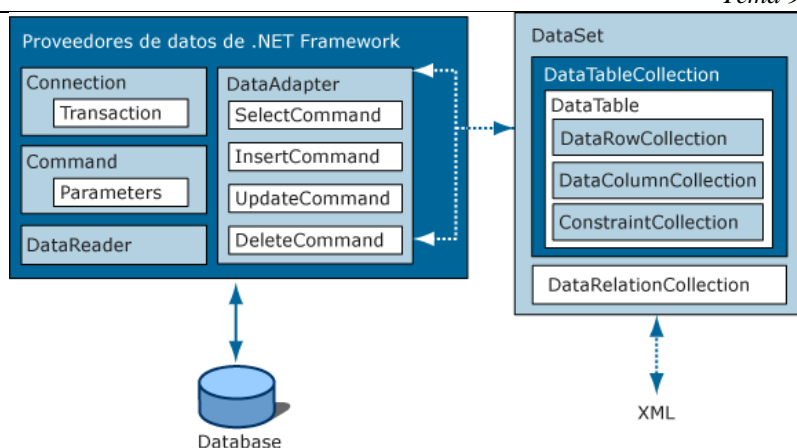
Con ayuda de ADO .NET podemos acceder y gestionar los datos de distintas fuentes de datos. Para permitir el acceso y la manipulación de distintas fuentes de datos, ADO .NET cuenta con varios tipos de proveedores de datos .NET.

En la tabla siguiente se muestran los proveedores de datos de .NET Framework que se incluyen en .NET Framework:

Proveedor de datos de .NET Framework	Descripción
Proveedor de datos de .NET Framework para SQL Server	Proporciona acceso de datos para Microsoft SQL Server versión 7.0 o posterior. Utiliza el espacio de nombres <i>System.Data.SqlClient</i>
Proveedor de datos de .NET Framework para OLE DB	Para orígenes de datos que se exponen mediante OLE DB. Utiliza el espacio de nombres <i>System.Data.OleDb</i>
Proveedor de datos de .NET Framework para ODBC	Para orígenes de datos que se exponen mediante ODBC. Utiliza el espacio de nombres <i>System.Data.Odbc</i>
Proveedor de datos de .NET Framework para Oracle	Para orígenes de datos de Oracle. El proveedor de datos de .NET Framework para Oracle es compatible con la versión 8.1.7 y posteriores del software de cliente de Oracle y utiliza el espacio de nombres <i>System.Data.OracleClient</i>

Para que una aplicación se pueda comunicar con una fuente de datos, el proveedor de datos .NET utiliza cuatro objetos denominados *objetos de proveedor de datos .NET*:

- El objeto *Connection*
- El objeto *Command*
- El objeto *DataReader*
- El objeto *DataAdapter*



Arquitectura de ADO .NET

Visual Studio proporciona adaptadores de datos para utilizarlos con bases de datos de diferentes proveedores.

CONJUNTO DE DATOS

El objeto *DataSet* es una representación de una o varias tablas de una base de datos con las que va a trabajar la aplicación, que se almacenan en una memoria caché desconectada.

Los datos contenidos en un conjunto de datos se pueden enlazar con los controles de un formulario.

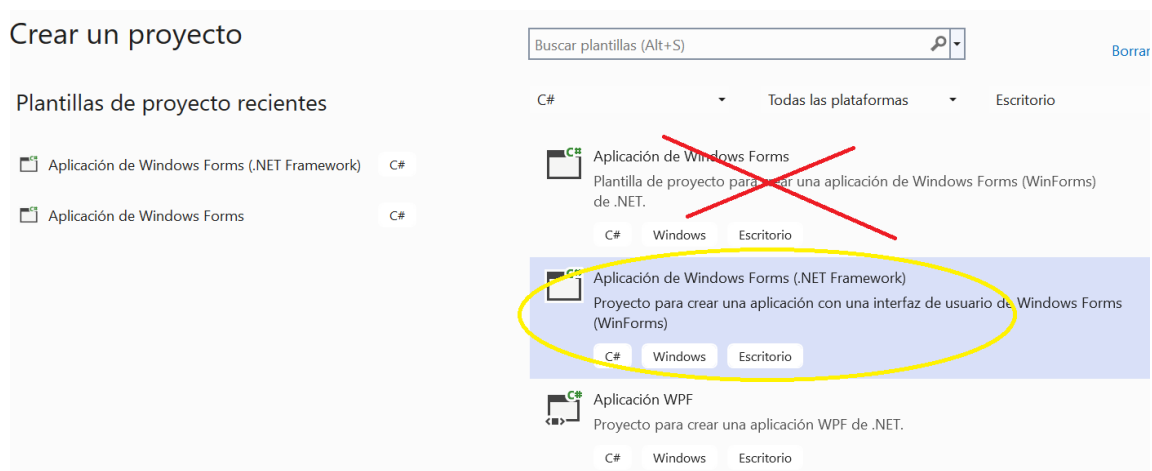
Para poder crear e inicializar las tablas del *DataSet* debemos hacer uso del objeto *DataAdapter* y su método *Fill()*, al que le pasaremos como parámetro una cadena que representa la consulta que se va a ejecutar y que va a rellenar de datos el *DataSet*. El método *Fill()*, posee dos parámetros; el primero es el *DataSet* a rellenar de información; y el segundo, una cadena con el nombre que tiene la tabla creada dentro del *DataSet*, producto de la ejecución de la consulta.

2. Conectar a una Base de Datos de SQL Server.

En este apartado vamos a ver cómo conectar con una base de datos de SQL Server. En concreto vamos a conectar con una base de datos llamada **Instituto** que tenéis en la plataforma. La BD tiene las tablas Profesores, Alumnos y Cursos.

Vamos a crear un nuevo proyecto C# en nuestro disco duro o lápiz de memoria y **copiaremos la BD en la carpeta** de nuestro proyecto.

Es importante que al crear el nuevo proyecto elijamos como tipo de proyecto Aplicación de Windows Forms (.NET Framework):

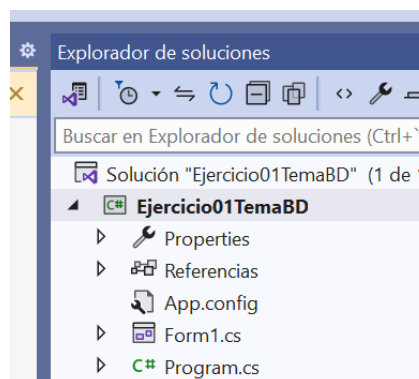


Al crear el nuevo proyecto vamos a entrar en el **evento Load** del formulario (hacemos doble click sobre el formulario en modo diseño). Este evento es el que se ejecuta cuando el formulario se carga.

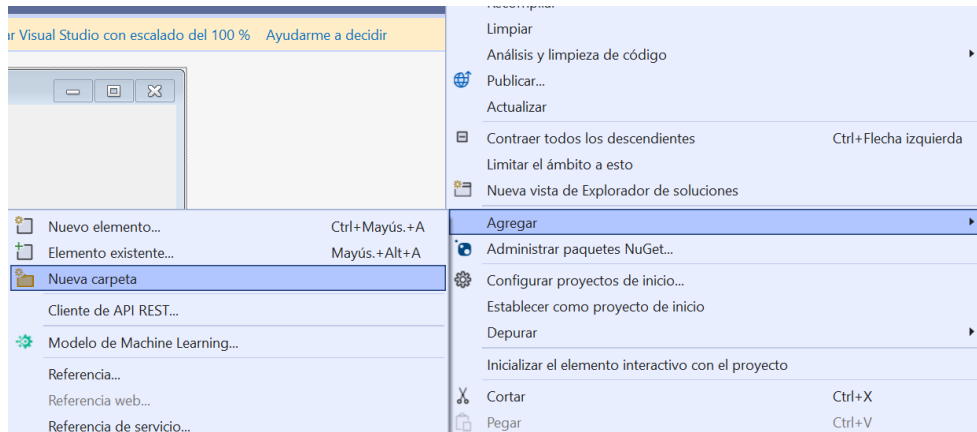
En el evento Load vamos a crear el **objeto conexión** y a darle la ruta de la BD.

Vamos a seguir una serie de pasos para añadir la BD a nuestro proyecto y recoger la ruta de la forma más sencilla posible.

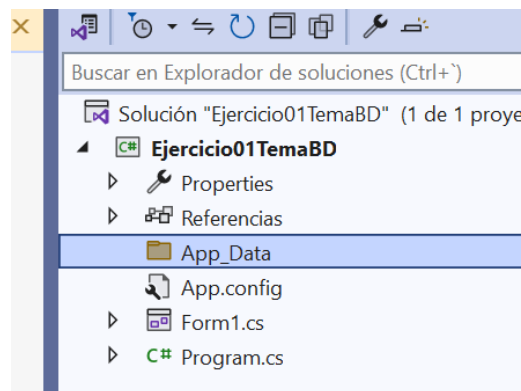
- En primer lugar, creamos una carpeta llamada App_Data en nuestro explorador de soluciones. Para ello pulsamos botón derecho sobre el nombre del proyecto:



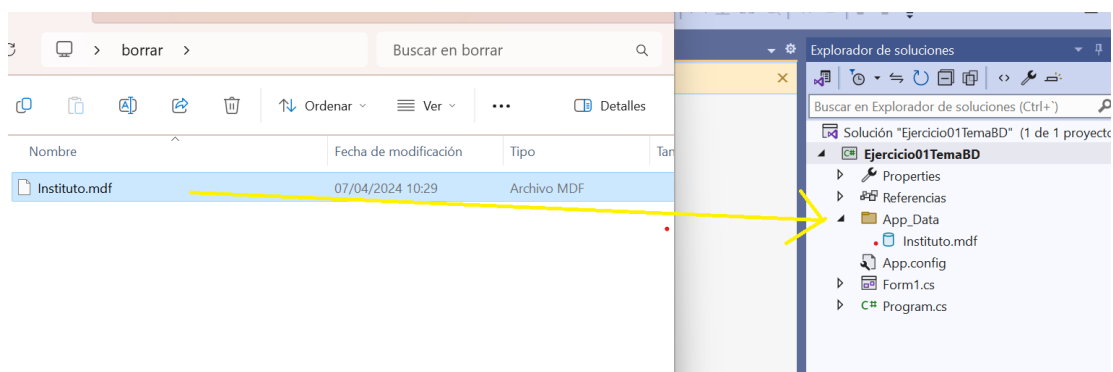
- Y elegimos Agregar->Nueva Carpeta:



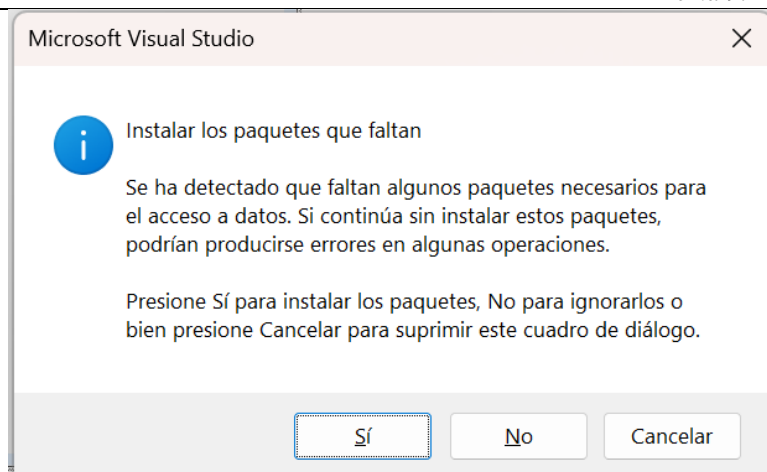
- Dándole el nombre **App_Data**:



- Arrastramos a esa carpeta el fichero **Instituto.mdf**.



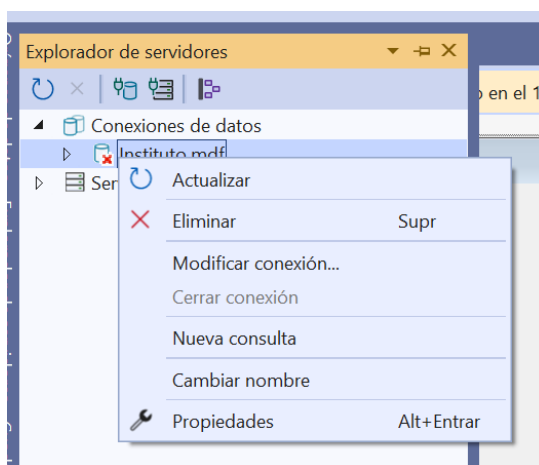
- Es posible, que nos de la primera vez que lo hacemos el siguiente mensaje que nos pide instalar los paquetes necesarios para Acceso a Datos. **Le decimos que sí.** Nos abrirá el Visual Studio Installer y le diremos que instale los paquetes.



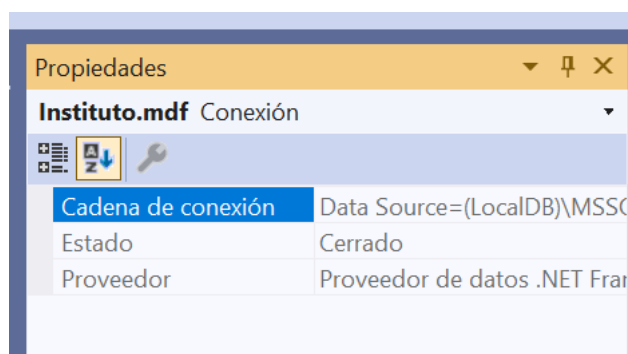
Una vez que hayamos hecho estos pasos (si ha instalado paquetes la primera vez, nos habrá reiniciado Visual Studio), vamos a hacer doble click sobre el formulario y de esa manera entramos al evento Load, que es el que se ejecuta cuando se carga el mismo.

A continuación, vamos a obtener la **cadena de conexión de la BD**.

- En el menú Ver->Explorador de soluciones abrimos el Explorador de soluciones, donde nos aparecerá la BD. La seleccionamos y abrimos sus propiedades:



- En sus propiedades seleccionamos y copiamos (Ctrl + C) su Cadena de conexión:



Vamos a **abrir y cerrar** la conexión en el evento Load.
Para poder trabajar debemos añadir la librería:

```
using System.Data.SqlClient;
```

Nuestra ventana de código quedará **de momento** así:

```
private void Form1_Load(object sender, EventArgs e)
{
    string cadenaConexion = "Data Source=(LocalDB)\\MSSQLLocalDB;
AttachDbFilename= C:\\ejemploBD1conConexion\\App_data\\Instituto.mdf;Integrated
Security=True;Connect Timeout=30";

    SqlConnection con = new SqlConnection(cadenaConexion);

    // Abrimos la conexión.
    con.Open();

    // Cerramos la conexión.
    con.Close();
}
```


3. Utilización de DataSet y DataAdapter.

Hemos visto en el apartado anterior cómo realizar la conexión con la Base de Datos. El siguiente paso es recoger los registros de la tabla que nos interese (en este caso la tabla Profesores). Para ello necesitamos un objeto **DataSet** y otro **DataAdapter**.

Un **DataSet** (conjunto de datos) es donde se almacenan los datos cuando se recogen de la BD. Podemos pensar que es algo así como una tabla o matriz, siendo las columnas cada columna de la tabla y las filas representarían cada registro completo de la tabla.

El **DataSet** debe ser rellenado con datos. El DataSet y el objeto conexión que abrimos en el apartado anterior no pueden interactuar entre ellos. Para ello necesitan el **DataAdapter**. El DataAdapter llena el DataSet con registros de la BD.

- Para crear un **DataSet** se hace de la siguiente forma.

Lo declaramos **fuera** del evento Load:

```
DataSet dataSetProfs;
```

y creamos el objeto en el evento Load:

```
dataSetProfs = new DataSet();
```

- Para crear el **DataAdapter**.

Lo declaramos **fuera** de Load:

```
SqlDataAdapter dataAdapterProfesores;
```

y creamos el objeto dentro del evento Load utilizando una sentencia de SQL para seleccionar los registros de la tabla que nos interese, así como la conexión que tenemos abierta:

```
string cadenaSQL = "SELECT * From Profesores";  
dataAdapterProfesores = new SqlDataAdapter(cadenaSQL, con);
```

- **Rellenamos** el DataSet con el DataAdapter.

Rellenamos y además damos un identificador a la tabla (Profesores).

```
dataAdapterProfesores.Fill(dataSetProfs, "Profesores");
```

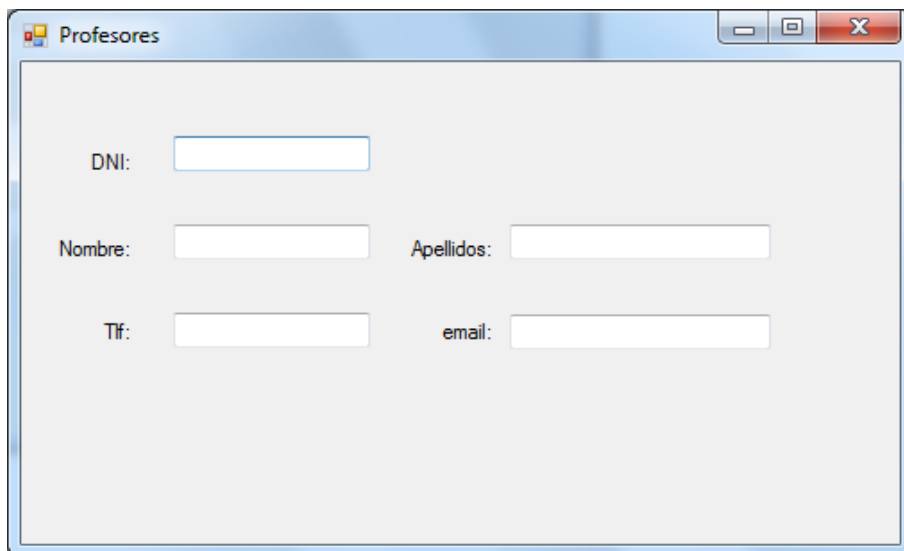
Nuestro código, por el momento, quedará de la siguiente forma:

```
DataSet dataSetProfs;  
SqlDataAdapter dataAdapterProfs;  
private void Form1_Load(object sender, EventArgs e)  
{  
    String cadenaConexion = "Data  
Source=(LocalDB)\\MSSQLLocalDB;AttachDbFilename=C:\\Corregir\\Ejercicio01Tema  
BD\\Ejercicio01TemaBD\\App_Data\\Instituto.mdf;Integrated Security=True";  
  
    SqlConnection con = new SqlConnection(cadenaConexion);  
  
    // Abrimos la conexión.  
    con.Open();  
  
    string cadenaSQL = "SELECT * From Profesores";  
    dataAdapterProfs = new SqlDataAdapter(cadenaSQL, con);  
  
    dataSetProfs = new DataSet();  
  
    dataAdapterProfs.Fill(dataSetProfs, "Profesores");  
  
    // Cerramos la conexión.  
    con.Close();  
}
```

4. Mostrar datos del Dataset.

De momento hemos rellenado el Dataset con los registros de la tabla Profesores. Vamos a ver cómo mostrar datos en textBox.

Crearemos un formulario similar al siguiente donde iremos mostrando los datos del dataSet.



The image shows a Windows application window titled "Profesores". Inside the window, there is a light gray background with five text input fields arranged in three rows. The first row contains a label "DNI:" followed by a single text box. The second row contains labels "Nombre:" and "Apellidos:" followed by two text boxes. The third row contains labels "Tlf:" and "email:" followed by two text boxes. The window has standard Windows window controls (minimize, maximize, close) in the top right corner.

De momento lo único que vamos a hacer es mostrar los datos del primer registro de la tabla cuando carguemos el formulario en el evento Load.

Para ello vamos a declarar una variable privada al formulario que nos indique la posición actual del registro en la tabla (de momento será 0).

Además, vamos **a declarar un subprograma** que se llamará mostrarRegistro que nos mostrará en los textBox el registro situado en la posición correspondiente.

En esta función primero cogeremos el registro situado en esa posición y a continuación **meteremos en cada textBox el campo o columna** correspondiente de ese registro.

Esto lo podemos hacer de dos formas distintas, **por posición o por nombre de campo**.

El código quedará (en amarillo las cosas que hemos añadido):

```
DataSet dataSetProfs;
SqlDataAdapter dataAdapterProfs;
// Variable que indica en qué registro estamos situados.
private int pos;

private void mostrarRegistro(int pos)
{
    // Objeto que nos permite recoger un registro de la tabla.
    DataRow dRegistro;

    // Cogemos el registro de la posición pos en la tabla Profesores
    dRegistro = dataSetProfs.Tables["Profesores"].Rows[pos];

    // Cogemos el valor de cada una de las columnas del registro
    // y lo ponemos en el textBox correspondiente.
    txtDNI.Text = dRegistro[0].ToString();
    txtNombre.Text = dRegistro[1].ToString();
    txtApellidos.Text = dRegistro[2].ToString();
    txtTlf.Text = dRegistro[3].ToString();
    txtEmail.Text = dRegistro[4].ToString();

    //También lo podemos hacer con el nombre del campo. Sería:
    //txtDNI.Text = dRegistro["DNI"].ToString();
    //txtNombre.Text = dRegistro["Nombre"].ToString();
    //txtApellidos.Text = dRegistro["Apellido"].ToString();
    //txtTlf.Text = dRegistro["Tlf"].ToString();
    //txtEmail.Text = dRegistro["Email"].ToString();

}

private void Form1_Load(object sender, EventArgs e)
{
    string cadenaConexión = "Data Source=(LocalDB)\\MSSQLLocalDB;
AttachDbFilename=C:\\ejemploBD1conConexion\\App_data\\Instituto.mdf;Integrated
Security=True;Connect Timeout=30";
    SqlConnection con = new SqlConnection(cadenaConexión);

    // Abrimos la conexión.
    con.Open();

    string cadenaSQL = "SELECT * From Profesores";
    dataAdapterProfs = new SqlDataAdapter(cadenaSQL, con);

    dataSetProfs = new DataSet();

    dataAdapterProfs.Fill(dataSetProfs, "Profesores");

    // Situamos la primera posición
    pos = 0;
    mostrarRegistro(pos);

    // Cerramos la conexión.
    con.Close();

}
```

5. Navegación por la Base de Datos

Vamos a continuación a ver cómo podemos “navegar” por los registros de la tabla de profesores de nuestra BD.

Hasta ahora únicamente hemos puesto en los textbox del formulario el primer registro de nuestra tabla. Cuando hablamos de navegar por la tabla, nos referimos a la posibilidad de ir al siguiente registro, al anterior, al primero o al último.

Nuestro formulario quedará así:



Para poder navegar necesitamos saber cuál es el número de registros que tiene nuestra tabla. Para ello definimos fuera de los eventos una variable llamada maxRegistros:

```
// Variable que indica en qué registro estamos situados.  
private int pos;  
// Variable que indica en número total de registros de nuestra tabla  
private int maxRegistros;
```

...

En el evento Load la inicializamos:

```
// Situamos la primera posición  
pos = 0;  
mostrarRegistro(pos);  
// Obtenemos el número de registros  
maxRegistros = dataSetProfs.Tables["Profesores"].Rows.Count;
```

Los botones que aparecen nos sirven para navegar al primer registro, al último, al siguiente y al anterior.

Quedaría así el código de los botones:

```
private void bPrimero_Click(object sender, EventArgs e)
{
    // Ponemos la primera posición
    pos = 0;
    mostrarRegistro(pos);
}

private void bAnterior_Click(object sender, EventArgs e)
{
    // Vamos a la posición anterior.
    pos--;
    mostrarRegistro(pos);
}

private void bSiguiente_Click(object sender, EventArgs e)
{
    // Vamos a la posición siguiente
    pos++;
    mostrarRegistro(pos);
}

private void bUltimo_Click(object sender, EventArgs e)
{
    // Vamos a la última posición.
    // Los registros van del 0 al numero de registros - 1
    pos = maxRegistros - 1;
    mostrarRegistro(pos);
}
```

Atención: Tener en cuenta que podemos salirnos de los límites de los registros de la tabla. Por ejemplo, si estamos en el primer registro y pulsamos **Anterior** o estamos en el último y pulsamos **Siguiente** nos salimos de los registros de la tabla y nos dará un error de ejecución.

Se deja como ejercicio al alumno evitar este error.

6. Añadir un nuevo registro a la Base de Datos.

Vamos a ver en este apartado como añadir un nuevo registro a la tabla correspondiente de la BD.

Para ello vamos a añadir a nuestro formulario dos nuevos botones:

- Botón Añadir que lo que hará será poner los textBox a blanco para que podamos escribir nuevos datos.
- Botón Guardar. Lo que hará este botón es guardar los datos que hay en el formulario en un nuevo registro al final de la tabla.

El código del botón Añadir será como sigue. Simplemente “limpiamos” el contenido de los textBox para que el usuario pueda introducir nuevos datos.

```
private void btnAnyadir_Click(object sender, EventArgs e)
{
    txtDNI.Clear();
    txtNombre.Clear();
    txtApellidos.Clear();
    txtTlf.Clear();
    txtEmail.Clear();
}
```

Una vez el usuario ha introducido un nuevo registro en los textbox debe poder guardarlo. Para ello hemos de hacer el botón Guardar. En él necesitamos hacer dos cosas: guardarlo en el Dataset y guardarlo en la Base de Datos.

Para añadir un registro al Dataset creamos un nuevo registro:

```
// Creamos un nuevo registro.
DataRow dRegistro = dataSetProfs.Tables["Profesores"].NewRow();
```

Ahora tenemos que poner los datos que tenemos en los textBox en este nuevo registro que hemos creado. Podemos hacer esto de dos formas. Bien con un índice que indique la posición de la columna:

```
dRegistro[0] = txtDNI.Text;
dRegistro[1] = txtNombre.Text;
...
```

o bien con el propio nombre de la columna en la tabla de la BD:

```
dRegistro["DNI"] = txtDNI.Text;
dRegistro["Nombre"] = txtNombre.Text;
```

Por ultimo añadimos el siguiente commando para añadir al Dataset:

```
dataSetProfs.Tables["Profesores"].Rows.Add(dRegistro);
```

Como hemos dicho con este código lo que conseguimos es añadir el registro al Dataset pero necesitamos hacerlo también con la Base de Datos original.

Para ello necesitamos **reconectar con la BD** ya que en el evento Load cerramos la conexión. En nuestro caso esto se hace mediante un objeto llamado **CommandBuilder**:

```
SqlCommandBuilder cb = new SqlCommandBuilder(dataAdapterProfs);
```

y por ultimo actualizamos la BD original utilizando el DataAdapter:

```
// Actualizamos la BD con el DataAdapter  
dataAdapterProfs.Update(dataSetProfs, "Profesores");
```

El código del botón quedaría:

```
private void btnGuardarNuevo_Click(object sender, EventArgs e)
{
    // Creamos un nuevo registro.
    DataRow dRegistro = dataSetProfs.Tables["Profesores"].NewRow();

    // Metemos los datos en el nuevo registro
    dRegistro[0] = txtDNI.Text;
    dRegistro[1] = txtNombre.Text;
    dRegistro[2] = txtApellidos.Text;
    dRegistro[3] = txtTlf.Text;
    dRegistro[4] = txtEmail.Text;

    /*
    // Si quisieramos hacerlo por nombre de columna en vez de posición
    dRegistro["DNI"] = txtDNI.Text;
    dRegistro["Nombre"] = txtNombre.Text;
    dRegistro["Apellido"] = txtApellidos.Text;
    dRegistro["Tlf"] = txtTlf.Text;
    dRegistro["Email"] = txtEmail.Text;*/

    // Añadimos el registro al Dataset
    dataSetProfs.Tables["Profesores"].Rows.Add(dRegistro);

    // Reconnectamos con el dataAdapter y actualizamos la BD
    SqlCommandBuilder cb = new SqlCommandBuilder(dataAdapterProfs);
    dataAdapterProfs.Update(dataSetProfs, "Profesores");

    // Actualizamos el número de registros y la posición en la tabla
    maxRegistros++;
    pos = maxRegistros - 1;
}
```


7. Actualizar y Eliminar Registros.

Lo primero que vamos a ver en este apartado es cómo actualizar los datos de un registro, es decir, guardar los datos modificados del registro en el que nos encontremos.

Al actualizar es importante que nuestra tabla **tenga clave principal**, porque si no, al actualizar, podemos tener un error de ejecución.

A continuación, tenemos el código para actualizar el registro. Es bastante parecido a añadir un nuevo registro, pero cogiendo el registro actual y dándole los datos de los textBox:

```
private void bActualizar_Click(object sender, EventArgs e)
{
    // Cogemos el registro situado en la posición actual.
    DataRow dRegistro = dataSetProfs.Tables["Profesores"].Rows[pos];

    // Metemos los datos en el registro
    dRegistro[0] = txtDNI.Text;
    dRegistro[1] = txtNombre.Text;
    dRegistro[2] = txtApellidos.Text;
    dRegistro[3] = txtTlf.Text;
    dRegistro[4] = txtEmail.Text;

    // Si quisieramos hacerlo por nombre de columna en vez de posición
    /*
        dRegistro["DNI"] = txtDNI.Text;
        dRegistro["Nombre"] = txtNombre.Text;
        dRegistro["Apellido"] = txtApellidos.Text;
        dRegistro["Tlf"] = txtTlf.Text;
        dRegistro["EMail"] = txtEmail.Text;*/

    // Reconectamos con el dataAdapter y actualizamos la BD
    SqlCommandBuilder cb = new SqlCommandBuilder(dataAdapterProfs);
    dataAdapterProfs.Update(dataSetProfs, "Profesores");
}
```

Para eliminar lo que hacemos es eliminar el registro de la posición actual y actualizar la BD:

```
private void bEliminar_Click(object sender, EventArgs e)
{
    // Eliminamos el registro situado en la posición actual.
    dataSetProfs.Tables["Profesores"].Rows[pos].Delete();

    // Tenemos un registro menos
    maxRegistros--;
    // Nos vamos al primer registro y lo mostramos
    pos = 0;
    mostrarRegistro(pos);

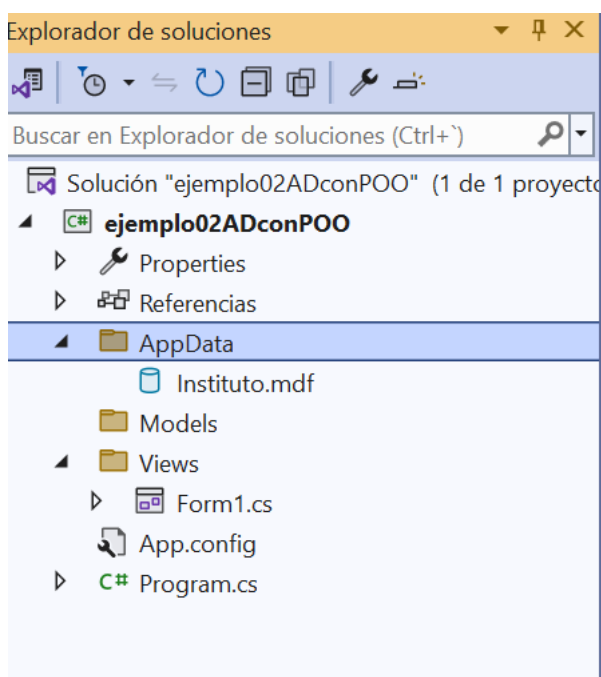
    // Reconectamos con el dataAdapter y actualizamos la BD
    SqlCommandBuilder cb = new SqlCommandBuilder(dataAdapterProfs);
    dataAdapterProfs.Update(dataSetProfs, "Profesores");
}
```

8. Acceso a datos utilizando POO.

Para este nuevo apartado podemos crear un nuevo proyecto, o copiar el anterior, ya que el formulario va a ser el mismo.

En el explorador de soluciones podemos crear la siguiente estructura de carpetas:

- **AppData** donde tendremos la BD local Instituto.mdf.
- **Views** donde tendremos los formularios.
- **Models** donde vamos a crear las clases que utilizaremos para manejar los datos de la BD.



En los apartados anteriores hemos visto cómo conectar con una BD local, y realizar las operaciones **CRUD** (create, read, update y delete), de manera que hemos creado nuevos registros, navegado por la BD, actualizado y borrado registros.

Lo hemos hecho teniendo el grueso del **código en el propio formulario**. Y, aunque esto hace que quizá entendamos mejor qué está pasando en cada situación vamos a continuación a utilizar la **Programación Orientada a Objetos** para **encapsular** todo ese código y que en el formulario únicamente trabajemos el código relacionado con la interfaz.

Para ello, en primer lugar, vamos a crear una **clase llamada Profesor**, la cual meteremos en la carpeta **Views**, que nos permitirá en un momento determinado tener en memoria los datos correspondientes a un profesor.

El código de esta clase podría ser algo así:

```
internal class Profesor
{
    // Miembros
    private string _Dni, _Nombre, _Apellidos, _Tlf, _EMail;

    // Propiedades
    public string Dni
    {
        get { return _Dni; }
        set { _Dni = value; }
    }

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

    // Otra posible forma de hacer la propiedad
    public string Apellidos
    {
        get => _Apellidos;
        set => _Apellidos = value;
    }

    public string Tlf
    {
        get => _Tlf;
        set => _Tlf = value;
    }

    public string eMail
    {
        get => _EMail;
        set => _EMail = value;
    }

    // Constructor
    public Profesor(string Dni, string Nombre, string Apellidos,
                    string Tlf, string eMail)
    {
        _Dni = Dni;
        _Nombre = Nombre;
        _Apellidos = Apellidos;
        _Tlf = Tlf;
        _EMail = eMail;
    }
}
```

Como vemos, simplemente es una clase con los miembros, propiedades y constructor para poder guardar los datos en memoria de un profesor.

Ahora, declaramos una clase (también la meteremos en la carpeta **Models**), donde vamos a tener toda la gestión de la tabla Profesor de nuestra base de datos. En ella gestionamos la conexión, creación del dataSet y dataAdapter, así como las operaciones CRUD.

```
internal class SqlDBHelper
{
    // Miembros para guardar el dataSet y el dataAdapter de profesores.
    private DataSet dataSetProfs;
    private SqlDataAdapter daProfesores;

    // Miembro para guardar el número de profesores.
    private int _numProfesores;
    // Propiedad de solo lectura.
    public int NumProfesores
    {
        get => _numProfesores;
    }

    // Constructor del objeto.
    // En el mismo hacemos la conexión y creamos dataSet y dataAdapter
    public SqlDBHelper()
    {
        string cadenaConexion = "Data
Source=(LocalDB)\\MSSQLLocalDB;AttachDbFilename=C:\\Corregir\\ejemplo02ADconP
OO\\ejemplo02ADconPOO\\AppData\\Instituto.mdf;Integrated Security=True";

        SqlConnection con = new SqlConnection(cadenaConexion);

        // Abrimos la conexión.
        con.Open();

        string cadenaSQL = "SELECT * From Profesores";
        daProfesores = new SqlDataAdapter(cadenaSQL, con);

        dataSetProfs = new DataSet();

        daProfesores.Fill(dataSetProfs, "Profesores");

        // Obtenemos el número de profesores
        _numProfesores = dataSetProfs.Tables["Profesores"].Rows.Count;

        // Cerramos la conexión.
        con.Close();
    }
}
```

```
// Método que a partir de una posición en la BD
// Devuelve un objeto profesor.
// Devuelve null si pos está fuera de los límites
public Profesor devuelveProfesor(int pos)
{
    Profesor profesor = null;

    if (pos >= 0 && pos < _numProfesores)
    {
        // Objeto que nos permite recoger un registro de la tabla.
        DataRow dRegistro;

        // Cogemos el registro de la posición pos en la tabla Profesores
        dRegistro = dataSetProfs.Tables["Profesores"].Rows[pos];

        // Cogemos el valor de cada una de las columnas del registro
        // y creamos el objeto profesor con esos datos.
        profesor = new Profesor(dRegistro[0].ToString(),
                                dRegistro[1].ToString(), dRegistro[2].ToString(),
                                dRegistro[3].ToString(), dRegistro[4].ToString()
                                );
    }

    return profesor;
}

// Metodos CRUD

// Método que reconecta y actualiza la BD
private void reconectarBD()
{
    // Reconectamos con el dataAdapter y actualizamos la BD
    SqlCommandBuilder cb = new SqlCommandBuilder(daProfesores);
    daProfesores.Update(dataSetProfs, "Profesores");
}

// Método que añade un profesor a nuestra BD
public void anyadirProfesor(Profesor profesor)
{
    // Creamos un nuevo registro.
    DataRow dRegistro = dataSetProfs.Tables["Profesores"].NewRow();

    // Metemos los datos en el nuevo registro
    dRegistro[0] = profesor.Dni;
    dRegistro[1] = profesor.Nombre;
    dRegistro[2] = profesor.Apellidos;
```

```
dRegistro[3] = profesor.Tlf;
dRegistro[4] = profesor.eMail;

// Si quisieramos hacerlo por nombre de columna en vez de posición
//      dRegistro["DNI"] = profesor.Dni;

// Añadimos el registro al Dataset
dataSetProfs.Tables["Profesores"].Rows.Add(dRegistro);

// Reconnectamos y actualizamos la BD
reconectarBD();

// Actualizamos el número de profesores
_numProfesores++;
}

// Actualizamos los datos del profesor
// situado en la posición pos
public void actualizarProfesor(Profesor profesor, int pos)
{
    // Cogemos el registro situado en la posición actual.
    DataRow dRegistro = dataSetProfs.Tables["Profesores"].Rows[pos];

    // Metemos los datos en el registro
    dRegistro[0] = profesor.Dni;
    dRegistro[1] = profesor.Nombre;
    dRegistro[2] = profesor.Apellidos;
    dRegistro[3] = profesor.Tlf;
    dRegistro[4] = profesor.eMail;

    // Si quisieramos hacerlo por nombre de columna en vez de posición
    // dRegistro["DNI"] = profesor.Dni;

    // Reconnectamos y actualizamos la BD
    reconectarBD();
}

public void eliminarProfesor(int pos)
{
    // Eliminamos el registro situado en la posición actual.
    dataSetProfs.Tables["Profesores"].Rows[pos].Delete();

    // Tenemos un profesor menos
    _numProfesores--;

    // Reconnectamos y actualizamos la BD
```

```
reconectarBD();  
}  
  
}
```

Y, finalmente, ¿cuál es el objetivo de pasar todo esto a una clase mediante POO? Pues conseguir que la utilización de la BD en el formulario sea mucho más sencilla:

```
public partial class Form1 : Form  
{  
    public Form1()  
    {  
        InitializeComponent();  
    }  
  
    // Instancia del objeto que maneja la BD.  
    SqlDBHelper sqlDBHelper;  
  
    // Variable que indica en qué registro estamos situados.  
    private int pos;  
  
    private void Form1_Load(object sender, EventArgs e)  
    {  
        // Creamos el objeto BD  
        sqlDBHelper = new SqlDBHelper();  
  
        // Situamos la primera posición  
        // y mostramos el registro  
        pos = 0;  
        mostrarRegistro(pos);  
    }  
  
    // Subprograma que muestra el registro situado en la posición pos  
    private void mostrarRegistro(int pos)  
    {  
        Profesor profesor;  
  
        profesor = sqlDBHelper.devuelveProfesor(pos);  
  
        // Ponemos los valores en los textBox correspondientes  
        txtDNI.Text = profesor.Dni;  
        txtNombre.Text = profesor.Nombre;  
        txtApellidos.Text = profesor.Apellidos;  
        txtTlf.Text = profesor.Tlf;  
        txtEmail.Text = profesor.eMail;  
    }  
  
    private void bPrimero_Click(object sender, EventArgs e)  
    {  
        // Ponemos la primera posición  
        pos = 0;  
        mostrarRegistro(pos);  
    }  
}
```

```
private void bAnterior_Click(object sender, EventArgs e)
{
    // Vamos a la posición anterior.
    pos--;
    mostrarRegistro(pos);
}

private void bSiguiente_Click(object sender, EventArgs e)
{
    // Vamos a la posición siguiente
    pos++;
    mostrarRegistro(pos);
}

private void bUltimo_Click(object sender, EventArgs e)
{
    // Vamos a la última posición.
    // Tener en cuenta que los registros van del 0 al numero de
    // registros - 1
    pos = sqlDBHelper.NumProfesores - 1;
    mostrarRegistro(pos);
}

private void bAnyadir_Click(object sender, EventArgs e)
{
    txtDNI.Clear();
    txtNombre.Clear();
    txtApellidos.Clear();
    txtTlf.Clear();
    txteMail.Clear();
}

private void bGuardarNuevo_Click(object sender, EventArgs e)
{
    // Creamos el profesor con los datos del formulario
    Profesor profesor = new Profesor(txtDNI.Text, txtNombre.Text,
        txtApellidos.Text, txtTlf.Text, txteMail.Text);

    sqlDBHelper.anyadirProfesor(profesor);

    // Actualizamos la posición
    pos = sqlDBHelper.NumProfesores - 1;
}

private void bActualizar_Click(object sender, EventArgs e)
{
    // Creamos el profesor con los datos del formulario
    Profesor profesor = new Profesor(txtDNI.Text, txtNombre.Text,
        txtApellidos.Text, txtTlf.Text, txteMail.Text);

    sqlDBHelper.actualizarProfesor(profesor, pos);
}

private void bEliminar_Click(object sender, EventArgs e)
{
    sqlDBHelper.eliminarProfesor(pos);

    // Nos vamos al primer registro y lo mostramos
    pos = 0;
    mostrarRegistro(pos);
}
```


}	
---	--