DATAGRIDVIEW Y ACCESO A BASE DE DATOS

Guía de laboratorio - Capítulo IX

Contenidos:

- ✓ Acceso a base de datos utilizando un control DataGridView.
- ✓ Vinculación con Access mediante ADO.NET.
- ✓ Agregar, modificar y eliminar registros mediante ADO.NET.
- ✓ Biblioteca de clases System.Data, sus componentes y proveedores de datos.

CARGA DE DATOS DE BD EN DATAGRIDVIEW MEDIANTE CÓDIGO

Se trata de crear una conexión a una base de datos mediante ADO.NET, una tecnología de acceso a datos potente y fácil de utilizar que permite mantener las conexiones activas durante el tiempo necesario de consulta o actualización de los datos.

Para ello requerimos de órdenes, objetos que encapsulan las sentencias SQL necesarias, almacenando en cache copias temporales para trabajar sin conexión.

Para ejercitar en la práctica este método de conexión a bases de datos, comenzaremos con un formulario que contendrá los siguientes controles:

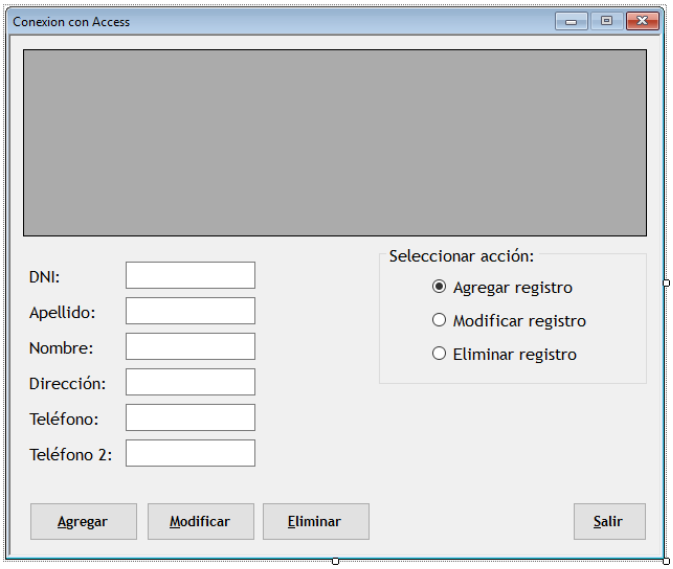
- ✓ Un DataGridView.
- ✓ Seis Labels y seis TextBox: txtDni, txtApellido, txtNombre, txtDireccion, txtTelefono y txtTelefono2.
- ✓ Un GroupBox con tres RadioButtons: rbAgregar, rbModificar y rbEliminar.
- ✓ Cuatro Buttons: btnAgregar, btnModificar, btnEliminar y btnSalir.

En cuanto al diseño dejaremos que utilicen su creatividad teniendo en cuenta criterios ya mencionados en la materia, pero en cuanto a funcionalidad indicamos a continuación las propiedades que deben establecer en cada caso:

dgvPersonas	
AllowUserToAddRows	False
ReadOnly	True
RowHeaderVisible	False (oculta el encabezado de las filas)
AutoSizeColumnsMode	AllCells
rbAgregar	
Checked	True

btnModificar	
Enabled	False (deshabilita el botón)
btnEliminar	
Enabled	False (deshabilita el botón)

Un ejemplo básico de la interfaz podría ser el siguiente:



Una vez completa la interfaz y establecidas las propiedades indicadas, procederemos a crear una **clase** denominada **ConexionConBD.cs** (Click derecho en el nombre del proyecto -> Agregar -> Nuevo -> Clase). Donde accederemos a la base de datos para las diferentes consultas y modificaciones.

La propuesta de esta actividad, es leer los datos de la tabla creada, de una base de datos de Access y crear un ABM (altas, bajas y modificaciones) para el mantenimiento de la misma. Para este ejemplo utilizaremos una base de datos llamada: ***Agenda.mdb***. *Tabla:* ***Personas***. *Campos:* ***DNI, Nombre, Apellido, Direccion, Telefono, Telefono2***.

Antes que todo, a la clase creada debemos habilitar las siguientes bibliotecas de clases para utilizar sus objetos más adelante:

```
using System.Data;
using System.Data.OleDb;
```

A continuación, analizaremos el por qué:

System.Data es un conjunto de clases (o biblioteca de clases) en el cual los componentes se separan en acceso a los datos y la manipulación de los mismos.

System.Data.Oledb: (Object Linking and Embedding for Databases o "Enlace e incrustación de objetos para bases de datos") es un proveedor de datos desarrollado por Microsoft que sirve como puente entre una aplicación y diferentes fuentes de información, o bases de datos.

Otros proveedores de datos:

- ODBC (SQL Access Group) - System.Data.Odbc.
- Oracle - System.OracleClient.
- SQL Server - System.Data.SqlCliente.

Cada uno de estos proveedores proporciona los drivers adecuados para acceder a las distintas bases de datos.

Al no depender de conexiones continuamente activas, **ADO.NET** sólo se conecta a la base durante el tiempo necesario de extracción o actualización de los datos, el resto del tiempo, la conexión permanecen inactiva almacenando los datos (obtenidos mediante órdenes que contienen sentencias SQL) en memoria cache, permitiéndonos trabajar sin conexión con una copia temporal de datos.

Los componentes que crearemos y utilizaremos de estas bibliotecas son:

DataSet, que es un conjunto de componentes que representan un esquema (o una base de datos entera o un subconjunto de una). Puede contener las tablas y las relaciones entre esas tablas (DataTable, DataRow, DataColumn, DataRelation, DataView).

DataAdapter: adaptador de datos entre una base de datos y un DataSet. En otras palabras, llena el DataSet con los datos de la base.

Objeto **Connection**: establecer una conexión a un origen de datos. Por ejemplo para OLE DB se utiliza el objeto OleDbConnection:

```
OleDbConnection Conexion = new OleDbConnection (strConexion);
```

Objeto **Command**: orden para acceso a los datos. Ejecuta sentencias SQL y devuelve resultados:

```
OleDbCommand OrdenSQL = new OleDbCommand ("Select nombre from telefonos", Conexion);
```

Objeto **DataReader**: Lector de datos. Cuando sólo se requieren consultas y no un ABM:

```
OleDbDataReader Lector = ordenSQL.ExecuteReader();
```

Modos de conexión

Debido a que la conexión es la acción más pesada es bueno mencionar que:

- La conexión debe realizarse preferentemente con los proveedores nativos como OLE DB y ODBC.

- La conexión debe abrirse lo más tarde posible. Es recomendable definir las necesidades de utilizarla posteriormente.

- La conexión debe cerrarse lo antes posible, siempre y cuando no tengamos la necesidad de utilizarla posteriormente.

Dicho todo esto, procederemos a crear nuestra conexión en la clase:

```
//creamos los objetos y variables a nivel clase
static private OleDbConnection Conexion;
static private OleDbCommand Orden;

//Indicamos el nombre de la base de datos dentro de cadena de conexion
static string strConexion = "Provider=Microsoft.Jet.OLEDB.4.0; " +
    "Data Source = Agenda.mdb;";
```

En este caso, la base de datos se encuentra dentro de la carpeta Debug del proyecto, en caso de ubicarse en otra dirección, se deberá especificar. Por ejemplo:

```
static string strConexion = "Provider=Microsoft.Jet.OLEDB.4.0; " +
    "Data Source = C:\\Users\\dell\\Desktop\\Agenda.mdb;";
```

La base de datos debe ser *.mdb ya que es el único formato que acepta el proveedor Microsoft.**Jet**.OLEDB.4.0.

De poseer una base de datos *.accdb o cualquiera superior a *.mdb, convertir yendo a **Archivo: Guardar&Publicar: Guardar base de datos como: Base de datos de Access 2000 (*.mdb).**

Si no se puede convertir de ninguna forma, se debería utilizar el proveedor Microsoft.**ACE**.OLEDB.12.0. Para ello primero verificar la configuración de compilación de la plataforma: **Explorador de soluciones: clic derecho sobre el nombre del proyecto: propiedades: solapa Build: Platform Target.** Debe estar en **X86.**

Lo siguiente es instalar "*Microsoft Access Database Engine 2010 Redistributable*" y ejecutar el programa. De no funcionar instalar el "*controlador del sistema de Office 2007: componentes de conectividad de datos"*.

Consulta de datos:

Una vez definida la cadena de conexión, creamos en **ConexionConBD,** un método para la consulta de datos:

```
//método para leer datos de Access
public OleDbDataReader Leer(string Consulta)
{
    //creamos la conexión con la base de datos
    Conexion = new OleDbConnection(strConexion);

    //establecemos la consulta dentro de la conexión
    Orden = new OleDbCommand(Consulta, Conexion);

    try
    {
        Conexion.Open();//abrimos la conexion
    //ejecutamos la consulta y retornamos el resultado de tipo OleDbDataReader
        return Orden.ExecuteReader();
    }
    catch
    {
 //ante cualquier error en la conexión, se devuelve un valor nulo al DataTable (Tabla)
        OleDbDataReader error = null;
        return error;
    }
}
```

Y un método para desconectar la conexión luego de cada acción llevada a cabo:

```
public void Desconectar()
{
    if (Conexion.State == ConnectionState.Open)
        Conexion.Close();
}
```

Para conectar a una base de datos de SQL Server, se requieren algunas mínimas modificaciones en nuestro código. Inicialmente la siguiente biblioteca de clases:

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

Luego, la cadena de conexión (**con autenticación de Windows**) sería de esta forma:

```
static string strConexion = "Data Source = Servidor SQL; Initial Catalog = BaseDeDatos;
Integrated Security = True;";
```

Donde "Servidor SQL" es el nombre del Servidor SQL a utilizar, y "BaseDeDatos" es la base de datos ubicada en su catálogo, previamente creada.

En caso de manejarse **sin autenticación de Windows**, la cadena de conexión sería de esta otra forma:

```
static string strConexion = "Data Source = Servidor SQL; Initial Catalog = BaseDeDatos; user id
= USUARIO;password = CONTRASEÑA;";
```

Una vez realizada la cadena de conexión, necesitaremos un objeto connection, un objeto command y un objeto dataReader, pero en este caso serán instancias de la nueva biblioteca de clases que agregamos al proyecto:

```
SqlConnection Conexion = new SqlConnection(strConexion);
SqlCommand Orden = new SqlCommand(Consulta, Conexion);
SqlDataReader Lector;
```

Por lo que, por ejemplo, el método de obtención de datos "Leer" creado anteriormente, quedaría de la siguiente forma:

```
SqlConnection Conexion;
SqlCommand Orden;

public SqlDataReader LeerSQL(string Consulta)
{
    SqlDataReader Lector;

    //creamos la conexión con la base de datos
    Conexion = new SqlConnection(strConexion);

    //establecemos la consulta dentro de la conexión
    Orden = new SqlCommand(Consulta, Conexion);

    try
    {
        Conexion.Open();//abrimos la conexion
        //ejecutamos la consulta y la asignamos al DataReader
        Lector = Orden.ExecuteReader();
        return Lector; //retornamos los datos obtenidos
    }
    catch
    {
        Lector = null;
        return Lector;
    }
}
```

Ahora sí, para lograr la carga de estos datos en la grilla **dgvPersonas**, colocamos las siguientes líneas de código debajo del constructor de nuestro **formulario**:

```
        //creamos una instancia de la clase para acceder a los métodos creados
        ConexionConBD claseConexion = new ConexionConBD();
        //creamos un objeto con la estructura de una tabla para recibir los datos
        DataTable Tabla = new DataTable();
```

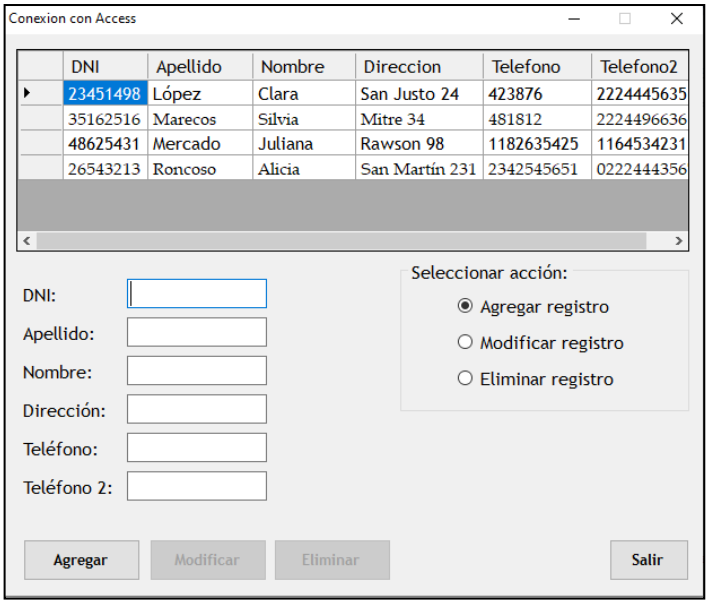
Entonces basta con acceder al evento Load del mismo y escribir lo siguiente:

```
    private void ConexionAccess_Load(object sender, EventArgs e)
    {
            //limpiamos la tabla que va a recibir la consulta
            Tabla.Clear();
            //ejecutamos el método Leer de la clase ConexionConBD y el objeto
            DataTable asigna el resultado a su estructura
            Tabla.Load(claseConexion.Leer("SELECT * FROM Personas ORDER BY Apellido"));

            //Asignamos el orígen de los datos de la grilla
            dgvPersonas.DataSource = Tabla;

            //Cerramos la conexión. Siempre desconectamos al terminar!
            claseConexion.Desconectar();
    }
```

Así es como se visualizarían los datos:



Agregar registros:

Para crear un nuevo registro en la base de datos primero crearemos un método, en la clase **ConexionConBD**, que nos permita realizar la consulta, indicando si la misma fue realizada con éxito o sucedió algún error:

```
public bool ABM(string Consulta)
{
    //variable de control
    bool Resultado = false;

    //creamos la conexión con la base de datos
    Conexion = new OleDbConnection(strConexion);

    //establecemos la consulta dentro de la conexión
    Orden = new OleDbCommand(Consulta, Conexion);

    try
    {
        Conexion.Open();//abrimos la conexion
        //ejecutamos la consulta
        Orden.ExecuteNonQuery();
        //indicamos que todo se realizó de forma correcta
        Resultado = true;
    }
    catch
    {
        //caso contrario, indicamos que hubo un error
        Resultado = false;
    }

    Desconectar();//deconectamos

    //devolvemos el valor asignado en la variable de control
    return Resultado;
}
```

Una vez creado el método, dentro del código del formulario se procede a llamarlo indicando cuál es la sentencia que debe ejecutar:

```
//creamos una nueva variable de control
bool Alta = false;

//llamamos al método y enviamos la consulta de inserción
Alta = claseConexion.ABM("INSERT INTO Personas (DNI, Apellido, Nombre, Direccion, Telefono,
Telefono2) VALUES ('" + txtDni.Text + "', '" + txtApellido.Text +
"', '" + txtNombre.Text + "', '" + txtDireccion.Text + "', '" + txtTelefono.Text + "', '" +
txtTelefono2.Text + "')"))

If (Alta == true) //chequeamos que el proceso se haya realizado correctamente
{
      MessageBox.Show("Se creó correctamente el registro", "Proceso finalizado:");

      ConexionAccess_Load(null, null); //Llamamos al evento Load para la actualización de datos
      en la grilla
}
else
{
      MessageBox.Show("No se pudo completar la operación", "Error en el procedimiento:");
}
```

Modificación de registros:

En cuanto a la modificación, lo único que cambiamos al código anterior es la consulta:

```
//creamos una nueva variable de control
bool Modif = false;

//llamamos al método y enviamos la consulta de modificación
Modif = claseConexion.ABM("UPDATE Personas SET Apellido='" + txtApellido.Text + "',
  Nombre='" + txtNombre.Text + "', Direccion='" + txtDireccion.Text + "', Telefono='"
  + txtTelefono.Text + "', Telefono2='" + txtTelefono2.Text + "' WHERE DNI=" +
  txtDni.Text + ";"))

If (Modif == true) //chequeamos que el proceso se haya realizado correctamente
{

MessageBox.Show("Se modificó correctamente el registro", "Proceso finalizado:");

      ConexionAccess_Load(null, null); //Llamamos al evento Load para la actualización
      de datos en la grilla
}
else
{
MessageBox.Show("No se pudo completar la operación", "Error en el procedimiento:");
}
```

También debemos recordar que el campo clave de la tabla es el DNI, por lo que no debe permitirse la modificación del mismo.

Eliminación de registros:

Volvemos a tener las mismas líneas de código, pero con modificación en la sintaxis de la consulta SQL:

```
if (claseConexion.ABM("DELETE * FROM Personas WHERE DNI=" + txtDni.Text + ";"))
{
MessageBox.Show("Se eliminó correctamente el registro", "Proceso finalizado:");

      ConexionAccess_Load(null, null); //Llamamos al evento Load para la actualización
      de datos en la grilla
}
else
{
MessageBox.Show("No se pudo completar la operación", "Error en el procedimiento:");
}
```

Nótese que en este caso no se utiliza una variable de control de tipo booleana, se utiliza el resultado booleano del método ABM directamente dentro del **if** para controlar la ejecución correcta de la consulta. El resultado obtenido es el mismo, sencillamente de esta forma se evita la creación de una nueva variable, aunque no es recomendable en los casos donde la consulta es confusamente larga, por lo que la variable puede funcionar para el programador como método de organización.

9.1. **Ejercicio:** En este caso la actividad será realizar un abm, de personas, de socios, de alumnos, de lo que ustedes prefieran, pero que cumpla con su finalidad de consultas y ABM con una conexión mediante código:

- Que los datos a ingresar contengan al menos un valor numérico, uno de tipo string y un booleano.
- Que se puedan consultar, insertar, modificar y eliminar registros de una base de datos.
- Que la interfaz sea sumamente intuitiva. Revisar el bloqueo y desbloqueo de botones u objetos de interacción con el usuario. Que no queden opciones válidas para el usuario si no está validada esa posibilidad.
- No permitir modificaciones sobre el DataGridView ni mostrar el último registro en blanco. Utilizar textBox y otros controles para el ingreso, búsqueda y modificación de los mismos.
- Tanto para modificar o eliminar algún registro, se deben mostrar los datos del mismo en los textBox correspondientes, tan sólo haciendo **clic** sobre dicho registro en la grilla.
- Al finalizar cualquier sentencia del ABM (alta, baja o modificación), se debe actualizar la grilla. Es lo que hacemos con la línea de código: `ConexionAccess_Load(null, null);`. Deben reemplazar esto por un método llamado "Actualizar". Tanto para el Load como para cada ABM.
- Al finalizar cualquier sentencia del ABM, se deben limpiar los textBox y los demás objetos de ingreso de datos. Crear un **método** para realizar esto.
- Diseñar el formulario creativamente sin dejar de ser sobrio y amigable a la vista (UX/UI).
- Agregar MessageBoxIcon a los MessageBox.
- Antes de la eliminación de cualquier registro, preguntar al usuario si está seguro de continuar con la eliminación, permitirle continuar o cancelar la operación.
- Validar los textBox para el ingreso de datos. **Crear una clase "Validaciones"** con los métodos necesarios para el control de ingreso de datos. *Propiedad para evitar el "pegado" de texto:* ***ShortcutsEnabled.***
- Se valorará evitar la repetición de código. Siempre que se pueda modularizar, háganlo.
- **Se requiere el uso de al menos una clase más**, de importancia primaria, es decir, que represente una entidad de importancia en el programa.