Unidad 04 - Laboratorio 04

1. ADO.Net con MySQL

Objetivos

Conectarse a una base de datos MySQL y consultar los datos y modificarlos

Duración Aproximada
30 minutos

Pasos

1) Abra el Visual Studio 2005 y haga clic en Archivo -> Nuevo -> Proyecto
2) Elija un proyecto de la categoría C# -> Windows -> Console Aplication.

   A) Completar los siguientes datos:
   - Nombre del Proyecto (**Name**): *Lab04*
   - Ubicación de la solución **(Location)**:

C:\Proyectos\Unidades\Unidad 04

3) Presione Aceptar.

4)
   En este proyecto nos conectaremos a una BD de MySQL. Para ello utilizaremos un driver de conexión a MySQL específicamente diseñado para ADO.Net.
   El mismo es una DLL que puede descargarse gratuitamente de la página de descargas de MySQL.
   Dicha DLL contiene clases específicas para manipular objetos de acceso a datos como DataReader, DataAdapter y Commands.

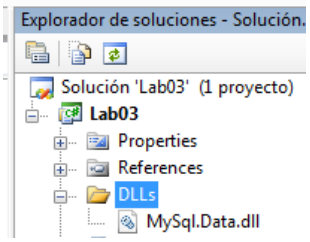
   Para poder utilizarla hay dos maneras:
   1. Instalar la DLL en el sistema e invocarla.
   2. Agregarla al proyecto y hacer referencia a la misma.

   Nosotros utilizaremos la segunda alternativa

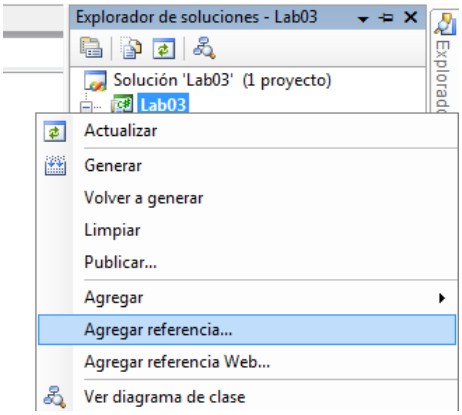
5) Para ello primero debemos agregar la DLL al proyecto. Una buena práctica es crear dentro del proyecto una carpeta donde almacenaremos las DLLs. De esta forma mantenemos la versión que utiliza nuestro proyecto en el mismo y si en algún momento hay que remplazarla por una versión más nueva es muy sencillo sólo cortar y pegar.

   Hacemos clic con el botón derecho sobre el proyecto Lab03. Allí elegimos Agregar → Nueva Carpeta. La llamamos DLLs.

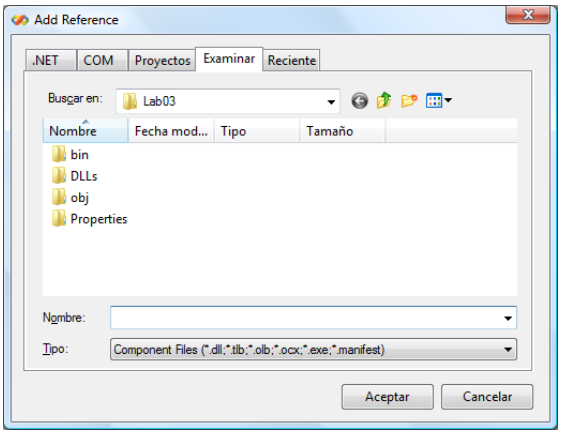
6) Luego tomamos la el archivo MySql.Data.dll desde el Windows Explorer y lo arrastramos y soltamos sobre la carpeta DLLs del explorador de Soluciones. Entonces el mismo se verá así:



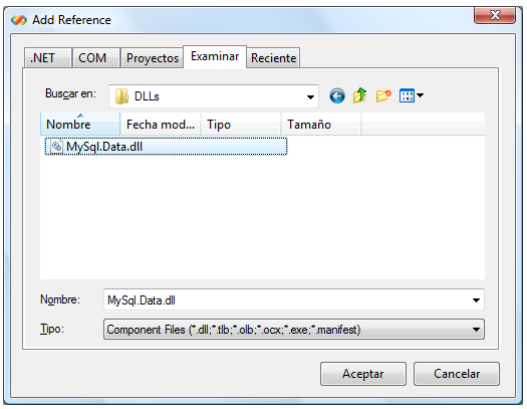
7) Una vez copiado el archivo agregamos la referencia. Para ello hacemos clic con botón derecho sobre el proyecto Lab03 y elegimos Agregar Referencia…



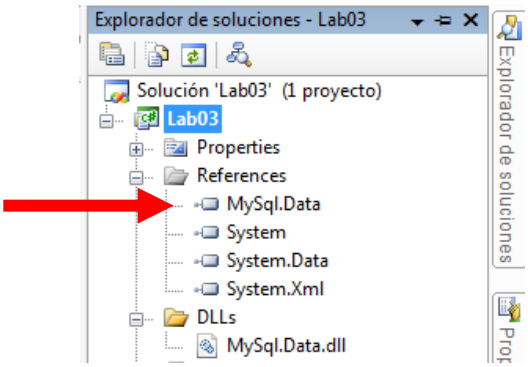
8) Aparecerá la siguiente ventana llamada Add Reference. En ella seleccionamos la pestaña Examinar



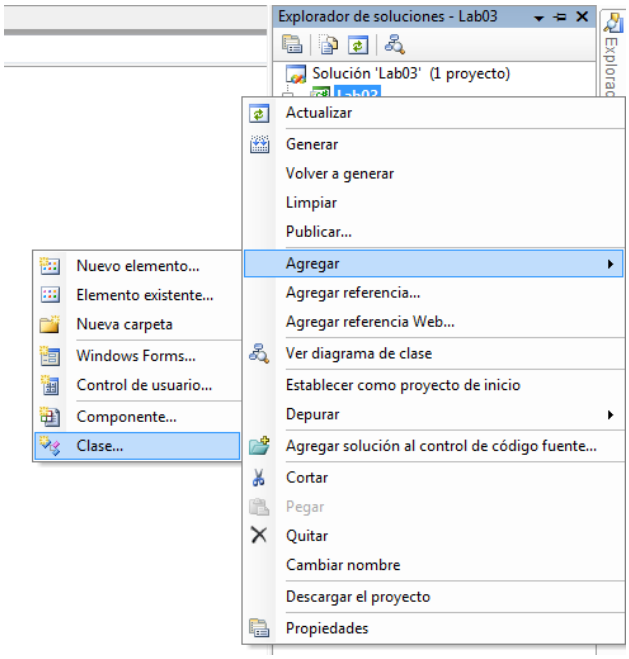
9) Dentro de DLLs elegimos MySQL.Data.dll y hacemos clic en Aceptar.



10) Entonces en el explorador de soluciones en la parte de References debería haberse agregado MySql.Data



11) Hacemos clic con el botón derecho sobre el proyecto Lab03. Allí elegimos Agregar → Clase



La llamamos Contactos.cs

12) En el archivo Contactos.cs agregamos la directiva using System.Data; y creamos la variable de instancia misContactos, la misma es del tipo DataTable.

Además agregamos los métodos getTabla, aplicaCambios, listar, editarFila y eliminarFila ***IDÉNTICOS*** a los que utilizamos en la clase ManejadorArchivo del Laboratorio 2.

La clase entonces queda así:

```
using System;
using System.Collections.Generic;
using System.Text;
```

```
using System.Data;

namespace Lab03
{
    public class Contactos
    {
        protected DataTable misContactos;

        public Contactos()
        {
            this.misContactos = this.getTabla();
        }

        public virtual DataTable getTabla()
        {

            return new DataTable();
        }

        public virtual void aplicaCambios()
        {

        }

        public void listar()
        {
            foreach (DataRow fila in this.misContactos.Rows)
            {
                if (fila.RowState != DataRowState.Deleted)
                {
                    foreach (DataColumn col in this.misContactos.Columns)
                    {
                        Console.WriteLine("{0}: {1}", col.ColumnName, fila[col]);
                    }
                    Console.WriteLine();
                }
            }
        }

        public void nuevaFila()
        {
            DataRow fila = this.misContactos.NewRow();
            foreach (DataColumn col in this.misContactos.Columns)
            {
                Console.Write("Ingrese {0}:",col.ColumnName);
                fila[col] = Console.ReadLine();
            }
            this.misContactos.Rows.Add(fila);
        }

        public void editarFila()
        {
            Console.WriteLine("Ingrese el número de fila a editar");
            int nroFila = int.Parse(Console.ReadLine());
            DataRow fila = this.misContactos.Rows[nroFila-1];
            for (int nroCol = 1; nroCol < this.misContactos.Columns.Count; nroCol++)
                //el 0 se omite por ser la ID
            {
                DataColumn col = this.misContactos.Columns[nroCol];
                Console.Write("Ingrese {0}:", col.ColumnName);
                fila[col] = Console.ReadLine();
            }

        }


        public void eliminarFila()
        {
            Console.WriteLine("Ingrese el número de fila a eliminar");
            int fila = int.Parse(Console.ReadLine());
            this.misContactos.Rows[fila-1].Delete();
        }
    }
   }
```

Los métodos getTabla y aplicaCambios tienen el modificador virtual para permitir ser sobrescritos en las clases hijas.

Listar Contactos

En el método listar los contactos recorremos la colección Rows de la DataTable.
Para listar el nombre de la columna recorremos la colección de columnas del DataTable y utilizamos la propiedad ColumnName y para acceder a cada una de las celdas de la DataTable recorremos la colección de filas (DataRows) con el foreach y luego para cada fila accedemos a los valores de las celdas como si fuese un array pero no sólo podemos acceder con el índice de la columna sino también con el nombre de la columna o como en este caso con el objeto columna.

Nuevos Contactos

Para crear un nuevo contacto creamos una nueva fila en la DataTable con:
this.misContactos.NewRow();

Es muy importante destacar que la única forma de crear una nueva fila es a partir de un DataTable. Esto se debe a que la fila debe tener una estructura de celdas y la misma es proporcionada por el DataTable a partir del cual se genera. Cabe aclarar que aunque una fila se genere a partir de una DataTable no significa que la misma esté agregada dentro de la colección de filas de la DataTable

Luego asignamos los valores a cada una de las celdas de la fila con:

```
foreach (DataColumn col in this.misContactos.Columns)
{
     Console.Write("Ingrese {0}:",col.ColumnName);
     fila[col] = Console.ReadLine();
}
```

Finalmente agregamos la fila a la tabla con:
this.misContactos.Rows.Add(fila);

Editar Contacto

Primero identificamos la fila de la tabla que queremos modificar aquí lo hacemos con el índice: DataRow fila = this.misContactos.Rows[nroFila-1];

También podría hacerse con un método de la DataTable llamado Select que permite filtrar un grupo de filas cuyos datos cumplen con determinados criterios.

Luego modificamos los datos de la fila con:

```
for (int nroCol = 1; nroCol < this.misContactos.Columns.Count; nroCol++)
//el 0 se omite por ser la ID
{
    DataColumn col = this.misContactos.Columns[nroCol];
    Console.Write("Ingrese {0}:", col.ColumnName);
    fila[col] = Console.ReadLine();
}
```

Aquí no es necesario agregarla a la DataTable como en el caso anterior porque la fila ya existía y se encontraba en la DataTable.

Eliminar Contacto

Para eliminar una fila simplemente localizamos la DataRow que deseamos eliminar. Luego simplemente eliminamos la fila con el método Delete de la DataRow.

13) Luego desde el explorador de soluciones agregamos otra nueva clase ContactosMySQL que hereda de Contactos.

14) En la clase ContactosMySQL utilizaremos el conector específico de ADO.Net para MySQL para realizar las operaciones sobre la base de datos. Estas clases se encuentran en el namespace MySql.Data.MySqlClient por lo que incluiremos la directiva using MySql.Data.MySqlClient; y además using System.Data; para poder manipular los datos.

15) Luego creamos la propiedad de sólo lectura connectionString la misma devolverá el siguiente valor:

server=serverisi;database=net;uid=net;pwd=net;

Un Connection String es un string que tiene los datos necesarios para conectarse a un origen de datos a través de una conexión. Para más información visitar http://www.connectionstrings.com/

Cada origen de datos requiere datos específicos, en el caso de MySQL se requiere:

- server: Indica el nombre o la dirección IP del servidor donde se encuentra la base de datos.
- database: Dentro de un servidor de bases de datos puede haber muchas bases de datos y por lo tanto le debemos especificar cuál es la que utilizaremos.
- uid: nombre del usuario de MySQL cuyos permisos utilizaremos.
- psw: la contraseña del usuario indicado en uid.

16) Sobrescribimos entonces el método getTabla. Dentro del mismo creamos una conexión y la abrimos.

17) Luego en el creamos un MySQLCommand con la sentencia SQL para obtener los datos. Para ejecutar el comando utilizamos el método ExecuteReader que crea un objeto DataReader que nos permite leer los datos de los archivos y cargarlos en el DataTable (a través del método Load).

18) A continuación cerramos la conexión.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data;
using MySql.Data.MySqlClient;

namespace Lab03
{
    class ContactosMysql:Contactos
    {
        protected string connectionString
        {
            get
            {
                return "server=localhost;database=net;uid=root;";//;pwd=123entre";
            }
        }

        public override DataTable getTabla()
        {
            using (MySqlConnection Conn = new MySqlConnection(connectionString))
            {
                MySqlCommand cmdSelect = new MySqlCommand("select * from contactos", Conn);
                Conn.Open();
                MySqlDataReader reader = cmdSelect.ExecuteReader();
                DataTable contactos = new DataTable();
                if (reader != null)
                {
                    contactos.Load(reader);
                }
                Conn.Close();
                return contactos;
            }
        }
```

19) Finalmente devolvemos la DataTable para utilizarla

20) A continuación sobrescribimos el método aplicaCambios

21) En el mismo creamos 3 comandos (MySQLCommand) uno para agregar filas otro para modificarlas y finalmente uno para eliminarlas.

Para cada uno de ellos escribimos la sentencia SQL que permite realizar dichas acciones y finalmente creamos los parámetros necesarios para que dada una fila se utilicen esos datos para insertar, modificar o eliminar dichos datos

22) Entonces creamos 3 comandos uno para agregar filas otro para modificarlas y finalmente uno para eliminarlas.

23) Luego abrimos la conexión

24) Sobre la DataTable misContactos buscamos cuales filas fueron agregadas con:

```
DataTable filasNuevas = this.misContactos.GetChanges(DataRowState.Added);
```

Sobre ellas ejecutamos el comando cmdInsert que realiza las inserciones.

25) Acto seguido buscamos las filas eliminadas con:

```
DataTable filasBorradas = this.misContactos.GetChanges(DataRowState.Deleted);
```

Sobre ellas ejecutaremos el comando cmdDelete que sirve para eliminar filas.

26) Luego buscamos las filas que tiene modificaciones con:

```
DataTable filasModificadas = this.misContactos.GetChanges(DataRowState.Modified);
```

Para ejecutar sobre ellas el comando cmdUpdate que es el encargado de modificar los datos.

```
public override void aplicaCambios()
{
     using (MySqlConnection Conn = new MySqlConnection(connectionString))
     {
          MySqlCommand cmdInsert = new MySqlCommand("insert into contactos"+
          "values(@id,@nombre,@apellido,@email,@telefono)", Conn);
          cmdInsert.Parameters.Add("@id", MySqlDbType.Int32);
          cmdInsert.Parameters.Add("@nombre", MySqlDbType.VarChar);
          cmdInsert.Parameters.Add("@apellido", MySqlDbType.VarChar);
          cmdInsert.Parameters.Add("@email", MySqlDbType.VarChar);
          cmdInsert.Parameters.Add("@telefono", MySqlDbType.VarChar);

          MySqlCommand cmdUpdate = new MySqlCommand(
          "update contactos set nombre=@nombre, apellido=@apellido, "+
                "email=@email,telefono=@telefono where id=@id", Conn);
          cmdUpdate.Parameters.Add("@id", MySqlDbType.Int32);
          cmdUpdate.Parameters.Add("@nombre", MySqlDbType.VarChar);
          cmdUpdate.Parameters.Add("@apellido", MySqlDbType.VarChar);
          cmdUpdate.Parameters.Add("@email", MySqlDbType.VarChar);
          cmdUpdate.Parameters.Add("@telefono", MySqlDbType.VarChar);

          MySqlCommand cmdDelete = new MySqlCommand("delete from contactos where id=@id",
                                                    Conn);
          cmdDelete.Parameters.Add("@id", MySqlDbType.Int32);

          DataTable filasNuevas = this.misContactos.GetChanges(DataRowState.Added);
          DataTable filasBorradas = this.misContactos.GetChanges(DataRowState.Deleted);
          DataTable filasModificadas = this.misContactos.GetChanges(DataRowState.Modified);

          Conn.Open();

          if (filasNuevas != null)
          {
               foreach (DataRow fila in filasNuevas.Rows)
               {
                    cmdInsert.Parameters["@id"].Value = fila["id"];
                    cmdInsert.Parameters["@nombre"].Value = fila["nombre"];
                    cmdInsert.Parameters["@apellido"].Value = fila["apellido"];
                    cmdInsert.Parameters["@email"].Value = fila["email"];
                    cmdInsert.Parameters["@telefono"].Value = fila["telefono"];
                    cmdInsert.ExecuteNonQuery();
               }
          }
          if (filasBorradas != null)
          {
               foreach (DataRow fila in filasBorradas.Rows)
               {
                    cmdDelete.Parameters["@id"].Value = fila["id", DataRowVersion.Original];
                    cmdDelete.ExecuteNonQuery();
               }
          }
          if (filasModificadas != null)
          {
               foreach (DataRow fila in filasModificadas.Rows)
               {
                    cmdUpdate.Parameters["@id"].Value = fila["id"];
                    cmdUpdate.Parameters["@nombre"].Value = fila["nombre"];
                    cmdUpdate.Parameters["@apellido"].Value = fila["apellido"];
                    cmdUpdate.Parameters["@email"].Value = fila["email"];
                    cmdUpdate.Parameters["@telefono"].Value = fila["telefono"];
                    cmdUpdate.ExecuteNonQuery();
               }
          }

          Conn.Close();
          this.misContactos.AcceptChanges();
     }
}
```

27) Luego asignamos a cada uno de los parámetros el valor que tiene la fila para ese campo.

28) Para ejecutar cada uno de los comandos utilizamos el método ExecuteNonQuery.

29) Cerramos la conexión y finalmente aceptamos los cambios en la DataTable.

Esto es muy importante ya que si volviera a aplicar los cambios, las inserciones seguirían figurando pendientes y al buscar las filas nuevas seguirán apareciendo. Entonces se volverá a ejecutar el comando INSERT con los datos de esa fila y producirá un error de clave repetida.

De la misma forma se volverán a tratar de ejecutar el comando UPDATE sobre las filas modificadas y el comando DELETE sobre las eliminadas. Aunque esto no producirá error como el comando INSERT, si producirá una baja en el rendimiento de la aplicación.

30) Escribimos el programa principal:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Lab03
{
    class Program
    {
        static void Main(string[] args)
        {
            Contactos contactos = new ContactosMysql();
            contactos.listar();
            menu(contactos);

        }

        static void menu(Contactos contactos)
        {
            string rta = "";
            do
            {
                Console.WriteLine("1 - Listar");
                Console.WriteLine("2 - Agregar");
                Console.WriteLine("3 - Modificar");
                Console.WriteLine("4 - Eliminar");
                Console.WriteLine("5 - Guardar Cambios");
                Console.WriteLine("6 - Salir");
                rta = Console.ReadLine();
                switch (rta)
                {
                    case "1":
                        contactos.listar();
                        break;
                    case "2":
                        contactos.nuevaFila();
                        break;
                    case "3":
                        contactos.editarFila();
                        break;
                    case "4":
                        contactos.eliminarFila();
                        break;
                    case "5":
```

```
                    contactos.aplicaCambios();
                    break;
                default:
                    break;
            }
        } while (rta != "6");
    }


}
}
```

31) Presionamos F5 y ejecutamos. Probamos crear nuevos contactos editar los existentes y eliminar. Luego elegimos la opción de guardar los cambios y salimos. Si volvemos a ejecutar veremos que los cambios furon guardados

32) Ahora modificaremos el programa para que utilice un DataAdapter en lugar de DateReader, GetChanges y ExecuteNonQuery.

33) Para ello agregamos en el proyecto una nueva clase ContactosMysqlConDataAdapter que también hereda de Contactos.

En la clase creamos la propiedad connectionString que devuelve el mismo valor que antes: server=serverisi;database=net;uid=net;pwd=net;

34) Luego creamos el constructor:

```
public ContactosMysqlConDataAdapter()
{
     this.adapater.InsertCommand = new MySqlCommand(
        "insert into contactos values(@id,@nombre,@apellido,@email,@telefono)");
     this.adapater.InsertCommand.Parameters.Add("@id", MySqlDbType.Int32, 1, "id");
     this.adapater.InsertCommand.Parameters.Add("@nombre", MySqlDbType.VarChar, 20, "nombre");
     this.adapater.InsertCommand.Parameters.Add("@apellido", MySqlDbType.VarChar, 20, "apellido");
     this.adapater.InsertCommand.Parameters.Add("@email", MySqlDbType.VarChar, 50, "email");
     this.adapater.InsertCommand.Parameters.Add("@telefono", MySqlDbType.VarChar, 10, "telefono");

     this.adapater.UpdateCommand = new MySqlCommand(
        "update contactos set nombre=@nombre, apellido=@apellido, email=@email,telefono=@telefono "+
        " where id=@id");
     this.adapater.UpdateCommand.Parameters.Add("@id", MySqlDbType.Int32, 1, "id");
     this.adapater.UpdateCommand.Parameters.Add("@nombre", MySqlDbType.VarChar, 20, "nombre");
     this.adapater.UpdateCommand.Parameters.Add("@apellido", MySqlDbType.VarChar, 20, "apellido");
     this.adapater.UpdateCommand.Parameters.Add("@email", MySqlDbType.VarChar, 50, "email");
     this.adapater.UpdateCommand.Parameters.Add("@telefono", MySqlDbType.VarChar, 10, "telefono");

     this.adapater.DeleteCommand = new MySqlCommand("delete from contactos where id=@id");
     this.adapater.DeleteCommand.Parameters.Add("@id", MySqlDbType.Int32, 1, "id");
}
```

Al comienzo del mismo está la instrucción:

```
public ContactosMysqlConDataAdapter()
            : base()
```

Esto significa que al crear un nuevo objeto de la clase ContactosMysqlConDataAdapter se invocará primero al constructor de la clase padre (base) sin parámetros y luego se ejecutarán las sentencias específicas del constructor de esta clase.

Como la clase padre es Contactos se ejecutará su constructor que invoca al método getTabla. Por lo cual sobrescribiremos este método para que además de instanciar la DataTable misContactos y cargar los datos de la base de datos, se encargue de crear un objeto DataAdapter (en este caso un MySqlDataAdapter) llamado adapter que es el que se encargará de realizar todas las operaciones de lectura y escritura sobre la BD.

Para poder utilizar un DataAdapter le debemos indicar una conexión a la base de datos (o en su defecto el connectionString).

También debe definirse un SelectCommand que es la sentencia SQL que permite recuperar los datos de la base de datos.

Todo esto lo hacemos con la sentencia

```
this.adapater = new MySqlDataAdapter("select * from contactos", this.connectionString);
```

Entonces el método getTabla queda así:

```
public override DataTable getTabla()
{
    this.adapater = new MySqlDataAdapter("select * from contactos",
                                         this.connectionString);
    DataTable contactos = new DataTable();
    this.adapater.Fill(contactos);
    return contactos;
}
```

Luego de ejecutarse el constructor de la clase Contactos continúa con la ejecución del resto del constructor.

Allí establecemos los comandos para agregar nuevas filas a la tabla de la base de datos, otro para modificar las filas y finalmente uno para borrarlos.

Además creamos los parámetros de cada uno de ellos. En este caso, para crear un parámetro para el DataAdapter indicamos el nombre del parámetro, el tipo de datos, el tamaño (en los tipos enteros, fecha, hora y booleanos no importa el tamaño pero el parámetro debe completarse con un valor cualquiera) y de cual columna obtendrá los valores para usar en el comando sql que lo utiliza.

Antes para cada una de las filas que debían agregarse, modificarse o eliminarse asignábamos manualmente el valor de la celda de la fila a cada parámetro según correspondía. Ahora el DataAdapter hace esto por nosotros pero debemos indicarle de cual celda de la fila obtiene el valor cada uno de los parámetros, eso es lo que hacemos con el último parámetro.

35) Luego sobrescribimos el aplicaCambios de la siguiente manera:

```
public override void aplicaCambios()
{
    using (MySqlConnection Conn = new MySqlConnection(this.connectionString))
    {
        this.adapater.InsertCommand.Connection = Conn;
        this.adapater.UpdateCommand.Connection = Conn;
        this.adapater.DeleteCommand.Connection = Conn;
        this.adapater.Update(this.misContactos);
    }
}
```

El DataAdapter se encarga de abrir la conexión y para cada fila que sea nueva o tenga cambios, decide cual comando ejecutar. Si la fila es nueva utilizará el InsertCommand, si fue modificada utilizará el UpdateComman

y si fue eliminada utilizará el DeleteCommand. Luego asignará los valores a cada parámetro con los valores de las celdas y lo ejecutará.

Todo esto lo realiza en el método Update.

36) Para terminar modificamos el Main de la siguiente forma

```
using System;
using System.Collections.Generic;
using System.Text;


namespace Lab02
{
    class Program
    {

        static void Main(string[] args)
        {
            ManejadorArchivo manejadorArch;
            Console.WriteLine("Elija el modo:");
            Console.WriteLine("1 - TXT");
            Console.WriteLine("2 - XML");
            if (Console.ReadLine() == "2")
            {
                manejadorArch=new ManejadorArchivoXml();
            }
            else
            {
                manejadorArch = new ManejadorArchivoTxt();
            }
            manejadorArch.listar();
            menu(manejadorArch);
        }

        static void menu(ManejadorArchivo manejadorArch)
        {
            string rta="";
            do
            {
                Console.WriteLine("1 - Listar");
                Console.WriteLine("2 - Agregar");
                Console.WriteLine("3 - Modificar");
                Console.WriteLine("4 - Eliminar");
                Console.WriteLine("5 - Guardar Cambios");
                Console.WriteLine("6 - Salir");
                rta = Console.ReadLine();
                switch (rta)
                {
                    case "1":
                        manejadorArch.listar();
                        break;
                    case "2":
                        manejadorArch.nuevaFila();
                        break;
                    case "3":
                        manejadorArch.editarFila();
                        break;
                    case "4":
                        manejadorArch.eliminarFila();
                        break;
                    case "5":
                        manejadorArch.aplicaCambios();
                        break;
                    default:
                        break;
                }
            } while (rta != "6");
        }
    }
}
```

Aclaraciones:

El manejador de archivos XML no realiza modificaciones en el archivo original. Simplemente lo borra y lo crea nuevamente. Hay muchas formas de manipular archivos XML, aquí se utilizó una pero también es muy popular utilizar el objeto XMLDocument o XMLDataDocument pero la manipulación del mismo no se realiza a través de una tabla sino de un árbol con nodos. El beneficio de no manipular el archivo XML como una tabla es que permite la flexibilidad de manipular un XML que no puede representarse con una estructura fija relacional (filas y columnas).

El manejador de archivos de texto sólo permitirá listar y aplicar los cambios de las filas agregadas. Desafortunadamente el conector OleDB permite realizar selecciones y agregar filas al final pero no permite realizar modificaciones ni eliminaciones porque para eso el conector OleDB necesita utilizar un índice, algo que los archivos planos no tienen.

Sin embargo aquí lo se agregó a este laboratorio ya que esa es una forma de aplicar las modificaciones y eliminaciones cuando se realizan los cambios en una tabla de la base de datos.

37) Presionamos F5 para ejecutar.

Se recomienda poner un punto de interrupción dentro del método getTabla del ManejadorArchivoTxt. Avanzar paso a paso (F10) hasta luego de ejecuta la sentencia contactos.Load(reader);
Posicionar el mouse sobre la variable contactos y aparecerá un menú flotante con una pequeña lupa. Haga Clic sobre ella para visualizar la estructura y datos del DataTable.