

# Programación con C# .NET

## Tema 9: Acceso a Bases de Datos



## Contenidos

- ❖ Introducción - Bases de Datos
- ❖ Lenguaje SQL
- ❖ ADO .NET y sus componentes
- ❖ Modos de Conexión
- ❖ Utilizando los componentes de .NET
- ❖ Alternativa al asistente de configuración del DataAdapter
- ❖ Utilizando las clases de .NET
- ❖ Utilizando clases generadas por el asistente
- ❖ Control BindingSource
- ❖ Control BindingNavigator
- ❖ LINQ to DataSet
- ❖ LINQ to SQL



## Introducción – Base de Datos

- Es una colección de datos clasificados y estructurados que son guardados en uno o varios ficheros pero referenciados como si de un único fichero se tratara.
- Existen en el mercado varios sistemas administradores de bases de datos. Ej: **Access**, **SQL Server**, **Oracle**, **DB2**, otros de libre distribución como **MySQL** y **PostgreSQL**.
- Los datos de una base de datos relacional se almacenan en tablas lógicamente relacionadas entre sí utilizando campos clave comunes.
- Cada tabla dispone los datos en filas y columnas.
- Una tabla es una colección de datos presentada en forma de una matriz bidimensional, donde las filas reciben el nombre de **tuplas** o **registros** y las columnas **campos**.
- Los usuarios de un sistema administrador de bases de datos pueden realizar operaciones como insertar, recuperar, modificar y eliminar datos de la base de datos, así como añadir nuevas tablas o eliminarlas.
- Todas estas operaciones se expresan utilizando el **lenguaje SQL**.



## SQL

- Es el lenguaje estándar para interactuar con bases de datos relacionales y es soportado por todos los sistemas administradores de bases de datos actuales.
- SQL incluye operaciones tanto de definición, por ejemplo **CREATE**, como de manipulación de datos, por ejemplo, **INSERT**, **UPDATE**, **DELETE** y **SELECT**.
- **Crear y eliminar una base de datos**
- Para crear una base de datos, SQL proporciona la sentencia **CREATE DATABASE** cuya sintaxis es:

**CREATE DATABASE** <base de datos>

- Para eliminar la base de datos se usa la orden:

**DROP DATABASE** <base de datos>



## Crear una tabla

- Para crear una tabla, SQL proporciona la sentencia CREATE TABLE.
- Esta sentencia especifica: el nombre de la tabla, los nombres y tipos de las columnas de la tabla y las claves primaria y externa de esa tabla (también llamada llave extranjera, en el sentido de que es importada de otra tabla). Su sintaxis es la siguiente:

CREATE TABLE <tabla> (<columna 1> [, <columna 2>] ... )

- donde *columna n* se formula según la sintaxis siguiente:

<columna n> <tipo de dato> [DEFAULT <expresion>]  
[<constante 1> [<constante 2>] ... ]

- Algunos de los tipos de datos mas utilizados son los siguientes:



## Tipos de Datos

Tipo SQL	Tipo SQL de .NET Framework
INTEGER	SqlInt32
REAL	SqlSingle
FLOAT	SqlDouble
CHAR	SqlString
VARCHAR	SqlString
BINARY	SqlBinary
DATE	SqlDateTime

- La cláusula DEFAULT permite especificar un valor por omisión para la columna y, opcionalmente, para indicar la forma o característica de cada columna, se pueden utilizar las constantes: NOT NULL (no se permiten valores nulos), NULL, UNIQUE o PRIMARY KEY.
- La cláusula PRIMARY KEY se utiliza para definir la columna como clave principal de la tabla. No puede contener valores nulos ni duplicados.
- Una tabla puede contener varias restricciones UNIQUE pero solo una restricción PRIMARY KEY.
- La cláusula UNIQUE indica que la columna no permite valores duplicados.



## Ejemplo

```
CREATE TABLE telefonos (  
    nombre          CHAR(30) NOT NULL,  
    direccion CHAR(30) NOT NULL,  
    telefono        CHAR(12) PRIMARY KEY NOT NULL,  
    observaciones   CHAR(240)  
)
```

- La diferencia entre los tipos CHAR (n) y VARCHAR (n) es que en el primer caso, el campo se rellena con espacios hasta n caracteres (longitud fija) y en el segundo no (longitud variable).
- **Escribir datos en la tabla:** Para escribir datos en una tabla, SQL proporciona la sentencia INSERT. Esta sentencia agrega una o mas filas nuevas a una tabla. Su sintaxis, de forma simplificada, es la siguiente:

```
INSERT [INTO] <tabla> [( <columna 1> [, <columna 2>]... )]  
VALUES (<expresión 1> [, <expresión 2>] ... ), ...
```



## Ejemplo

```
INSERT INTO telefonos VALUES  
(‘Leticia Aguirre’, ‘Managua’, ‘79145’, ‘Ninguna’)
```

- **Modificar datos de una tabla:** Para modificar datos en una tabla, SQL proporciona la sentencia UPDATE. Esta sentencia puede cambiar los valores de filas individuales, grupos de filas o todas las filas de una tabla. Sintaxis:

```
UPDATE <tabla> SET <columna 1 = (<expresión 1> | NULL)  
[, <columna 2 = (<expresión 2> | NULL)] ...  
WHERE <condición de búsqueda>
```

- Ejemplo:

```
UPDATE telefonos SET direccion = “Zaragoza, Leon”  
WHERE telefono = ‘8547854’
```





## Borrar y seleccionar registros de una tabla

- Para borrar registros en una tabla, SQL proporciona la sentencia DELETE. Esta sentencia quita una o varias filas de una tabla. Sintaxis simplificada:

```
DELETE FROM <tabla> WHERE <condición de búsqueda>
```

- Ejemplo:

```
DELETE FROM telefonos WHERE telefono = "589632"
```

- **Seleccionar datos de una tabla:** Para seleccionar datos de una tabla, SQL proporciona la sentencia SELECT. Las cláusulas principales de esta sentencia se pueden resumir del modo siguiente:

```
SELECT [ALL | DISTINCT] <lista de seleccion>  
FROM <tablas>  
WHERE <condiciones de seleccion>  
[ORDER BY <columna 1> [ASC|DESC] [, <columna 2> [ASC|DESC] ] ... ]
```

- Las cláusulas de una instrucción SELECT deben especificarse en el orden indicado.



## Ejemplos

- `SELECT * FROM telefonos`
- `SELECT * FROM telefonos ORDER BY nombre`
- `SELECT * FROM telefonos WHERE telefono > "896500"`
- `SELECT * FROM telefonos WHERE telefono LIKE '91*'`



## Crear una base de datos

- La operación de crear una base de datos puede variar cuando se realiza desde el entorno de desarrollo aportado por el sistema administrador de bases de datos.
- Pero no varía si la creamos desde la línea de órdenes utilizando el lenguaje SQL.
- **Base de datos de Microsoft SQL Server:** Es muy sencillo crearla si tiene instalado el administrador corporativo que incorpora el paquete de SQL Server. Si no lo tiene, se puede crear con la utilidad SQLCMD, ejecutable desde la línea de órdenes.
- Los pasos son los siguientes:
  - Ejecute la orden Inicio -> Ejecutar -> cmd para abrir la consola del sistema.
  - Localice el fichero SQLCMD.EXE (o bien el fichero SQLCMD90.EXE), cambie a ese directorio y ejecute la orden:  
  
`SQLCMD -S nombre-del-ordenador\SqIExpress`
  - Ejecute el conjunto de sentencias SQL correspondientes para crear la base de datos. Este conjunto de sentencias SQL las puede almacenar en un fichero de texto (script).



## ADO .NET

- ADO.NET es una tecnología de acceso a datos.
- No depende de conexiones continuamente activas lo que permite dar servicio a muchos más usuarios.
- Las interacciones con la base de datos se realizan mediante órdenes para acceso a los datos, que son objetos que encapsulan las sentencias SQL o los procedimientos almacenados que definen la operación a realizar sobre el origen de datos.
- Los datos requeridos se almacenan en memoria caché en conjuntos de datos, lo que permite trabajar sin conexión sobre una copia temporal de los datos obtenidos.
- El formato de transferencia de datos es XML (no utiliza información binaria, se basa en texto). Esto permite enviar la información mediante cualquier protocolo, por ejemplo, HTTP.



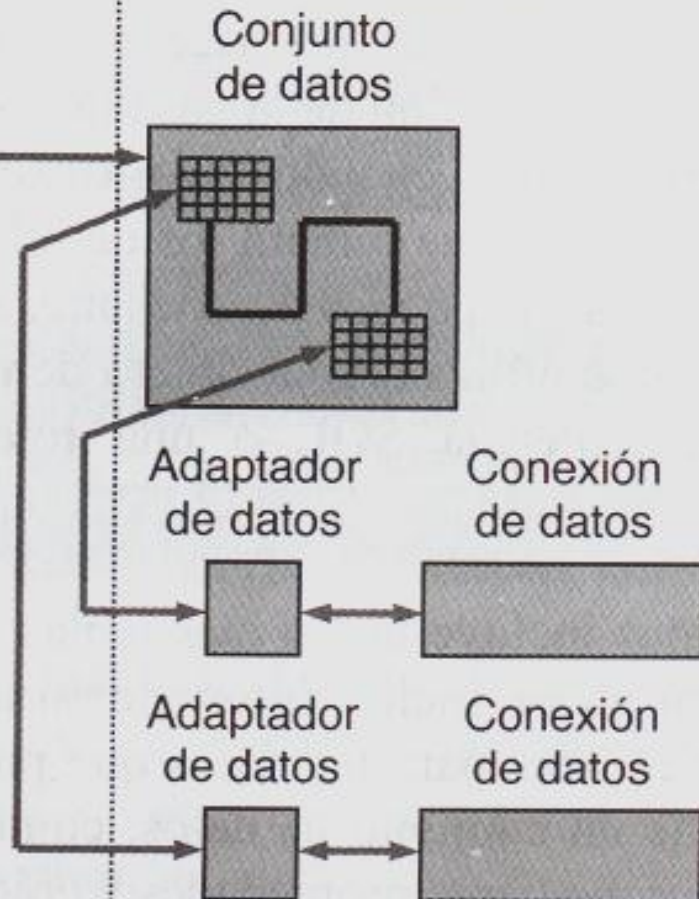
## Componentes de ADO .NET

- ADO.NET es un conjunto de clases, pertenecientes al espacio **System.Data**, para acceso a los datos de un origen de datos.
- Los componentes están diseñados para separar el acceso a los datos de la manipulación de los mismos y son: **DataSet** y el **proveedor de datos de .NET Framework**.
- El proveedor de datos es un conjunto de componentes entre los que se incluyen los objetos conexión (**Connection**), de órdenes (**Command**), lector de datos (**DataReader**) y adaptador de datos (**DataAdapter**).
- La figura siguiente muestra cómo trabajan conjuntamente los objetos mencionados entre sí, para que una aplicación pueda interactuar con un origen de datos:

## Capa de presentación



## Capa de la lógica de negocio



## Capa de datos

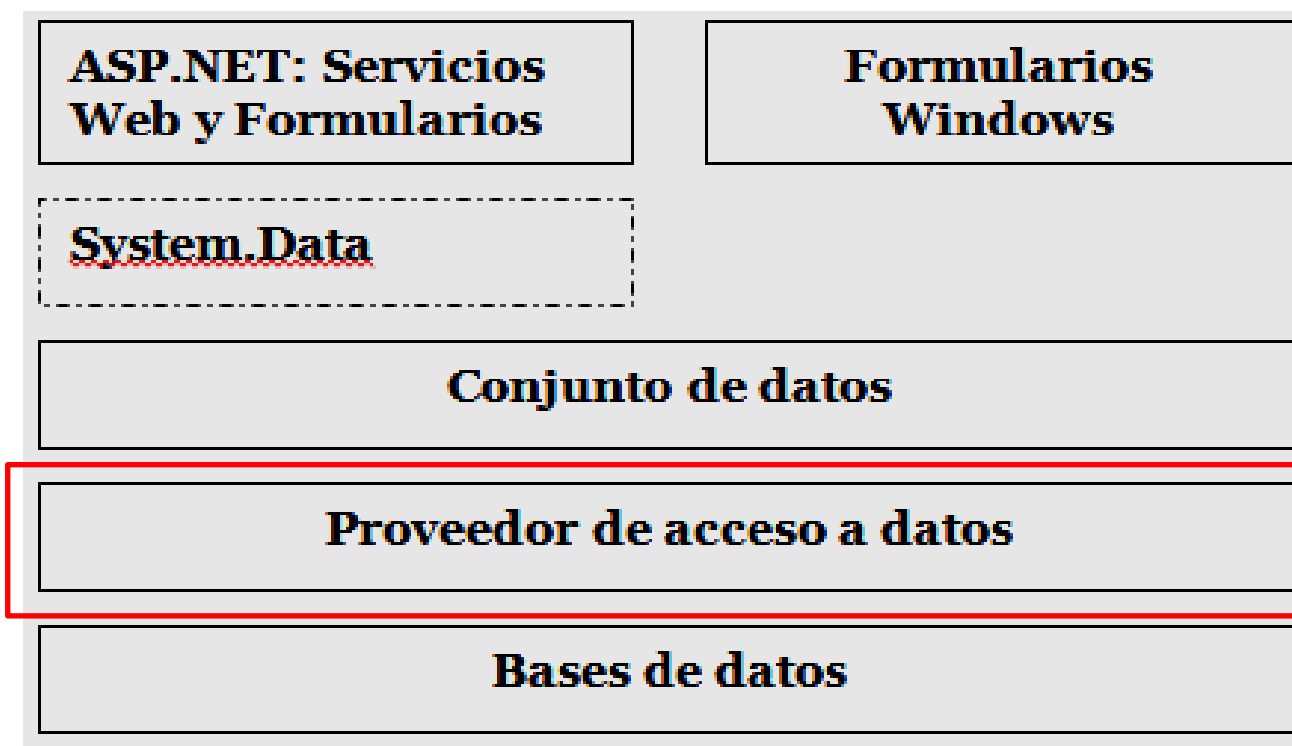
Origen de datos





## Conexión y ejecución de sentencias

- El trabajo de conexión con la base de datos, o la ejecución de una sentencia SQL determinada, la realiza el proveedor de acceso a datos.





## Conjunto de datos

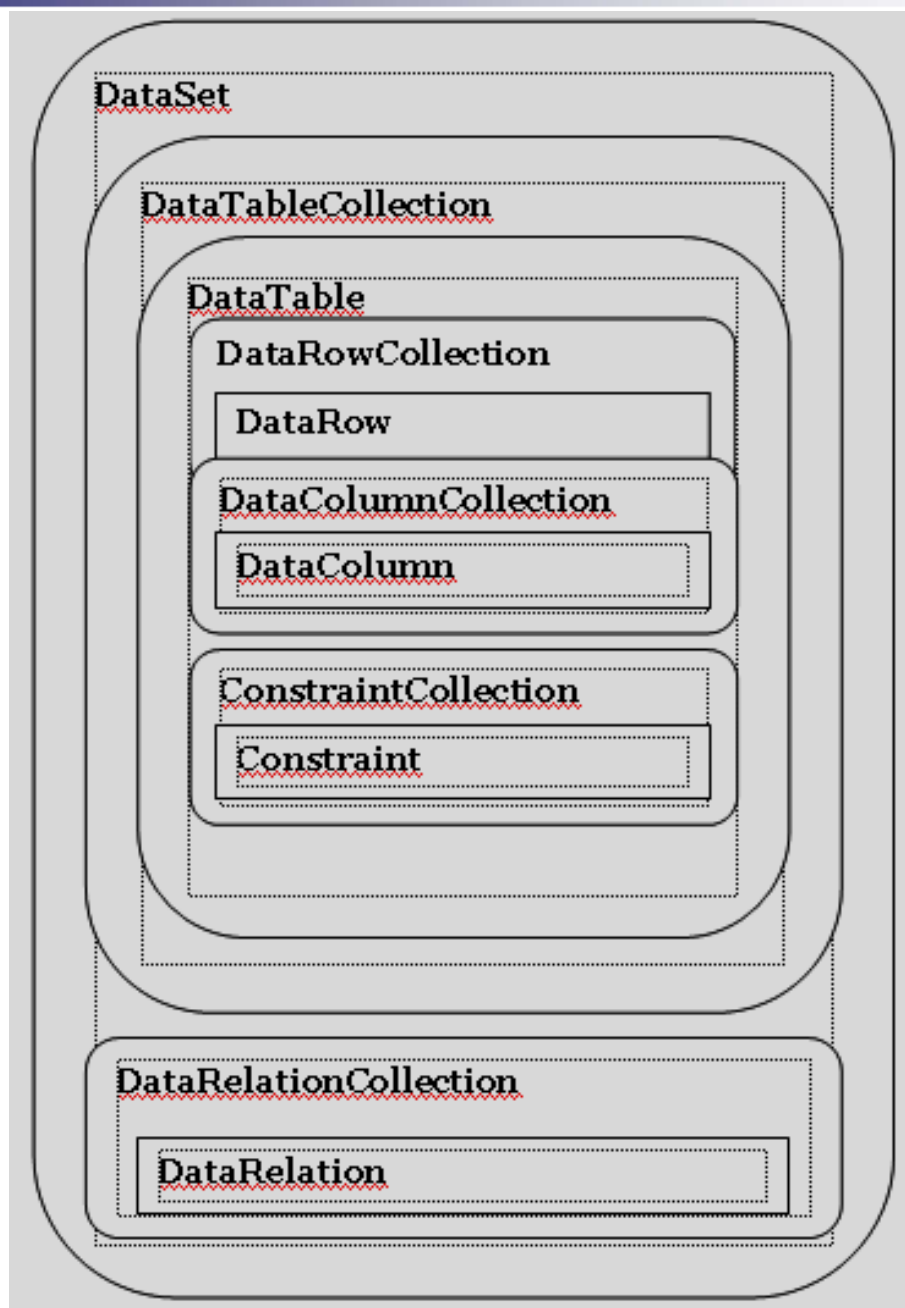
- Un formulario Windows utiliza un adaptador de datos para leer la información de la base de datos y la almacena en un conjunto de datos.
- Cuando quiera escribir en el origen de datos, volverá a utilizar el adaptador el que tomará los datos del conjunto de datos.
- Un conjunto de datos incluye una o más tablas basadas en las tablas del origen de datos y también puede incluir información acerca de las relaciones entre estas tablas y las restricciones para los datos que pueden contener las tablas.
- El conjunto de datos está representado por la clase **DataSet** del espacio **System.Data**.





## Estructura de un conjunto de datos

- Estructura de la clase **DataSet**:
  - Colección **DataTableCollection** de objetos **DataTable**.
  - Colección **DataRelationCollection** de objetos **DataRelation**.
- Estructura de la clase **DataTable**:
  - Colección **DataRowCollection** de objetos **DataRow**.
  - Colección  **DataColumnCollection** de objetos **DataColumn**.
  - Colección **ConstraintCollection** de objetos **Constraint**.

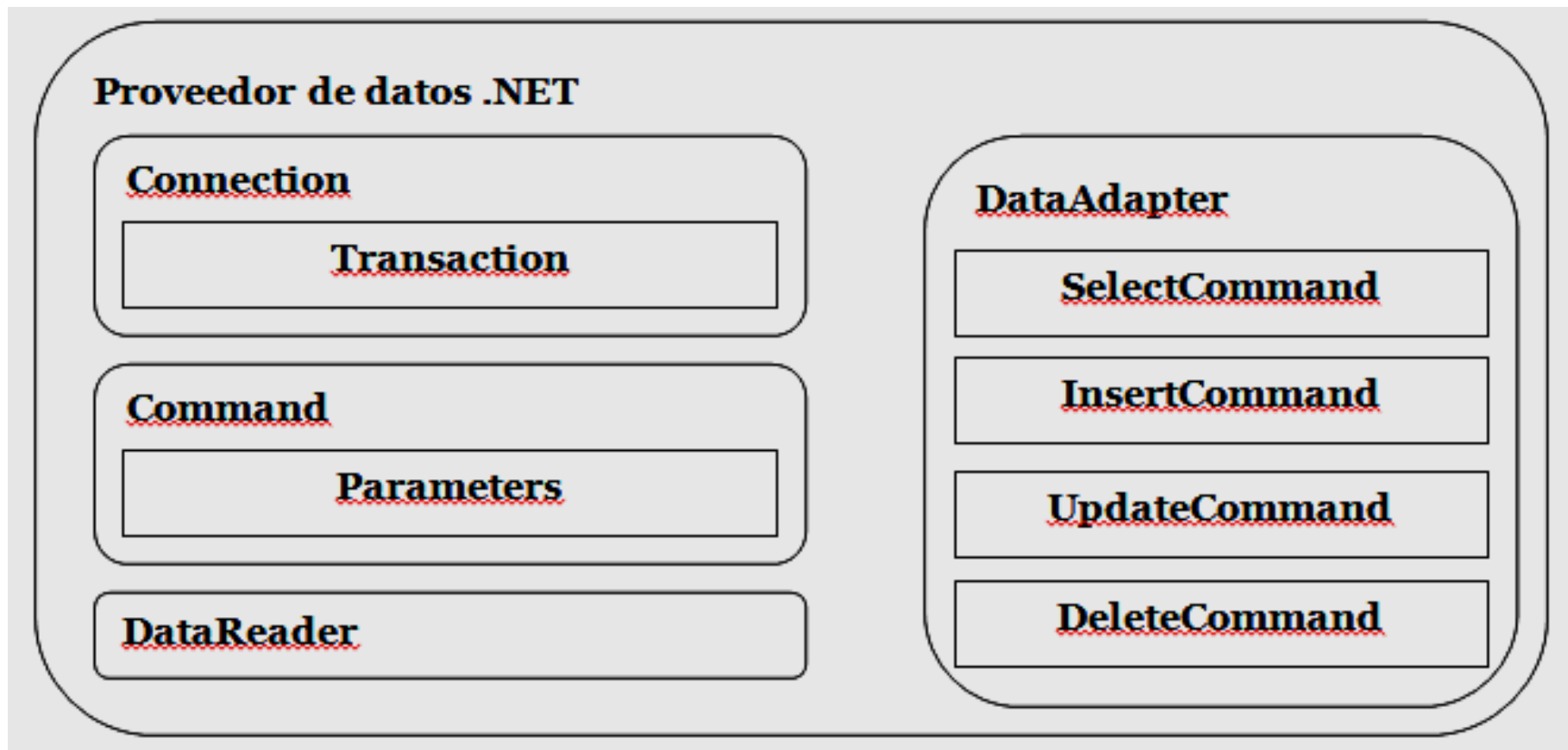




## Proveedor de datos

- Sirve como puente entre una aplicación y un origen de datos.
- Se utiliza tanto para recuperar datos de un origen como para actualizarlos.
- Componentes principales:
  - Conexión con el origen de datos (objeto **Connection**). Establece una conexión a un origen de datos determinado.
  - Orden para acceso a los datos (objeto **Command**). Ejecuta una orden en un origen de datos.
  - Lector de datos (objeto **DataReader**). Lee una secuencia de datos se sólo avance y sólo lectura desde un origen de datos.
  - Adaptador de datos (objeto **DataAdapter**). Llena un **DataSet** y realiza las actualizaciones necesarias en el origen de datos.

## Proveedor de datos





## Proveedor de datos

- .NET incluye los siguientes proveedores de datos:

Proveedor	Espacio de nombre
ODBC	System.Data.Odbc
OLE DB	System.Data.OleDb
Oracle	System.Data.OracleClient
SQL Server	System.Data.SqlClient

- Cada uno de estos proveedores proporciona los drivers adecuados para acceder a las distintas bases de datos. Microsoft proporciona los proveedores de acceso a datos más corrientes.
- Pero hay otros de igual importancia como MySQL. En este caso, debemos acudir al fabricante para que nos proporcione el conjunto de clases que definen ese proveedor en particular.
- Por cuestiones de rendimiento, es aconsejable utilizar un proveedor de acceso a datos nativo.



## Objeto Conexión

- Para establecer una conexión a un origen de datos, ADO.NET proporciona el objeto **Connection**.
- Su función es presentar atributos y métodos para permitir establecer y modificar las propiedades de la conexión, por ejemplo, el identificador de usuario y la contraseña, entre otras.

Objeto	Descripción
SqlConnection	Permite establecer una conexión a un origen de datos Microsoft SQL Server
OleDbConnection	Permite establecer una conexión a un origen de datos OLE DB
OdbcConnection	Permite establecer una conexión a un origen de datos ODBC
OracleConnection	Permite establecer una conexión a un origen de datos Oracle

- Ejemplo:  
`OleDbConnection conexion = new OleDbConnection(strConexion);`

## Objeto Orden

- Después de establecer una conexión con un origen de datos, puede utilizar un objeto **Command** para ejecutar sentencias SQL y devolver resultados desde ese origen de datos.
- Después de crear el objeto, la instrucción SQL puede ser consultada o modificada a través de su propiedad **CommandText**.

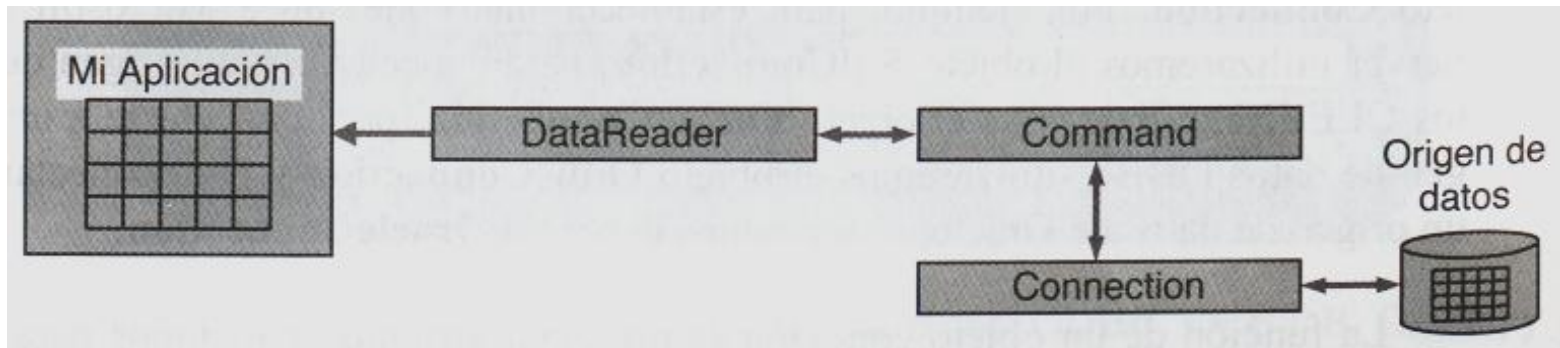
Objeto	Descripción
SqlCommand	Para a un origen de datos Microsoft SQL Server
OleDbCommand	Para orígenes de datos compatibles con OLE DB
OdbcCommand	Para orígenes de datos compatibles con ODBC
OracleCommand	Para orígenes de datos compatibles con Oracle

- Ejemplo:

```
OleDbCommand ordenSQL = new OleDbCommand("SELECT nombre, telefono  
FROM telefono ", conexion);
```

## Objeto lector de datos

- Cuando una aplicación sólo necesite leer datos (no actualizarlos), no será necesario almacenarlos en un conjunto de datos, basta utilizar un objeto **lector de datos** en su lugar.
- Un objeto lector de datos obtiene los datos del origen de datos y los pasa directamente a la aplicación (un adaptador de datos utiliza un objeto lector de datos para llenar un conjunto de datos).
- El objeto lector de datos proporcionado por .NET para SQL Server es **SqlDataReader** y para orígenes OLE-DB es **OleDbDataReader**. La figura siguiente muestra cómo se utilizan estos objetos:







## Objeto lector de datos (ejemplo)

```
conexion.Open();  
OleDbDataReader lector = ordenSQL.ExecuteReader();  
  
while(lector.Read())  
    Console.WriteLine(lector.GetString(0) + “, ” +  
        lector.GetString(1));  
lector.close();
```



## Adaptador de Datos

- Un adaptador es un conjunto de objetos utilizado para intercambiar datos entre un origen de datos y un conjunto de datos (objeto **DataSet**).
- Esto significa que una aplicación leerá datos de una base de datos para un conjunto de datos y, a continuación, manipulará dichos datos.
- También, en algunas ocasiones, volverá a escribir en la base de datos los datos modificados del conjunto de datos.

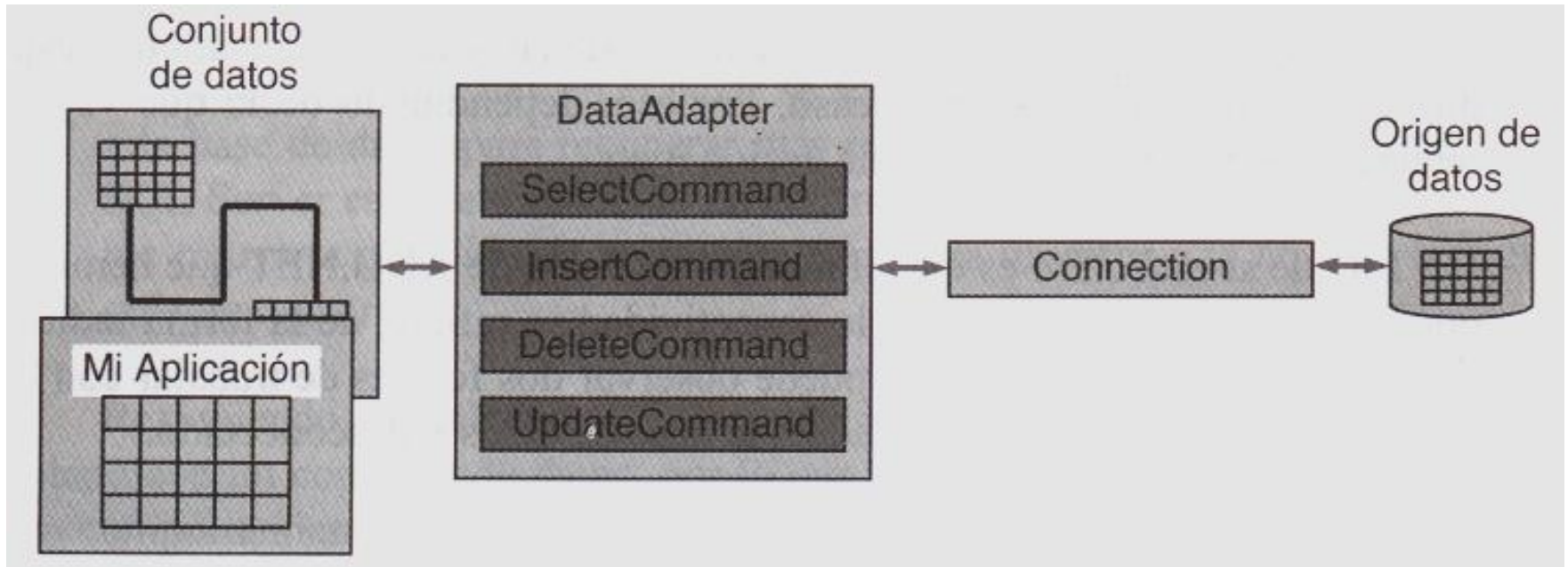
Objeto Adaptador	Objeto Orden	Objeto Conexión
OleDbAdapter	OleDbCommand	OleDbConnection
OdbcDataAdapter	OdbcCommand	OdbcConnection
SqlDataAdapter	SqlCommand	SqlConnection
OracleDataAdapter	OracleCommand	OracleConnection



## Adaptador de Datos

- Generalmente, cada adaptador de datos intercambia datos entre una sola tabla de un origen de datos y un solo objeto **DataTable** (tabla de datos) del conjunto de datos.
- Esto quiere decir que lo normal es utilizar tantos adaptadores como tablas tenga el conjunto de datos.
- De esta forma, cada tabla del conjunto de datos tendrá su correspondiente tabla en el origen de datos.
- En la siguiente figura se puede observar la utilización de un adaptador de datos para llenar un conjunto de datos (objeto **DataSet**):

## Adaptador de Datos





## Propiedades de un Adaptador de Datos

- Según se observa en la figura anterior, un adaptador contiene también las propiedades **SelectCommand**, **InsertCommand**, **DeleteCommand**, **UpdateCommand** y **TableMappings** para facilitar la lectura y actualización de los datos en un origen de datos.

Propiedad	Significado
SelectCommand	Hace referencia a una orden que recupera filas del origen de datos, entendiendo por orden un objeto Command que almacena una instrucción SQL o un nombre de procedimiento almacenado.
InsertCommand	Hace referencia a una orden para insertar filas en el origen de datos.
UpdateCommand	Hace referencia a una orden para modificar filas en el origen de datos.
DeleteCommand	Hace referencia a una orden para eliminar filas del origen de datos.

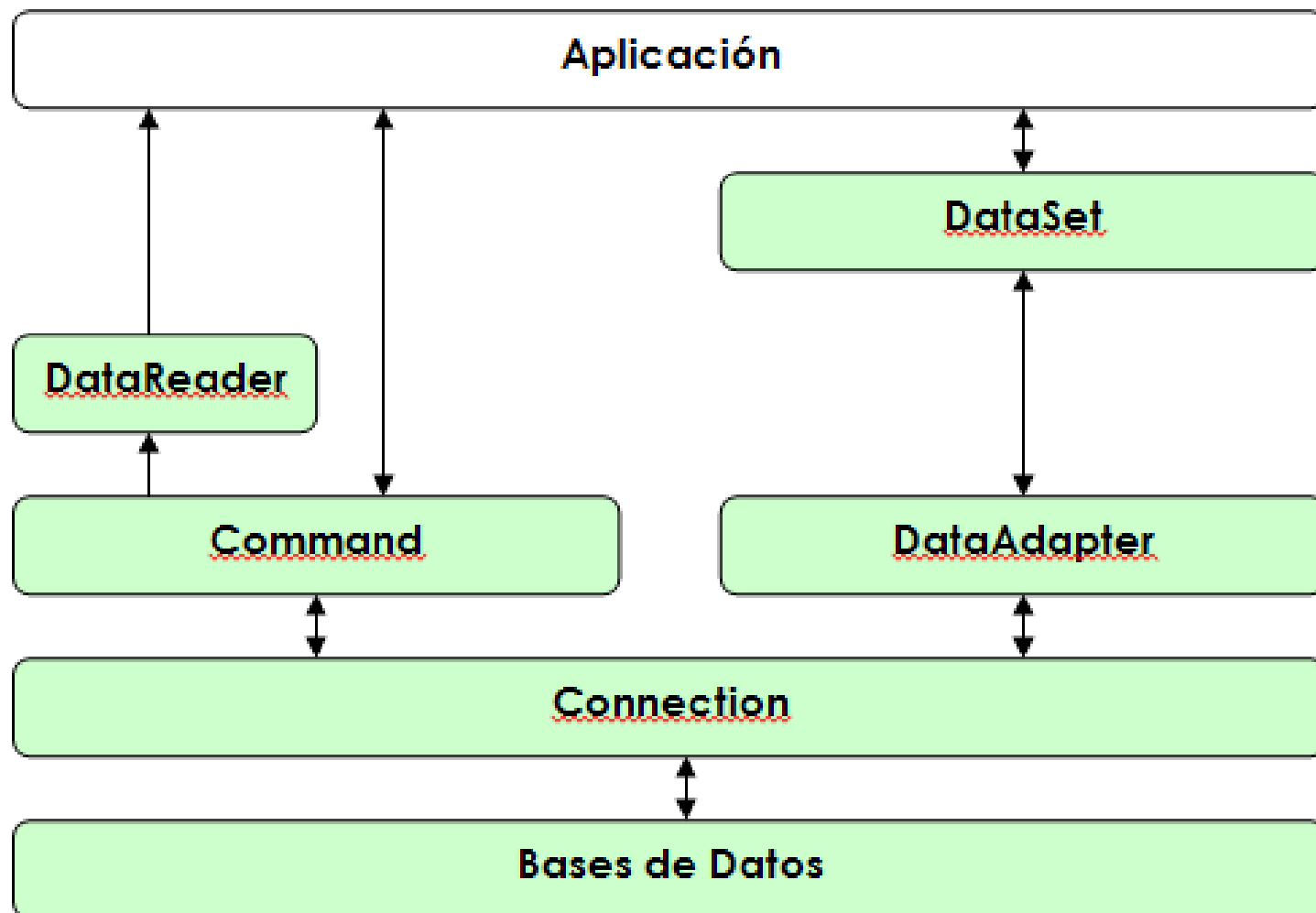


## Modos de Conexión

- La acción más pesada cuando realizamos un acceso a una base de datos, se encuentra en la conexión con la base de datos. Esa tarea tan simple, es la que más recursos del sistema consume.
- Por ello, es bueno tener presente las siguientes consideraciones:
  - La conexión debe realizarse, siempre que se pueda, con los proveedores de acceso datos nativos, simplemente porque son más rápidos que los proveedores del tipo OLE DB y ODBC.
  - La conexión debe abrirse lo más tarde posible. Es recomendable definir todas las variables que podamos antes de realizar la conexión.
  - La conexión debe cerrarse lo antes posible, siempre y cuando no tengamos la necesidad de utilizarla posteriormente.
- Los modos de conexión que existen son: **conectado** a la base de datos y **desconectado**.



## Modos de Conexión





## Modos de Conexión

- El objeto **DataSet** nos ofrece la posibilidad de almacenar datos de una determinada base de datos (tablas de una base de datos).
- Esto hace posible que una aplicación pueda trabajar con los datos procedentes de una base de datos estando desconectada de dicha base.
- En otras ocasiones será necesario trabajar estando la aplicación conectada a la base de datos.
- Esto es precisamente lo que nos ofrece el objeto **DataReader**, la posibilidad de trabajar con bases de datos conectadas.



## Acceso conectado a bases de datos

- El siguiente ejemplo, establece una conexión con el origen de datos tfnos, base de datos Access, y utilizando un lector muestra en una ventana de consola los datos *nombre* y *telefono* de cada uno de los registros de la base.

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

public class BaseDeDatos
{
    private OleDbConnection ConexionConBD;
    private OleDbCommand Orden;
    private OleDbDataReader Lector;

    public void LeerDeBaseDeDatos()
    {
        // Abrir la base de datos
        string strConexion = "Provider=Microsoft.Jet.OLEDB.4.0;" + "Data
Source=C:\\..\\..\\tfnos.mdb;";
        ConexionConBD = new OleDbConnection(strConexión);
        ConexionConBD.Open();
```

## Acceso conectado a bases de datos

```
// Crear una consulta
string Consulta = "SELECT nombre, telefono FROM telefonos";
Orden = new OleDbCommand(Consulta, ConexionConBD);

// ExecuteReader hace la consulta y devuelve un OleDbDataReader
Lector = Orden.ExecuteReader();
// Llamar siempre a Read antes de acceder a los datos
while (Lector.Read()) // siguiente registro
Console.WriteLine(Lector["nombre"] + " " + Lector["telefono"]);

// Llamar siempre a Close una vez finalizada la lectura
Lector.Close();
}

public void CerrarConexion()
{
// Cerrar la conexión cuando ya no sea necesaria
if (Lector != null)
Lector.Close();
if (ConexionConBD != null)
ConexionConBD.Close();
}
```

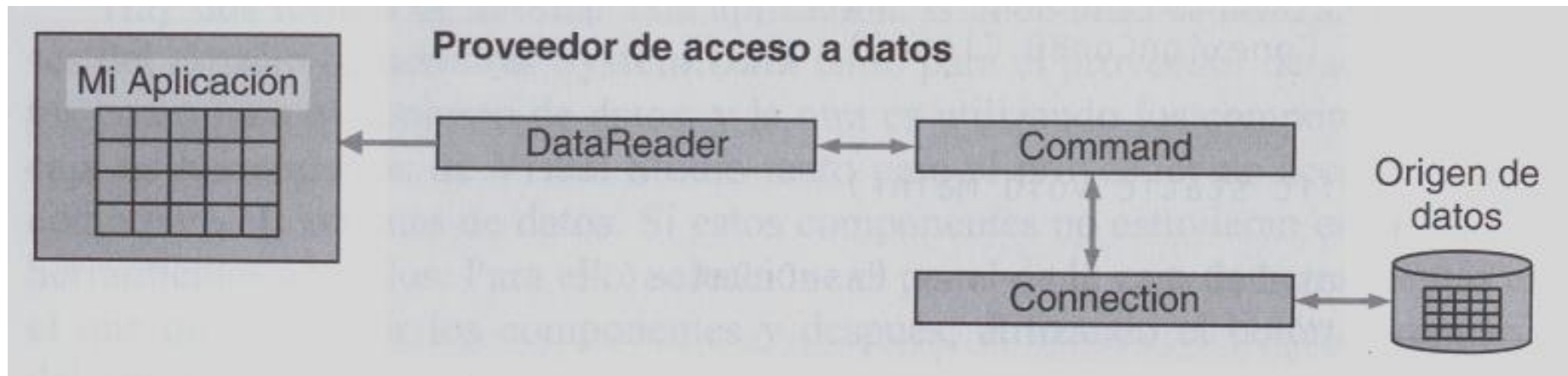


## Acceso conectado a bases de datos

```
public static void Main()
{
    BaseDeDatos bd = new BaseDeDatos();
    try
    {
        Bd.LeerDeBaseDeDatos();
    }
    catch(Exception e)
    {
        Console.WriteLine("Error: " + e.Message);
    }
    finally
    {
        bd.CerraConexion();
    }
}
```

## Acceso conectado a bases de datos

- Observe que, ocurra lo que ocurra, la conexión siempre se cierra, ya que el bloque finally, independientemente de cómo finalice la aplicación, siempre se ejecuta.
- El código anterior, desde un punto de vista gráfico, da lugar a la arquitectura siguiente:



## Acceso conectado a bases de datos

- A continuación se presenta otra versión de la aplicación anterior. Ahora el acceso se hace a la base de datos bd\_telefonos de SQL Server.

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

```
public class BaseDeDatos  
{  
    private SqlConnection ConexionConBD;  
    private SqlCommand Orden;  
    private SqlDataReader Lector;  
  
    public void LeerDeBaseDeDatos()  
    {  
        // Abrir la base de datos  
        string strConexión = "Data Source=.\sqlexpress;" + "Initial  
Catalog=bd_telefonos;Integrated Security=True";  
        ConexionConBD = new SqlConnection(strConexión);  
        ConexionConBD.Open();  
    }  
}
```



## Acceso conectado a bases de datos

```
// Crear una consulta
string Consulta = "SELECT nombre, telefono FROM telefonos";
Orden = new SqlCommand(Consulta, ConexionConBD);
// ExecuteReader hace la consulta y devuelve un SqlDataReader
Lector = Orden.ExecuteReader();
// Llamar siempre a Read antes de acceder a los datos
while (Lector.Read()) // siguiente registro
{
    Console.WriteLine(Lector["nombre"] + " " + Lector["telefono"]);
}
// Llamar siempre a Close una vez finalizada la lectura
Lector.Close();
}

public void CerrarConexion()
{
    // Cerrar la conexión cuando ya no sea necesaria
    if (Lector != null)
        Lector.Close();
    if (ConexionConBD != null)
        ConexionConBD.Close();
}
```



## Acceso conectado a bases de datos

```
public static void Main()
{
    BaseDeDatos bd = new BaseDeDatos();
    try
    {
        bd.LeerDeBaseDeDatos();
    }
    catch (Exception e)
    {
        Console.WriteLine("Error: " + e.Message);
    }
    finally
    {
        bd.CerrarConexion();
    }
}
```



## Acceso desconectado a bases de datos

- Una aplicación que interaccione con una base de datos generalmente mostrará los datos en uno o más formularios, permitirá manipularlos, y finalmente, actualizará la base de datos.
- Este ejemplo presenta un formulario Windows que muestra datos en una rejilla de datos, de una sola tabla de una base de datos. La rejilla es editable, lo que permitirá realizar cambios en los datos y actualizar la base de datos.
- La base de datos que vamos a utilizar la construiremos con *Microsoft SQL Server* (para este ejemplo utilizaremos la base *bd\_telefonos.mdf* creada anteriormente).
- Utilizar otros gestores de bases de datos como Access u Oracle, no cambia el procedimiento a seguir.
- El desarrollo de esta aplicación lo vamos a dividir en los siguientes pasos:





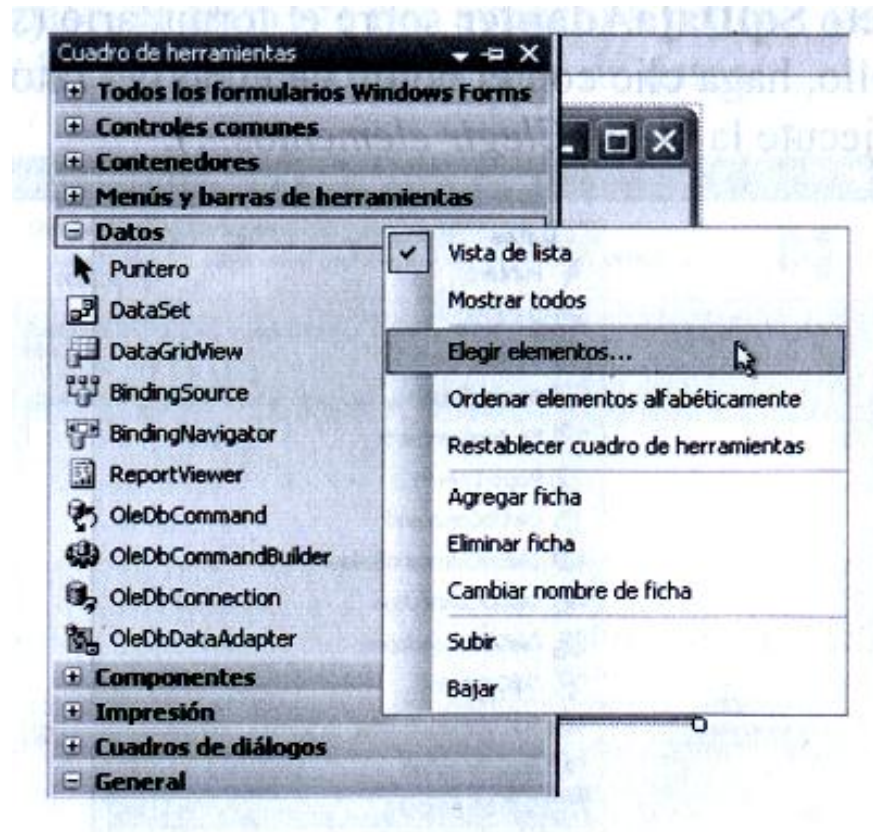
## Acceso desconectado a bases de datos

1. Crear la base de datos con Microsoft SQL Server, si aún no está creada.
2. Crear una aplicación Windows. Utilizaremos Visual Studio.
3. Establecer la conexión con el origen de datos. Esto incluye crear una consulta que permita llenar el conjunto de datos a partir de la base de datos.
4. Crear el conjunto de datos.
5. Agregar el control rejilla al formulario y enlazarlo a los datos.
6. Agregar código para llenar el conjunto de datos y código para enviar los cambios del conjunto de datos de vuelta a la base de datos.

*Hay dos formas de abordar esta aplicación. Una de ellas es utilizando las clases del espacio de nombres **System.Data** tanto para el proveedor de acceso a datos como para el conjunto de datos, y la otra es utilizando los componentes de la caja de herramientas de Visual Studio tanto para el proveedor de acceso a datos como para el conjunto de datos.*

*Si estos componentes no estuvieran en la caja de herramientas añádalos. Para ello, seleccione el panel de la caja de herramientas en el que quiere añadir los componentes y después, utilizando el botón secundario del ratón haga clic en su barra de título y ejecute la orden Elegir elementos... del menú contextual que se visualiza.*

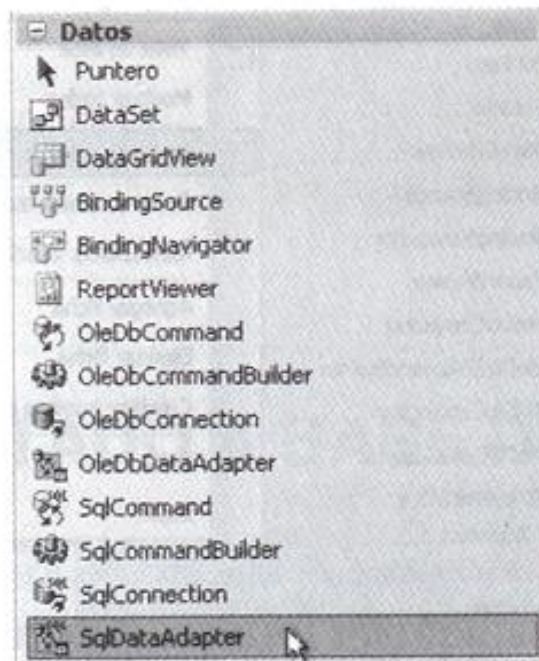
## Acceso desconectado a bases de datos



- En la ventana que se visualiza, seleccione los componentes **SqlCommand**, **SqlCommandBuilder**, **SqlConnection**, **SqlDataAdapter** y **SqlDataSource**.

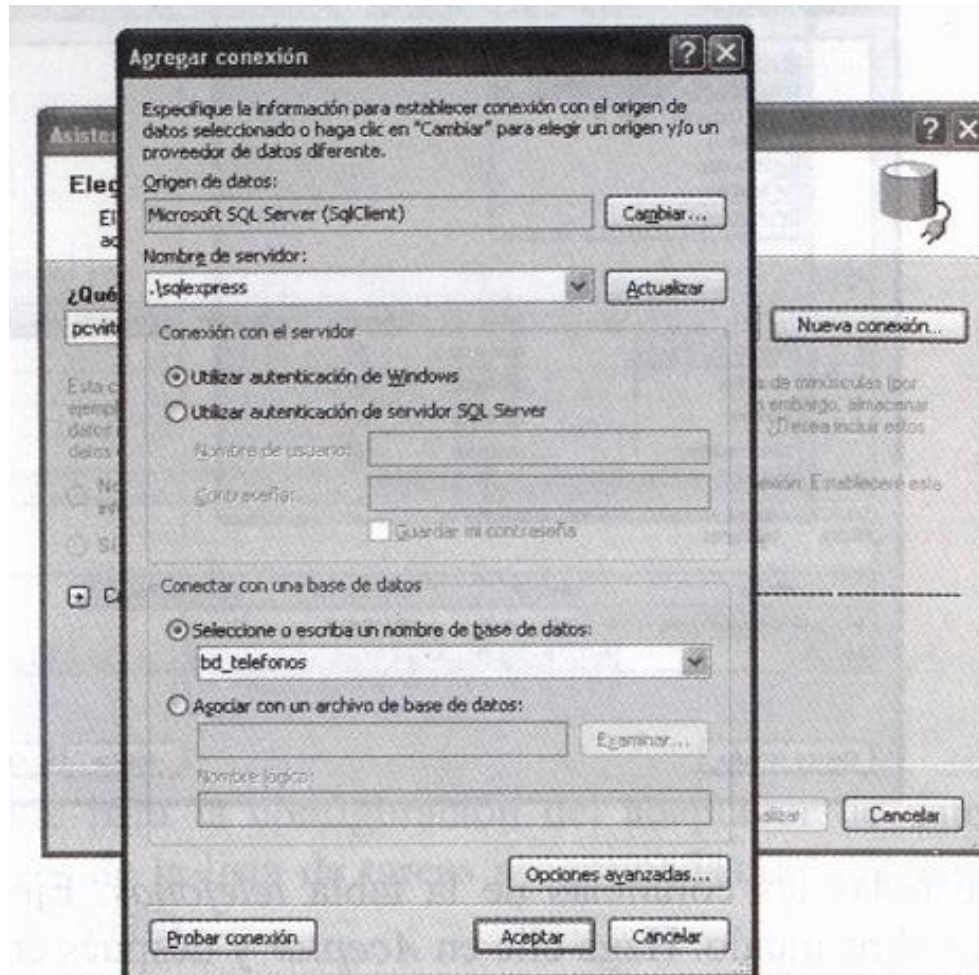
## Utilizando los componentes de .NET

- **Proveedor de datos:** Para empezar, añada a la aplicación un adaptador de datos que contenga la instrucción SQL que se utilizará más adelante para llenar el conjunto de datos que mostrará la rejilla.
- Para ello, arrastre desde la ficha *Datos* del cuadro de herramientas un objeto **SqlDataAdapter** sobre el formulario (si no aparece este control añádalo; para ello, haga clic con el botón derecho del ratón sobre el título de la ficha, *Datos*, y ejecute la orden *Elegir elementos...*).



## Utilizando los componentes de .NET

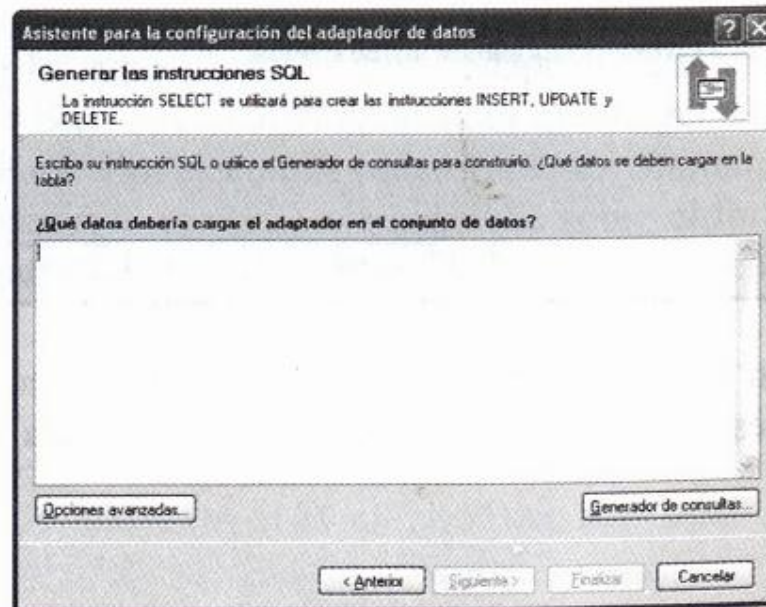
- La acción anterior arrancará el asistente para la configuración del adaptador de datos. Haga clic en el botón *Nueva conexión*:



## Utilizando los componentes de .NET

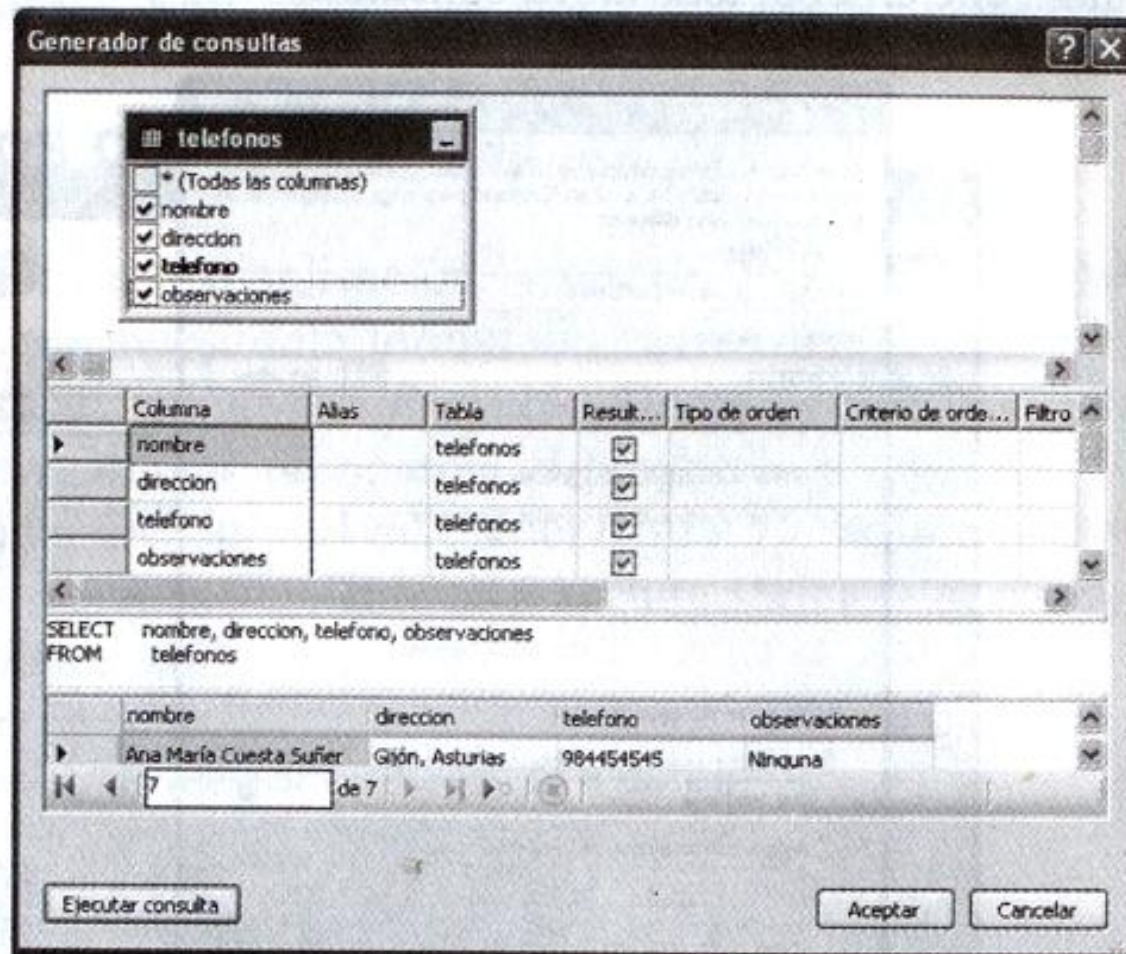
*En la ventana que se visualiza, elija como nombre de servidor .\sqlexpress y seleccione un nombre de base de datos de la lista correspondiente; en nuestro caso, bd\_telefonos.mdb; después haga clic en el botón Probar conexión y si el test resultó satisfactorio haga clic en Aceptar. Ahora, en la ventana del asistente, puede observar la nueva conexión y, si quiere, la cadena de conexión. Haga clic en Siguiente para pasar al siguiente paso: elegir el modo de acceso del adaptador a la base de datos; elija "Usar instrucciones SQL" y después haga clic en Siguiente.*

- En el siguiente paso, el asistente nos ayudará a generar las instrucciones SQL para realizar más tarde la consulta sobre la base de datos.





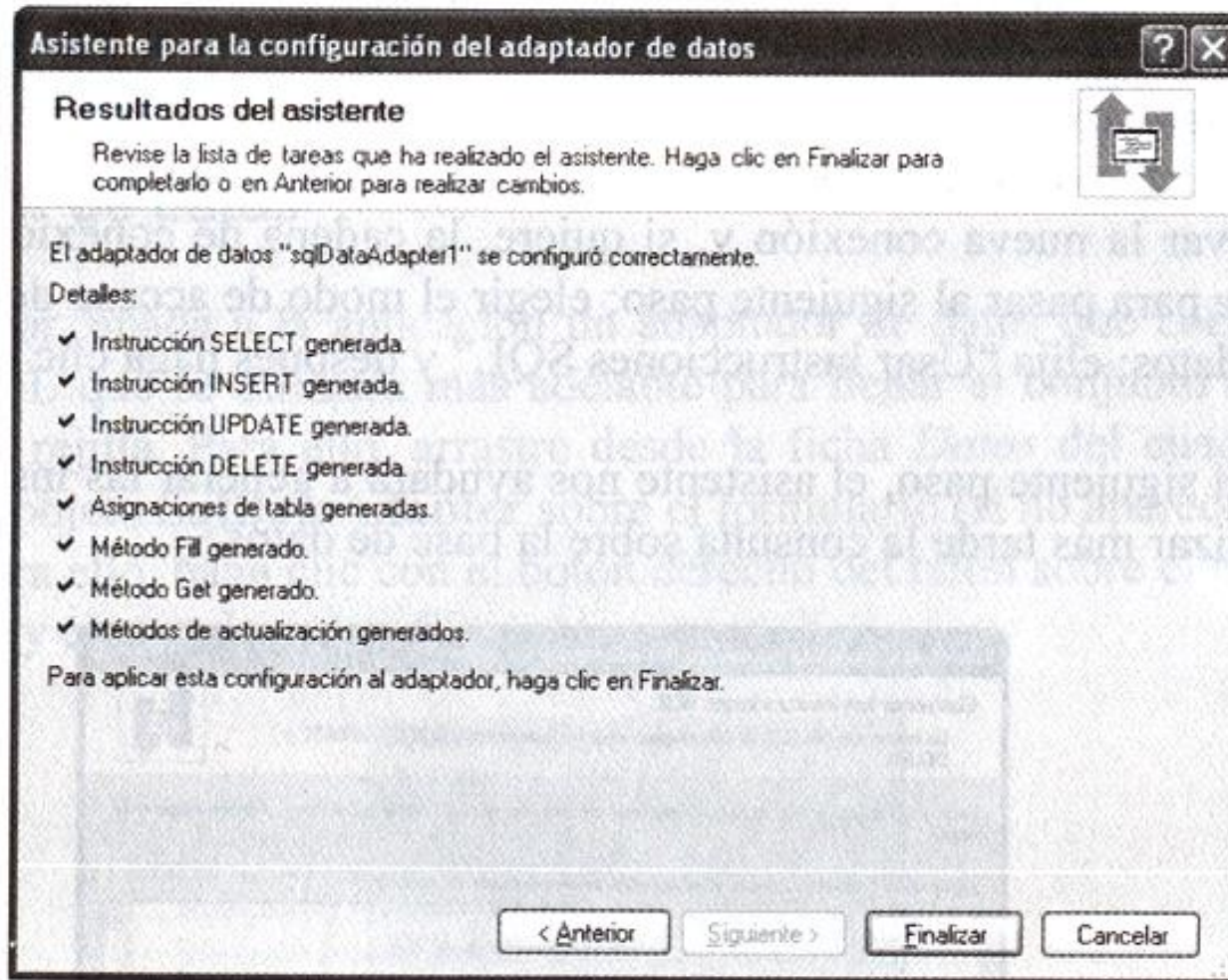
## Utilizando los componentes de .NET



- Seleccione todas las columnas de la tabla *telefonos*. Ejecute la consulta si quiere observar el resultado. Haga clic en *Aceptar* y después en *Siguiente*

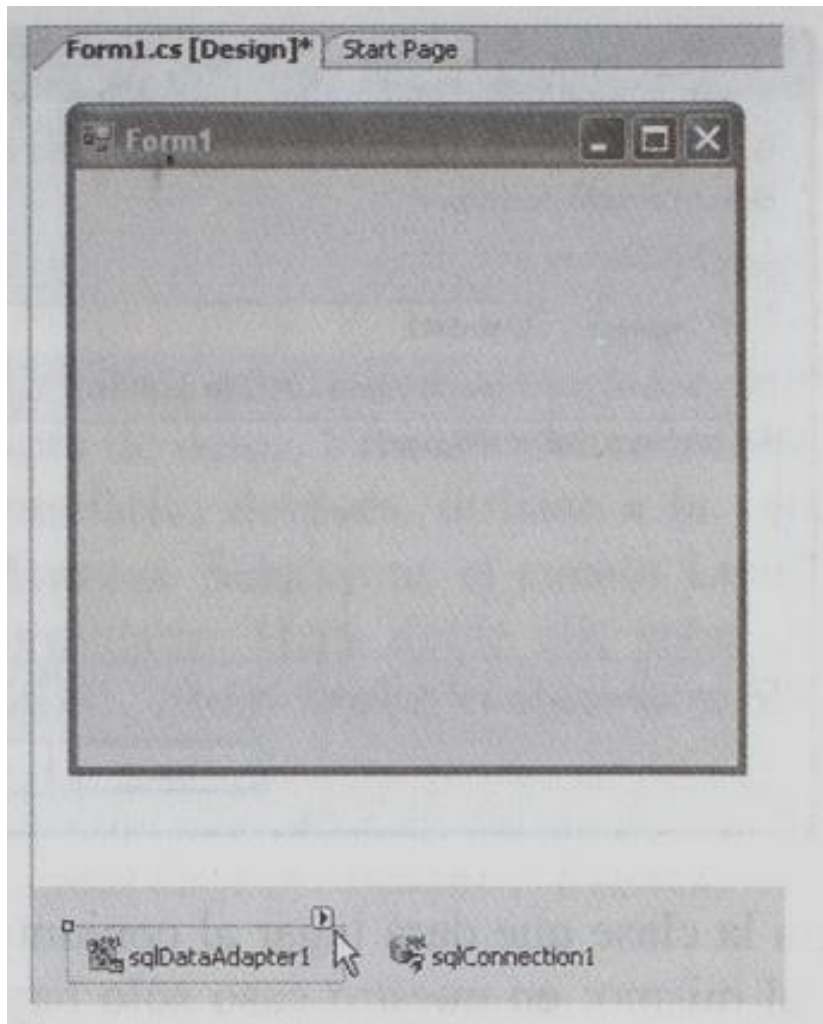
## Utilizando los componentes de .NET

- El asistente le mostrará las tareas realizadas:



## Utilizando los componentes de .NET

- Resultado hasta ahora:

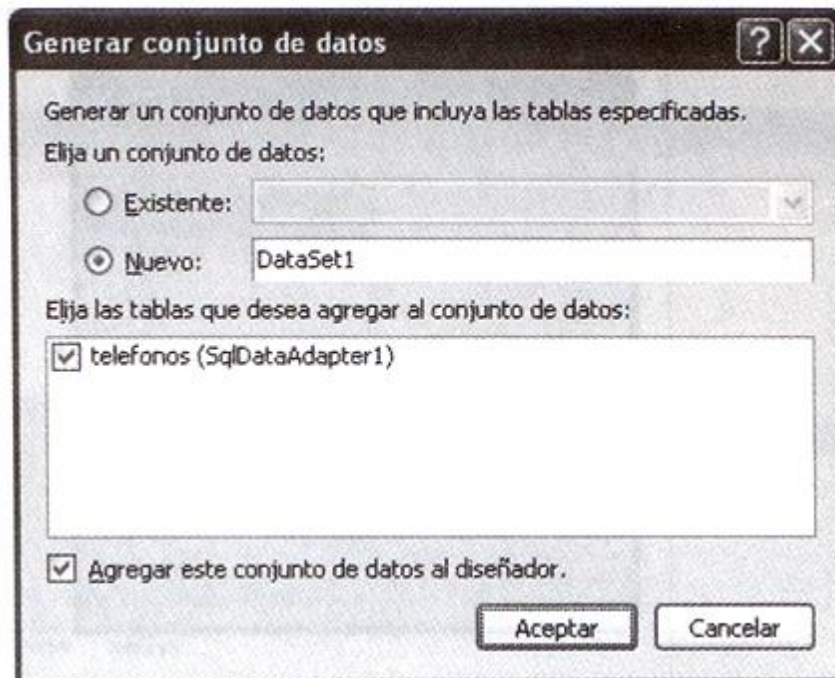


- Observe la figura; corresponde al diseñador de Visual Studio; en su parte inferior se pueden observar dos objetos: ***SqlDataAdapter1*** y ***SqlConnection1***. Esto significa que el asistente ha creado un adaptador de datos que contiene la consulta (instrucción SQL) que se utilizará para llenar el conjunto de datos y, como parte de este proceso, ha definido una conexión para obtener acceso a la base de datos.
- Por lo tanto, el proveedor de datos está construido. El paso siguiente es crear el conjunto de datos que almacenará los datos obtenidos de la base de datos por medio del proveedor de datos.



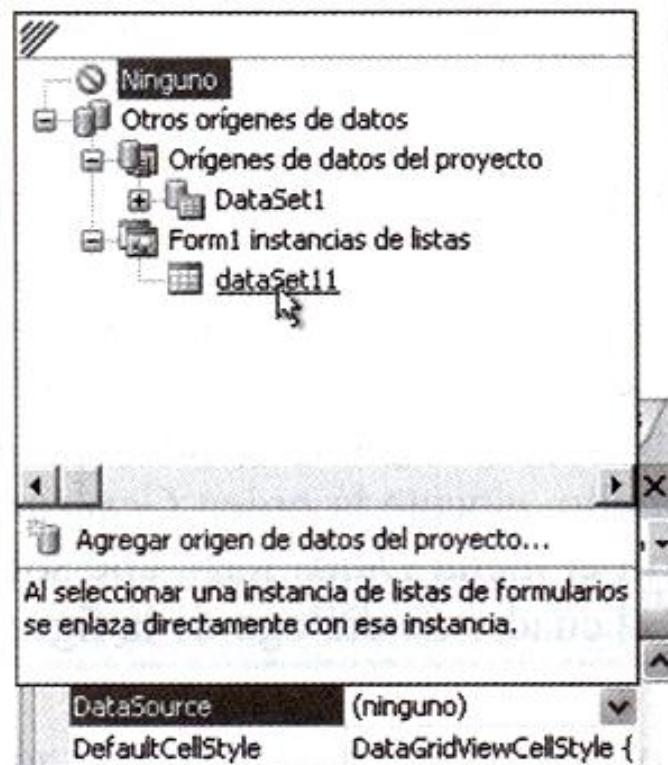
## Utilizando los componentes de .NET...Crear el conjunto de datos

- Una forma sencilla de generar automáticamente el conjunto de datos basándose en la consulta que se ha especificado para el adaptador de datos es utilizando los asistentes de Visual Studio; el conjunto de datos que se generará será un objeto de la clase **DataSet**.
- Para ello, ejecute la orden *Generar conjunto de datos* del menú *Datos* (si no se muestra el menú *Datos* haga clic sobre el formulario para que se muestre). Se mostrará el cuadro de diálogo de la figura siguiente:



## Utilizando los componentes de .NET...Agregar un control rejilla al formulario

- Arrastre desde la página *Windows Forms* del cuadro de herramientas un control **DataGridView**.
- El paso siguiente es vincular esta rejilla con la tabla *telefonos* del conjunto de datos. Para ello, asigne a su propiedad **DataSource** el valor *DataSet11* (véase la figura siguiente) y a su propiedad **DataMember** la tabla *telefonos*.





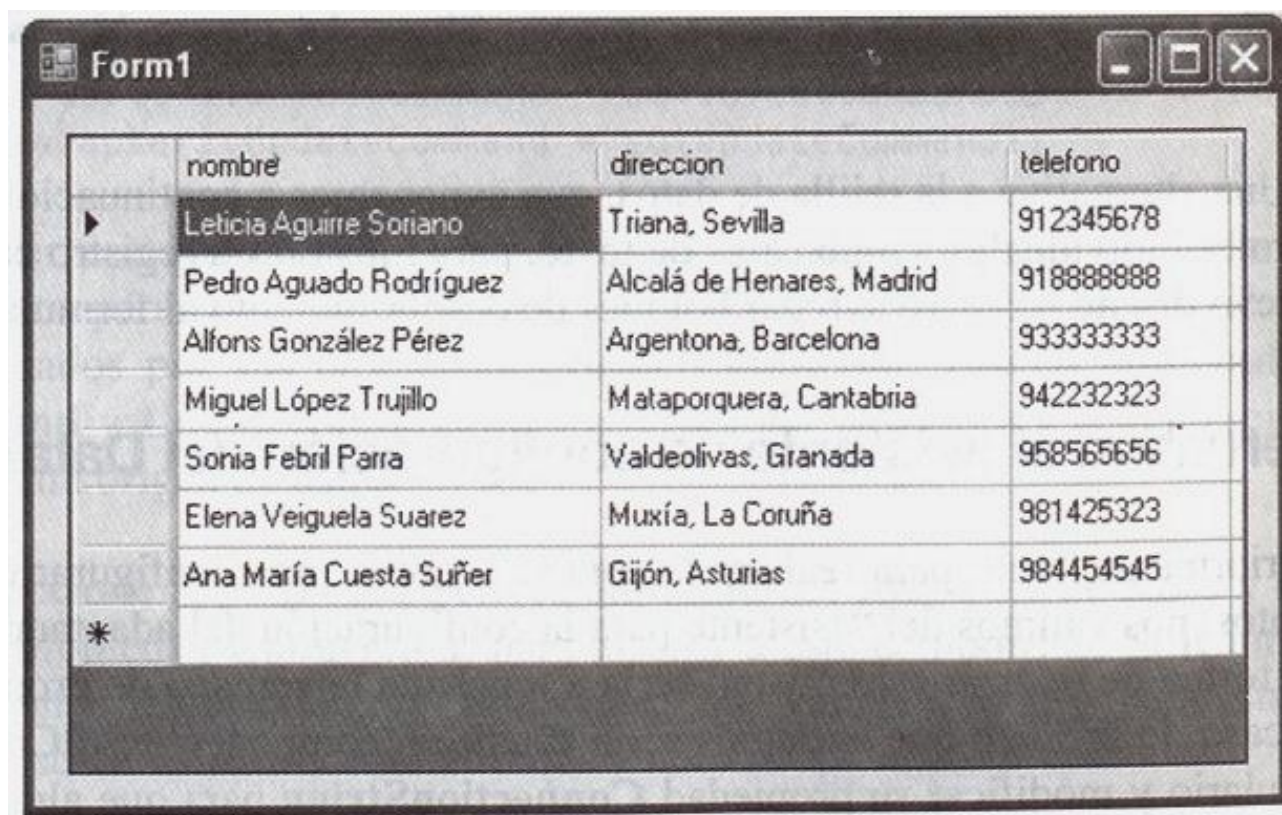
## Utilizando los componentes de .NET...Código

- Lo primero que vamos a hacer es añadir el código necesario para llenar la rejilla con los datos del conjunto de datos.
- Para ello, si se encuentra en la página de diseño, seleccione el formulario; después, diríjase a la ventana de propiedades y haga clic en el botón *Eventos*. Seleccione el evento **Load** que se producirá justo cuando se cargue el formulario. Haga doble clic sobre **Load**. Esto hará que se añada el controlador *Form1\_Load* de este evento. Complételo como se muestra a continuación.

```
private void Form1_Load(object sender, EventArgs e)
{
    dataSet11.Clear();
    sqlDataAdapter1.Fill(dataSet11);
}
```

## Utilizando los componentes de .NET...Código

- El método anterior, primero borra el conjunto de datos actual y, a continuación, llama al método **Fill** del adaptador de datos, pasándole el conjunto de datos que desea llenar.
- Si ahora ejecutamos la aplicación el resultado será similar al siguiente:



	nombre	direccion	telefono
▶	Leticia Aguirre Soriano	Triana, Sevilla	912345678
	Pedro Aguado Rodríguez	Alcalá de Henares, Madrid	918888888
	Alfons González Pérez	Argentona, Barcelona	933333333
	Miguel López Trujillo	Mataporquera, Cantabria	942232323
	Sonia Febril Parra	Valdeolivas, Granada	958565656
	Elena Veiguela Suarez	Muxía, La Coruña	981425323
	Ana María Cuesta Suñer	Gijón, Asturias	984454545
*			



## Utilizando los componentes de .NET...Código

- Finalmente añadiremos el código para actualizar la base de datos con las modificaciones que se realicen sobre la rejilla.
- Cuando el usuario de nuestra aplicación haga un cambio en el control rejilla de datos, dicho control guardará automáticamente el registro actualizado en el conjunto de datos (esto sucede justo en el momento de cambiar el punto de inserción a otro registro), pero no en el origen de datos.
- Esto tiene que hacerlo explícitamente el adaptador de datos invocando a su método **Update**, que examina cada registro de la tabla de datos especificada del conjunto de datos y, si se ha modificado alguno, ejecuta las órdenes apropiadas de las referenciadas por las propiedades **InsertCommand**, **UpdateCommand** o **DeleteCommand**.



## Utilizando los componentes de .NET...Código

- Según lo expuesto, agregue un nuevo controlador al formulario, en este caso para manipular el evento **FormClosing** que se produce cuando se cierra dicho formulario. Después complete dicho controlador así:

```
private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
    if (dataSet11.HasChanges())
    {
        sqlDataAdapter1.Update(dataSet11);
        MessageBox.Show( "Origen de datos actualizado");
    }
}
```



## Alternativa al asistente de configuración del DataAdapter

- Otra forma de realizar esto mismo sería a través de la ventana de propiedades.
- En este caso, lo primero que haríamos sería añadir un componente **SqlConnection** al formulario y modificar su propiedad **ConnectionString** para que almacene la cadena de conexión válida para nuestra base de datos. Por ejemplo:

Data Source=.\sqlexpress;Initial Catalog=bd\_telefonos;Integrated Security=True

- El código generado para realizar esta operación es el siguiente:

```
private SqlConnection sqlConnection1;
```

```
sqlConnection1 = new SqlConnection();
```

```
sqlConnection1.ConnectionString = "Data Source=.\sqlexpress;Initial  
Catalog=bd_telefonos;" + "Integrated Security=True";
```



## Alternativa al asistente de configuración del DataAdapter

- A continuación añadiríamos el componente **SqlDataAdapter** al formulario. Si esta acción abre la ventana del asistente, ciérrala, ya que, como hemos dicho, realizaremos su configuración a través de la ventana propiedades.
- Añadir un adaptador de datos requiere el código siguiente, generado por Visual Studio (véase el fichero *Form1.Designer.cs*):

```
private SqlDataAdapter sqlDataAdapter1;  
private SqlCommand sqlSelectCommand1;  
private SqlCommand sqlInsertCommand1;  
private SqlCommand sqlUpdateCommand1;  
private SqlCommand sqlDeleteCommand1;
```

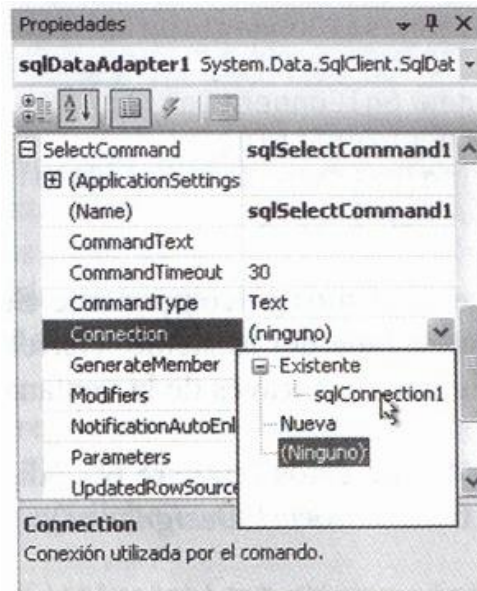
```
sqlDataAdapter1 = new SqlDataAdapter();  
sqlSelectCommand1 = new SqlCommand();  
sqlInsertCommand1 = new SqlCommand();  
sqlUpdateCommand1 = new SqlCommand();  
sqlDeleteCommand1 = new SqlCommand();
```

```
sqlDataAdapter1.DeleteCommand = sqlDeleteCommand1;  
sqlDataAdapter1.InsertCommand = sqlInsertCommand1;  
sqlDataAdapter1.SelectCommand = sqlSelectCommand1;  
sqlDataAdapter1.UpdateCommand = sqlUpdateCommand1;
```



## Alternativa al asistente de configuración del DataAdapter

- Obsérvese que cuando se añade un objeto **DataAdapter** se añaden también los objetos para las órdenes SELECT, INSERT, UPDATE Y DELETE que serán referenciados por sus propiedades **SelectCommand**, **InsertCommand**, **DeleteCommand** y **UpdateCommand** para posibilitar la lectura y actualización de los datos en un origen de datos.
- Seleccione la propiedad **SelectCommand** del adaptador de datos y dentro de ésta, la propiedad **Connection**, para asignar a este componente **SqlCommand** el componente de conexión *SqlConnection1* que vamos a utilizar para establecer la comunicación entre la fuente de datos y nuestra aplicación:





## Alternativa al asistente de configuración del DataAdapter

- La operación especificada genera el código siguiente:

```
sqlSelectCommand1.Connection = sqlConnection1;
```

- Finalmente, arrastramos desde la caja de herramientas un **DataSet** y un **DataGridView**:

```
private DataSet dataSet1;  
private DataGridView dataGridView1;  
  
dataSet1 = new DataSet();  
dataGridView1 = new DataGridView();
```



## Alternativa al asistente de configuración del DataAdapter

- El paso siguiente sería volcar en la rejilla los datos que se obtengan de la base de datos.
- Para ello, escribiríamos un método, que responda al evento **Load** del formulario, que ejecute la sentencia SELECT válida para nuestro ejemplo:

```
private void Form1_Load(object sender, EventArgs e)
{
    // Sentencia SELECT a ejecutar
    sqlSelectCommand1.CommandText = "SELECT * FROM telefonos";
    // Rellenar el DataSet con el contenido obtenido por SELECT
    sqlDataAdapter1.Fill(dataSet1, "telefonos");
    // Mostrar la información obtenida en una rejilla
    dataGridView1.DataSource = dataSet1.Tables["telefonos"];
}
```



## Alternativa al asistente de configuración del DataAdapter

- Si la conexión está cerrada antes de llamar a Fill, ésta se abrirá para recuperar datos y, a continuación, se cerrará.
- Si la conexión está abierta antes de llamar a Fill, ésta permanecerá abierta. Según esto, también podríamos escribir:

```
private void Form1_Load(object sender, EventArgs e)
{
    // Sentencia SELECT a ejecutar
    sqlSelectCommand1.CommandText = "SELECT * FROM telefonos";
    // Abrir la conexión
    sqlConnection1.Open();
    // Rellenar el DataSet con el contenido obtenido por SELECT
    sqlDataAdapter1.Fill(dataSet1, "telefonos");
    // Cerrar la conexión
    sqlConnection1.Close();
    // Mostrar la información obtenida en una rejilla
    dataGridView1.DataSource = dataSet1.Tables["telefonos"];
}
```



## Alternativa al asistente de configuración del DataAdapter

- Finalmente añadiremos el código para que, al salir de la aplicación, se actualice la base de datos con las modificaciones que se hubieran realizado sobre la rejilla.
- Cuando el usuario de nuestra aplicación haga un cambio en el control rejilla de datos, dicho control guardará automáticamente el registro actualizado en el conjunto de datos (esto sucede justo en el momento de cambiar el punto de inserción a otro registro), pero no en el origen de datos.
- Esto tiene que hacerlo explícitamente el adaptador de datos invocando a su método **Update**, que examina cada registro de la tabla de datos especificada del conjunto de datos y, si se ha modificado alguno, ejecuta las órdenes apropiadas de las referenciadas por las propiedades **InsertCommand**, **UpdateCommand** o **DeleteCommand**.

## Alternativa al asistente de configuración del DataAdapter

- Según lo expuesto, agregue un nuevo controlador al formulario, en este caso para manipular el evento **FormClosing** que se produce cuando se cierra dicho formulario. Después complete dicho controlador así:

```
private void Form1_FormClosing(object sender, FormClosingEventArgs e) {  
    try {  
        if (dataSet1.HasChanges()) {  
            // IMPLEMENTACION DE LA ORDEN UPDATE  
  
            // IMPLEMENTACION DE LA ORDEN INSERT  
  
            // IMPLEMENTACION DE LA ORDEN DELETE  
  
            sqlDataAdapter1.Update(dataSet1, "telefonos");  
            MessageBox.Show("Origen de datos actualizado");  
        }  
    }  
    catch (SqlException ex) {  
        MessageBox.Show("No se actualizó el origen de datos " +  
            "porque ocurrió el error: " + ex.Message);  
    }  
}
```



## Alternativa al asistente de configuración del DataAdapter

- Cuando se ejecuta el método **Update**, el objeto **DataAdapter** examina la propiedad **RowState** de cada fila y ejecuta las órdenes INSERT, UPDATE o DELETE necesarias.
- En la actualización, los campos de cada objeto **DataRow** (fila) se trasladan a los valores de los parámetros de las órdenes.
- Si estas órdenes no se implementaron, el método **Update** genera una excepción.
- De acuerdo con lo expuesto, pasamos a implementar las órdenes UPDATE, INSERT y DELETE referenciadas por las propiedades **UpdateCommand**, **InsertCommand** y **DeleteCommand** del adaptador de datos.



## Alternativa al asistente de configuración del DataAdapter

- Igual que hizo con la propiedad **SelectCommand**, seleccione cada una de las propiedades especificadas en el párrafo anterior y dentro de ella, la propiedad **Connection**, para asignar al componente **SqlCommand** el componente de conexión *SqlConnection1* que vamos a utilizar para establecer la comunicación entre la fuente de datos y nuestra aplicación.
- Las operaciones especificadas generan el código siguiente:

```
SqlUpdateCommand1.Connection = SqlConnection1;  
SqlInsertCommand1.Connection = SqlConnection1;  
SqlDeleteCommand1.Connection = SqlConnection1;
```

- Queda, para, cada una de las órdenes, especificar los parámetros involucrados y escribir la acción que se desea ejecutar.



## Alternativa al asistente de configuración del DataAdapter

- UPDATE. Actualizar en la base de datos, los datos que se han modificado en la rejilla:

```
// IMPLEMENTACIÓN DE LA ORDEN UPDATE
// Añadir los parámetros a la orden para cada campo de la fila
SqlUpdateCommand1.Parameters.Add("@nombre",
SqlDbType.VarChar, 30, "nombre");
SqlUpdateCommand1.Parameters.Add("@direccion",
SqlDbType.VarChar, 30, "direccion");
SqlUpdateCommand1.Parameters.Add( "@telefono",
SqlDbType.VarChar, 12, "telefono");
SqlUpdateCommand1.Parameters.Add("@observaciones",
SqlDbType.VarChar, 240, "observaciones");
// ...

// Sentencia UPDATE a ejecutar
SqlUpdateCommand1.CommandText = "UPDATE telefonos " +
"SET nombre = @nombre, direccion = @direccion, " +
"telefono = @telefono, observaciones = @observaciones " +
"WHERE (...
```

## Alternativa al asistente de configuración del DataAdapter

- INSERT. Insertar en la base de datos las nuevas filas añadidas en la rejilla:

```
// IMPLEMENTACIÓN DE LA ORDEN INSERT
// Añadir los parámetros a la orden para cada campo de la fila
SqlInsertCommand1.Parameters.Add("@nombre",
SqlDbType.VarChar, 30, "nombre");
SqlInsertCommand1.Parameters.Add~"@direccion",
SqlDbType.VarChar, 30, "direccion");
SqlInsertCommand1.Parameters.Add("@telefono",
SqlDbType.VarChar, 12, "telefono");
SqlInsertCommand1.Parameters.Add("@observaciones",
SqlDbType.VarChar, 240, "observaciones");

// Sentencia INSERT a ejecutar
SqlInsertCommand1.CommandText = "INSERT INTO telefonos " +
"(nombre, direccion, telefono, observaciones) VALUES " +
"(@nombre, @direccion, @telefono, @observaciones)";
```



## Alternativa al asistente de configuración del DataAdapter

- DELETE. Borrar en la base de datos las filas borradas en la rejilla:

```
// IMPLEMENTACIÓN DE LA ORDEN DELETE
```

```
// Añadir los parámetros a la orden para cada campo de la fila
```

```
SqlDeleteCommand1.Parameters.Add("@nombre",
```

```
SqlDbType.VarChar, 30, "nombre");
```

```
SqlDeleteCommand1.Parameters.Add("@direccion",
```

```
SqlDbType.VarChar, 30, "direccion");
```

```
SqlDeleteCommand1.Parameters.Add("@telefono",
```

```
SqlDbType.VarChar, 12, "telefono");
```

```
SqlDeleteCommand1.Parameters.Add("@observaciones",
```

```
SqlDbType.VarChar, 240, "observaciones");
```

```
// Sentencia DELETE a ejecutar
```

```
SqlDeleteCommand1.CommandText = "DELETE FROM telefonos " +
```

```
"WHERE (nombre = @nombre AND direccion = @direccion " +
```

```
"AND telefono = @telefono AND observaciones = @observaciones)";
```



## Utilizando las clases de .NET

- El problema que tenemos que resolver es mostrar en un control rejilla de datos de un formulario Windows, la información almacenada en la tabla *telefonos* de la base de datos *bd\_telefonos*.
- Esto implica crear e iniciar los objetos conexión, adaptador de datos y conjunto de datos, y añadir una rejilla al formulario que muestre los datos obtenidos de la base de datos.
- Este proceso lo podemos hacer como respuesta al evento **Load** del formulario:

```
private void Form1_Load(object sender, EventArgs e)
{
    CrearIniciarComponentes();
    ObtenerMostrarDatos();
}
```

- Para realizar una conexión con la base de datos, utilizando un adaptador de datos, para obtener la información de la misma y volcarla en un conjunto de datos, previamente hay que declarar las variables que referenciarán los objetos conexión, adaptador de datos y conjunto de datos:



## Utilizando las clases de .NET

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

// Conexión
private SqlConnection SqlConnection1;
// Adaptador de datos
private SqlDataAdapter SqlDataAdapter1;
private SqlCommand SqlSelectCommand1;
private SqlCommandcSqlInsertCommand1;
private SqlCommand SqlUpdateCornmand1;
private SqlCornmand SqlDeleteCornmand1;
// Conjunto de datos
private DataSet DataSet1;
```

- Nótese que el hecho de declarar la utilización de un **DataAdapter** sin usar componentes, obliga a añadir las referencias a los espacios de nombres *System.Data* y *System.Xml*.

## Utilizando las clases de .NET



- El paso siguiente será crear e iniciar los objetos conexión, adaptador de datos y conjunto de datos:



## Utilizando las clases de .NET

```
private void CrearIniciarComponentes() {  
    // Crear una conexión con la base de datos  
    SqlConnection1 = new SqlConnection();  
    SqlConnection1.ConnectionString = "Data Source=.\sqlexpress;" + "Initial  
Catalog=bd_telefonos;" +  
    "Integrated Security=True";  
    // Crear el DataAdapter y las órdenes SQL asociadas  
    // estableciendo la conexión con la fuente de datos  
    SqlDataAdapter1 = new SqlDataAdapter();  
    SqlCommand1 = new SqlCommand();  
    SqlCommand1.CommandText = "SELECT * FROM telefonos";  
    SqlCommand1.Connection = SqlConnection1;  
    SqlDataAdapter1.SelectCommand = SqlCommand1;  
    SqlCommand1.CommandText = "INSERT INTO telefonos (numero, nombre, direccion)  
VALUES (@numero, @nombre, @direccion)";  
    SqlCommand1.Connection = SqlConnection1;  
    SqlDataAdapter1.InsertCommand = SqlCommand1;  
    SqlCommand1.CommandText = "UPDATE telefonos SET nombre = @nombre, direccion = @direccion  
WHERE numero = @numero";  
    SqlCommand1.Connection = SqlConnection1;  
    SqlDataAdapter1.UpdateCommand = SqlCommand1;  
    SqlCommand1.CommandText = "DELETE FROM telefonos WHERE numero = @numero";  
    SqlCommand1.Connection = SqlConnection1;  
    SqlDataAdapter1.DeleteCommand = SqlCommand1;  
    // Crear el DataSet donde se volcarán los datos  
    DataSet1 = new DataSet(); }  
}
```



## Utilizando las clases de .NET

*Una vez creados los componentes conexión, adaptador de datos y conjunto de datos, implementamos la orden SELECT del DataAdapter, abrimos la conexión con el origen de datos, recuperamos las filas del origen de datos mediante la sentencia SELECT especificada por la propiedad SelectCommand del adaptador de datos, que volcamos en la tabla "telefonos" del DataSet, y cerramos la conexión:*

```
private void ObtenerMostrarDatos()
{
    // Sentencia SELECT a ejecutar
    SqlSelectCommand1.CommandText = "SELECT * FROM telefonos";
    // Abrir la conexión
    SqlConnection1.Open();
    // Rellenar el DataSet con el contenido obtenido por SELECT
    SqlDataAdapter1.Fill(DataSet1, "telefonos");
    // Cerrar la conexión
    SqlConnection1.Close();
}
```





## Utilizando las clases de .NET

- El paso siguiente es vincular un `DataGridView` con la tabla *telefonos* del conjunto de datos.
- Para ello, asigne a su propiedad **DataSource** el objeto **DataTable** correspondiente a la tabla *telefonos* del **DataSet**.
- Modifique las propiedades de la rejilla que considere necesarias. Por ejemplo, para asignar un nombre adecuado a las columnas. Según esto, añada al método anterior las líneas de código indicadas a continuación:

```
private void ObtenerMostrarDatos()
{
    // ...
    // Mostrar la información obtenida en una rejilla
    DataGridView1.DataSource = DataSet1.Tables["telefonos"];
    // Establecer los títulos de las columnas
    DataGridView1.Columns["nombre"].HeaderText = "Nombre";
    DataGridView1.Columns["direccion"].HeaderText = "Dirección";
    DataGridView1.Columns["telefono"].HeaderText = "Teléfono";
    DataGridView1.Columns["observaciones"].HeaderText = "Observaciones";
}
```



## Utilizando las clases de .NET

### **Actualizar la base de datos**

- Finalmente añadiremos el código para que, al salir de la aplicación, se actualice la base de datos con las modificaciones que se hubieran realizado sobre la rejilla.
- Esto ya fue explicado anteriormente en el apartado "Alternativa al asistente de configuración del DataAdapter".

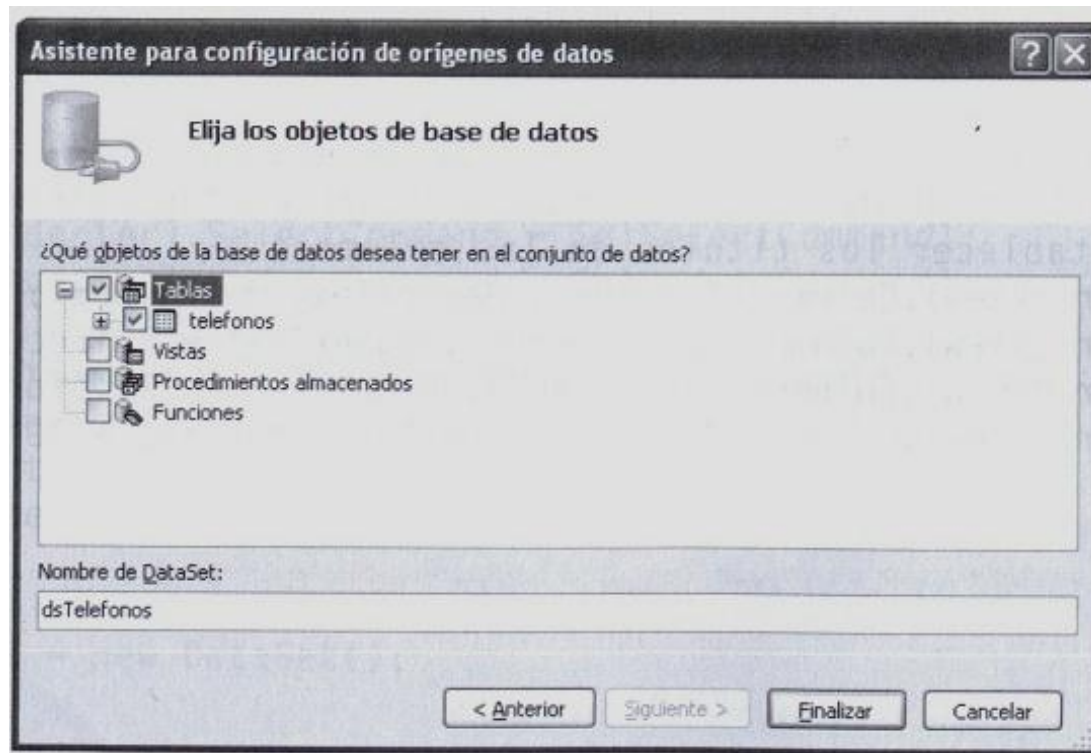


## Utilizando clases generadas por el asistente

- Otra forma de proceder es crear, utilizando los asistentes de Visual Studio, una capa de acceso a datos que nos permita abstraernos de las operaciones con la base de datos.
- Esta capa proporcionará las clases necesarias para crear un adaptador, un conjunto de datos, una tabla, una fila de la tabla, etc., y serán generadas automáticamente por el asistente para la configuración de orígenes de datos.
- Por ejemplo, cree una aplicación Windows. Después, diríjase al panel *Orígenes de datos*, o bien ejecute la orden *Mostrar orígenes de datos* del menú *Datos* para mostrarlo, y haga clic en *Agregar nuevo origen de datos*.
- Se le mostrará un asistente que le permitirá crear una nueva conexión, en nuestro caso, con un origen de datos *Microsoft SQL Server*, utilizando un proveedor de datos de .NET Framework para un servidor *SQL Server*.

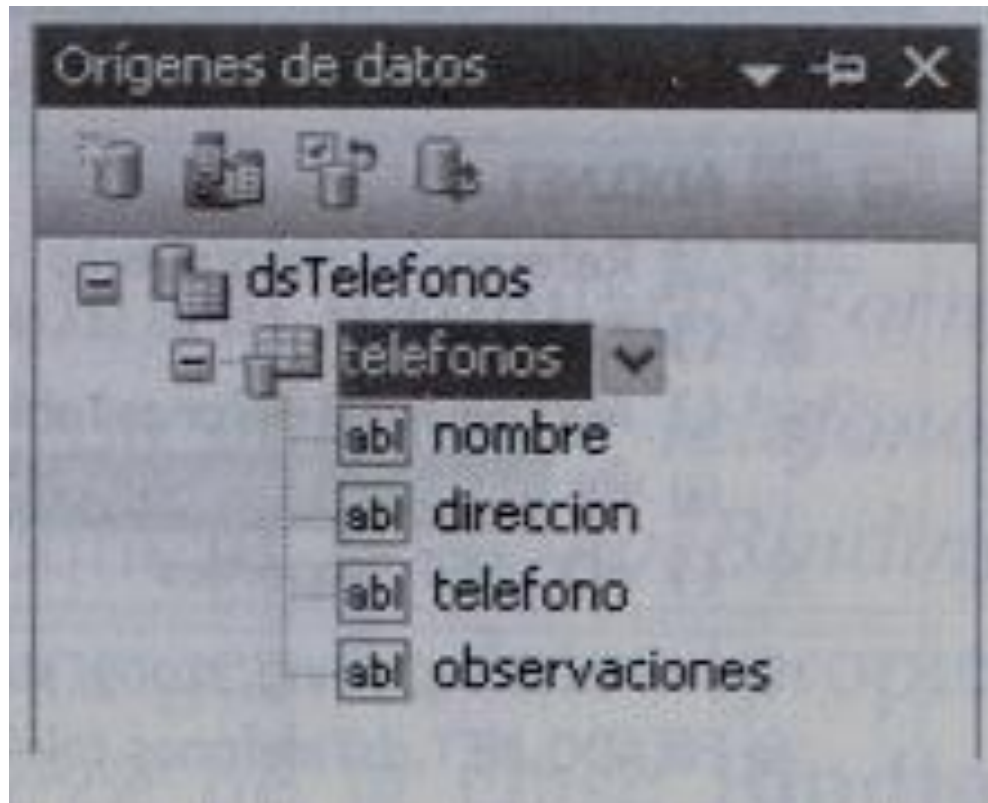
## Utilizando clases generadas por el asistente

- Una vez creada la conexión, cree un conjunto de datos (**DataSet**): *dsTelefonos*. Para crear este conjunto de datos elija:
  - Tipo de origen de base de datos: *Base de datos*.
  - Conexión de datos: *bd\_telefonos.mdf*.
  - Objetos de base de datos: tabla *telefonos*.
  - Nombre del **DataSet**: *dsTelefonos*.



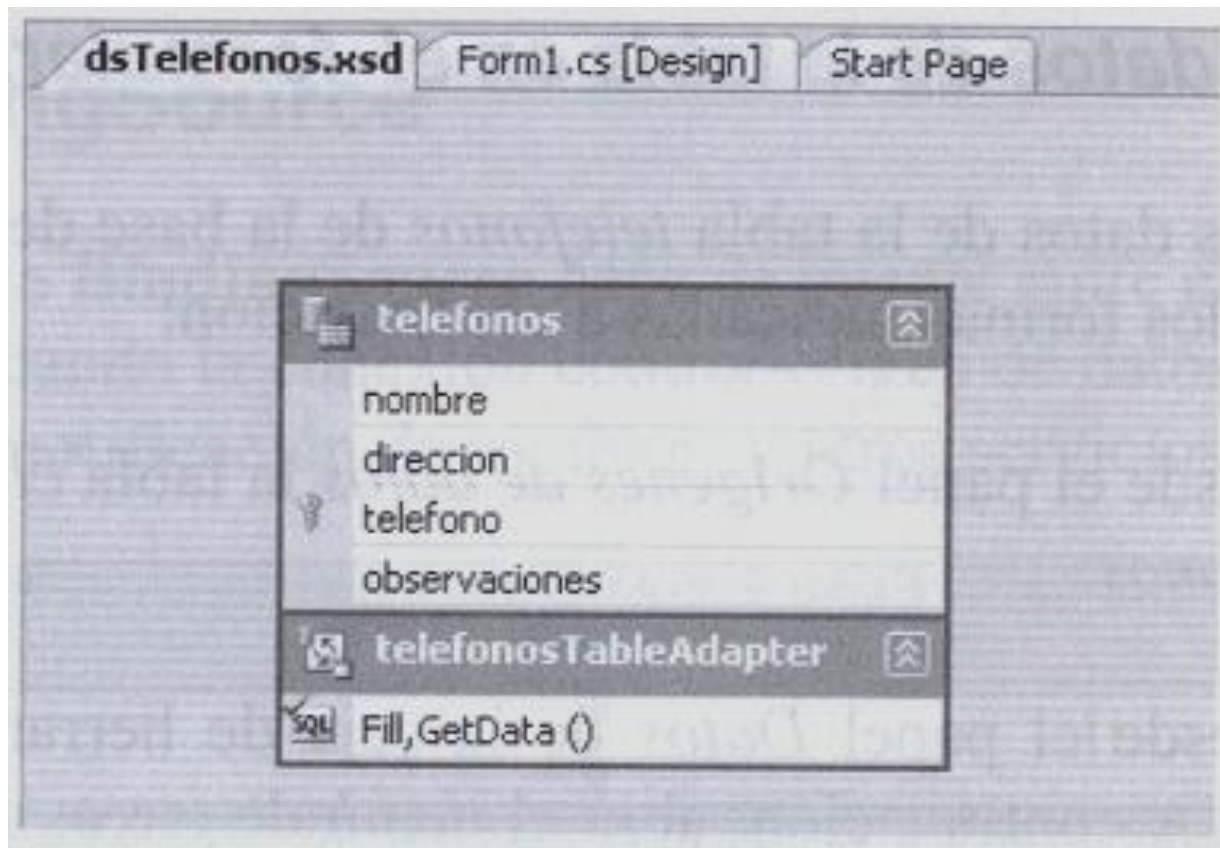
## Utilizando clases generadas por el asistente

- Cuando haga clic en el botón *Finalizar* habrá creado la clase que define el conjunto de datos. Si ahora, desde Visual Studio, ejecuta la orden *Datos > Mostrar orígenes de datos* podrá observar en el panel *Orígenes de datos* la estructura de este origen de datos:



## Utilizando clases generadas por el asistente

- Asimismo, si hace clic con el botón secundario de ratón sobre el nombre del DataSet y ejecuta la orden **Editar DataSet** con el diseñador del menú contextual que se visualiza, se mostrará la siguiente ventana:





## Utilizando clases generadas por el asistente

- La figura anterior muestra dos objetos: *telefonos* de la clase **DataTable** y *telefonosTableAdapter* de la clase **TableAdapter**.
- El primero hace referencia a la tabla del **DataSet** y el segundo al adaptador que se utilizará para, ejecutando la orden SQL adecuada, llenar la tabla del **DataSet**.
- Esto es lo que anteriormente denominados acceso desconectado a base de datos.
- El adaptador presenta dos métodos: *Fill* y *GetData*.
- El método *Fill* toma como parámetro un **DataTable** o un **DataSet** y ejecuta la orden SQL programada (puede verla haciendo clic con el botón secundario del ratón en el título de cualquiera de los dos paneles y ejecutando la orden *Configurar* del menú contextual que se visualiza).
- El método *GetData* devuelve un nuevo objeto **DataTable** con los resultados de la orden SQL.
- También se han creado métodos *Insert*, *Update* y *Delete* que se pueden llamar para enviar cambios de filas individuales directamente a la base de datos.

## Utilizando clases generadas por el asistente

- Asimismo, en la vista de clases puede ver la funcionalidad proporcionada por la clase del conjunto de datos añadido, funcionalidad que utilizaremos cuando sea necesario.
- Por ejemplo, la clase *telefonosTableAdapter* (véase la figura siguiente) permitirá crear un adaptador para acceder a la tabla *telefonos* y su método *Fill* o *GetData*, obtener los datos de dicha tabla.



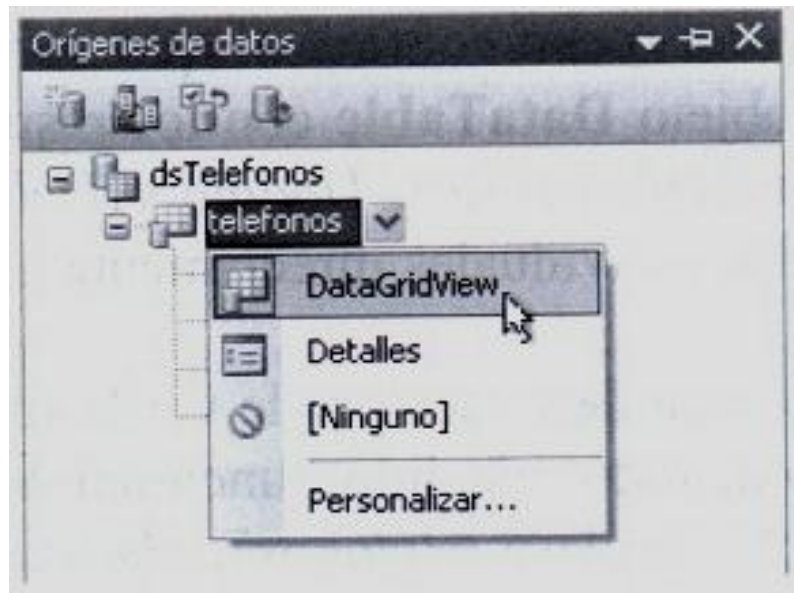




## Utilizando clases generadas por el asistente

- Para mostrar los datos de la tabla *telefonos* de la base de datos, puede proceder de alguna de las dos formas indicadas a continuación:
  - Arrastre desde el panel *Orígenes de datos* la tabla *telefonos* del conjunto de datos *dsTelefonos*.
  - Arrastre desde el panel *Datos* de la caja de herramientas un control *DataGridView*. A continuación, abra el menú de tareas de la rejilla y configúrela, asignando, como primer paso, el origen de datos.
- Quizás, la forma más sencilla sea la primera. Para hacer lo indicado en ese punto 1, diríjase al panel *Orígenes de datos* y arrastre sobre el formulario la tabla *telefonos* del conjunto de datos *dsTelefonos*.
- Observando la figura siguiente, vemos a la izquierda de la entidad *telefonos* un icono: *DataGridView*, *Detalles*, etc (refresque la vista si es necesario haciendo clic en el botón *Refrescar* de la barra de herramientas de este panel):

## Utilizando clases generadas por el asistente



- El icono al que nos hemos referido en el párrafo anterior, seleccionable haciendo clic con el ratón en el botón situado a la derecha de la entidad, indica los controles que se añadirán sobre el formulario para acceder al origen de datos.
- Nosotros vamos a dejar el seleccionado, *DataGridView*, para que se añada una rejilla. Una vez añadida, fije su propiedad **Anchor** con los valores *Top*, *Bottom*, *Left* y *Right*, y su propiedad **AutoSizeColumnsMode** con el valor *Fill*.

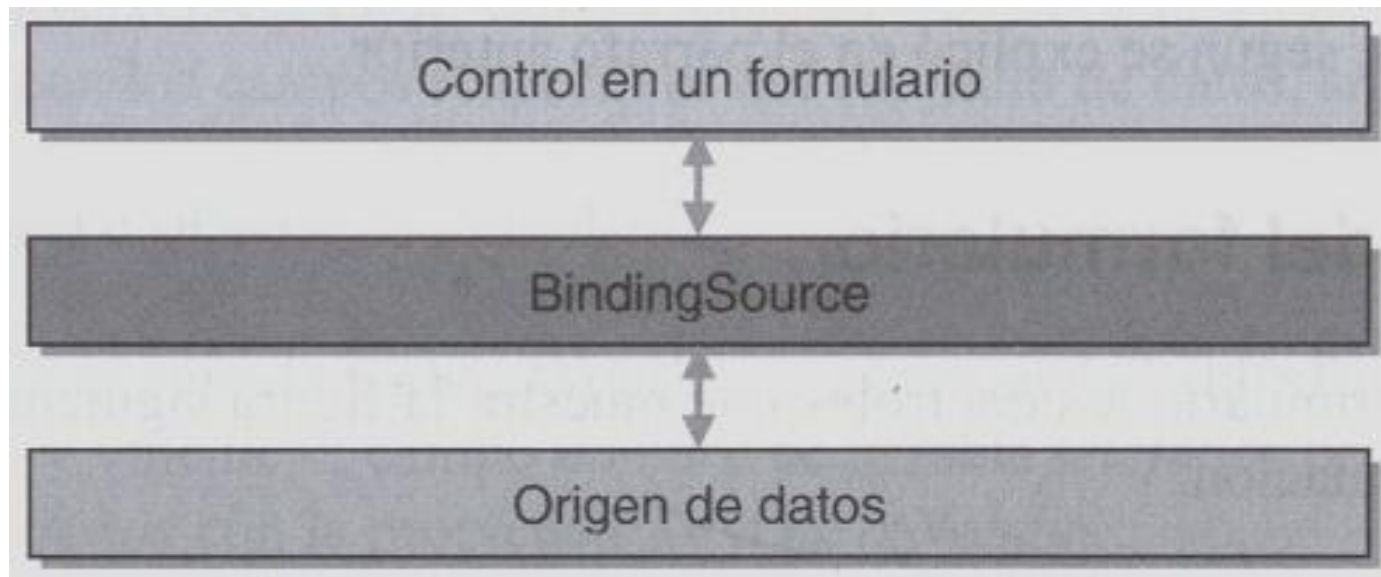


## Utilizando clases generadas por el asistente

- La operación descrita añadirá al formulario *Form1*:
  - un conjunto de datos *dsTelefonos*, de tipo *dsTelefonos*,
  - un adaptador *telefonosTableAdapter* para llenar la tabla de *dsTelefonos* y
  - un control *telefonosBindingNavigator* de la clase **BindingNavigator** que tiene por origen de datos (propiedad **BindingSource**) al componente *telefonosBindingSource* de la clase **BindingSource** que, a su vez, está conectado al origen de datos *dsTelefonos*.
- Si la barra de navegación no se necesita puede ocultarla poniendo su propiedad **Visible** a **false**.

## Control BindingSource

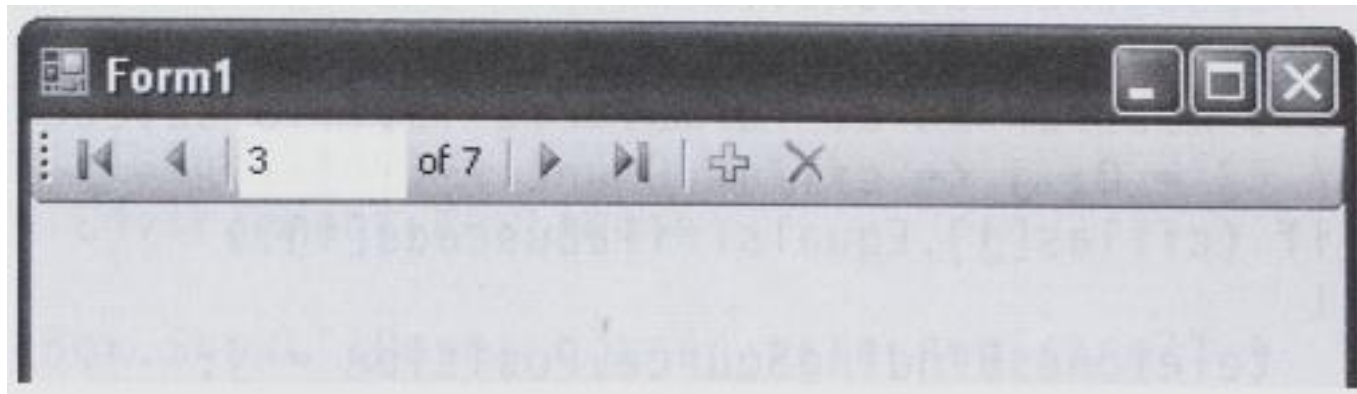
- En general, un objeto **BindingSource** hace de puente entre el control y el conjunto de datos, simplificando la conexión control-origen de datos, la actualización del contenido actual, la notificación de cambios, y otros servicios.



- Realizada la conexión control-origen de datos, toda interacción con los datos, incluyendo navegación, ordenación, filtrado y actualización, puede ser llevada a cabo utilizando la funcionalidad del componente **BindingSource**

## Control BindingNavigator

- El control **BindingNavigator** está compuesto de un **ToolStrip** con los siguientes objetos **ToolStripxxx** (**ToolStripButton**, **ToolStripSeparator**, etc.):
  - Botón para ir al registro primero.
  - Botón para ir al registro anterior.
  - Caja de texto para mostrar la posición del registro actual.
  - Etiqueta para mostrar el número total de registros.
  - Botón para ir al registro siguiente.
  - Botón para ir al último registro.
  - Botón para añadir un registro.
  - Botón para eliminar un registro.





## Control BindingNavigator

- Cada uno de estos objetos es asignado a la propiedad correspondiente del control **BindingNavigator**. Por ejemplo:

```
private BindingNavigator bindingNavigator1;  
private ToolStripButton bindingNavigatorMoveFirstItem;  
private ToolStripTextBox bindingNavigatorPositionItem;  
// ...  
bindingNavigator1 = new BindingNavigator();  
bindingNavigatorMoveFirstItem = new ToolStripButton();  
bindingNavigatorPositionItem = new ToolStripTextBox();  
// ...  
bindingNavigator1.MoveFirstItem = bindingNavigatorMoveFirstItem;  
bindingNavigator1.PositionItem = bindingNavigatorPositionItem;  
// ...
```



## Control BindingNavigator

- Para conectar este control con el origen de datos, simplemente hay que asignar a su propiedad **BindingSource** el componente **BindingSource** que tiene la conexión con ese origen de datos. Por ejemplo:

```
BindingSource1.DataMember = "telefonos";
```

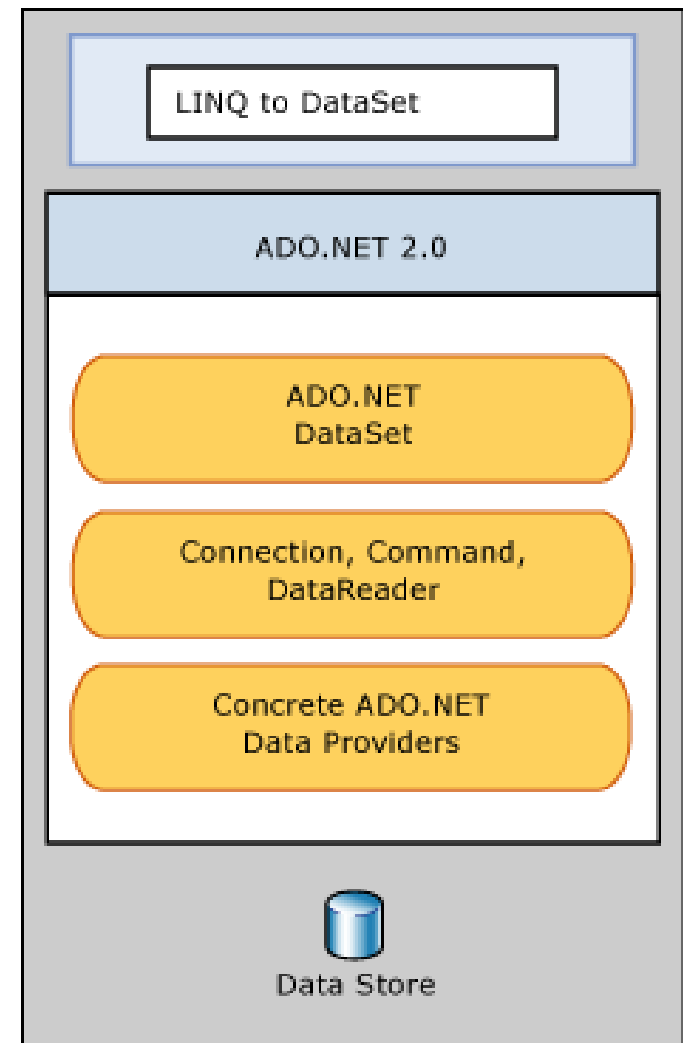
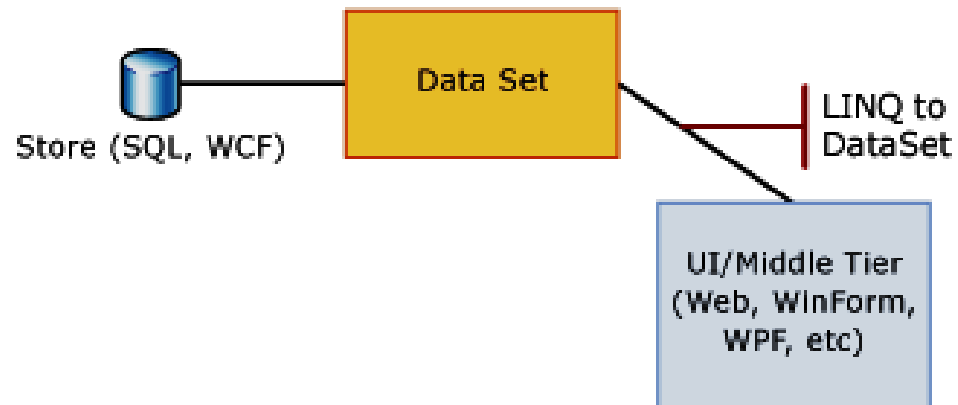
```
BindingSource1.DataSource = dsTelefonos;
```

```
BindingNavigator1.BindingSource = BindingSource1;
```

- Cada uno de los controles de BindingNavigator tiene su correspondiente propiedad o método en el componente BindingSource proporcionando la misma funcionalidad.
  - Por ejemplo, el botón MoveFirstItem se corresponde con el método MoveFirst de BindingSource, la posición PositionItem se corresponde con la propiedad Position de BindingSource, etc

## LINQ to DataSet

- Simplificado mucho, llenamos el DataSet y después construimos y ejecutamos consultas LINQ sobre esos datos en memoria.





## LINQ to DataSet

```
static void Main(string[] args)
{
    try
    {
        DataSet ds = new DataSet();
        ds.Locale = CultureInfo.InvariantCulture;
        FillDataSet(ds);
        DataTable products = ds.Tables["Products"];

        IEnumerable<DataRow> query = from product in products.AsEnumerable()
                                     select product;

        Console.WriteLine("Product Names:");
        foreach(DataRow p in query)
        {
            Console.WriteLine(p.Field<string>("ProductName"));
        }
        Console.Read();
    }
    catch (Exception ex)
    {
        Console.WriteLine("Excepcion en Main " + ex.Message);
        Console.Read();
    }
}

//main
```



## LINQ to DataSet

```
/*
 * Metodo FillDataSet, se encarga de llenar el DataSet _ds
 * por medio de un DataAdapter
 */
private static void FillDataSet(DataSet _ds)
{
    SqlConnection conn = new SqlConnection();
    try
    {
        conn.ConnectionString = cadenaconexion;

        string q = "SELECT * FROM dbo.Products";
        SqlDataAdapter da = new SqlDataAdapter(q, conn);

        da.Fill(_ds, "Products");
    }
    catch (Exception ex)
    {
        Console.WriteLine("Exception en FillDataSet " + ex.Message);
        Console.Read();
    }
    finally
    {
        if (conn.State == ConnectionState.Open)
        {
            conn.Close();
        }
    }
}
} //function
```



## LINQ to SQL

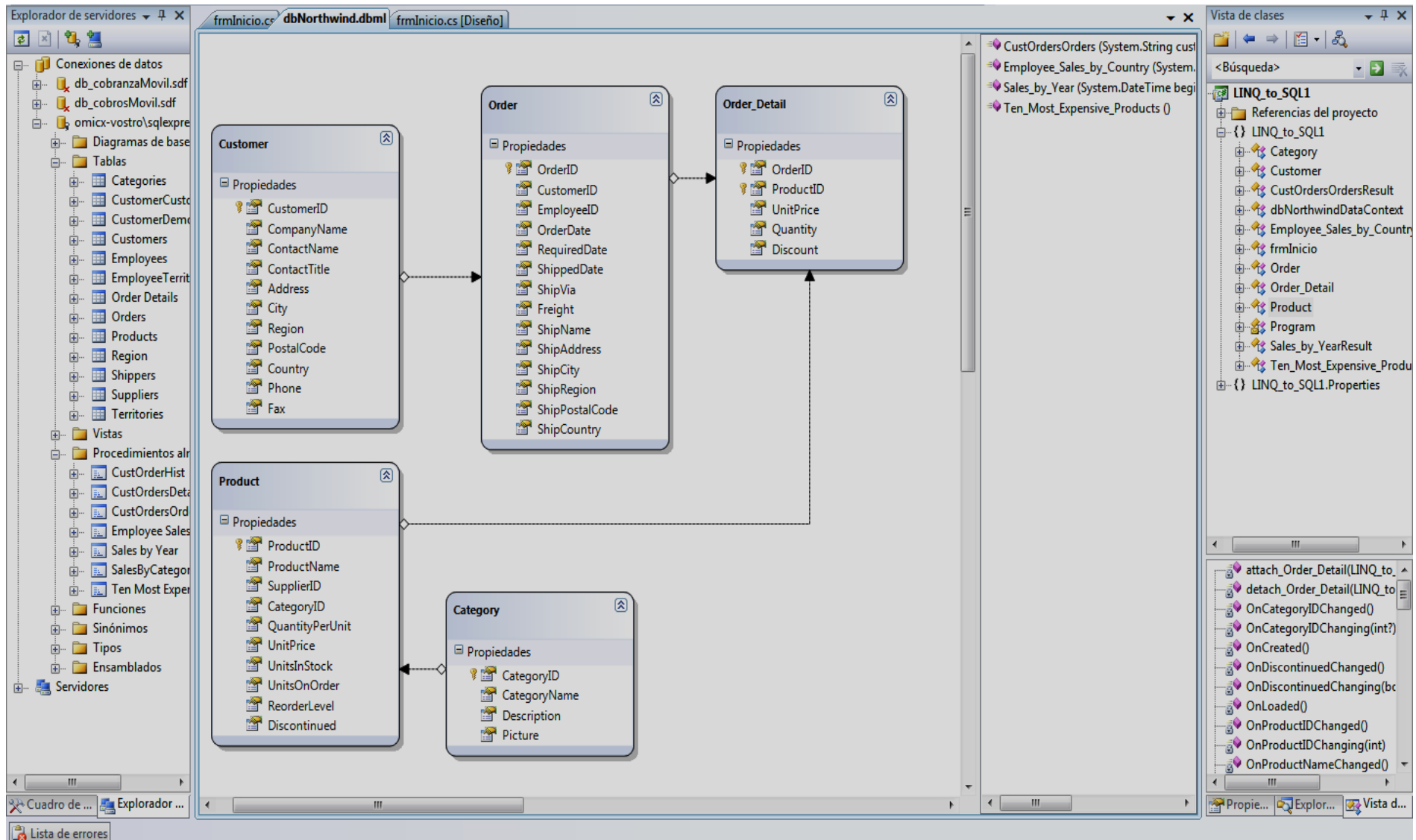
- Permite interactuar con SQL Server (solo soporta SQL Server y SQL Server Compact 3.5)
- Implementación de OR/M (mapeador de objetos relacionales)
- Modelar bases de datos relacionales con clase .NET
- Podemos consultar, actualizar, añadir, borrar
- LINQ to SQL convierte las consultas integradas en el lenguaje del modelo de objetos (C#) a SQL y las envía a la base de datos para su ejecución.
- Incluye compatibilidad con los procedimientos almacenados y las funciones definidas por el usuario en la base de datos



## LINQ to SQL

- Visual Studio 2008 viene con un diseñador de LINQ to SQL que nos aporta una forma fácil de modelar y visualizar una base de datos como un modelo de objeto
- Usando este diseñador LINQ to SQL podemos crear fácilmente una representación de la base de datos
- Con este diseñador crear clases, métodos y demás solo es cuestión de ¡arrastrar y soltar !
- Con el diseñador de objetos relacionales, arrastraremos las clases y los procedimientos almacenados (o funciones) y se generará el código para acceder a ellos directamente

# LINQ to SQL – O/R Designer





## LINQ to SQL – O/R Designer

```
/*
 * Metodo que se encarga de obtener y mostrar los datos de la tabla Products
 * de la bd Northwind
 */
private void fillDataGrid()
{
    dbNorthwindDataContext db = new dbNorthwindDataContext();

    var products = from product in db.Product
                   where product.Category.CategoryName == "Beverages"
                   select new {
                       product.ProductID,
                       product.ProductName,
                       product.QuantityPerUnit,
                       product.UnitPrice,
                       product.UnitsInStock,
                       product.UnitsOnOrder,
                       product.ReorderLevel,
                       product.Discontinued,
                       product.Category.CategoryName
                   };

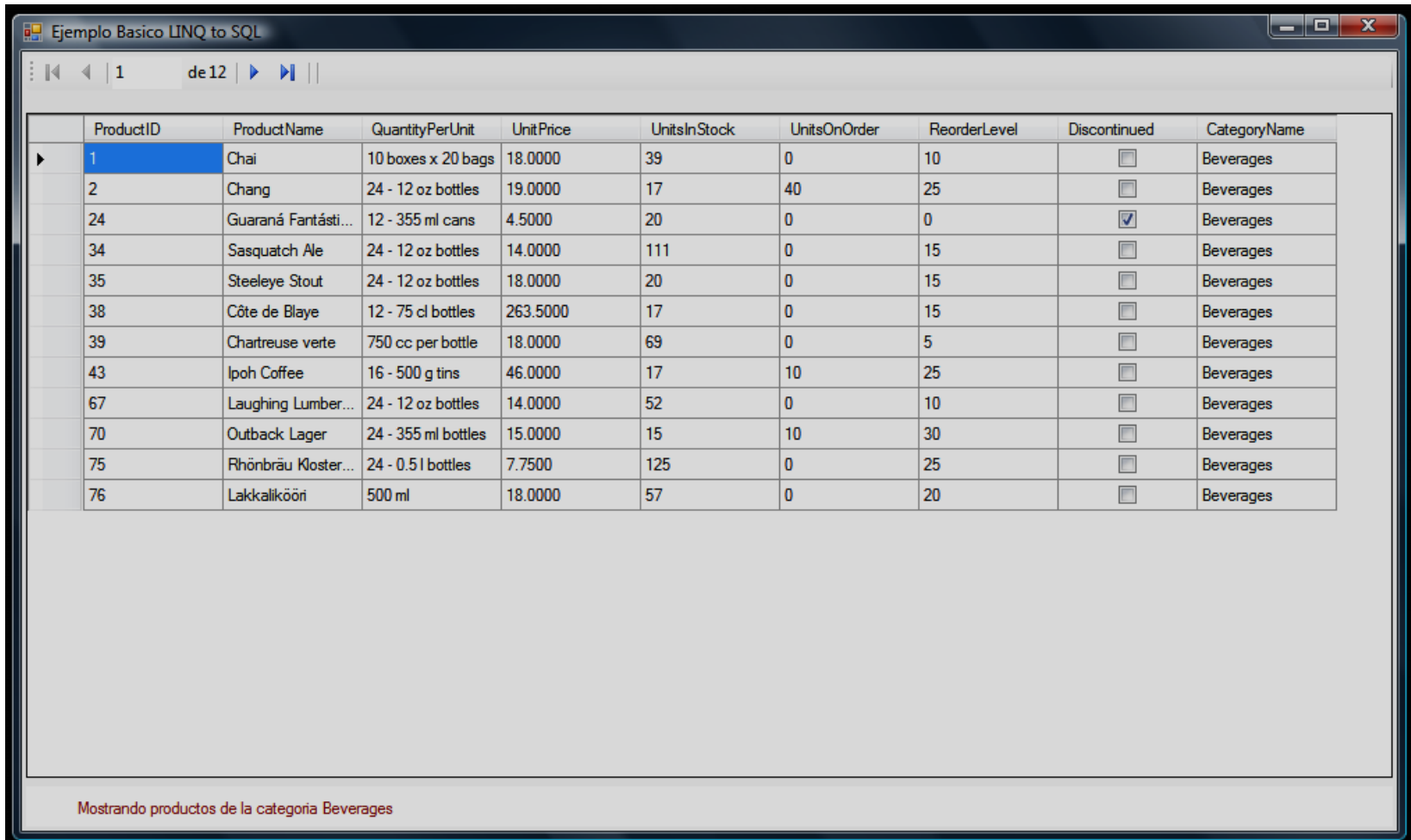
    BindingSource bind = new BindingSource();
    bind.DataSource = products.ToList(); //asignar el resultado de la consulta

    this.bndNavigator.BindingSource = bind;
    this.dgvDatos.DataSource         = bind;

    lblStatus.Text = "Mostrando productos de la categoria Beverages";

} //function
```

# LINQ to SQL



Ejemplo Basico LINQ to SQL

1 de 12

ProductID	ProductName	QuantityPerUnit	UnitPrice	UnitsInStock	UnitsOnOrder	ReorderLevel	Discontinued	CategoryName
1	Chai	10 boxes x 20 bags	18.0000	39	0	10	<input type="checkbox"/>	Beverages
2	Chang	24 - 12 oz bottles	19.0000	17	40	25	<input type="checkbox"/>	Beverages
24	Guaraná Fantástico	12 - 355 ml cans	4.5000	20	0	0	<input checked="" type="checkbox"/>	Beverages
34	Sasquatch Ale	24 - 12 oz bottles	14.0000	111	0	15	<input type="checkbox"/>	Beverages
35	Steeleye Stout	24 - 12 oz bottles	18.0000	20	0	15	<input type="checkbox"/>	Beverages
38	Côte de Blaye	12 - 75 cl bottles	263.5000	17	0	15	<input type="checkbox"/>	Beverages
39	Chartreuse verte	750 cc per bottle	18.0000	69	0	5	<input type="checkbox"/>	Beverages
43	Ipoh Coffee	16 - 500 g tins	46.0000	17	10	25	<input type="checkbox"/>	Beverages
67	Laughing Lumberjack Lager	24 - 12 oz bottles	14.0000	52	0	10	<input type="checkbox"/>	Beverages
70	Outback Lager	24 - 355 ml bottles	15.0000	15	10	30	<input type="checkbox"/>	Beverages
75	Rhönbräu Kloster Pils	24 - 0.5 l bottles	7.7500	125	0	25	<input type="checkbox"/>	Beverages
76	Lakkaikööri	500 ml	18.0000	57	0	20	<input type="checkbox"/>	Beverages

Mostrando productos de la categoria Beverages

- Este ejemplo muestra los registros en la tabla Products de la bd Northwind donde el nombre de la categoría es Beverages