

# ADO.NET (2/2)

---

*Herramientas Avanzadas para el Desarrollo de  
Aplicaciones*

1. Repaso
2. Acceso desconectado a BD
3. Controles de datos
4. Concurrencia
5. Acceso conectado vs desconectado

**1**

Repaso

## Objetos de acceso a datos (*ActiveX Data Objects*)

- ADO.NET es la tecnología que las aplicaciones .net utilizan para comunicarse con la BD.
- Optimizada para aplicaciones distribuidas (como aplicaciones web).
- Basada en XML
- Modelo de objetos completamente nuevo.
- **Entorno conectado vs desconectado.**

# Entorno conectado

CONEXIÓN ABIERTA

Form2

Request Update

Mostrar y modificar datos

Conectar a una base de datos

Solicitar datos específicos

Devolver datos

Transmitir actualizaciones

Cerrar la conexión

Base de datos

SIN CONEXIÓN

# Entorno conectado (II)

- Un entorno conectado es aquel en que los usuarios están conectados continuamente a una fuente de datos
- Ventajas:
  - El entorno es más fácil de mantener
  - La concurrencia se controla más fácilmente
  - Es más probable que los datos estén más actualizados que en otros escenarios
- Inconvenientes:
  - Debe existir una conexión de red constante
  - Escalabilidad limitada

# Entorno desconectado

CONEXIÓN ABIERTA



Conectar a una base de datos

Solicitar datos específicos

Devolver datos

Cerrar la conexión

Conectar a una base de datos

Transmitir actualizaciones

Cerrar la conexión

Base de datos

SIN CONEXIÓN

# Entorno desconectado (II)

- Un entorno desconectado es aquel en el que los datos pueden modificarse de forma independiente y los cambios se escriben posteriormente en la base de datos.
- Ventajas:
  - Las conexiones se utilizan durante el menor tiempo posible, permitiendo que menos conexiones den servicio a más usuarios
  - Un entorno desconectado mejora la escalabilidad y el rendimiento de las aplicaciones
- Inconvenientes:
  - Los datos no siempre están actualizados
  - Pueden producirse conflictos de cambios que deben solucionarse



# Tabla cliente...

WindowsApplication1 - Microsoft Visual Studio

File Edit View Project Build Debug Data Table Designer Tools Window Community Help

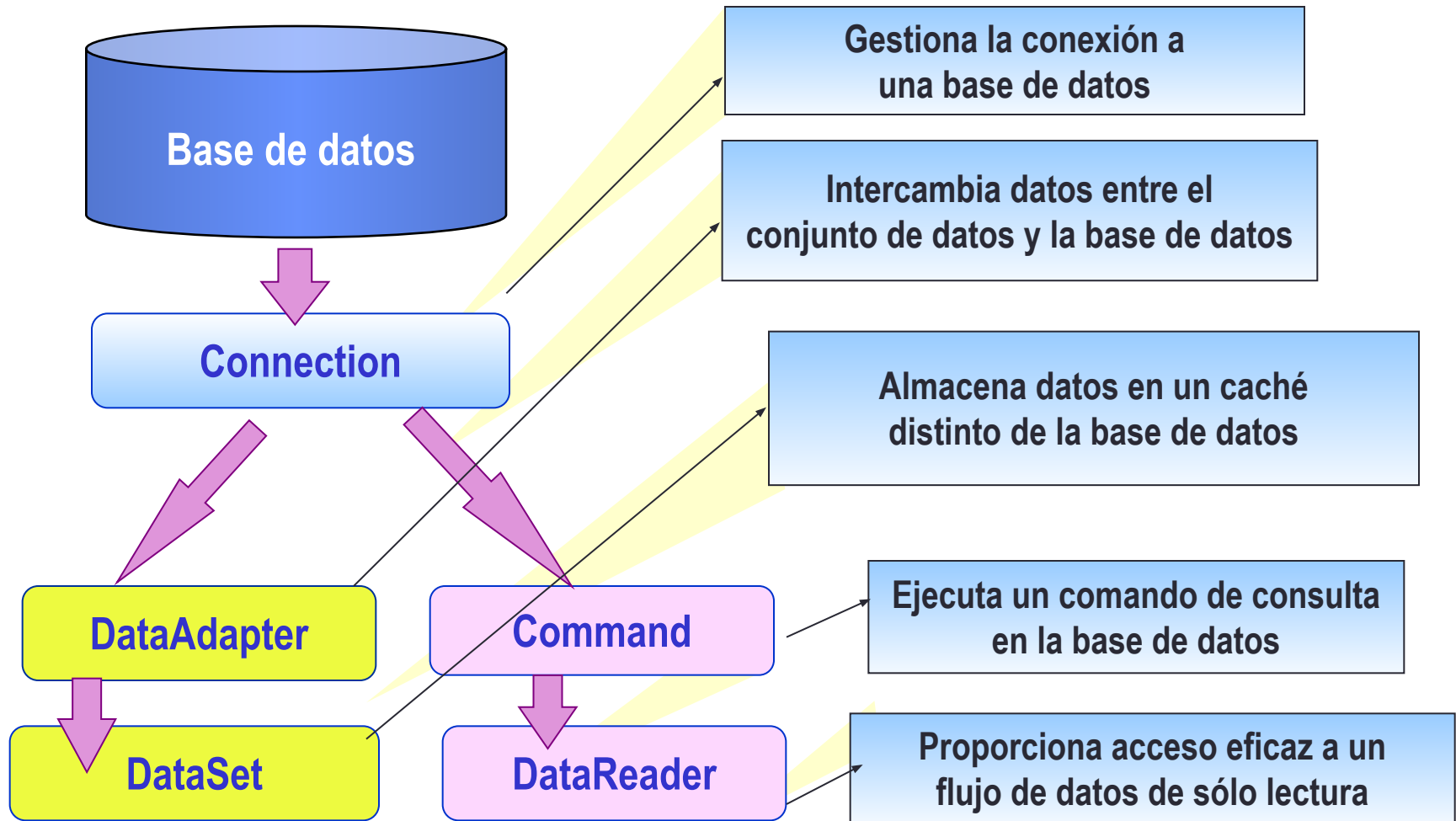
Server Explorer

- Data Connections
  - irene1\sqlexpress.prueba.dbo
  - irene1\sqlexpress.prueba1.dbo
    - Database Diagrams
    - Tables
      - cliente
        - usuario
        - contraseña
        - dni
    - Views
    - Stored Procedures
    - Functions
    - Synonyms
    - Types
    - Assemblies
  - Servers
    - irene1

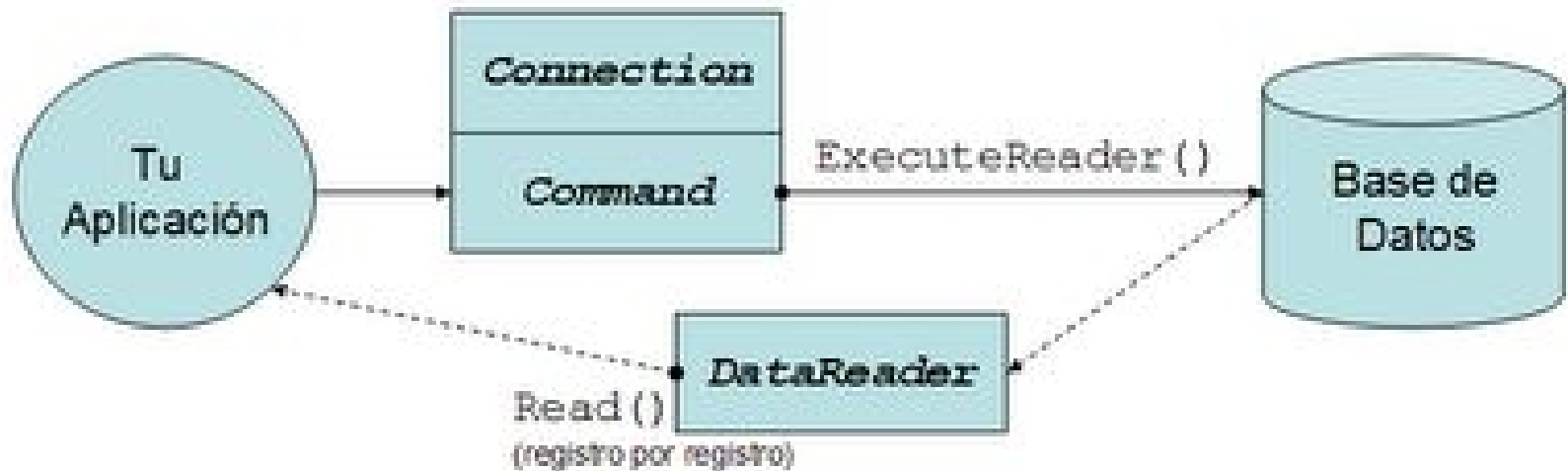
dbo.cliente: Tabl...qlexpress.prueba1)

Column Name	Data Type	Allow Nulls
usuario	nchar(10)	<input checked="" type="checkbox"/>
contraseña	nchar(10)	<input checked="" type="checkbox"/>
dni	nchar(10)	<input type="checkbox"/>
		<input type="checkbox"/>

# Objetos de ADO.NET



# Modo conectado



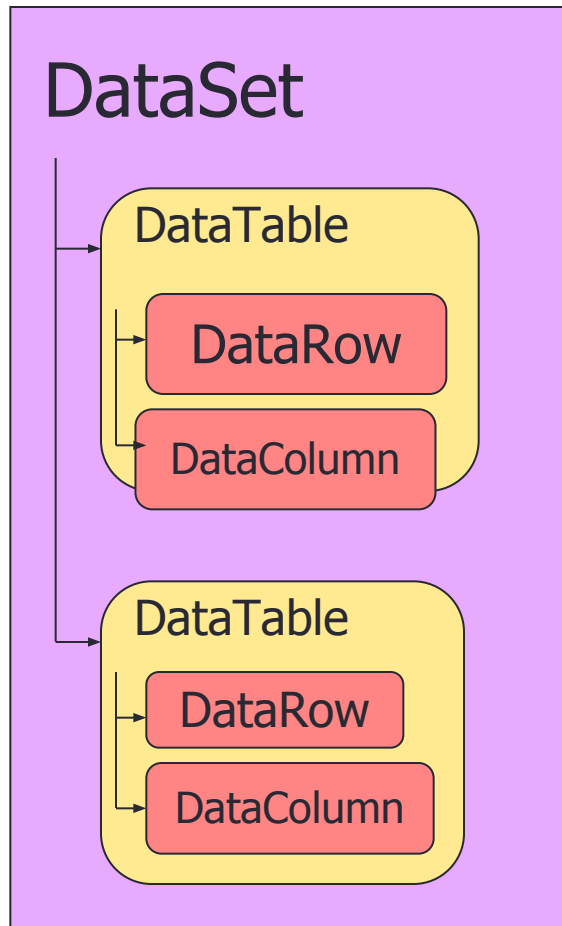
2

Acceso  
desconectado

# 1. Objeto Connection

- **Connection:** se utiliza para establecer las conexiones al proveedor de datos adecuado (método Open).

## 2. Objeto DataSet



- Nuevo objeto **DataSet**: representación de una **base de datos** relacional **en memoria**:
  - No necesidad de conexión continua.

# Objeto DataSet (II)

- Podemos trabajar con una BD que es copia de las partes con las que queremos trabajar de la BD real, liberando la conexión.
- Si queremos reflejar los cambios en la BD real, debemos confirmar nuestro objeto DataSet.

# Objetos DataRow, DataColumn

- DataRow
  - Representa una única fila de información de la tabla.
  - Se puede acceder a los valores individuales usando el nombre de campo o un índice.
- DataColumn
  - No contienen ningún dato real.
  - Almacenan información sobre la columna (tipo de datos, valor predeterminado, restricciones..)



# Objetos DataRelation, DataView

- **DataRelation**
  - Especifica una relación padre/hijo entre dos tablas diferentes de un objeto DataSet.
- **DataView**
  - Proporciona una vista sobre una DataTable.
  - Cada DataTable tiene al menos un DataView (a través de la propiedad DefaultView), que se usa para la vinculación de datos.
  - DataView muestra los datos de DataTable sin cambios o con opciones especiales de filtro u ordenación.

### 3. Objeto Adaptador

- El objeto Adaptador de datos se encarga de manejar la conexión por nosotros.
- Se utiliza para insertar datos en un objeto **DataSet**.
- El objeto **DataAdapter** utiliza comandos para actualizar el origen de datos después de hacer modificaciones en el objeto **DataSet**.

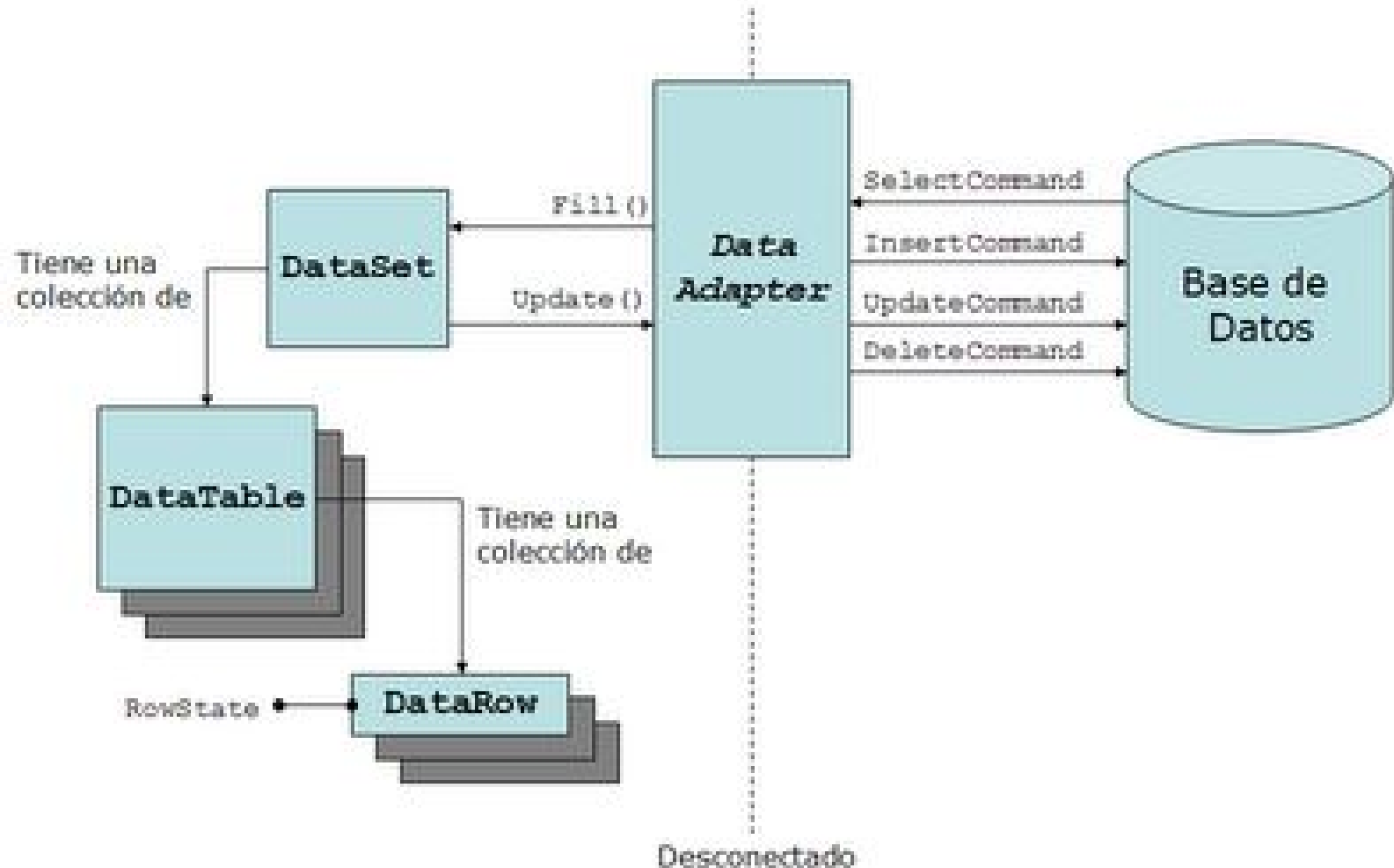
# DataAdapter (II)

- Una ventaja al usar un DataAdapter es que **no tengo que preocuparme por abrir y cerrar la conexión**. Estos métodos lo hacen automáticamente

## 4. Objeto CommandBuilder

- Este objeto (opcional) lo utiliza el **DataAdapter** para crear los comandos SQL necesarios.
- También se pueden proporcionar las sentencias SQL explícitamente o por medio de procedimientos almacenados.

# Modo desconectado



# Recordad...

- `System.Data.OleDb` y `System.Data.SqlClient`: clases responsables del acceso a datos desde fuentes SQL Server y OLE DB.
- Incluyen clases que al trabajar con SQL llevarán el prefijo `Sql` y al emplear Ole DB llevarán `OleDb`:
  - `SqlConnection` y `OleDbConnection`
  - `SqlDataAdapter` y `OleDbDataAdapter`
  - `SqlCommandBuilder` y `OleDbCommandBuilder`
  - **PERO NO CON `DataSet` (ni `dataRow`, `dataColumn`...)**

# EJEMPLO: inserción de datos

Usuario	<input type="text" value="luis"/>
Contraseña	<input type="text" value="1256"/>
DNI	<input type="text" value="55555"/>
<input type="button" value="Insertar usuario"/>	

# Conexión a una BD en Sql Server

## Importar namespaces

```
using System.Data;  
using System.Data.Common;  
using System.Data.SqlClient;  
using System.Data.SqlTypes;
```



## Crear la conexión


```
string s = "data source=.\SQLEXPRESS;Integrated  
Security=SSPI;AttachDBFilename=|DataDirectory|\\Database1.mdf;  
User Instance=true";  
SqlConnection c=new SqlConnection(s);
```

## Definición de un comando Select

- Para recuperar los datos se necesita:
  - Una sentencia SQL que seleccione la información deseada
  - Un objeto Command que ejecute la sentencia SQL
  - Un objeto DataReader que capture los registros recuperados

**ATENCIÓN:** En modo desconectado siempre necesitamos recuperar los datos (SELECT) para poder trabajar con ellos (INSERT, UPDATE, DELETE)

## Definición de un comando Select

- Para recuperar los datos se necesita:
    - ~~Una sentencia SQL que seleccione la información deseada~~
    - ~~Un objeto Command que ejecute la sentencia SQL~~
    - ~~Un objeto DataReader que capture los registros recuperados~~
  - Un objeto **DataAdapter** que ejecute la sentencia SQL
  - Un objeto **DATASET** donde guardar el resultado de la sentencia SQL
- 

## Objetos DataSet y DataAdapter

- Creamos una BD virtual, mediante un objeto DataSet

```
DataSet bdvirtual = new DataSet();
```

- La llenamos con las tablas que estamos interesados en trabajar:
  - Objeto DataAdapter
    - Método Fill()

...

```
SqlDataAdapter da = new  
SqlDataAdapter("select * from Cliente", c);  
  
da.Fill(bdvirtual,"cliente");
```

# Ahora trabajamos en local

- Ahora en “bdvirtual” tenemos nuestra base de datos local.

# Para trabajar en local

- Lo hacemos modificando filas y columnas de las tablas almacenadas en local.
- En dataset tenemos almacenada la bd virtual, copiamos a un datatable la tabla a modificar.

```
DataTable t = new DataTable();  
t = bdvirtual.Tables["cliente"];
```

# Operaciones

- Obtener una tabla:
  - `DataTable t = new DataTable();`
  - `t = bdvirtual.Tables["cliente"];`
- Acceder a los elementos filas de esa tabla (podemos usar un bucle):
  - `DataRow fila = t.Rows[0];`
  - `fila[0] = "Andrés";`
  - **IGUAL QUE**
  - `t.Rows[0][0] = "Andrés";`



# Primera fila segunda columna:

- `t.Rows[0][1]`
- Información de las columnas: (nombre, tipo)
  - `t.Columns[0].ColumnName`
  - `t.Columns[0].DataType`

# Queremos insertar un nuevo cliente

- Esto equivale a insertar una fila en nuestra tabla local...

```
DataRow nuevafila = t.NewRow();  
nuevafila[0] = textBox1.Text;  
nuevafila[1] = textBox2.Text;  
nuevafila[2] = textBox3.Text;  
t.Rows.Add(nuevafila);
```

# Validar los cambios

- **Objeto Adaptador:**
  - nos ha servido para llenar la tabla, también para actualizar los datos en la BD real.
- **Método Update**
  - El *DataAdapter* analizará los cambios que se han hecho en el *DataSet* y ejecutará los comandos apropiados para *insertar*, *actualizar* o *borrar* en la BD real.

# Constructor de comandos

- Objeto CommandBuilder:
  - Constructor de comandos
  - Le pasamos como argumento el adaptador
  - Construye los comandos SQL que nos hagan falta para actuar sobre la BD

```
• SqlCommandBuilder cbuilder = new SqlCommandBuilder(da);  
• da.Update(bdvirtual, "cliente");  
• label4.Text = "Cambiado";
```

# Ahora faltaría actualizar los cambios en la BD real

- Objeto CommandBuilder
- Objeto DataAdapter → Método UPDATE

```
SqlCommandBuilder cbuilder = new  
    SqlCommandBuilder(da);
```

```
da.Update(bdvirtual, "cliente");
```

```
string s = "data source=.\SQLEXPRESS;Integrated  
Security=SSPI;AttachDBFilename=|DataDirectory|\\Database1.mdf;User  
Instance=true";  
SqlConnection c=new SqlConnection(s);  
DataSet bdvirtual = new DataSet();  
SqlDataAdapter da = new SqlDataAdapter("select * from Cliente", c);  
da.Fill(bdvirtual, "cliente");
```

```
DataTable t = new DataTable();  
t = bdvirtual.Tables["cliente"];  
DataRow nuevafila = t.NewRow();  
nuevafila[0] = textBox1.Text;  
nuevafila[1] = textBox2.Text;  
nuevafila[2] = textBox3.Text;  
t.Rows.Add(nuevafila);
```

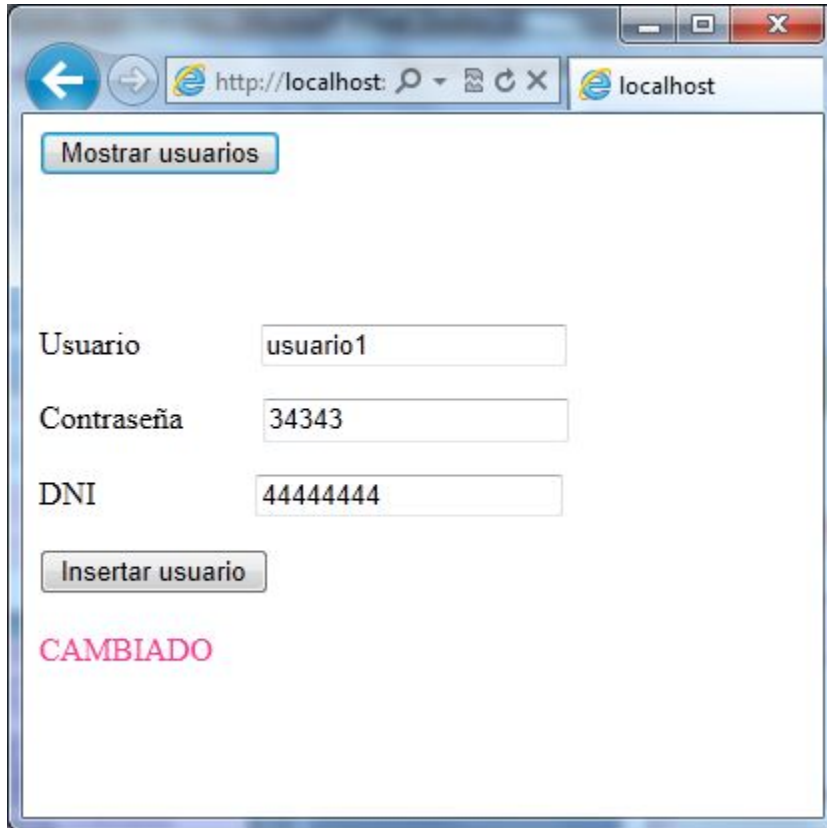
```
SqlCommandBuilder cbuilder = new SqlCommandBuilder(da);  
da.Update(bdvirtual, "cliente");  
label4.Text = "CAMBIADO";
```

**¿Qué faltaría en el código?**

# En las 3 capas, modificamos la función CAD

```
public bool InsertarCliente(ENCliente cli)    {
    bool cambiado;
    ENCliente cl = cli;
    DataSet bdvirtual = new DataSet() ;
    SqlConnection c = new SqlConnection(s);
    try    {
        SqlDataAdapter da = new SqlDataAdapter("select * from Cliente",c);
        da.Fill(bdvirtual,"cliente");
        DataTable t = new DataTable();
        t = bdvirtual.Tables["cliente"];
        DataRow nuevafila = t.NewRow();
        nuevafila[0] = cl.Usuario;
        nuevafila[1] = cl.Contraseña;
        nuevafila[2] = cl.Dni;
        t.Rows.Add(nuevafila);
        SqlCommandBuilder cbuilder = new SqlCommandBuilder(da);
        da.Update(bdvirtual, "cliente");
        cambiado = true;    }
    catch (Exception ex) {    cambiado = false;    }
    finally    {    c.Close();    }
    return cambiado; }
```

# Ejecución



A screenshot of a web browser window showing a user management interface. The browser's address bar displays 'http://localhost:'. The page has a blue header with a 'Mostrar usuarios' button. Below this, there are three input fields: 'Usuario' with the value 'usuario1', 'Contraseña' with the value '34343', and 'DNI' with the value '44444444'. Below these fields is an 'Insertar usuario' button. At the bottom of the page, the word 'CAMBIADO' is displayed in pink.

Mostrar usuarios

Usuario

Contraseña

DNI

Insertar usuario

CAMBIADO



A screenshot of a SQL Server Enterprise Manager window showing a table named 'usuario'. The table has four columns: 'usuario', 'contraseña', and 'dni'. The table contains several rows of user data. The first row is highlighted in blue.

	usuario	contraseña	dni
►	irenee	345	4343
	usuario1	34343	44444444
	laura	123	45698
	irene	123	45896
	luis	1256	55555
	jose	258	85964
	ii	343	rr
*	NULL	NULL	NULL



3

Control GridView

# Control GridView

- Control para presentación de datos en forma de tabla (filas y columnas)
- Propiedades:
  - Selección
  - Paginación
  - Ordenación
  - Edición
  - Extensible mediante plantillas
- [http://msdn.microsoft.com/es-es/library/cc295223\(v=expression.40\).aspx](http://msdn.microsoft.com/es-es/library/cc295223(v=expression.40).aspx)

# GridView

The screenshot shows the Visual Studio IDE with the 'Datos' (Data) menu open. The 'GridView' control is selected, and its tasks are displayed on the right. The tasks include 'Formato automático...', 'Elegir origen de datos:' (set to '(Ninguno)'), 'Editar columnas...', 'Agregar nueva columna...', and 'Editar plantillas'.

**Datos**

- Puntero
- GridView
- DataList
- DetailsView
- FormView
- ListView
- Repeater
- DataPager
- SqlDataSource
- AccessDataSource
- LinqDataSource
- ObjectDataSource
- XmlDataSource
- SiteMapDataSource

**asp:gridview#GridView1**

Column0	Column1	Column2
abc	abc	abc
abc	abc	abc
abc	abc	abc
abc	abc	abc
abc	abc	abc

**Tareas de GridView**

- Formato automático...
- Elegir origen de datos: (Ninguno)
- Editar columnas...
- Agregar nueva columna...
- Editar plantillas

**asp:sqldatasource#SqlDataSource1**

**SqlDataSource - SqlDataSource1**

**Tareas de SqlDataSource**

- Configurar origen de datos...

## Objetivo:

Vamos a mostrar en un gridview los datos de la tabla cliente.

- *Con el asistente*
- *Con código*

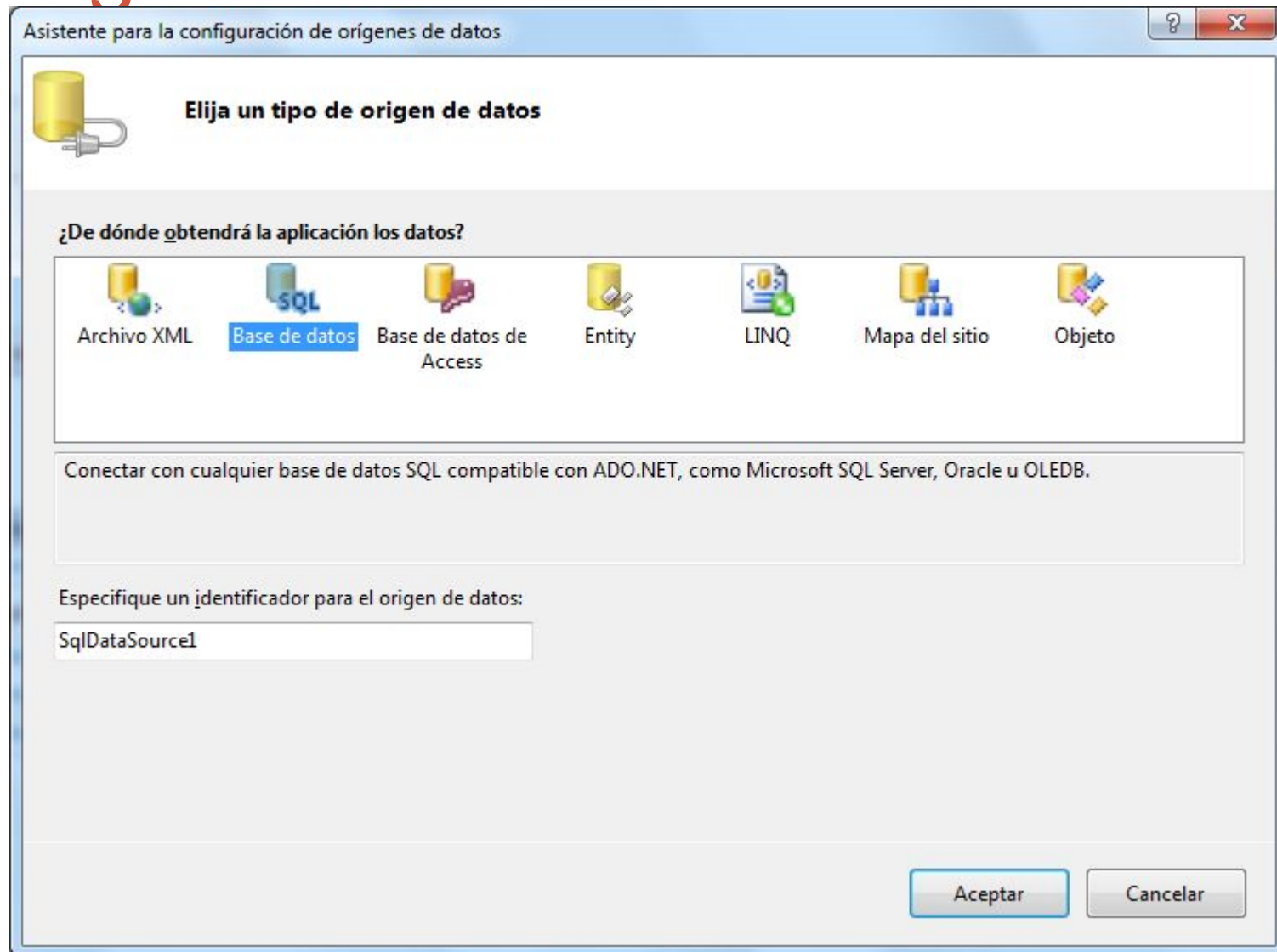
# GridView → DataSource

- Añadir un objeto GridView y elegir como origen de datos la bd.

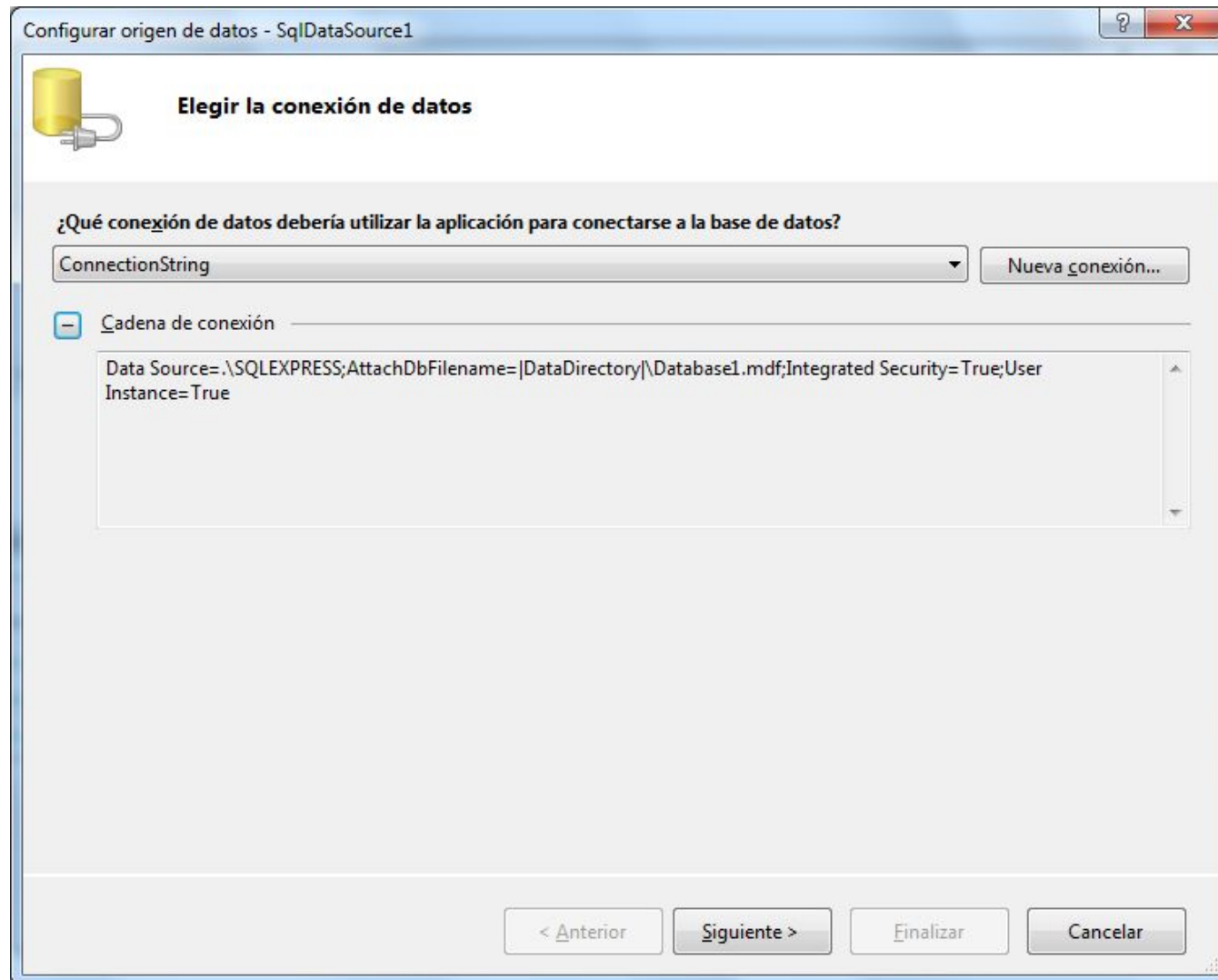
# Control GridView

- Podemos asignarle una tabla de la BD como origen de datos, o los registros obtenidos como resultado de una sentencia SQL.

# Elegir BD



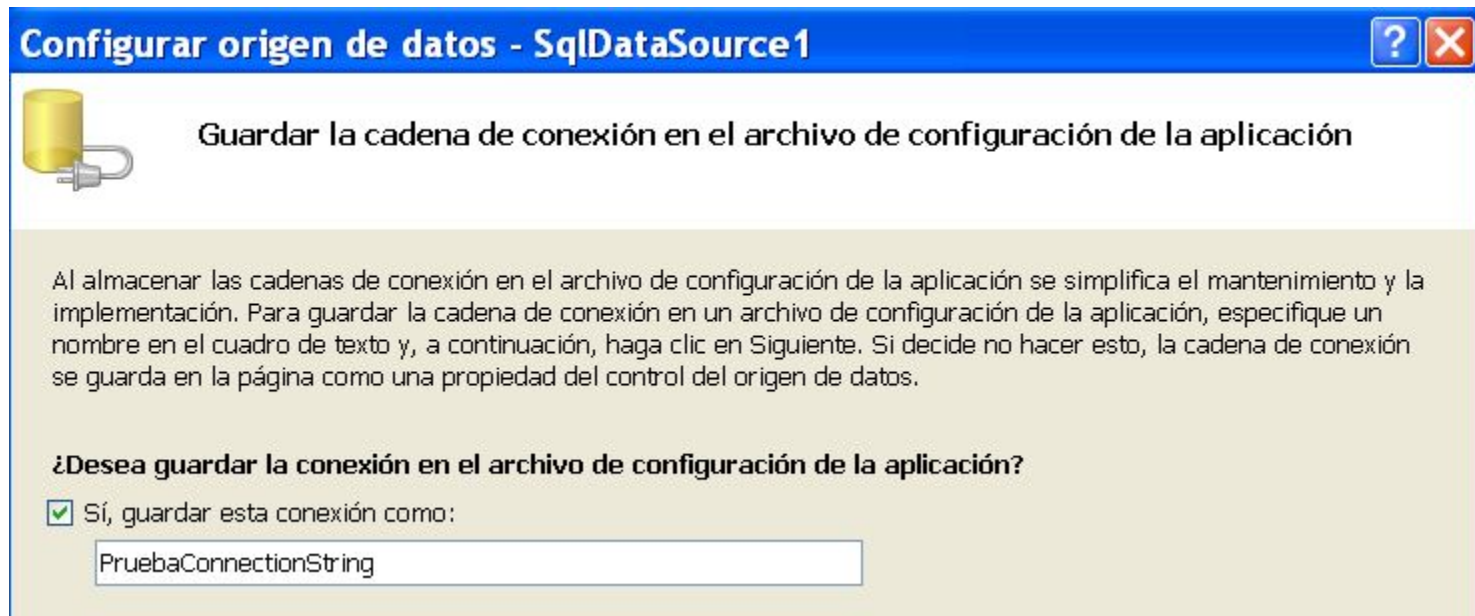
# Elegir conexión de datos






# GridView

- Guardamos la cadena de conexión en el archivo Web.config



Configurar origen de datos - SqlDataSource1

 **Configurar la instrucción Select**

¿Cómo desea recuperar los datos de la base de datos?

☐ Especificar una instrucción SQL o un procedimiento almacenado personalizado

☒ Especificar columnas de una tabla o vista

Nombre:

cliente

Columnas:

☒ \*

☐ usuario

☐ contraseña

☐ dni

☐ Devolver sólo filas únicas

WHERE...

ORDER BY...

Avanzadas...

Instrucción SELect:

SELECT \* FROM [cliente]

< Anterior

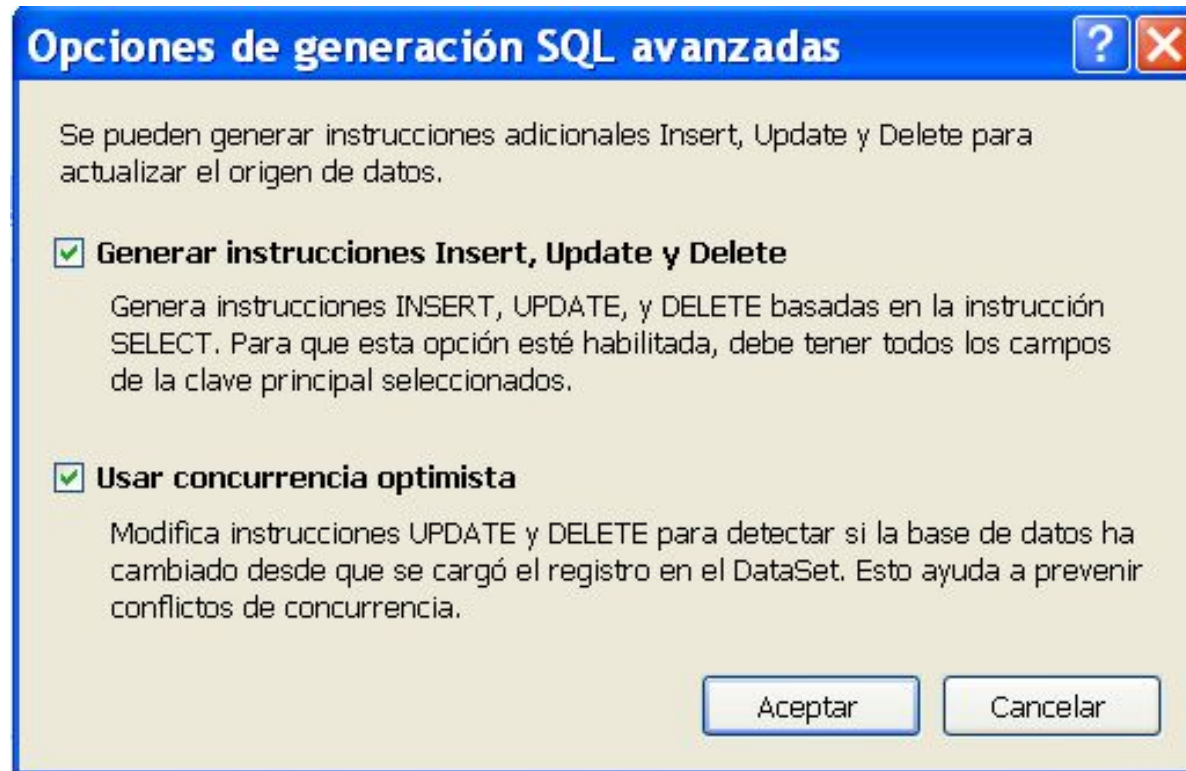
Siguiente >

Finalizar

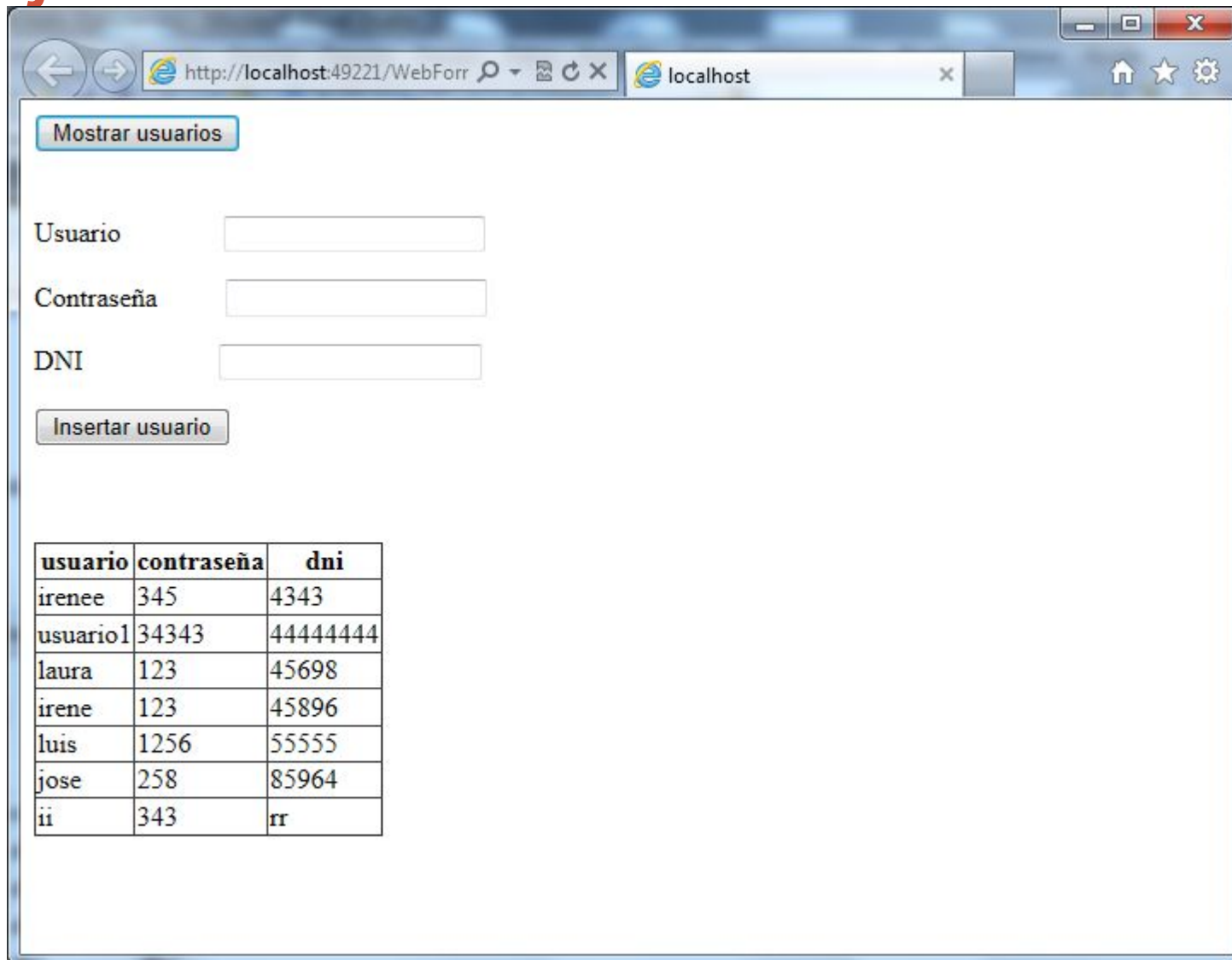
Cancelar

# GridView (asistente)

- Avanzadas
  - Podemos generar las instrucciones Insert, Update y Delete



# Ejecución...



Mostrar usuarios

Usuario

Contraseña

DNI

Insertar usuario

usuario	contraseña	dni
irenee	345	4343
usuario1	34343	44444444
laura	123	45698
irene	123	45896
luis	1256	55555
jose	258	85964
ii	343	rr

# Código para mostrar datos de un dataset

```
ENCliente enl = new ENCliente();  
DataSet d = new DataSet();  
  
protected void Page_Load(object sender, EventArgs e)  
{  
  
    if (!Page.IsPostBack)  
    {  
        d = enl.listarClientesD();  
  
        GridView1.DataSource = d;  
        GridView1.DataBind();  
    }  
  
}
```

# Donde en el CAD...

```
public DataSet ListarClientesD()
{
    DataSet bdvirtual = new DataSet();

    SqlConnection c = new SqlConnection(s);
    SqlDataAdapter da = new SqlDataAdapter("select * from Cliente", c);
    da.Fill(bdvirtual, "cliente");
    return bdvirtual;
}
```

# Ejercicio



## Ejercicio

- Modificar el formulario anterior para editar los datos de una persona en la BD.

5 min



# AutoGenerateSelectButton=true

Usuario

Contraseña

DNI

Insertar usuario

Editar usuario

	<u>usuario</u>	<u>contraseña</u>	<u>dni</u>
<a href="#">Seleccionar</a>	laura	1237	44444444
<a href="#">Seleccionar</a>	laura2	1239	4569
<a href="#">Seleccionar</a>	irene	123	45896
<a href="#">Seleccionar</a>	luis	1256	55555
<a href="#">Seleccionar</a>	jose	258	85964
1 <u>2</u>			



# Evento SelectedIndexChanged

```
protected void GridView2_SelectedIndexChanged(object sender,
EventArgs e)
{
    TextBox1.Text = GridView2.SelectedRow.Cells[1].Text;
    TextBox2.Text = GridView2.SelectedRow.Cells[2].Text;
    TextBox3.Text = GridView2.SelectedRow.Cells[3].Text;
    TextBox3.Enabled = false;
}
```

# Botón editar (click)

```
{  
  
    ENCliente en = new ENCliente();  
    en.Usuario = TextBox1.Text;  
    en.Contraseña = TextBox2.Text;  
    d = en.ModificarCliente(GridView2.SelectedIndex);  
  
    GridView2.DataSource = d;  
    GridView2.DataBind();  
}
```

# EN

```
public DataSet ModificarCliente(int i)
{
    CADcliente c = new CADcliente();
    DataSet a = c.ModificarCliente(this,i);
    return a;
}
```

# CAD (simplificado)

```
public DataSet ModificarCliente(ENCliente cli, int i)
{
    ENCliente cl = cli;
    DataSet bdvirtual = new DataSet();
    SqlConnection c = new SqlConnection(s); SqlDataAdapter da = new
    SqlDataAdapter("select * from Cliente", c);
        da.Fill(bdvirtual, "cliente");
        DataTable t = new DataTable();
        t = bdvirtual.Tables["cliente"];

        t.Rows[i]["usuario"]=cl.Usuario;
        t.Rows[i]["contraseña"] = cl.Contraseña;

        SqlCommandBuilder cbuilder = new SqlCommandBuilder(da);
        da.Update(bdvirtual, "cliente");

        return bdvirtual;
}
```

# Ejercicio



## Ejercicio

- Modificar el formulario anterior para eliminar los datos de una persona en la BD.

5 min



# AutoGenerateDeleteButton=true

```
protected void GridView2_RowDeleting(object sender, GridViewDeleteEventArgs e)
{
    ENCliente en = new ENCliente();

    d = en.BorrarCliente(e.RowIndex);

    GridView2.DataSource = d;
    GridView2.DataBind();
}
```

# EN

```
public DataSet BorrarCliente(int i)
{
    CADcliente c = new CADcliente();
    DataSet a = c.BorrarCliente(this, i);
    return a;
}
```

# CAD

```
public DataSet BorrarCliente(ENCliente cli, int i)
{
    ENCliente cl = cli;
    DataSet bdvirtual = new DataSet();
    SqlConnection c = new SqlConnection(s);

    SqlDataAdapter da = new SqlDataAdapter("select * from Cliente", c);
    da.Fill(bdvirtual, "cliente");
    DataTable t = new DataTable();
    t = bdvirtual.Tables["cliente"];

    t.Rows[i].Delete();

    SqlCommandBuilder cbuilder = new SqlCommandBuilder(da);
    da.Update(bdvirtual, "cliente");

    return bdvirtual;    }
```

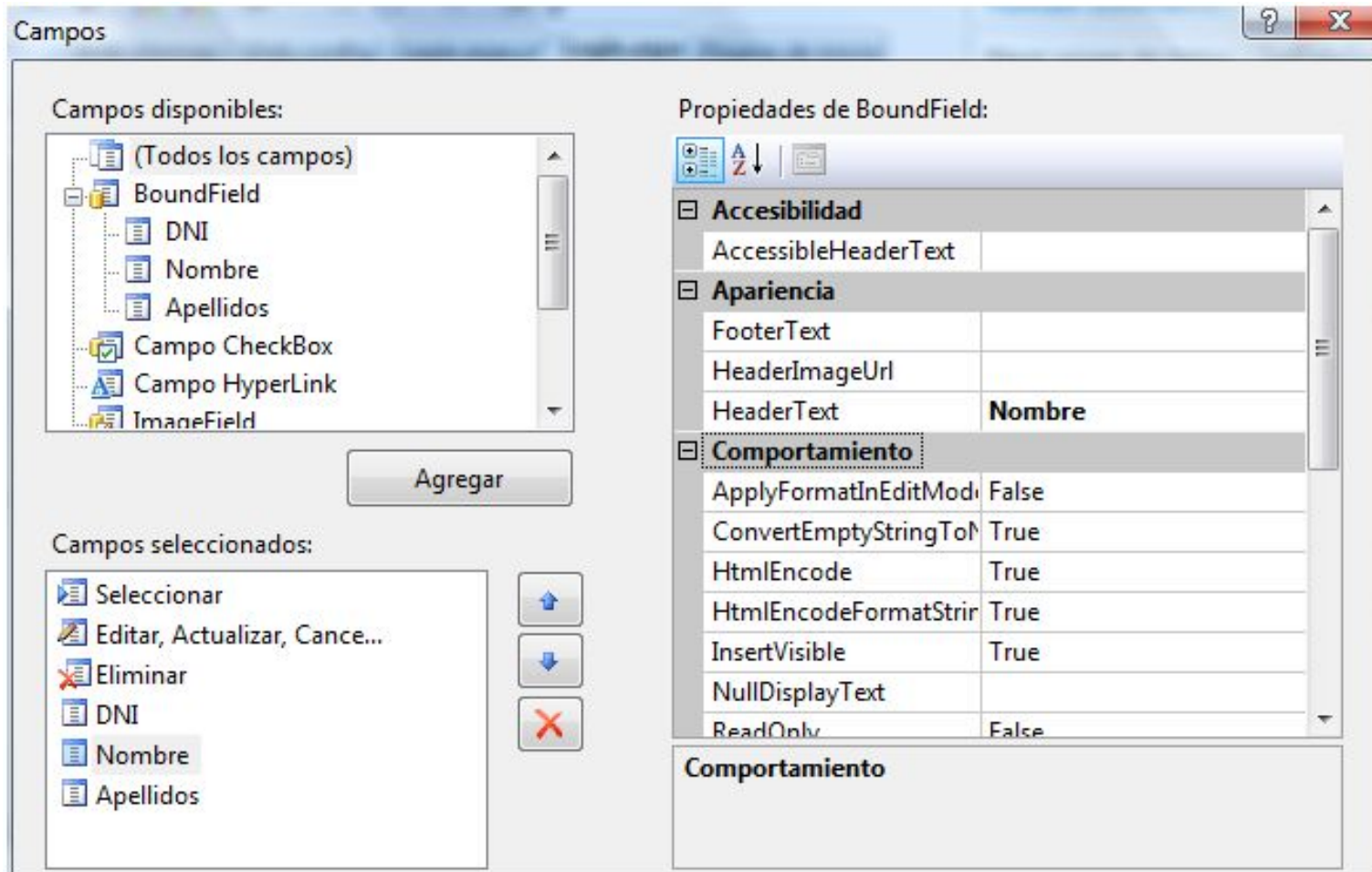


# Paginación en GridView

- Propiedades
  - AllowPaging = true
  - PageSize = 5 (numero de elementos por página)
- Al mostrar los datos con asistente no hay que escribir código, pero al enlazarlo nosotros necesitamos escribir el código para el evento

```
protected void GridView2_PageIndexChanging(object sender,  
    GridViewPageEventArgs e)  
{  
    d = enl.listarClientesD();  
    GridView2.PageIndex = e.NewPageIndex;  
    GridView2.DataSource = d;  
    GridView2.DataBind();  
}
```

# GridView: editar columnas



# GridView

- Tipos de columnas:
  - BoundField: Muestra el texto de un campo de la BBDD
  - ButtonField: Muestra un botón para cada item
  - CheckBoxField: Muestra un checkbox para cada item
  - CommandField: Proporciona funciones de selección, edición y borrado
  - HyperLinkField: Muestra el texto de un campo de la BBDD como un hipervínculo
  - ImageField: Muestra una imagen
  - TemplateField: Permite especificar múltiples campos y controles personalizados

# GridView

- Modificamos el aspecto del GridView

asp:gridview#GridView1

			<u>DNI</u>	<u>Nombre</u>	<u>Apellidos</u>
Seleccionar	Editar	Eliminar	abc	abc	abc
Seleccionar	Editar	Eliminar	abc	abc	abc
Seleccionar	Editar	Eliminar	abc	abc	abc
Seleccionar	Editar	Eliminar	abc	abc	abc
Seleccionar	Editar	Eliminar	abc	abc	abc

1 2

SqlDataSource - SqlDataSource1

**Tareas de GridView**

- Formato automático...
- Elegir origen de datos: SqlDataSource1
- Configurar origen de datos...
- Actualizar esquema
- Editar columnas...
- Agregar nueva columna...
- ☒ Habilitar paginación
- ☒ Habilitar ordenación
- ☒ Habilitar edición
- ☒ Habilitar eliminación
- ☒ Habilitar selección
- Editar plantillas

# GridView

- Resultado final

			<u>DNI</u>	<u>Nombre</u>	<u>Apellidos</u>
<div>Actualizar</div> <div>Cancelar</div>			11111111	<div>Laura</div>	<div>Sanchez</div>
<div>Seleccionar</div>	<div>Editar</div>	<div>Eliminar</div>	22222222	Alberto	Lopez
<div>Seleccionar</div>	<div>Editar</div>	<div>Eliminar</div>	44444444	Juan	Perez
<div>Seleccionar</div>	<div>Editar</div>	<div>Eliminar</div>	55555555	Sara	Jover
<div>Seleccionar</div>	<div>Editar</div>	<div>Eliminar</div>	66666666	Berta	Belda
			1 2		

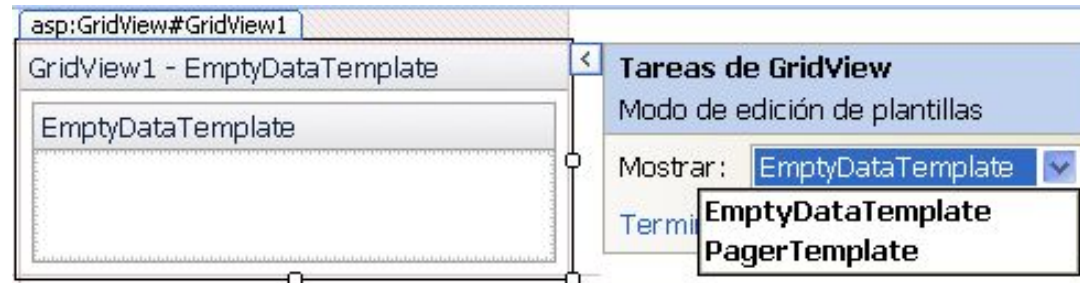
# Editar plantillas

- **EmptyDataText**

- Se utiliza para mostrar un mensaje cuando no existen datos que mostrar en el GridView

- **EmptyDataTemplate**

- Podemos personalizar el mensaje mostrado cuando el GridView está vacío.



No se encontraron datos referentes  
a su consulta. Inténtelo de nuevo.

Buscar

**4**

Concurrencia

# Entorno desconectado: conflictos

- En un entorno desconectado, varios usuarios pueden modificar los datos de los mismos registros al mismo tiempo.
- **Formas de gestión del conflicto:**
  - Concurrencia pesimista.
  - Concurrencia positiva.
  - Last Win.
  - Escribir código para gestionar el conflicto.



# Concurrencia

- **Concurrencia pesimista:** Cuando una fila es leída, esta queda bloqueada para su lectura para cualquier otro que la demande hasta que aquel que la posee la libere.

# Concurrencia (I)

- **Concurrencia positiva:** Las filas están disponibles para su lectura en todo momento, estas pueden ser leídas por distintos usuarios al mismo tiempo.
- Cuando alguno intenta modificar una fila ya modificada se produce un error y no se modifica.

# Concurrencia (II)

- “**Last win**”: esta técnica implica que no existe control. El último cambio en escribirse es el que permanece.

# ADO.NET: Concurrency positiva

- El objeto DataSet mantiene dos versiones de las filas que leímos:
  - Versión original, idéntica a la leída en la BD
  - Versión actualizada, representa los cambios del usuario
- Cuando se actualiza la fila, se comparan los valores originales con la fila real de la BD, para ver si ha sido modificada.
  - Si ha sido modificada, hay que capturar una excepción
  - Sino, la actualización es efectuada

# Evento RowUpdated

- Escribir código en la aplicación que permita a los usuarios determinar qué cambios deberían conservarse. Las soluciones específicas pueden variar dependiendo de los requerimientos de negocio de una determinada aplicación.
- Evento RowUpdated:
  - Al actualizar una fila: después de cada operación pero antes de lanzar cualquier excepción.
  - Podemos examinar los resultados e impedir que se lance una excepción.

5

Conectado vs  
Desconectado

# Conectado vs Desconectado

- Acceso conectado a datos (conexión viva)
  - **DataReader**
    - Podemos recuperar rápidamente todos los resultados.
    - Utiliza una conexión viva. Más ligero y veloz que DataSet
    - Acceso a los resultados sólo hacia delante de sólo lectura.
    - Mejor rendimiento que DataSet, por lo que es mejor elección para acceso a datos simple.
- Acceso desconectado a datos
  - **DataSet**

# Conectado vs Desconectado

- Para realizar consultas de sólo lectura, que únicamente serán necesarias realizarlas una vez (no tendremos que volver a acceder a filas anteriores) el objeto recomendado es **DataReader**.
- *Por ejemplo, para comprobar si un artículo se encuentra entre una tabla que guarda la lista de artículos del inventario de un almacén, basta con realizar una única consulta de sólo lectura.*
- Sin embargo, si vamos a realizar un acceso a datos más complicado, como puede ser la consulta de todos los artículos de diferentes tipos que pertenecen a un proveedor, la elección correcta sería utilizar **DataSet**.



# Conectado vs Desconectado

- **Acceso a datos.**
  - Como hemos dicho, si tenemos previsto recibir y almacenar datos, optamos por *DataSet*, ya que *DataReader* sólo permite lecturas.
- **Trabajar con más de una tabla o más de una base de datos.**
  - Si la función que estamos desarrollando requiere información situada en varias tablas de una misma base de datos o de varias, utilizaremos el objeto *DataSet*. Con *DataReader* sólo podemos construir consultas SQL que accedan a una base de datos.