Hada T6: ADO.NET

Herramientas Avanzadas para el Desarrollo de Aplicaciones

Objetivos

- ADO.net 2.0
- Creación de una BD SQL desde VStudio.net
- Acceso conectado a BD
- Creación cadena conexión: Web.config
- Propiedad DataDirectory

ADO.Net 2.0

1

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

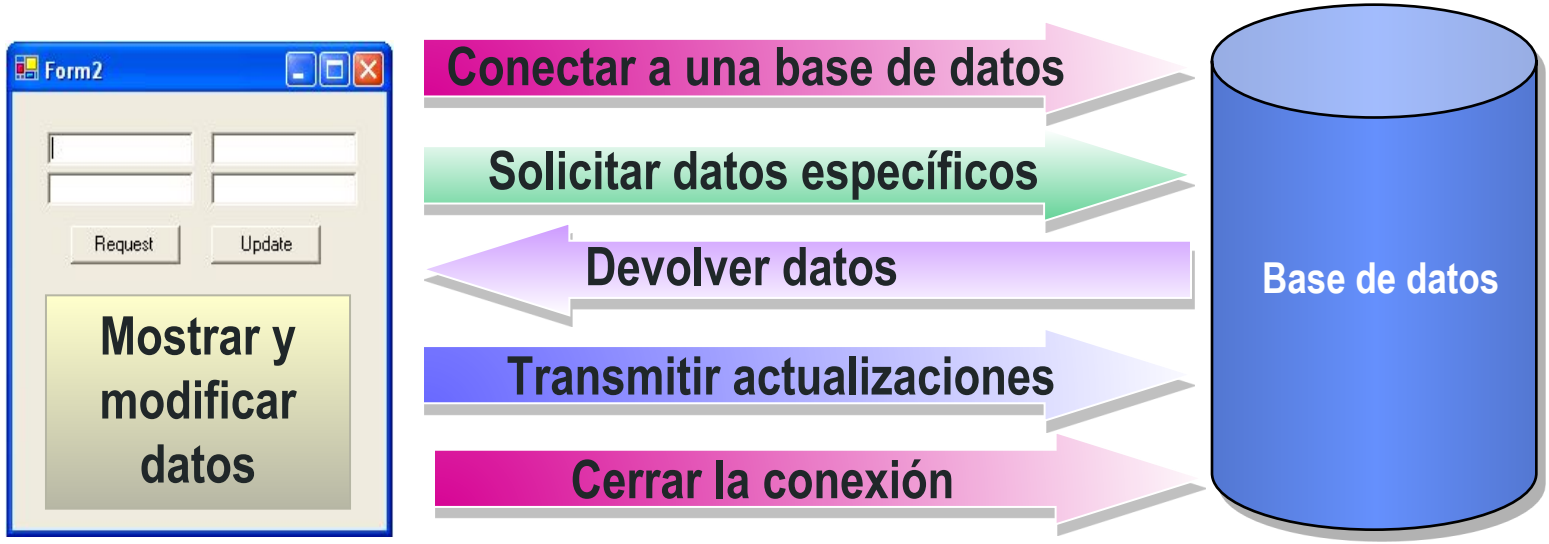
- ADO.NET es la tecnología que las aplicaciones asp.net utilizan para comunicarse con la BD.
- Optimizada para aplicaciones distribuidas (como aplicaciones web).
- Basada en XML
- Modelo de objetos completamente nuevo.
- **Entorno conectado vs desconectado.**

Entorno conectado

- 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 conectado (II)

CONEXIÓN ABIERTA



SIN CONEXIÓN

Entorno desconectado

- 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

Entorno desconectado (II)

CONEXIÓN ABIERTA

SIN CONEXIÓN

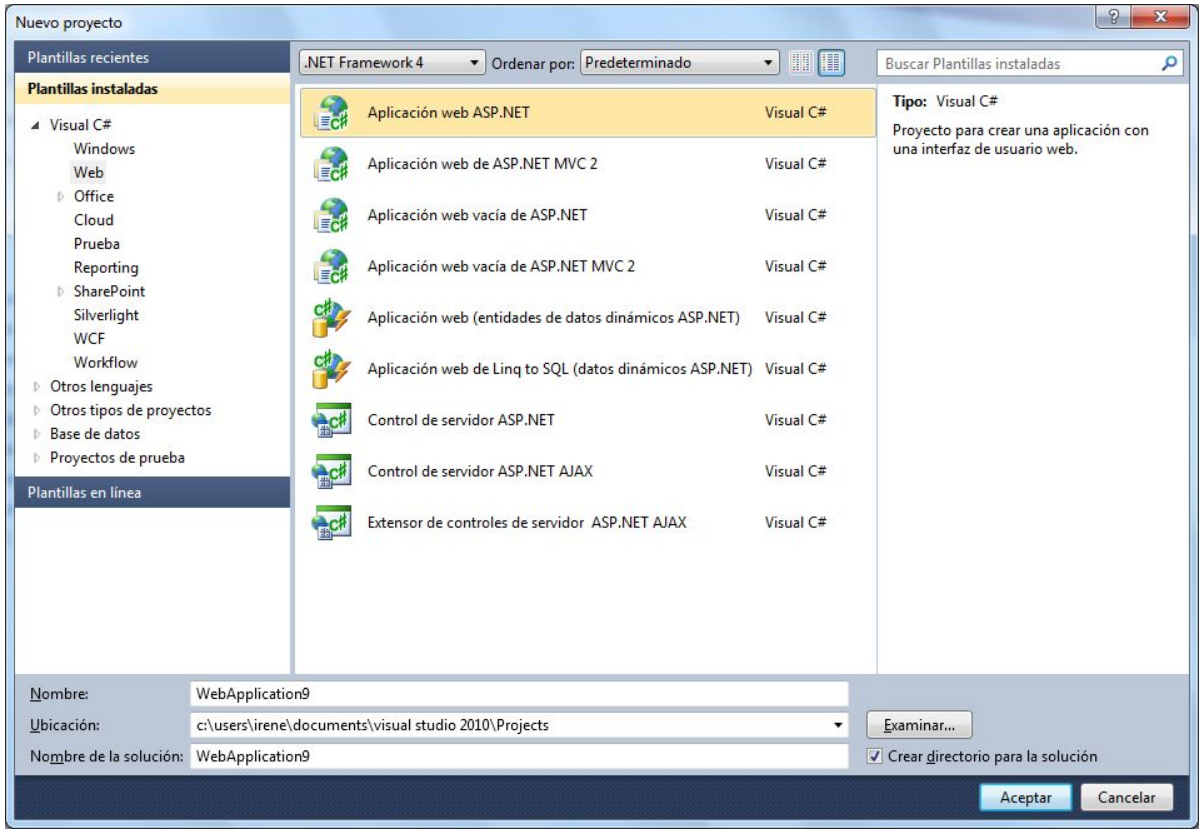
Ejemplos de entornos

- Un supermercado, donde en cada punto de venta (TPV) se almacenan las ventas realizadas. Cada cierto tiempo se las ventas se han de actualizar en la BD central.
- **Una fábrica que requiere una conexión en tiempo real para controlar la salida de producción y el almacén.**
- Una aplicación que mantiene datos de clientes en un equipo portátil de un representante.
- **Una aplicación que hace un seguimiento de lluvias y precipitaciones.**
- Un agente de bolsa que requiere una conexión constante a los valores del mercado.
- **Una aplicación que un granjero utiliza para contar el ganado. La aplicación está en el dispositivo basado en Microsoft Windows CE del granjero que ejecuta Microsoft SQL Server 2000 Windows CE Edition.**

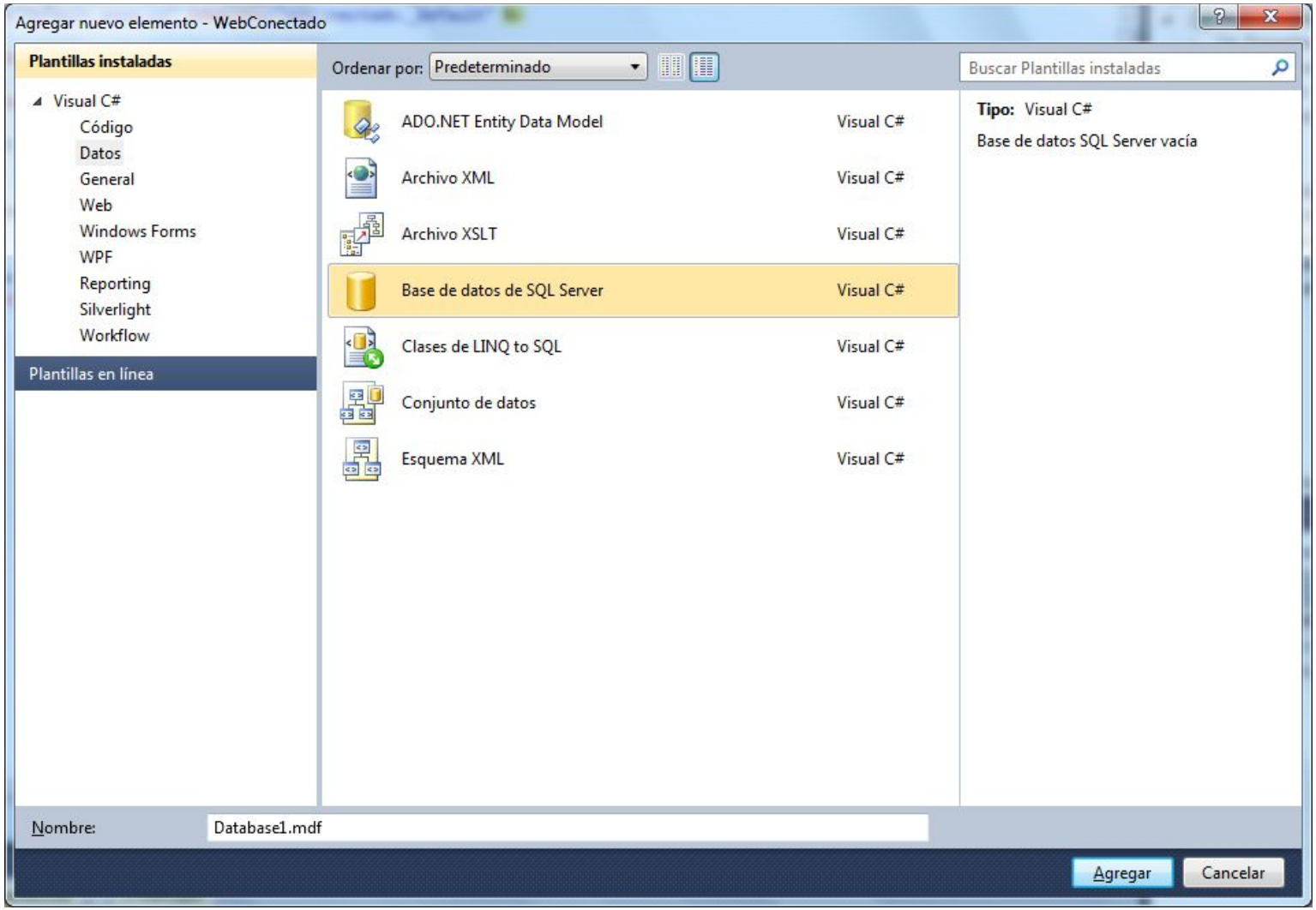
2

Creación BD desde Vstudio.net

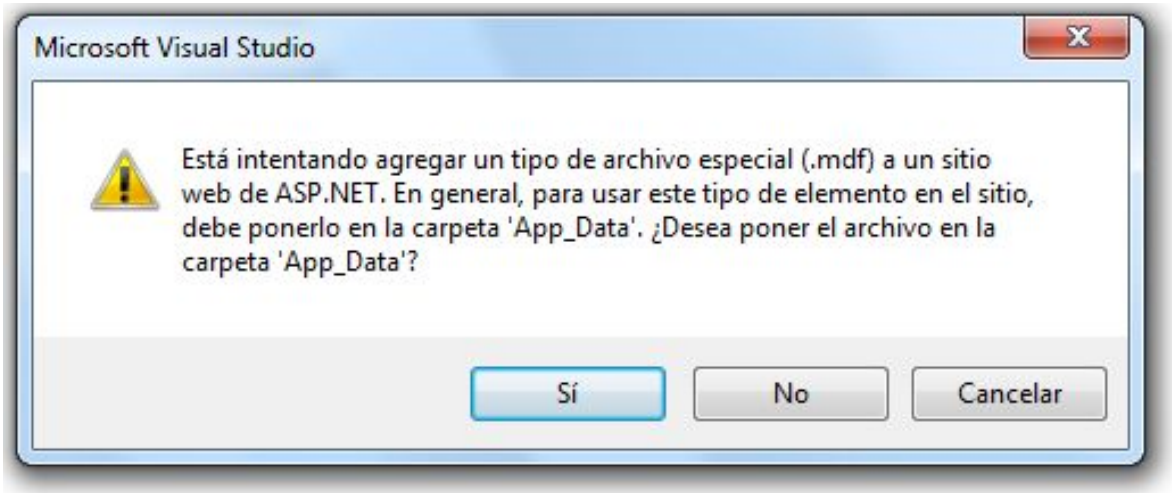
Nuevo proyecto: Aplicación Web C#



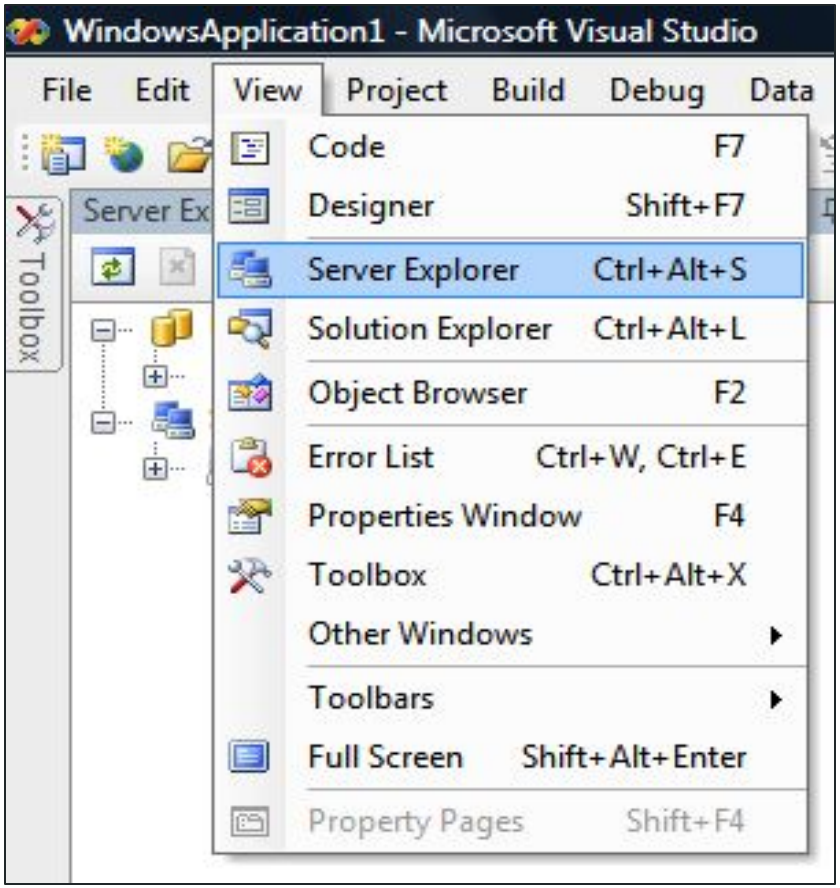
Agregar nuevo elemento: BD SQL server

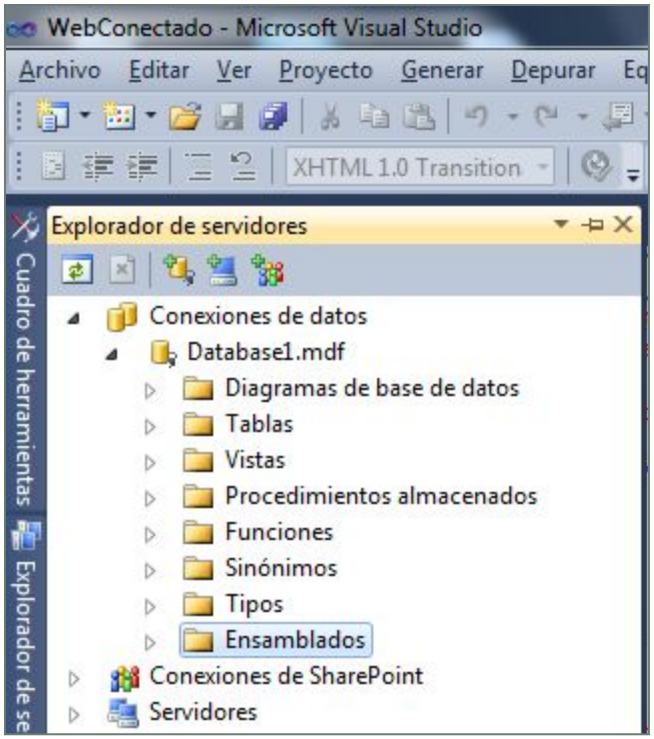


Carpeta especial App_Data

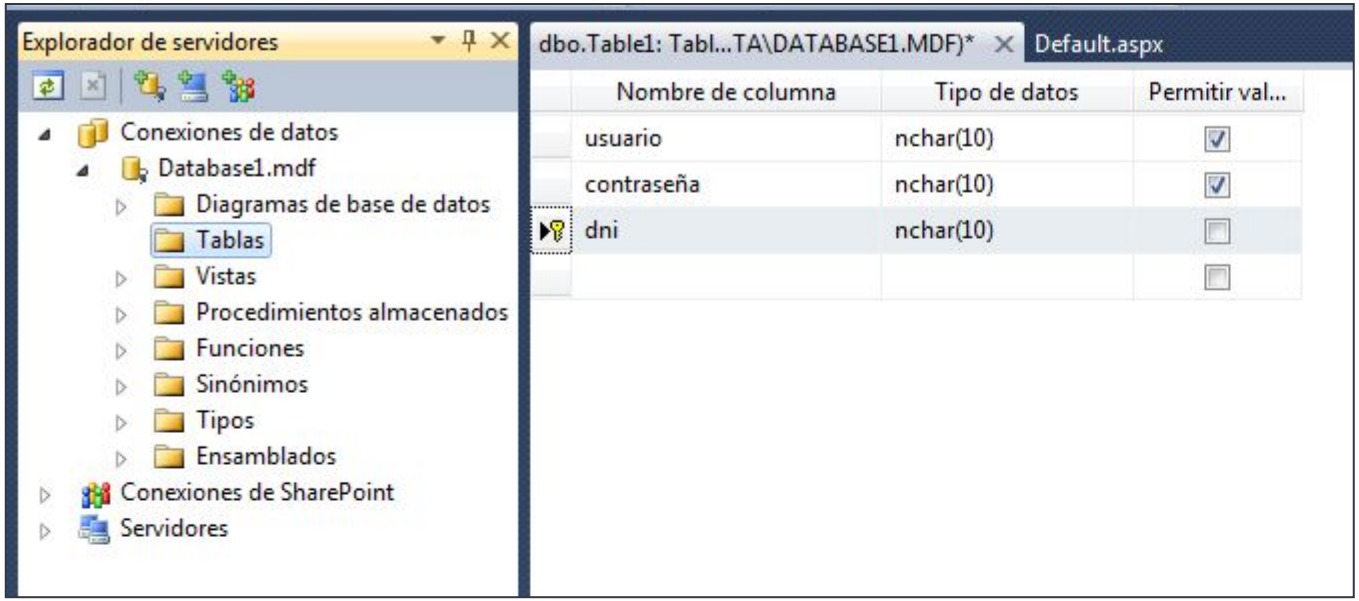


Ver→ explorador de servidores





Añadimos tablas, claves… y datos!



3 Acceso Conectado

Modo conectado

1. Objetos Connection y Command

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

- Command: sirven para ejecutar sentencias SQL y procedimientos almacenados.

Objetos Connection y Command

- System.Data.SqlClient: clases responsables del acceso a datos desde fuentes SQL Server.
- Incluyen clases que al trabajar con SQL Server llevarán el prefijo Sql:
  - SqlConnection
  - SqlCommand

2. Objeto DataReader

- Proporcionan un flujo de datos firme.

- Proporcionan un cursor de sólo lectura que avanza por los registros sólo hacia delante.
  - Mantienen una **conexión viva** con el origen de datos, pero no permiten realizar ningún cambio.

Espacios de nombres de datos

- System.Data
- System.Data.Common
- System.Data.SqlClient →Ms SQL Server DB
- System.Data.SqlTypes → contiene clases para trabajar con tipos de datos nativos de SQL Server

EJEMPLO: 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";

```
SqlConnection c=new SqlConnection(s);
```

Abrir la conexión

```
c.Open();
```

Cadena de Conexión

Parámetro	Descripción
Connection TimeOut	Define el tiempo de espera máximo que debe esperar una conexión para intentar conectar con éxito con el servidor de base de datos. En caso de superar este tiempo se genera una excepción. El tiempo por defecto definido es de 15 segundos.
Data Source	Recibe el nombre del servidor SQL Server utilizado en la conexión.
Integrated Security	Configura nuestra conexión de un modo seguro o no. Recibe como valores True, False y SSPI, siendo True y SSPI el mismo modo de seguridad.
AttachDBFilename	Si se utiliza un nombre de archivo para conectar con la base de datos, se simplifica la implementación de la base de datos con la aplicación (especificamos el archivo mdf)

Propiedad DataDirectory

Propiedad DataDirectory

- **DataDirectory** es una cadena de sustitución que indica la ruta de acceso de la base de datos.

- **DataDirectory** facilita el uso compartido de un proyecto y la impresión de una aplicación al eliminar la necesidad de definir la ruta de acceso completa. Por ejemplo, en vez de tener la siguiente cadena de conexión:
  - "AttachDbFilename= c:\program files\MyApp\Mydb.mdf“

- Al usar |**DataDirectory|**, puede tener la siguiente cadena de conexión:
  - "AttachDbFilename= |DataDirectory|\Mydb.mdf”

- **Para aplicaciones Web, se tendrá acceso a la carpeta App_Data.**

Dónde crear la cadena de conexión???

El archivo web.config

- **Archivo de configuración de la aplicación ASP.NET basado en XML**
- Incluye las opciones de seguridad personalizada, administración de estado, administración de memoria..etc

```
<?xml version="1.0" encoding="utd-8" ?>
    <configuration>
        <system.web>
        </system.web>
        <connectionString>

        </connectionString>

    </configuration>
```

Ejemplo

```
<?xml version="1.0" encoding="utf-8"?>
<!--
  For more information on how to configure your ASP.NET application, please visit
  http://go.microsoft.com/fwlink/?LinkId=301880
  -->
<configuration>
  <configSections>
    <!-- For more information on Entity Framework configuration, visit http://go.microsoft.com/
     fwlink/?LinkID=237468 -->
    <section name="entityFramework"
     type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework,
     Version=6.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" requirePermission="false" />
  </configSections>
  <connectionStrings>
    <add name="DefaultConnection" connectionString="Data Source=(LocalDb)\v11.0;AttachDbFilename=|
     DataDirectory|\aspnet-MvcMovie-20140425082247.mdf;Initial Catalog=aspnet-
     MvcMovie-20140425082247;Integrated Security=True"
      providerName="System.Data.SqlClient" />
  </connectionStrings>
  <appSettings>
    <add key="webpages:Version" value="3.0.0.0" />
    <add key="webpages:Enabled" value="false" />
    <add key="ClientValidationEnabled" value="true" />
    <add key="UnobtrusiveJavaScriptEnabled" value="true" />
  </appSettings>
```

Connectionstring en webconfig

```
<connectionStrings>
 <add name="miconexion"
connectionString="Data
Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=|DataDirector
y|\Database.mdf;Integrated Security=True"
providerName="System.Data.SqlClient" />
</connectionStrings>
```

cadena de conexión llamada miconexion que hace referencia a una cadena de conexión que se conecta a una instancia de SQL Server.

Recuperar cadenas de conexión de archivos de configuración

- El espacio de nombres **System.Configuration** proporciona clases para trabajar con información de configuración almacenada en archivos de configuración.

C#

```
using System.Configuration;

string cadena;
cadena =
ConfigurationManager.ConnectionStrings["miconexion"].ToString
();
```

Se recupera la cadena de conexión del archivo de configuración pasando el nombre de la dicha cadena al ConfigurationManager

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

Objeto Command

- Los objetos Command representan sentencias SQL. Para utilizar un objeto Command se define la sentencia SQL a utilizar y la conexión disponible y se ejecuta el comando:

```
SqlCommand com=
new SqlCommand("Select * from clientes",c);
```

Objeto DataReader (I)

- Se utiliza el método ExecuteReader del objeto Command:
  - `SqlDataReader dr= com.ExecuteReader();`
- Se recupera una fila con el método Read
  - `dr.Read()`

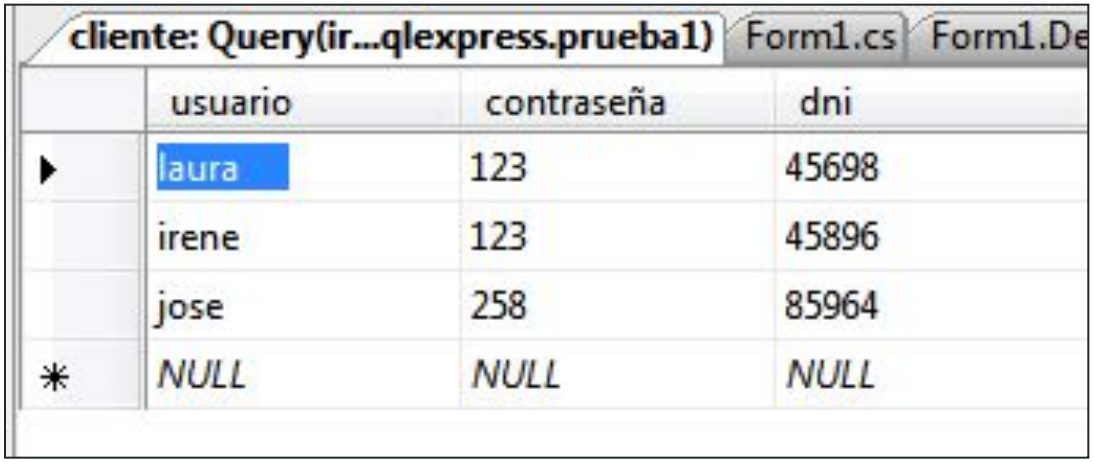
Objeto DataReader (II)

- Podemos acceder a los valores de esa fila con el nombre del campo correspondiente:
  - `MyDataReader[“nombrecampo”];`

- Para leer la siguiente fila, volvemos a usar el método Read
  - devuelve true si se ha recuperado una fila de información correctamente
  - si devuelve false es porque hemos intentado leer después del final del conjunto de resultados

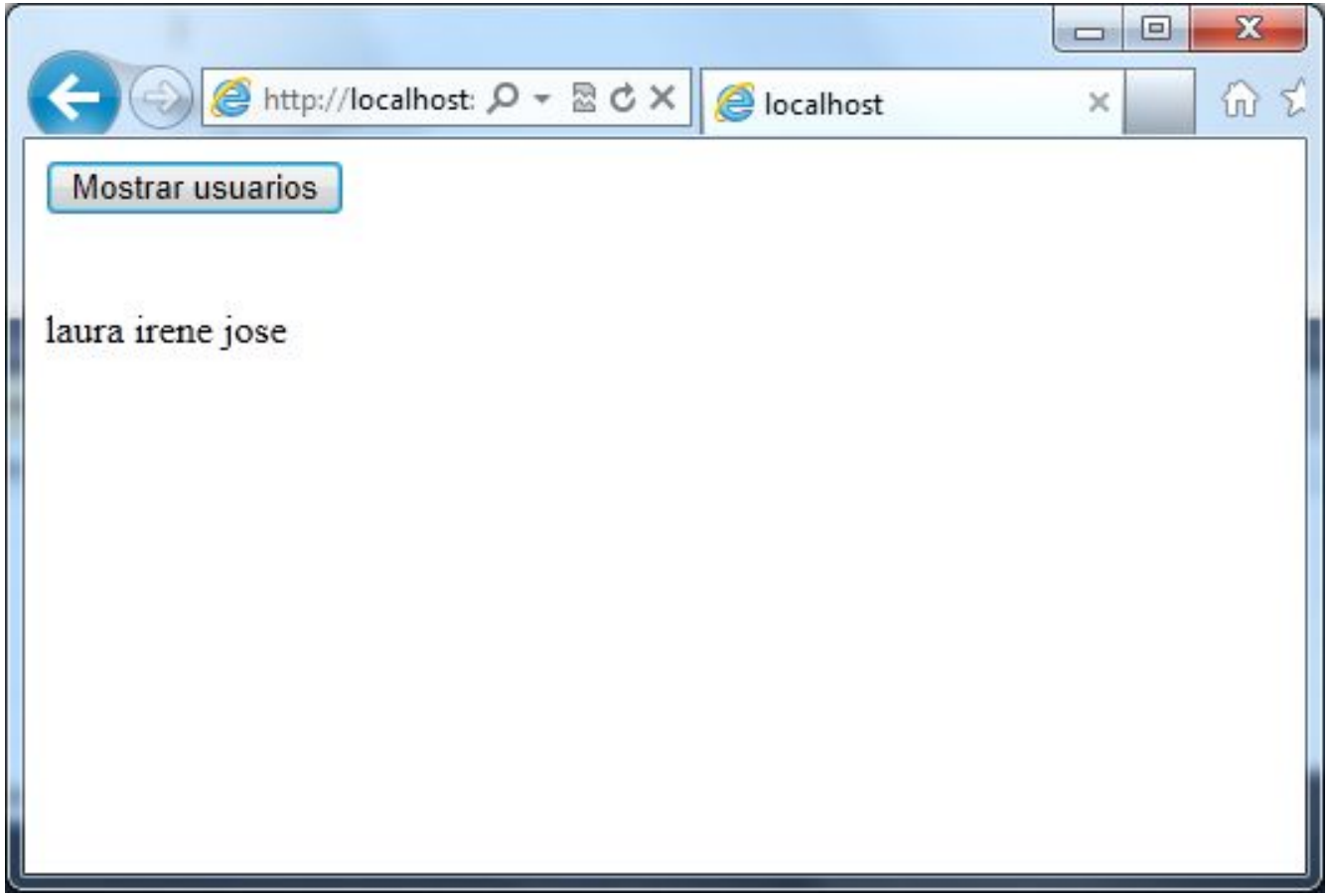
¿Cómo se haría en el ejemplo que estamos viendo??

- Escribir el código necesario para mostrar en una etiqueta el nombre de los clientes (tabla cliente, columna usuario)



	usuario	contraseña	dni
▶	laura	123	45698
	irene	123	45896
	jose	258	85964
*	*NULL*	*NULL*	*NULL*

Ejecución


http://localhost:
localhost
Mostrar usuarios
laura irene jose

En el ejemplo…

```
while ()
{
}
```

Cerrar los objetos DataReader y Connection

- `dr.Close();`
- `c.Close();`

Importante

- Utilizar manejo de excepciones para conexión a BD:
  - Try/catch

Dónde añadimos try/catch?

```
string s = "data source=(LocalDB)\MSSQLLocalDB;Integrated
  Security=SSPI;AttachDBFilename=|DataDirectory|\\Database1.mdf";

SqlConnection c=new SqlConnection(s);
c.Open();
SqlCommand com= new SqlCommand("Select * from cliente",c);
SqlDataReader dr= com.ExecuteReader();

while (dr.Read())
{      this.label1.Text+=dr["usuario"].ToString();
      label1.Text += " ";
}

dr.Close();
c.Close();
```

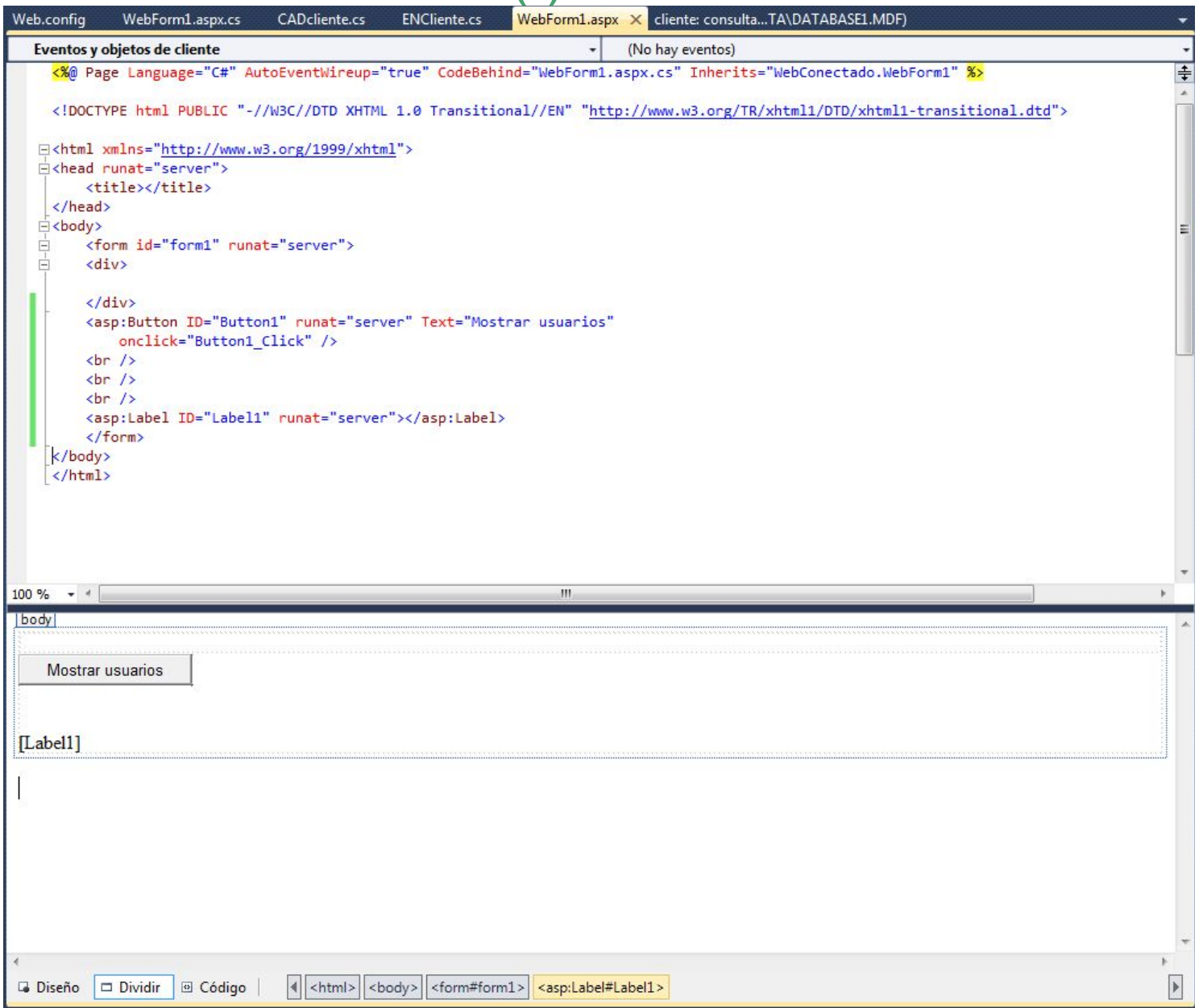
```
string s = "data source=.\\SQLEXPRESS;Integrated
Security=SSPI;AttachDBFilename=|DataDirectory|\\Database1.mdf";
SqlConnection c=new SqlConnection(s);
try
{
c.Open();
SqlCommand com= new SqlCommand("Select * from cliente",c);
SqlDataReader dr= com.ExecuteReader();

while (dr.Read())
{      this.label1.Text+=dr["usuario"].ToString();
      label1.Text += " ";
}

dr.Close();
}
 catch (Exception ex) { label2.Text = ex.Message; }
 finally
 {
     c.Close();}
```

Y cómo sería el código teniendo en cuenta las 3 capas?

CAPA INTERFAZ (I)



CAPA INTERFAZ (II)

```
using System;
…
using System.Collections;
namespace WebConectado
{
    public partial class WebForm1 : System.Web.UI.Page
    {
        ArrayList a = new ArrayList();

        protected void Button1_Click(object sender, EventArgs e)
        {
            ENCliente en = new ENCliente();
            a=en.listarClientes();

            foreach (string s in a)
                Label1.Text += s + " ";
        }}}
```

CAPA EN

```
namespace WebConectado
{
    public class ENCliente
    {
        private string usuario;
        public string Usuario
        {  get { return usuario; }
            set { usuario = value; }
        }
        private string dni;
        public string Dni
        {  get { return dni; }
            set { dni = value; }
        }
        private string contraseña;
        public string Contraseña
        {   get { return contraseña;}
            set { contraseña = value;}
        }
```

```
        public ArrayList listarClientes()
        {
            ArrayList a = new ArrayList();
            CADcliente c = new
  CADcliente();
            a=c.ListarClientes();

            return a;
        }


}
}
```

SE PODRIA SIMPLIFICAR???

CAPA CAD

```
public class CADcliente{
        ArrayList lista = new ArrayList();
        string s = "data source=.\\SQLEXPRESS;Integrated
 Security=SSPI;AttachDBFilename=|DataDirectory|\\Database1.mdf";
        public ArrayList ListarClientes()
        {
                SqlConnection c = new SqlConnection(s);
                 c.Open();
                 SqlCommand com = new SqlCommand("Select * from cliente", c);
                 SqlDataReader dr = com.ExecuteReader();
                 while (dr.Read())
                 {
                     lista.Add(dr["usuario"].ToString());
                 }
                 dr.Close();
                 c.Close();
     return lista;
        }}
```

Cómo sería poniendo la cadena de conexión en Web.config??

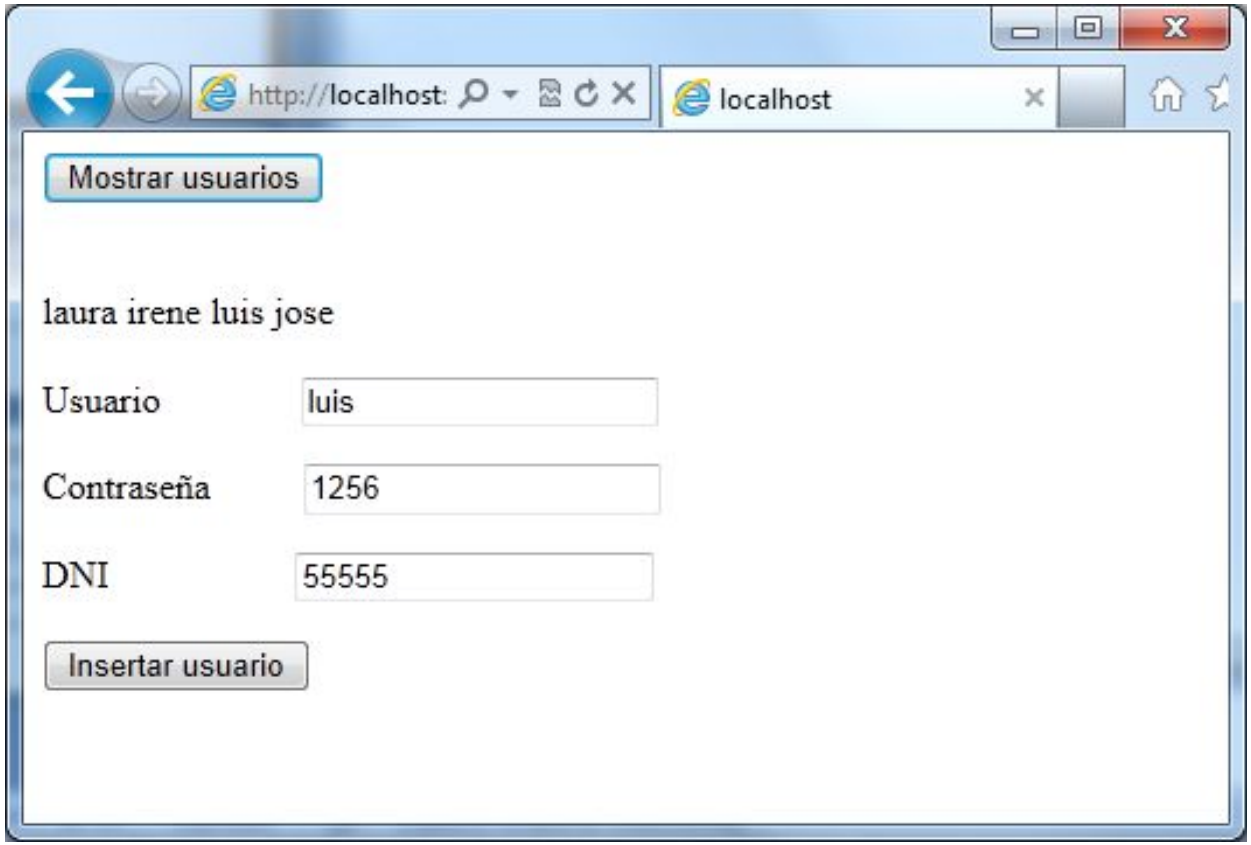
Modificación de datos: Insert, Update, Delete

- En este caso no es necesario un objeto DataReader ya que no se recupera ningún resultado.
- Tampoco un comando Select de SQL, sino:
  - Update
  - Insert
  - Delete
- Necesitamos crear un objeto Command para ejecutar la sentencia SQL apropiada.
- Método ExecuteNonQuery: obtiene el número de registros afectados.

Ejecución de comandos

- ExecuteNonQuery
  - Ejecuta un comando y no devuelve ningún resultado (obtiene el número de registros afectados.)
- ExecuteReader
  - Ejecuta un comando y devuelve un comando que implementa DataReader (Permite iterar a partir de los registros recibidos)

Ejemplo, Insertar usuarios



```
string s = "data source=.\\SQLEXPRESS;Integrated
  Security=SSPI;AttachDBFilename=|DataDirectory|\\Database1.mdf";

SqlConnection c=new SqlConnection(s);
try
{
c.Open();
SqlCommand com= new SqlCommand("Select * from cliente",c);
SqlDataReader dr= com.ExecuteReader();

while (dr.Read())
{      this.label1.Text+=dr["usuario"].ToString();
      label1.Text += " ";
}

dr.Close();
}
 catch (Exception ex) { label2.Text = ex.Message; }
 finally
 {
     c.Close();
 }
```

```
string s = "data source=.\\SQLEXPRESS;Integrated
  Security=SSPI;AttachDBFilename=|DataDirectory|\\Database1.mdf"
  ;

SqlConnection c=new SqlConnection(s);
try
{
c.Open();
SqlCommand com = new SqlCommand("Insert Into Cliente
  (usuario,contraseña,dni) VALUES ('" + textBox1.Text + "','" +
  textBox3.Text + "','" + textBox2.Text +"')", c);

com.ExecuteNonQuery();

}
 catch (Exception ex) { label2.Text = ex.Message; }
 finally
 {
     c.Close();
 }
```

Interfaz

```
protected void Button2_Click(object sender,
    EventArgs e)
        {
            ENCliente en = new ENCliente();
            en.Usuario = TextBox1.Text;
            en.Contraseña = TextBox2.Text;
            en.Dni = TextBox3.Text;

            en.InsertarCliente();
        }
```

ENCliente

```
public void InsertarCliente()
        {
            CADcliente c = new CADcliente();
            c.InsertarCliente(this);
        }
```

CADCliente

```
public void InsertarCliente(ENCliente cli)
{
   ENCliente cl = cli;
   SqlConnection c = new SqlConnection(s);
   c.Open();

   SqlCommand com = new SqlCommand("Insert Into Cliente
   (usuario,contraseña,dni) VALUES ('" + cl.Usuario + "','" +
  cl.Contraseña + "','" +
    cl.Dni + "')", c);

    com.ExecuteNonQuery();
    c.Close();
}
```