

ADO.NET

*Herramientas Avanzadas para el Desarrollo de
Aplicaciones*

Escuela Politécnica Superior
Universidad de Alicante

Goals

- ADO.net 2.0
- Creating a DB using VStudio.net
- Connected data access
- Creating the connection string: Web.config
- DataDirectory Property

ADO.Net 2.0

1

ActiveX Data Objects

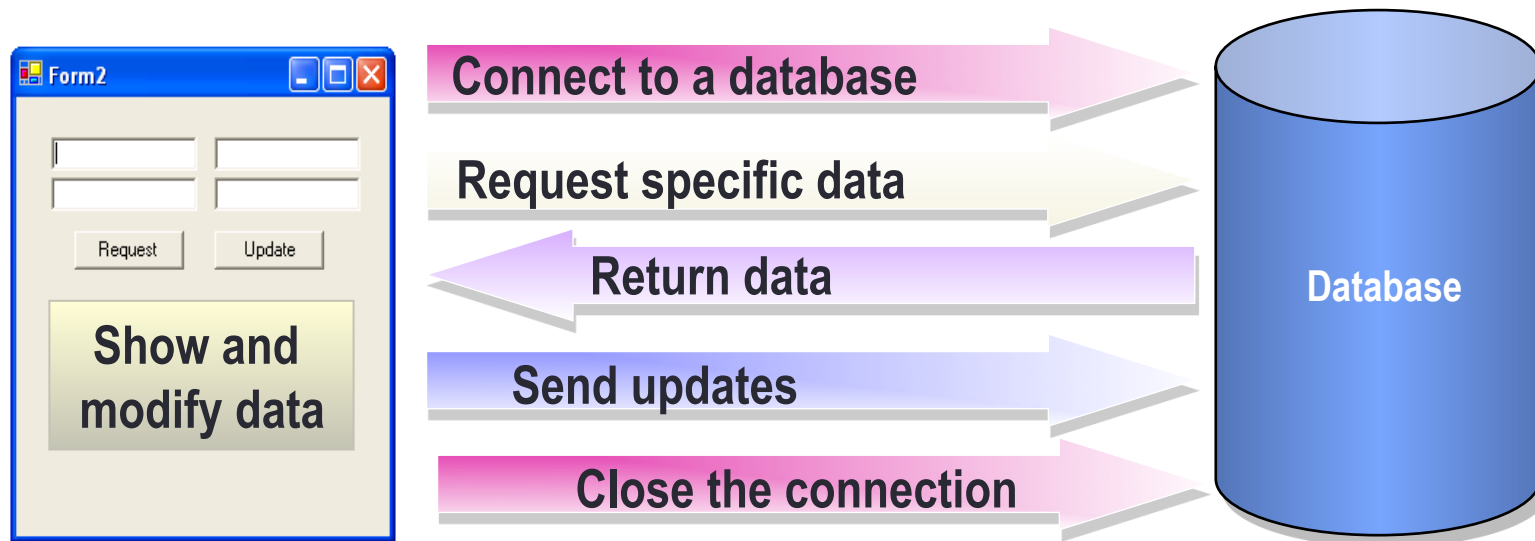
- ADO.NET is the technology used by the asp.net applications to communicate with the DB.
- Optimized for distributed applications (such as Web applications).
- Based on XML
- New set of model objects.
- **Connected and disconnected access to data.**

Connected environment

- A connected environment is that one in which the users are continuously connected to a data source
- Advantages:
 - The environment is easier to maintain
 - The concurrency is more easily controlled
 - It is more probable that the data are updated than in other environments
- Disadvantages:
 - We need a constant network connection
 - Limited scalability

Connected environment(II)

CONNECTION OPEN



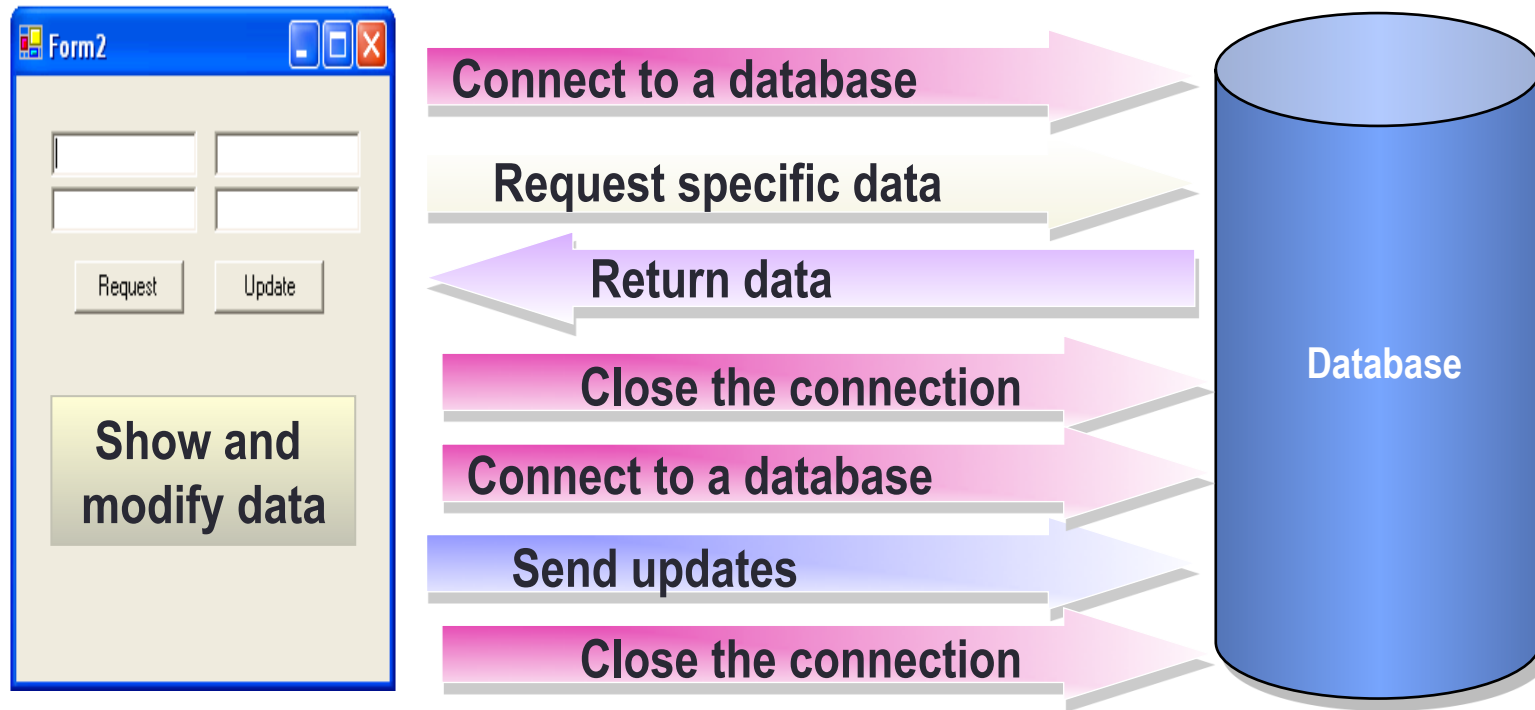
WITHOUT CONNECTION

Disconnected environment

- In a disconnected environment data can be modified in an independent way and the updates are written in the database afterwards.
- Advantages:
 - Connections are used during the minimal time needed, allowing that less connections are used by more users
 - A disconnected environment improves the scalability and performance of the applications
- Disadvantages:
 - Data are not allways updated
 - There can be conflicts with updates that must be solved

Disconnected environment (II)

CONNECTION OPEN



WITHOUT CONNECTION

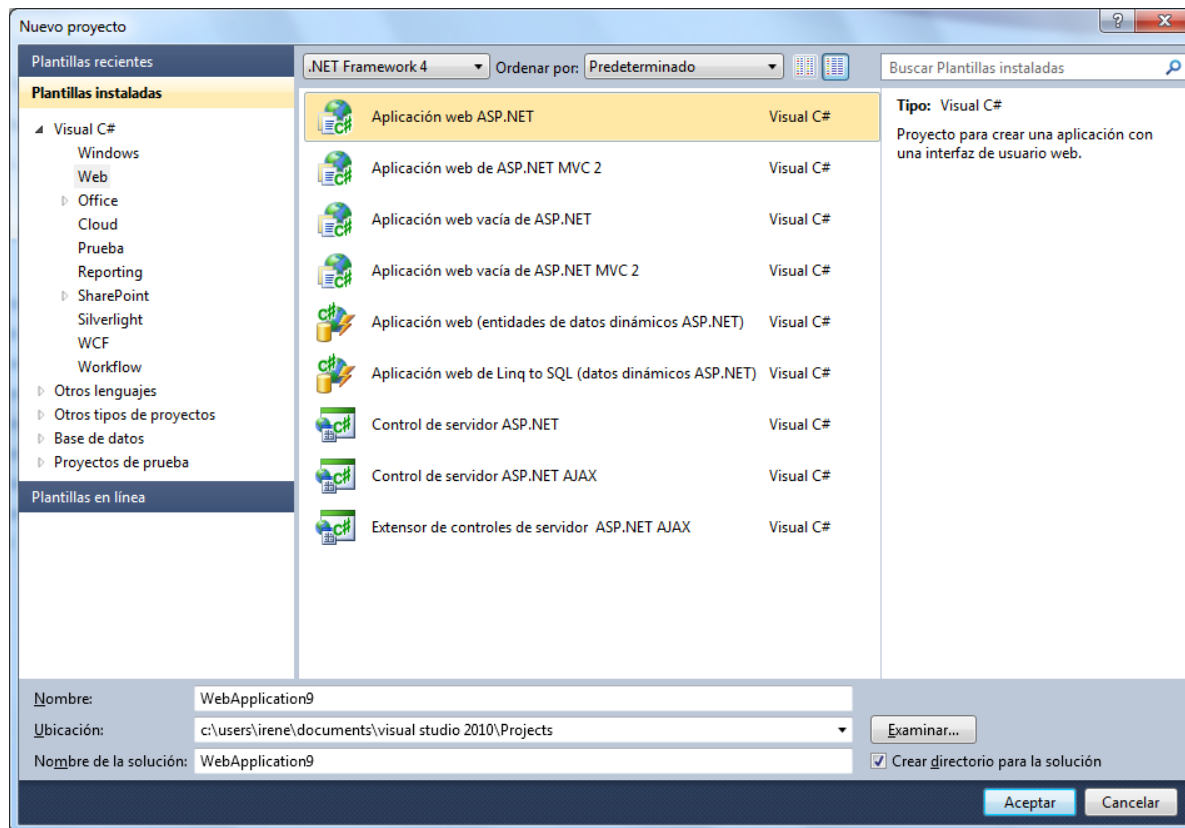
Examples

- A supermarket in which every selling point stores the sales done. Every certain time the sales have to be updated in the central DB.
- A factory that requires a constant real-time connection in order to control the production output and the store.
- An application that store client data in the laptop of a sales representative.
- An application that do a tracking of rainfalls.
- A stockbroker that requires a constant connection to the stock market values.
- A farmer application used to count the cattle (sheeps, pigs, goats..). The application is in a device based on Microsoft Windows CE which performs Microsoft SQL Server 2000 Windows CE Edition.

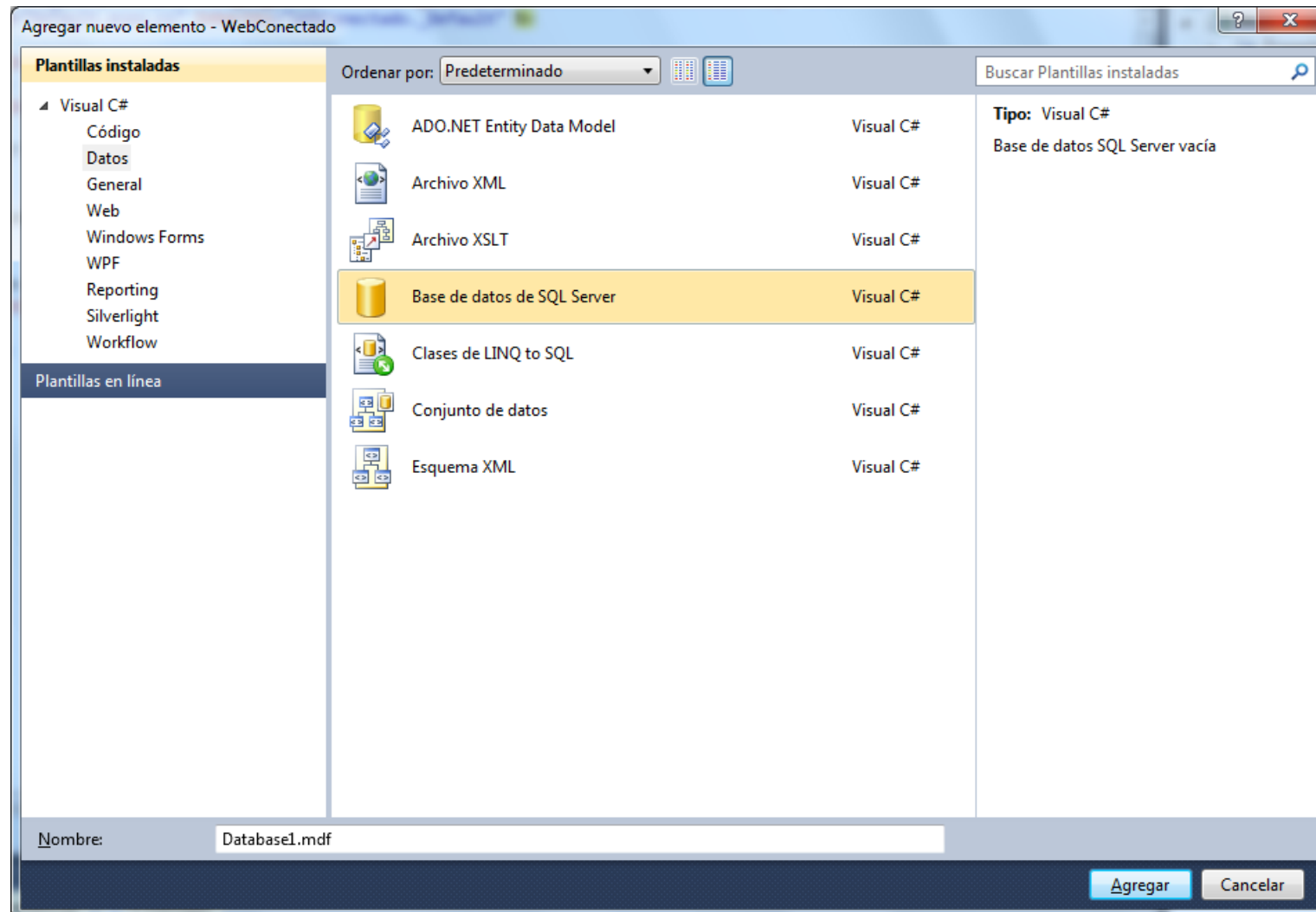
2

Creating a DB in
Vstudio.net

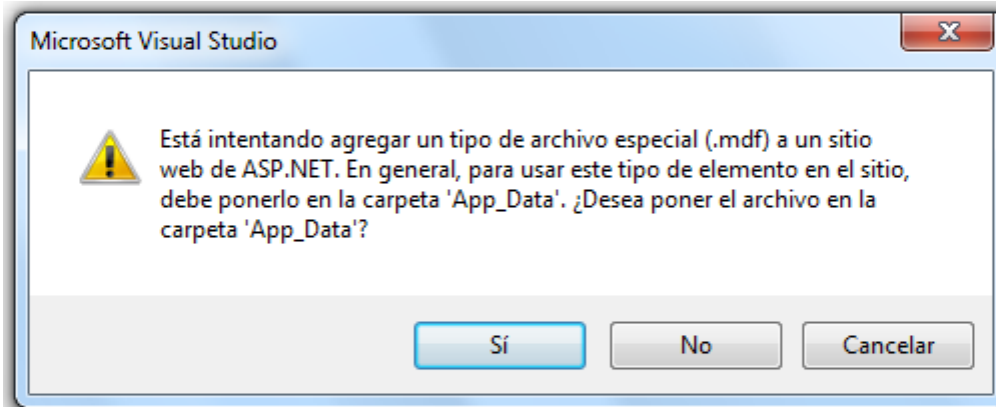
New project: Web Application C#



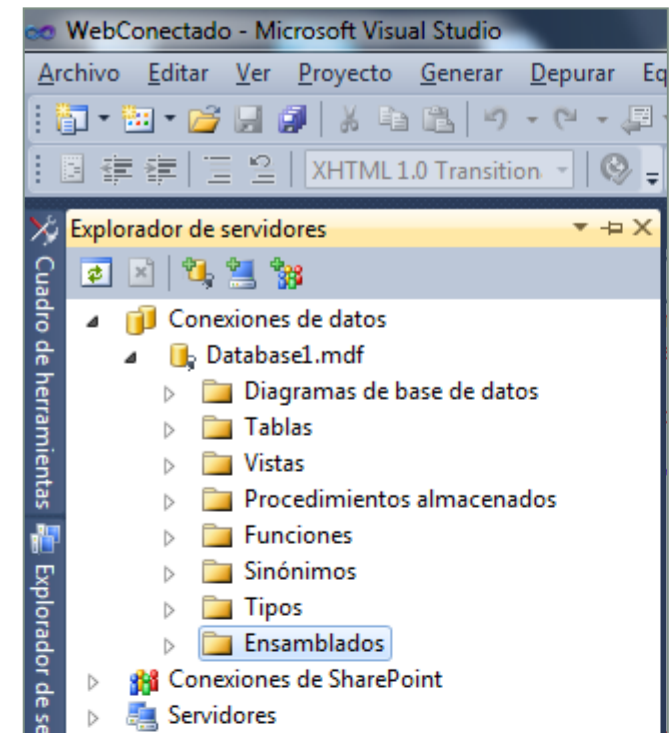
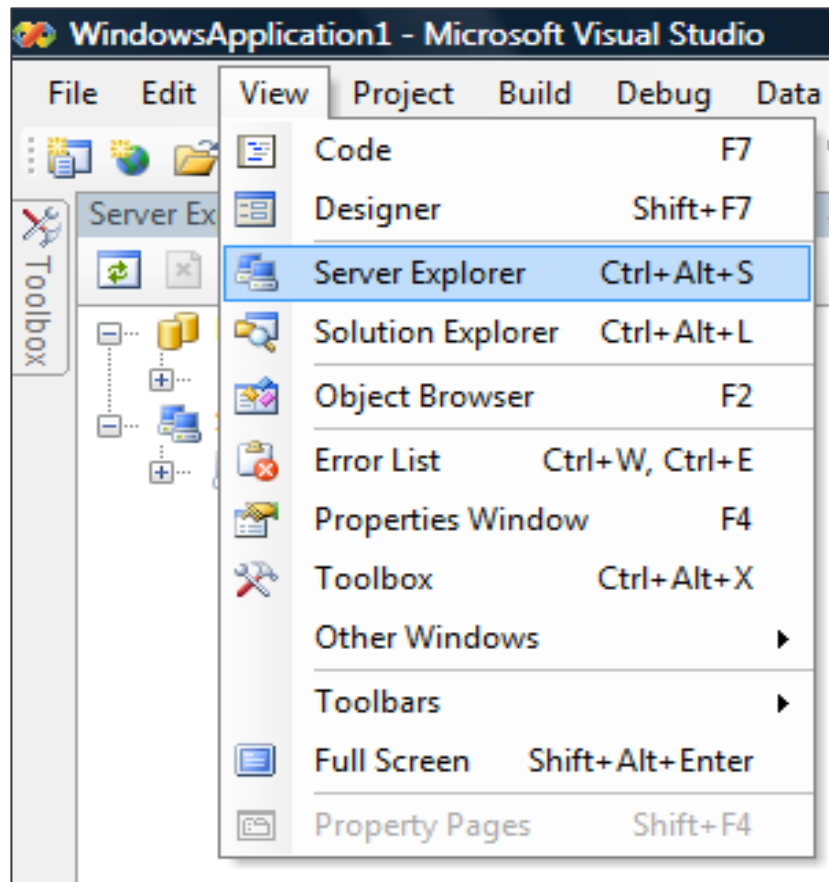
Add new item: BD SQL server



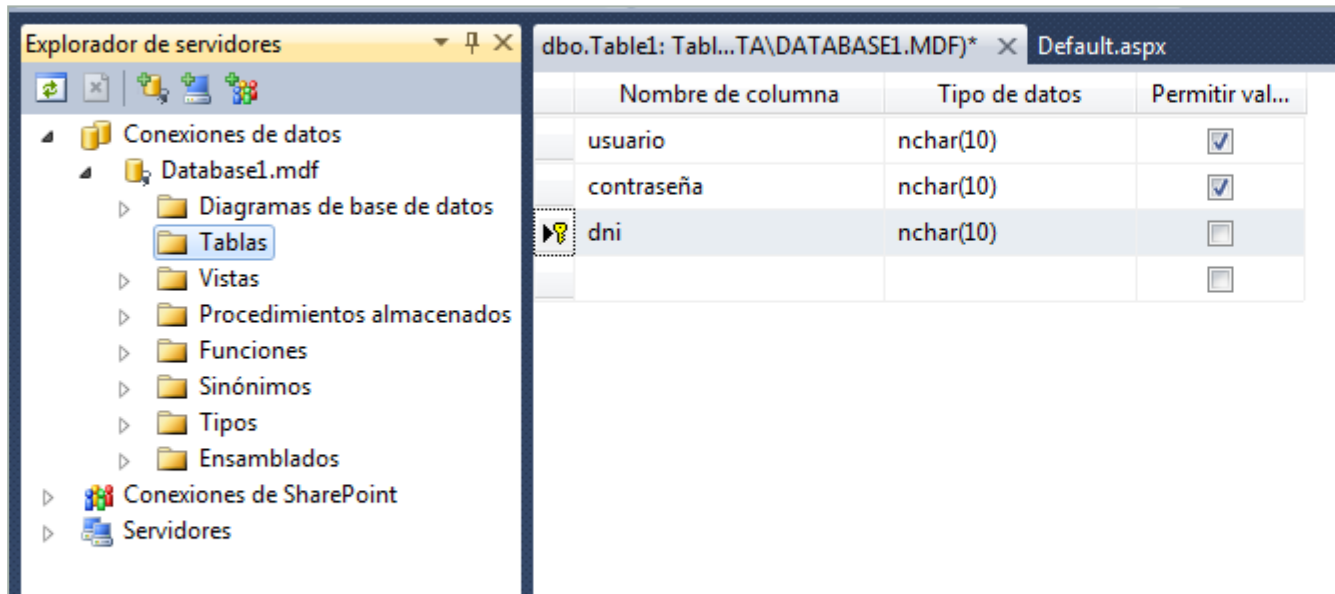
Special folder App_Data



View → Server explorer



We add tables, keys... and data!



Possible configuration problems

- Install SQLServer Management Studio Express
 - In the connection option → allow remote connections
- In SQL Configuration Manager
 - Protocols of SQLEXPRESS
 - Enable TCP/IP and named pipes
- Stop the SQLEXPRESS service and restart it



3

Connected
environment

1. Connection and Command Objects

- **Connection:** used to establish connections with the proper data provider (Open method).
- **Command:** used to perform SQL sentences and stored procedures.

Connection and Command Objects

- `System.Data.OleDb` and `System.Data.SqlClient`: responsible classes for data access from SQL Server and OLE DB sources.
- They include classes that when working with SQL will have the `Sql` prefix and when using Ole DB will have the `OleDb` prefix:
 - `SqlConnection` and `OleDbConnection`
 - `SqlCommand` and `OleDbCommand`

2. DataReader Object

- Provide a solid data flow.
- It allows retrieving a read-only, forward-only stream of data from a database.
 - It maintains a **live connection** with the data source, but it does not allow doing any modification.
- Using the **DataReader** can increase application performance both by retrieving data as soon as it is available, rather than waiting for the entire results of the query to be returned, and (by default) storing only one row at a time in memory, reducing system overhead.

Namespaces

- `System.Data`
- `System.Data.Common`
- `System.Data.OleDb` → Ms access, Oracle.. DB
- `System.Data.SqlClient` → Ms SQL Server 7.0 DB
- `System.Data.SqlTypes` → contains classes for working with the native data types of SQL Server

EXAMPLE: Connection to a Sql Server DB

Import namespaces

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

Connection string (I)

Parameter	Description
Connection Timeout	The length of time (in seconds) to wait for a connection to the server before terminating the attempt and generating an error.
Data Source	The name or network address of the instance of SQL Server to which to connect, or in case of working with Access, the name of the file used
Initial Catalog / Database	The name of the database
Integrated Security	When false, User ID and Password are specified in the connection. When true, the current Windows account credentials are used for authentication. Recognized values are true, false, yes, no, and sspi (strongly recommended), which is equivalent to true.

Connection string (II)

Parameter	Description
AttachDBFilename	The name of the primary file, including the full path name, of an attachable database.
Persist Security Info	When set to false or no (strongly recommended), security-sensitive information, such as the password, is not returned as part of the connection if the connection is open or has ever been in an open state. Resetting the connection string resets all connection string values including the password. Recognized values are true, false, yes, and no.
Password	The password for the SQL Server account logging on
Provider	Only used within OleDbConnection, it establishes or returns the provider's name.
User ID	The SQL Server login account

Provider

- SQLOLEDB: OLEDB provider for SQL
- MSDAORA: OLEDB provider for a Oracle DB
- Microsoft.Jet.OLEDB.4.0: OLEDB provider for Access

Creating the connection

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

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

Opening the connection

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



DataDirectory Property

DataDirectory Property

- **DataDirectory** is a substitution string that indicates the path to the database.
- **DataDirectory** eliminates the need to hard-code the full path which leads to several problems as the full path to the database could be serialized in different places. DataDirectory also makes it easy to share a project and also to deploy an application.
- For example, instead of having the following connection string:
 - "AttachDbFilename= c:\program files\MyApp\Mydb.sdf"
- Using |**DataDirectory**|, you can have the following connection string:
"AttachDbFilename= |DataDirectory|\Mydb.sdf"
- To set the **DataDirectory** property, call the **AppDomain.SetData** method.
 - AppDomain.CurrentDomain.SetData("DataDirectory", newpath);

DataDirectory Property

- If you do not set the **DataDirectory** property, the following default rules will be applied to access the database folder:
- For applications that are put in a folder on the user's computer, the database folder uses the application folder. For instance if MyApp.exe is located in the folder /MyDir, it will have access to the folder /MyDir.
-
- For applications that are located in a directory in the client, the database folder uses the specific data folder that is created. **In Web applications, it will have access to the folder App_Data.**

|DataDirectory|

```
<connectionStrings>
```

```
<add name="miconexion"
```

```
connectionString="Data Source=localhost\SQLEXPRESS;
```

```
AttachDbFilename=|DataDirectory|\bd.mdf;Integrated
```

```
Security=True;Connect Timeout=30;User Instance=True"
```

```
providerName="System.Data.SqlClient" />
```

```
</connectionStrings>
```

Definition of a Select command

- For recovering the data we need:
 - A SQL sentence which selects the desired information
 - A Command object which performs the SQL sentence
 - A DataReader object which captures the recovered entries

Command object

- The Command objects represent SQL sentences. For using a Command object we define the SQL sentence to be used and the available connection and the command is performed:

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


DataReader Object (I)

- We use the method ExecuteReader from the Command object:
 - `SqlDataReader dr= com.ExecuteReader();`
- We recover a row with the Read method
 - `dr.Read()`

DataReader object (II)

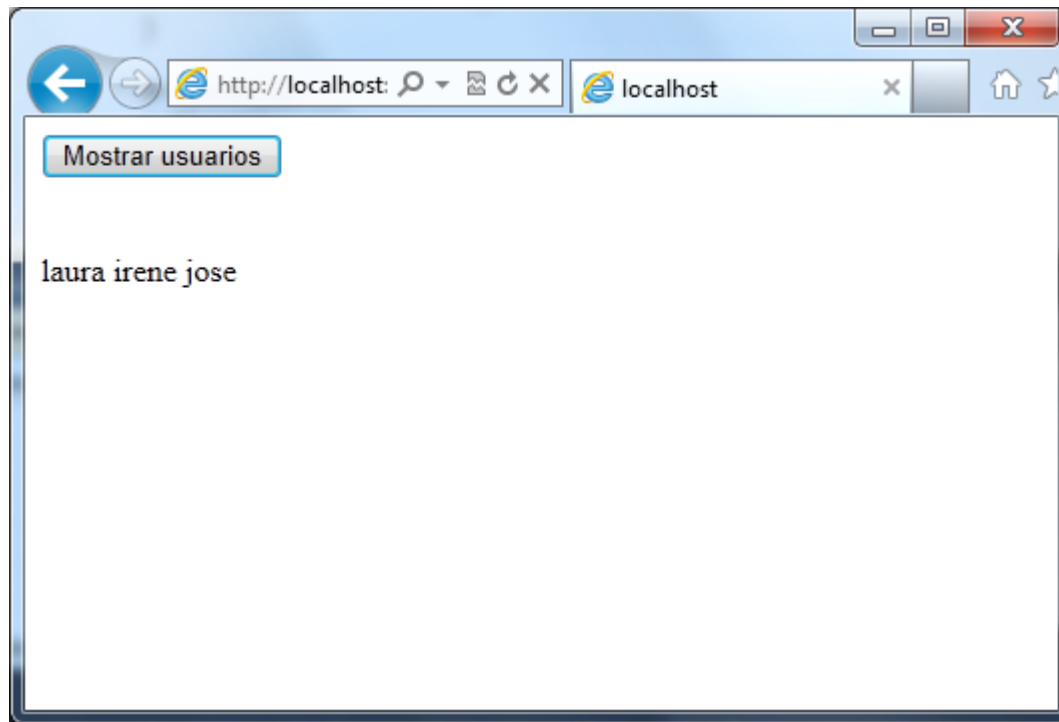
- We can access the value of that row with the name of the corresponding field:
 - `MyDataReader["fieldname"];`
- For reading the next row, we need the Read method
 - returns `true` if we have successfully recovered a row of information
 - If it returns `false` is because we have attempted reading after the end of the set of results

How would this be done in the example??

- Write the needed code for showing in a label the name of the clients (Client table, user column)

cliente: Query(ir...qlxpress.prueba1)			
	usuario	contraseña	dni
▶	laura	123	45698
	irene	123	45896
	jose	258	85964
*	NULL	NULL	NULL

Execution



In the example...

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

Closing the DataReader and Connection objects

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

Important

- Use exception management when connecting to a DB:
 - Try/catch

Where do we add try/catch?

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

```
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();
```

```
}
```



How would this code be, applying the 3 layers architecture?

INTERFACE LAYER(I)

The screenshot displays the Visual Studio IDE with the 'WebForm1.aspx' file open. The top toolbar shows the 'Código' (Code) view is active. The code editor contains the following HTML and ASP.NET markup:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="WebForm1.aspx.cs" Inherits="WebConectado.WebForm1" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title></title>
</head>
<body>
<form id="form1" runat="server">
<div>

</div>
<asp:Button ID="Button1" runat="server" Text="Mostrar usuarios"
onclick="Button1_Click" />
<br />
<br />
<br />
<asp:Label ID="Label1" runat="server"></asp:Label>
</form>
</body>
</html>
```

The 'Diseño' (Design) view at the bottom shows a visual representation of the code. It features a button labeled 'Mostrar usuarios' and a label placeholder '[Label1]'. The bottom status bar indicates the current selection is on the 'Label1' control within the 'form1' container.

INTERFACE LAYER (II)

```
using System;
```

```
...
```

```
using System.Collections;
```

```
namespace WebConectado
```

```
{
```

```
    public partial class WebForm1 : System.Web.UI.Page
```

```
    {
```

```
        ArrayList a = new ArrayList();
```

```
        protected void Page_Load(object sender, EventArgs e)
```

```
        {
```

```
            protected void Button1_Click(object sender, EventArgs e)
```

```
            {
```

```
                ENCliente en = new ENCliente();
```

```
                a=en.listarClientes();
```

```
                foreach (string s in a)
```

```
                    Label1.Text += s + " ";
```

```
            }}}
```

EN LAYER

```
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;
}
}
```

CAD LAYER

```
public class CADcliente
{
    ArrayList lista = new ArrayList();
    string s = "data source=.\SQLEXPRESS;Integrated Security=SSPI;AttachDBFilename=|
DataDirectory|\\Database1.mdf;User Instance=true";

    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;
}
```



Where do we create the connection
string???

The web.config file

- Configuration file of the ASP.NET applications based on XML
- It includes the security personalized options, memory management, etc

```
<?xml version="1.0" encoding="utf-8" ?>  
  <configuration>  
    <system.web>  
      <!-- CONFIGURATION SECTION -->  
    </system.web>  
  </configuration>
```


Important, where do we store the connection string?

- To avoid storing them in the code, we can store them in the **web.config** file in a ASP.NET application.
- The connection string can be stored inside the element **<connectionStrings>**. Connection strings are stored as pairs name-value, where the name can be used to search the value stored in the **connectionString** attribute at execution time.

Web.config configuration file: based on XML

```
<connectionStrings>  
  <add name="DatabaseConnection"  
    connectionString="Persist Security Info=False;Integrated  
Security=SSPI;database=Northwind;server=(local);"  
    providerName="System.Data.SqlClient" />  
</connectionStrings>
```

Connection string called DatabaseConnection which refers to a connection string which connects to a local instance of SQL Server.

Recovering connection strings from configuration files

- The namespace **System.Configuration** provides classes in order to work with configuration data stored in the Web.config file.

C#

```
using System.Configuration;
```

```
string cadena;
```

```
cadena =
```

```
ConfigurationManager.ConnectionStrings["DatabaseConnection"].ToString  
();
```

We recover the connection string, passing the name of the string to the ConfigurationManager

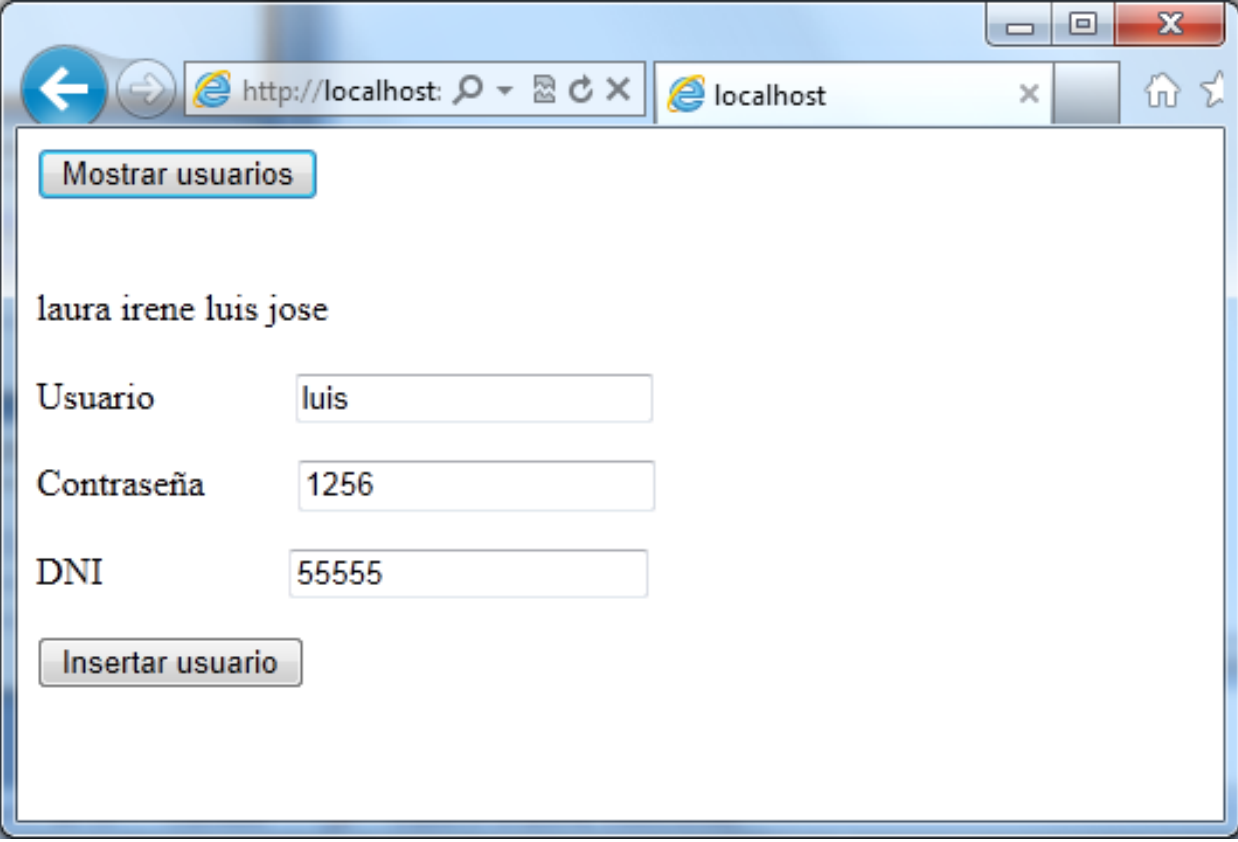
Modifying data: Insert, Update, Delete

- In this case, we do not need a DataReader object because we do not recover any result.
- We also do not need a SQL Select command, we need:
 - Update
 - Insert
 - Delete
- We have to create a Command object in order to perform the proper SQL sentence.
- We will use the ExecuteNonQuery method: it obtains the number of affected entries.

Execution of Commands

- **ExecuteNonQuery**
 - Executes a command and does not return any result (it obtains the number of affected entries)
- **ExecuteReader**
 - Executes a command and returns a command which DataReader implements (it allows iterate over the received entries)

Example, Insert users



A screenshot of a web browser window displaying a user management interface. The browser's address bar shows 'http://localhost:'. The page contains a button labeled 'Mostrar usuarios'. Below this, the text 'laura irene luis jose' is displayed. The form includes three input fields: 'Usuario' with the value 'luis', 'Contraseña' with the value '1256', and 'DNI' with the value '55555'. At the bottom of the form is a button labeled 'Insertar usuario'.

Mostrar usuarios

laura irene luis jose

Usuario

Contraseña

DNI

Insertar usuario

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

```
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;User Instance=true";
```

```
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();
```

```
}
```

Interface

```
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();
}
```