



# ASP.NET 3.5

## ACCESO A DATOS

CLIPSA



DISTRIBUIDO POR:

**CENTRO DE INFORMÁTICA PROFESIONAL S.L.**

C/ URGELL, 100  
08011 BARCELONA  
TFNO: 93 426 50 87

C/ RAFAELA YBARRA, 10  
48014 BILBAO  
TFNO: 94 448 31 33

[www.cipsa.net](http://www.cipsa.net)

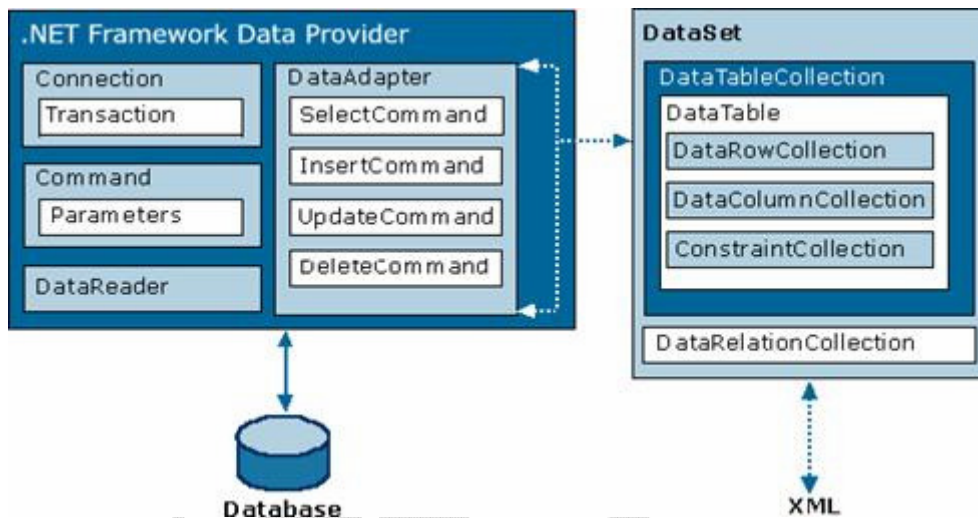
**RESERVADOS TODOS LOS DERECHOS. QUEDA PROHIBIDO TODO TIPO DE REPRODUCCIÓN TOTAL O PARCIAL DE ESTE MANUAL, SIN PREVIO CONSENTIMIENTO POR EL ESCRITOR DEL EDITOR**

CLIPSA

## Fundamentos ADO.Net

El objetivo de ADO.NET es proporcionar un puente entre los objetos propios de una aplicación Web / WinForms, y una base de datos contra la que trabajar.

Las clases que conforman ADO.NET tienen como misión encapsular la mayoría de los detalles complejos que intervienen en el acceso y manipulación de una base de datos. A continuación se muestra una vista general de la organización de dichas clases:



Los modelos de objetos de acceso de datos tradicionales se basan en establecer una conexión contra una base de datos manteniéndola el tiempo necesario para llevar a cabo todas las operaciones. Esto implica el mantenimiento de múltiples conexiones contra una misma base de datos largos periodos de tiempo. A esto se le denomina comúnmente modelo conectado. El modelo conectado puede producir que los servidores se saturen o pierdan rendimiento cuando se conecten muchos usuarios al mismo tiempo, lo cual es especialmente grave si se trata de conexiones a través de Internet.

La tecnología ADO.NET introduce un nuevo modelo de conexión a las bases de datos; el modelo desconectado. El modelo desconectado se basa en establecer una conexión inicial a la base de datos, descargar una copia de la información almacenada y cerrar acto seguido la conexión. La aplicación trabaja a partir de ese momento con una copia local de los datos independientemente a la base de datos. Posteriormente; es posible reestablecer la conexión con la base de datos para actualizarla con los cambios realizados en la copia local de datos.

ADO.NET puede trabajar tanto en modo conectado empleando las diferentes clases **DataReader**, o en modo desconectado empleando la clase **DataSet**.

# ACCESO A DATOS CON ADO.NET

## 1.1.- MODELO DE OBJETOS

ADO.NET ofrece un conjunto de clases y herramientas integradas en el Framework .NET que permite la conexión y manipulación de bases de datos SQL Server, Access, Oracle... etc. Todas las clases que componen ADO.NET se encuentran definidas en el espacio de nombres System.Data.

La jerarquía de clases ADO.NET está diseñada para permitir trabajar de manera única con bases de datos de distintos sistemas. Las clases que componen ADO.NET pueden subdividirse en dos tipos:

Clases independientes del origen de datos: Estas son clases que no dependen del origen de datos empleado siendo comunes a todos.

Espacios de nombres:

- System.Data:
- System.Data.Common:

La clase DataSet ( System.Data ) es una de las más importantes en ADO.NET cuya función es almacenar los datos recuperados desde un origen de datos para su manipulación. Las instancias de DataSet suelen obtenerse por lo tanto a partir de consultas a la base de datos.

Clases Proveedores de Datos: Estas son agrupaciones de clases especializadas en la conexión y manipulación de orígenes de datos concretos tales como Access, SQL Server, Oracle.. etc

Proveedor de datos de .NET Framework	Descripción
Proveedor de datos de .NET Framework para SQL Server	Proporciona acceso de datos para Microsoft SQL Server versión 7.0 o posterior. Utiliza el espacio de nombres <a href="#">System.Data.SqlClient</a> .
Proveedor de datos de .NET Framework para OLE DB	Para orígenes de datos que se exponen mediante OLE DB. Utiliza el espacio de nombres <a href="#">System.Data.OleDb</a> .
Proveedor de datos de .NET Framework para ODBC	Para orígenes de datos que se exponen mediante ODBC. Utiliza el espacio de nombres <a href="#">System.Data.Odbc</a> .
Proveedor de datos de .NET Framework para Oracle	Para orígenes de datos de Oracle. El proveedor de datos de .NET Framework para Oracle es compatible con la versión 8.1.7 y posteriores del software de cliente de Oracle y utiliza el espacio de nombres <a href="#">System.Data.OracleClient</a> .

Cada Adaptador de Datos está formado por una serie de clases que facilitan las operaciones básicas para el trabajo contra una base de datos:

- Conexión con el origen de datos.
- Definición y ejecución de comandos de consulta y modificación de datos.
- Lectura secuencial inmediata de datos desde el origen de datos.
- Obtención de instancias DataSet a partir de comandos de consulta.

Cada una de las funciones indicadas tiene asociada una o varias clases dentro del Adaptador de un determinado origen de datos:

Objeto	Descripción
<b>Connection</b>	Establece una conexión a un origen de datos determinado. La clase base para todos los objetos <b>Connection</b> es <a href="#">DbConnection</a> .
<b>Command</b>	Ejecuta un comando en un origen de datos. Expone <b>Parameters</b> y puede ejecutarse en el ámbito de un objeto <b>Transaction</b> de <b>Connection</b> . La clase base para todos los objetos <b>Command</b> es <a href="#">DbCommand</a> .
<b>DataReader</b>	Lee una secuencia de datos de sólo avance y sólo lectura desde un origen de datos. La clase base para todos los objetos <b>DataReader</b> es <a href="#">DbDataReader</a> .
<b>DataAdapter</b>	Llena un <b>DataSet</b> y realiza las actualizaciones necesarias en el origen de datos. La clase base para todos los objetos <b>DataAdapter</b> es <a href="#">DbDataAdapter</a> .

Adicionalmente a las clases anteriores, los Adaptadores de datos exponen más clases para otras funcionalidades:

Objeto	Descripción
<b>Transaction</b>	Permite incluir comandos en las transacciones que se realizan en el origen de datos. La clase base para todos los objetos <b>Transaction</b> es <a href="#">DbTransaction</a> .
<b>CommandBuilder</b>	Un objeto auxiliar que genera automáticamente las propiedades de comando de un <b>DataAdapter</b> o que obtiene de un procedimiento almacenado información acerca de parámetros con la que puede rellenar la colección <b>Parameters</b> de un objeto <b>Command</b> . La clase base para todos los objetos <b>CommandBuilder</b> es <a href="#">DbCommandBuilder</a> .
<b>ConnectionStringBuilder</b>	Un objeto auxiliar que proporciona un modo sencillo de crear y administrar el contenido de las cadenas de conexión utilizadas por los objetos <b>Connection</b> . La clase base para todos los objetos <b>ConnectionStringBuilder</b> es <a href="#">DbConnectionStringBuilder</a> .
<b>Parameter</b>	Define los parámetros de entrada, salida y valores devueltos para los comandos y procedimientos almacenados. La clase base para todos los objetos <b>Parameter</b> es <a href="#">DbParameter</a> .
<b>Exception</b>	Se devuelve cuando se detecta un error en el origen de datos. En el caso de que el error se detecte en el cliente, los proveedores de datos de .NET Framework inician una excepción de .NET Framework. La clase base para todos los objetos <b>Exception</b> es <a href="#">DbException</a> .
<b>Error</b>	Expone la información relacionada con una advertencia o error devueltos por un origen de datos.
<b>ClientPermission</b>	Se proporciona para los atributos de seguridad de acceso a código de los proveedores de datos de .NET Framework. La clase base para todos los objetos <b>ClientPermission</b> es <a href="#">DbDataPermission</a> .

## 1.2.- PROVEEDORES DE DATOS

### Proveedor de Datos para Microsoft SQL Server ( **System.Data.SqlClient** )

Para utilizar el proveedor de datos de .NET Framework para SQL Server, se requiere SQL Server 7.0 ó posterior. Para versiones anteriores se recomienda emplear el proveedor de datos OLE DB con el controlador OLE DB para SQL Server ( SQLOLEDB ).

### Proveedor de Datos para Orígenes OLE DB. ( **System.Data.OleDb** )

Este proveedor de datos permite acceder a orígenes de datos mediante controladores OLEDB. Se emplea como proveedor genérico cuando no se dispone de uno más específico para un determinado tipo de base de datos.

Controlador	Proveedor
SQLOLEDB	Proveedor OLE DB para SQL Server de Microsoft
MSDAORA	Proveedor OLE DB para Oracle de Microsoft
Microsoft.Jet.OLEDB.4.0	Proveedor OLE DB para Microsoft Jet

## ACCESO A DATOS CON ADO.NET

### Proveedor de Datos para Orígenes ODBC. ( **System.Data.Odbc** )

Este proveedor de datos realiza el acceso mediante un driver ODBC ( Object Data Base Control ), que constituye una tecnología anterior aún a OLE DB. Con este proveedor se puede acceder a la práctica totalidad de las bases de datos existentes.

#### **Controlador**

SQL Server

Microsoft ODBC para Oracle

Microsoft Access Driver (\*.mdb)

### Proveedor de Datos para Oracle ( **System.Data.OracleClient** )

El proveedor de datos de .NET Framework para Oracle permite el acceso a datos de orígenes de datos de Oracle mediante el software de conectividad de cliente de Oracle. El proveedor de datos es compatible con la versión 8.1.7 o posterior del software de cliente de Oracle.

El proveedor de datos de .NET Framework para Oracle requiere que el software de cliente de Oracle (8.1.7 o posterior) esté instalado en el sistema antes de conectarse a un origen de datos de Oracle.

### Otros proveedores de datos y consideraciones.

Además de éstos proveedores de datos incluidos en el Framework de .NET, existen otros muchos proveedores específicos para diferentes tipos de bases de datos, como por ejemplo MySQL.

Siempre que se disponga de un proveedor de datos específico para un tipo de base de datos será preferible su uso antes que cualquier otro proveedor tipo OLEDB ó ODBC. Estos proveedores genéricos, no pueden aprovechar las características específicas de cada sistema de bases de datos, y el rendimiento que ofrecen siempre es inferior al del proveedor específico.



### **1.3.- ONEXION A BASE DE DATOS**

En ADO.NET se utiliza el objeto Connection para la conexión con el origen de datos. La clase Connection es una clase dependiente del origen de datos:

- *OdbcConnection*
- *OleDbConnection*
- *OracleConnection*
- *SqlConnection*

Todas estas poseen un conjunto común de métodos y propiedades heredados de la clase base DbConnection.

### **1.4.- CADENA DE CONEXION**

Para que una clase conexión pueda establecer comunicación con un determinado origen de datos es necesario fijar previamente una *Cadena de Conexión*. La Cadena de conexión es una simple cadena de texto que contiene toda la información necesaria para conectar con un origen de datos concreto.

Ejemplos:

Cadena de conexión a base de datos Access vía OLEDB:

```
Provider=Microsoft.Jet.OLEDB.4.0;  
Data Source=d:\Northwind.mdb;  
User ID=Admin;  
Password=;
```

Cadena de conexión a base de datos SQL Server:

```
Persist Security Info=False;  
Integrated Security=true;  
Initial Catalog=Adventureworks;  
Data Source=192.168.1.5"
```

Las cadenas de conexión tienen una sintaxis distinta dependiendo del origen de datos. Éstas pueden componerse directamente en código como en el ejemplo anterior, o emplear la clase generadora apropiada:

- *SqlConnectionStringBuilder*
- *OdbcConnectionStringBuilder*
- *OleDbConnectionStringBuilder*
- *OracleConnectionStringBuilder*

Las clases generadoras de cadenas de conexión funcionan como una colección de pares Clave-Valor configurables. Las claves y valores dependen de cada origen de datos.

## ACCESO A DATOS CON ADO.NET

Generación de cadena de conexión para SQL Server:

```
[C#]
using System.Data;
using System.Data.SqlClient;

SqlConnectionStringBuilder builder = new SqlConnectionStringBuilder();
builder["Data Source"] = "192.168.10.150"; // Dirección IP del servidor SQL
builder["Integrated Security"] = true;
builder["Initial Catalog"] = "Adventureworks"; //Nombre de la base de datos
String cadenaConexion = builder.ConnectionString;
```

Resultado:

Source=(local);Initial Catalog="Adventureworks"; Integrated Security=True

Generación de cadena de conexión para Access:

```
[C#]
using System.Data;
using System.Data.OleDb;
OleDbConnectionStringBuilder builder = new OleDbConnectionStringBuilder();
builder["Provider"] = "Microsoft.Jet.OLEDB.4.0";
builder["Data Source"] = "C:\\Sample.mdb";
builder["User Id"] = "Admin";
String cadenaConexion = builder.ConnectionString;
```

Resultado:

Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\\Sample.mdb;User ID=Admin

### 1.5.- CONEXION Y DESCONEXION CON EL ORIGEN DE DATOS

La conexión con la base de datos se realiza a través de una instancia de Connection correspondiente al origen de datos:

- *SqlConnection*
- *OdbcConnection*
- *OleDbConnection*
- *OracleConnection*

Para abrir la conexión se emplea el método Open. Para cerrar la conexión se emplea igualmente el método Close.

```
[C#]
using System.Data;
using System.Data.SqlClient;

SqlConnectionStringBuilder builder = new SqlConnectionStringBuilder();







// parametros de generación de la conexión
SqlConnection conn = new SqlConnection( builder.ConnectionString );

// Conexion
conn.Open();

// comprobación estado de la conexión
if ( conn.State == ConnectionState.Open ) {
    Console.WriteLine("Conexion abierta");

    // Desconexion
    conn.Close();
}
```

La propiedad `State` de la clase `Connection` permite conocer el estado de la conexión en un momento dado. La propiedad devuelve un valor perteneciente al tipo enumerado `ConnectionState` ( `System.Data` ). Los valores posibles son los siguientes:

	Nombre de miembro	Descripción
	<b>Broken</b>	Se ha perdido la conexión con el origen de datos. Esto sólo puede ocurrir tras abrir la conexión. Una conexión en este estado se puede cerrar y volver a abrir. Este valor se reserva para versiones futuras del producto.
	<b>Closed</b>	La conexión está cerrada.
	<b>Connecting</b>	El objeto de conexión está conectando con el origen de datos. Este valor se reserva para versiones futuras del producto.
	<b>Executing</b>	El objeto de conexión está ejecutando un comando. Este valor se reserva para versiones futuras del producto.
	<b>Fetching</b>	El objeto de conexión está recuperando datos. Este valor se reserva para versiones futuras del producto.
	<b>Open</b>	La conexión está abierta.

Listado de los métodos comunes a todas las clases `Connection`.

	Nombre	Descripción
	<a href="#">BeginTransaction</a>	Sobrecargado. Inicia una transacción de base de datos.
	<a href="#">ChangeDatabase</a>	Cambia la base de datos actual para una conexión abierta.
	<a href="#">Close</a>	Cierra la conexión con la base de datos. Éste es el método recomendado para cerrar conexiones abiertas.
	<a href="#">CreateCommand</a>	Crea y devuelve un objeto <a href="#">DbCommand</a> asociado a la conexión actual.
	<a href="#">CreateObjRef</a>	Crea un objeto que contiene toda la información relevante necesaria para generar un proxy utilizado para comunicarse con un objeto remoto. (Se hereda de <a href="#">MarshalByRefObject</a> ).
	<a href="#">Dispose</a>	Sobrecargado. Libera los recursos utilizados por el objeto <a href="#">Component</a> . (Se hereda de <a href="#">Component</a> ).
	<a href="#">EnlistTransaction</a>	Da de alta en la transacción especificada.
	<a href="#">Equals</a>	Sobrecargado. Determina si dos instancias de <a href="#">Object</a> son iguales. (Se hereda de <a href="#">Object</a> ).
	<a href="#">GetHashCode</a>	Sirve como función hash para un tipo concreto. <a href="#">GetHashCode</a> es apropiado para su utilización en algoritmos de hash y en estructuras de datos como las tablas hash. (Se hereda de <a href="#">Object</a> ).
	<a href="#">GetLifetimeService</a>	Recupera el objeto de servicio de duración actual que controla la directiva de duración de esta instancia. (Se hereda de <a href="#">MarshalByRefObject</a> ).
	<a href="#">GetSchema</a>	Sobrecargado. Devuelve información de esquema para el origen de datos de <a href="#">DbConnection</a> .
	<a href="#">GetType</a>	Obtiene el objeto <a href="#">Type</a> de la instancia actual. (Se hereda de <a href="#">Object</a> ).
	<a href="#">InitializeLifetimeService</a>	Obtiene un objeto de servicio de duración para controlar la directiva de duración de esta instancia. (Se hereda de <a href="#">MarshalByRefObject</a> ).
	<a href="#">Open</a>	Abre una conexión a base de datos con la configuración que especifica <a href="#">ConnectionString</a> .
	<a href="#">ReferenceEquals</a>	Determina si las instancias de <b>Object</b> especificadas son la misma instancia. (Se hereda de <a href="#">Object</a> ).
	<a href="#">ToString</a>	Devuelve un objeto <a href="#">String</a> que contiene el nombre del objeto <b>Component</b> , en caso de que exista. Este método no debe reemplazarse. (Se hereda de <a href="#">Component</a> ).

### 1.6.- CONSULTAS CONTRA BASE DE DATOS

Una vez establecida una conexión con el origen de datos es posible ejecutar una consulta para recuperar, modificar, insertar o eliminar datos. Para ello se emplean las diferentes clases comando según el origen de datos:

- *SqlCommand*
- *OleDbCommand*
- *OdbcCommand*
- *OracleCommand*




Todas las clases comparten una serie de métodos y propiedades comunes recogidas en la clase *DbCommand* de la que heredan todas ellas.

Todas las operaciones de consulta y manipulación contra un origen de datos siguen los siguientes pasos:

- 1.- Creación de la instancia de la clase Comando adecuado
- 2.- Establecer parámetros de la instancia comando a ejecutar
- 3.- Ejecución de la instancia comando y recuperación de resultados ( si hay ).
- 4.- Manipulación de los datos.

La creación de la instancia de la clase comando se puede obtener invocando el método *CreateCommand* de la clase conexión que se esté empleando. Una vez creada deben configurarse sus propiedades para ejecutar la operación deseada:

*Propiedad CommandType*: Indica el tipo de operación que va a llevarse a cabo:  
Valores → Enumerado Command Type:

	Nombre de miembro	Descripción
	<b>StoredProcedure</b>	Nombre del procedimiento almacenado.
	<b>TableDirect</b>	Nombre de una tabla.
	<b>Text</b>	Comando de texto SQL. Predeterminado.

*Propiedad CommandText*: Indica la sentencia SQL o el nombre del procedimiento a ejecutar en función del valor de *CommandType*.

*Colección Parameters*: Permiten añadir valores a los parámetros en caso de emplearse una consulta parametrizada o un procedimiento almacenado.

## 1.7.- EJECUCION DE UNA CONSULTA

Puede decirse que existen cuatro tipos distintos de consultas:

- Consultas con retorno de datos
- Consultas con retorno de valor escalar
- Sentencias de modificación que no retornan resultados ( altas, baja, modificaciones ).
- Ejecución de procedimientos almacenados

### 1.7.1.- Ejecución de consultas escalares

Algunas operaciones contra una base de datos devuelven un único valor en vez de un conjunto de filas y columnas. Este tipo de operaciones son consultas escalares de tipo “SELECT COUNT(\*) FROM CLIENTES” en las que se retorna un único registro con una única columna y valor.

La única diferencia con la ejecución de un comando de consulta convencional radica en el método a invocar para la ejecución: **ExecuteScalar**.

```
[C#]
using System.Data;
using System.Data.SqlClient;

SqlConnection conn = new SqlConnection( ... );
conn.Open();
SqlCommand cmd = conn.CreateCommand();
cmd.CommandText = "SELECT COUNT(*) FROM CLIENTES";
cmd.CommandType = CommandType.Text;
// Recuperación del valor de la primera columna del primer registro.
int cantidad = Convert.ToInt32(cmd.ExecuteScalar());
```

### 1.7.2.- Ejecución de consultas sin retorno de datos

Las operaciones de modificación de datos tales como altas, bajas y modificaciones de datos no provocan ningún retorno de datos. En estos casos, la única información que se recibe es el número de registros afectados por la operación.

Para ejecutar este tipo de comando es necesario invocar el método **ExecuteNonQuery**.

```
[C#]
using System.Data;
using System.Data.SqlClient;

SqlConnection conn = new SqlConnection( ... );
conn.Open();
SqlCommand cmd = conn.CreateCommand();
cmd.CommandText = "INSERT INTO Usuarios( nombre, apellido, edad ) VALUES
                  ( @nom, @ape, @ed )";
cmd.CommandType = CommandType.Text;
// Inclusión + instanciación de parámetros
cmd.Parameters.Add( new SqlParameter( "@nom", "Roger" ));
cmd.Parameters.Add( new SqlParameter( "@ape", "Petrov" ));
cmd.Parameters.Add( new SqlParameter( "@ed", 24 ));
// Recuperación de número de registros actualizados
int regs = cmd.ExecuteNonQuery();
```

### 1.7.3.- Ejecución de consultas con retorno de datos

El tipo más simple de consulta es la que se ejecuta en base a una instrucción SQL y retorna unos resultados que pueden ser leídos. La propiedad `CommandType` determina el tipo de comando que se va a ejecutar. En el caso de una consulta SQL el valor es `CommandType.Text`. La sentencia SQL se establece mediante la propiedad `CommandText`.

Los datos resultantes de la consulta pueden obtenerse en modo conectado, o en modo desconectado. El modo conectado implica que se mantiene la conexión con el origen de datos y se recuperan los registros de uno en uno mediante una instancia específica de la clase `DataReader`. El modo desconectado implica la descarga de toda la información en una instancia de la clase `DataSet` y la desconexión de la base de datos.

### 1.7.4.- Llamadas a procedimientos almacenados

Un procedimiento almacenado es una mini-aplicación programada en SQL que se almacena y ejecuta en el servidor de bases de datos. Para ejecutar un procedimiento almacenado es necesario invocarlo por su nombre enviándole los parámetros que requiera.

La propiedad `CommandType` debe establecerse con el valor `CommandType.StoredProcedure`. La propiedad `CommandText` debe recibir el nombre del procedimiento almacenado.

Para cada parámetro de entrada al procedimiento almacenado, será necesario crear una instancia de la clase `Parameter`, y añadirla a la correspondiente colección `Parameters` de la instancia `Command`.

Cada instancia representa un parámetro para el procedimiento almacenado, con su nombre, tipo, longitud y valor asociado. Los parámetros pueden ser de entrada o de salida. Los parámetros de entrada envían datos al procedimiento almacenado, los de salida permiten recibir datos resultantes de la ejecución del procedimiento almacenado. Para declarar un parámetro de tipo salida debe definirse la propiedad `Direction` al valor `ParameterDirection.Output`.

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

SqlCommand cmd = conn.CreateCommand();
cmd.CommandText = "BuscarCliente";
cmd.CommandType = CommandType.StoredProcedure;
// Creación de parámetro @nombre de tipo NVarchar de 15 caracteres.
SqlParameter param = new SqlParameter("@nombre", SqlDbType.NVarChar, 15 );
param.Value = "Roger Petrov";

// Creación de parámetro de retorno @retorno de tipo Integer
SqlParameter retorno = new SqlParameter();
retorno.ParameterName="@retorno";
retorno.SqlDbType = SqlDbType.Int;
retorno.Direction = ParameterDirection.Output;

// Inclusión de parámetro en la SqlParameterCollection del commando.
cmd.Parameters.Add( param );
cmd.Parameters.Add( retorno );

SqlDataReader reader = cmd.ExecuteReader();
int valorRetorno = Int.Parse( retorno.Value );
```

## 1.8.- RECUPERACION DE DATOS EN MODO CONECTADO

El acceso a base de datos en modo conectado emplea las clases *DataReader*. Normalmente, se emplea el modo de acceso conectado para operaciones puntuales sobre la base de datos, tales como; inserciones, actualizaciones, localización y/o eliminación de registros. También se puede emplear para realizar consultas mediante invocando el método ***ExecuteReader*** del objeto *command* correspondiente.

Cada origen de datos posee una clase *DataReader* específica derivada de la clase padre *DbDataReader*.

- *SqlDataReader*
- *OracleDaraReader*
- *OdbcDataReader*
- *OleDbDataReader*

Las clases *DataReader* representan secuencias de registros de solo lectura recuperados desde el origen de datos de uno en uno según se leen. Los datos de cara registro recuperado por una instancia de *DataReader* se obtienen mediante sus métodos conforme el tipo de dato. Para indicar la columna se puede emplear su nombre, o su índice en el orden de columna:

	<a href="#">GetBoolean</a>	Obtiene el valor de la columna especificada como tipo Boolean.
	<a href="#">GetByte</a>	Obtiene el valor de la columna especificada como byte.
	<a href="#">GetBytes</a>	Lee una secuencia de bytes de la columna especificada, a partir de la posición que indica <i>dataIndex</i> , y los copia en el búfer comenzando en la ubicación que indica <i>bufferIndex</i> .
	<a href="#">GetChar</a>	Obtiene el valor de la columna especificada como un único carácter.
	<a href="#">GetChars</a>	Lee una secuencia de caracteres de la columna especificada, a partir de la posición que indica <i>dataIndex</i> , y los copia en el búfer comenzando en la ubicación que indica <i>bufferIndex</i> .
	<a href="#">GetData</a>	Devuelve un objeto <b>DbDataReader</b> para el ordinal de columna solicitado.
	<a href="#">GetDataTypeName</a>	Obtiene el nombre del tipo de datos de la columna especificada.
	<a href="#">GetDateTime</a>	Obtiene el valor de la columna especificada como un objeto <a href="#">DateTime</a> .
	<a href="#">GetDecimal</a>	Obtiene el valor de la columna especificada como un objeto <a href="#">Decimal</a> .
	<a href="#">GetDouble</a>	Obtiene el valor de la columna especificada como un número de punto flotante de precisión doble.
	<a href="#">GetEnumerator</a>	Devuelve una interfaz <a href="#">IEnumerator</a> que se puede utilizar para recorrer en iteración las filas en el lector de datos.
	<a href="#">GetFieldType</a>	Obtiene el tipo de datos de la columna especificada.
	<a href="#">GetFloat</a>	Obtiene el valor de la columna especificada como un número de punto flotante de precisión simple.
	<a href="#">GetGuid</a>	Obtiene el valor de la columna especificada como un identificador global único (GUID).
	<a href="#">GetHashCode</a>	Sirve como función hash para un tipo concreto. <a href="#">GetHashCode</a> es apropiado para su utilización en algoritmos de hash y en estructuras de datos como las tablas hash. (Se hereda de <a href="#">Object</a> ).
	<a href="#">GetInt16</a>	Obtiene el valor de la columna especificada como un entero de 16 bits con signo.
	<a href="#">GetInt32</a>	Obtiene el valor de la columna especificada como un entero de 32 bits con signo.
	<a href="#">GetInt64</a>	Obtiene el valor de la columna especificada como un entero de 64 bits con signo.

## ACCESO A DATOS CON ADO.NET

Una instancia de *DataReader* solo posee en cada momento un único registro del conjunto total recuperado por una consulta. Para recuperar el siguiente registro es necesario invocar al método *Read* que retorna un valor booleano indicando si hay o no más registro a continuación.

Ejemplo de recorrido completo de los datos en una instancia de *DataReader*:

```
[C#]
OleDbDataReader reader = cmd.ExecuteReader();
if (reader.HasRows)
    // Si Read() retorna TRUE -> existe otro registro para leer.
    while (reader.Read())
        Console.WriteLine("\t{0}\t{1}", reader.GetInt32(0), reader.GetString(1));
else
    Console.WriteLine("No rows returned.");
// Cierre del DataReader
reader.Close();
```

Los métodos *GetInt32(0)* y *GetString(1)*, retornan el valor Integer de la primer columna y la cadena de texto de la segunda del registro en la memoria del *DataReader*.

También es posible recuperar los campos del registro apuntado por un *DataReader* empleando el operador de indización con el nombre de la columna asociada.

```
while ( reader.Read() ) {
    lblTitulos.Text = reader["titulo"].ToString() + "<br>" + reader["autor"].ToString();
}
```

Utilizando el indizador los valores retornados son de tipo *Object*; por lo que es necesario realizar la conversión al tipo de dato correspondiente en .NET

### Acerca de los *DataReaders* y el uso del modo conectado...

Un *DataReader* usa de forma exclusiva la instancia conexión y no permite ejecutar ningún otro comando mientras esté abierto. Ello es debido a que el *DataReader* recupera de uno en uno los registros desde la base de datos empleando la conexión abierta. Para liberar la conexión es muy importante cerrar siempre el objeto *DataReader* tras su uso. De igual manera; la conexión no debe cerrarse mientras un *DataReader* permanezca abierto, ya que interrumpiría el flujo de datos del objeto *DataReader* en uso.

Es importante destacar que el uso del modo conectado implica que existe una conexión abierta con el servidor de bases de datos por cada usuario durante el uso de la base de datos por parte de éste último. Esto puede conllevar problemas de rendimiento, ya que cada conexión consumirá recursos del servidor incluso aunque el usuario no haga nada. Es por ello que es necesario siempre minimizar en lo posible el tiempo que la aplicación mantiene la conexión con la base de datos.

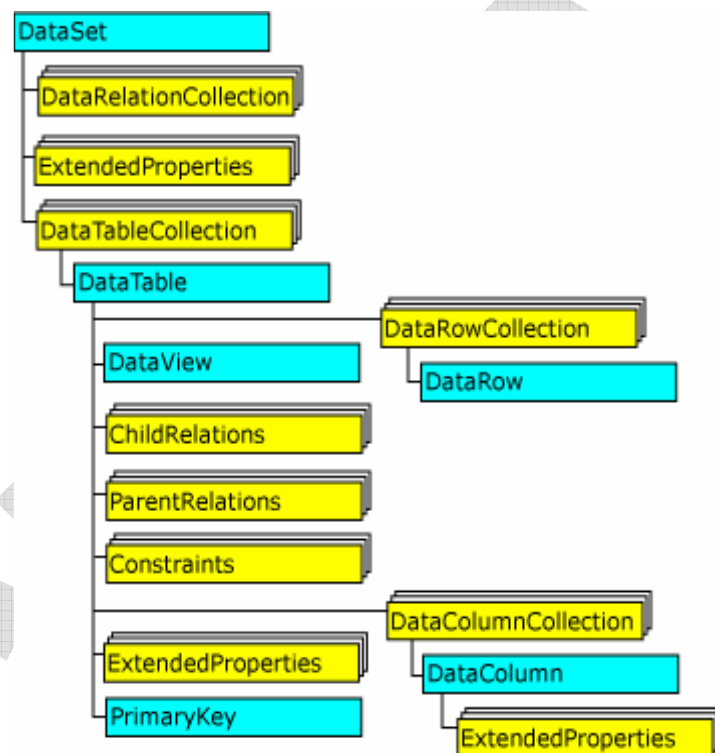
La ventaja que introduce el modo desconectado es que cada usuario establece conexión con la base de datos el tiempo mínimo necesario para recuperar los datos que requiere y cerrarla. Esto permite dar servicio a muchos más usuarios de los que el servidor tolera a la vez, puesto que es difícil que todos se conecten al mismo tiempo en un lapso de tiempo muy pequeño.



## 1.9.- RECUPERACION DE DATOS EN MODO CONECTADO

La clase *DataSet* permite recuperar información de una base de datos sin necesidad de mantener la conexión abierta con la base de datos. Cada objeto *DataSet* constituye una copia local de datos descargados de una base de datos independientemente de la misma. Debido a ello, no existen diferentes clases *DataSet* para cada origen de datos, si no una única para todos.

Los datos almacenados en un objeto *DataSet*, se organizan en múltiples objetos *DataTable*. Cada objeto *DataTable* representa una tabla de datos que contiene unos registros ( objetos *DataRow* ), y está organizada en un número de columnas ( objetos *DataColumn* ); reflejando así siempre el mismo esquema que el de la base de datos de donde provienen.



***Jerarquía de clases asociadas a la clase DataSet***

Es importante entender que un *DataSet* y los datos que contiene son una *copia desconectada* de la información en la base de datos. Si modificamos los datos en el *DataSet*, los cambios no se reflejan en el acto en la base de datos. De igual manera; no son visibles los cambios realizados por otros usuarios en la base de datos una vez cargado el *DataSet*.

Para ello es necesario restablecer la conexión con la base de datos para, bien volver a recargar los datos del *DataSet*, o actualizar la base de datos con los cambios realizados en los datos del *DataSet*.

### 1.9.1.- Clases integrantes de un DataSet.

Las principales clases que intervienen en la estructura de los datos almacenados en un DataSet son:

- **DataTable:** Representa cada una de las tabla almacenadas en el DataSet
- **DataRow:** Representa cada uno de los registros contenidos en una tabla.
- **DataColumn:** Representa cada una de las columnas que conforman una tabla.
- **DataRelation:** Representa cada una de las relaciones entre tablas.

El conjunto de las instancias de cada una de estas clases se almacenan en colecciones accesibles a través de propiedades:

- **DataTableCollection:** Colección con todas las tablas ( DataTable ) presentes en un DataSet. Se obtiene a partir de la propiedad Tables de DataSet.
- **DataRowCollection:** Colección con todas los registros ( DataRow ) presentes en una tabla. Se obtiene a partir de la propiedad Rows de DataTable
- **DataColumnCollection:** Colección con todas las columnas ( DataColumn ) presentes en una tabla. Se obtiene a partir de la propiedad Columns de DataTable.
- **DataRelationCollection:** Colección con todas las relaciones que mantiene una tabla con otras tablas. Se obtienen a partir de las propiedades ParentRelations y ChildRelations de DataTable.

### 1.9.2.- Recuperación de datos de un DataSet.

El siguiente ejemplo muestra por pantalla el nombre de cada tabla contenida en un DataSet seguido del nombre y tipo de sus columnas, y todos los datos de los registros contenidos.

```
[C#]
DataSet oDataSet;
// Bucle de recorrido de tablas presentes en el Dataset
foreach (DataTable oDTTable in oDataSet.Tables)
{
    Console.WriteLine("NOMBRE TABLA: " + oDTTable.TableName);
    // visualizacion nombres y tipos de datos de columnas
    foreach (DataColumn oDCol in oDTTable.Columns)
    {
        Console.WriteLine("COLUMNA:" + oDCol.ColumnName + " TIPO:" +
oDCol.DataType.Name);
    }
    // Bucle de recorrido de todos los registros en una tabla
    foreach (DataRow oDRow in oDTTable.Rows)
    {
        // visualizacion del valor de cada campo de cada registro.
        for (int indx = 0; indx < oDTTable.Columns.Count; indx++)
        {
            Console.Write(oDRow[indx].ToString());
        }
    }
    Console.WriteLine("FIN COLUMNA");
}
```






## 1.9.3.- Manipulación de datos en un DataSet.

Los datos contenidos en los registros de las tablas de un DataSet pueden manipularse, y los registros pueden ser eliminados y creados. Para ello se trabaja principalmente con las instancias DataRow contenidas en la DataRowCollection de una DataTable.

Las principales operaciones de manipulación de datos son:

- 1.- Modificar el valor de los campos de un registro
- 2.- Insertar nuevos registros a una tabla
- 3.- Eliminar un registro de una tabla

Las operaciones de modificación, eliminación o inserción de registros en un DataSet son registradas mediante la propiedad RowState de las instancias DataRow que contiene. Esta propiedad indica si un registro ha sido modificado, creado o eliminado mediante un valor de la enumeración DataRowStates.

Nombre de miembro	Descripción
 <b>Added</b>	La fila se ha agregado a <b>DataRowCollection</b> y no se ha llamado a <a href="#">AcceptChanges</a> .
 <b>Deleted</b>	La fila se ha eliminado mediante el método <b>Delete</b> del <b>DataRow</b> .
 <b>Detached</b>	Se ha creado la fila, pero no forma parte de ningún <b>DataRowCollection</b> . <b>DataRow</b> se encuentra en este estado inmediatamente después de haber sido creado y antes de que se agregue a una colección, o bien si se ha quitado de una colección.
 <b>Modified</b>	La fila se ha modificado y no se ha llamado a <b>AcceptChanges</b> .
 <b>Unchanged</b>	La fila no ha cambiado desde que se llamó a <b>AcceptChanges</b> por última vez.

*Valores de la enumeración DataRowStates que indican el estado de modificación de un DataRow*

Para enviar los cambios en un DataSet a la base de datos es necesario convertirlos en sentencias SQL. Por cada registro modificado, insertado, o eliminado en el DataSet; se genera y envía una sentencia SQL al servidor. Para ello se emplea la propiedad RowState de cada registro. Esto se denomina proceso de actualización y mientras no se lleva a cabo los cambios sólo existen en la memoria del ordenador.

Una vez completado con éxito el proceso de actualización se pueden confirmar los cambios en el DataSet invocando al método *AcceptChanges*, o anularlos antes de enviarlos a la base de datos invocando *RejectChanges*.

### 1.9.4.- Modificación de los campos de un registro.

Para modificar los datos de un determinado registro basta con asignar los nuevos valores a los campos de la instancia DataRow que represente el registro en cuestión.

```
[C#]
// Obtenemos la instancia DataRow del segundo registro de la tabla
DataRow dr = dt.Rows[2];
// Visualiza por consola el valor antiguo del campo nombre del registro
Console.WriteLine( dr["NOMBRE"].ToString());
// Modificamos el valor del campo "NOMBRE" del registro.
dr["NOMBRE"] = "Petrov";
```

La clase DataRow define un indizador para poder acceder a los valores de sus campos como si de una matriz se tratase. De esta manera cada uno de los campos que conforman el registro pueden ser accedidos mediante su índice ordinal o el propio nombre de la columna.

### 1.9.5.- Inserción de un nuevo registro.

Si deseamos crear un nuevo registro, se debe crear una nueva instancia DataRow. Esta instancia representará a un nuevo registro de la tabla con sus campos correspondientes. Para ello es necesario emplear el método NewRow de la propia DataTable en vez del constructor de la clase DataRow.

```
[C#]
// Obtengo una nuevo registro
DataRow dr = dt.NewRow();
// Asigno valores a los campos del nuevo registro
dr["NOMBRE"] = "Jhonny";
dr["EDAD"] = 21;
// Inserto nuevo registro en la colección de registros de la tabla.
dt.Rows.Add(dr);
```

Una vez obtenida la instancia del nuevo registro debe darse valor a sus campos y finalmente añadir el registro a la colección de registros de la tabla ( DataRowCollection ). Para acceder a la colección debemos emplear la propiedad Rows, y al igual que con cualquier otro tipo de colección para añadir un nuevo elemento debe emplearse el método Add.

### 1.9.6.- Eliminación de un registro

Si deseamos eliminar un registro de la tabla debemos invocar al método Delete de la instancia DataRow. Esto no elimina el registro si no que altera su propiedad RowState con el valor DataRowState.Deleted. La eliminación real de los registros en la base de datos tiene lugar al actualizar el DataSet.

```
[C#]
// Obtengo la instancia de DataRow del segundo registro de la tabla.
DataRow dr = dt.Rows[2];
// Elimina el registro de la colección de registros de la tabla.
dr.Delete();
```

## 1.10.- EL ADAPTADOR DE DATOS

El Adaptador de datos es un objeto que se encarga de hacer de intermediario entre los datos desconectados almacenados en una instancia DataSet, y el origen de datos del que proceden. Sus principales dos funciones son rellenar una instancia DataSet con los resultados de una consulta, y actualizar posteriormente en el origen de datos las modificaciones que se hayan realizado en el DataSet. En ambos casos se mantiene una conexión con la base de datos el mínimo tiempo posible.

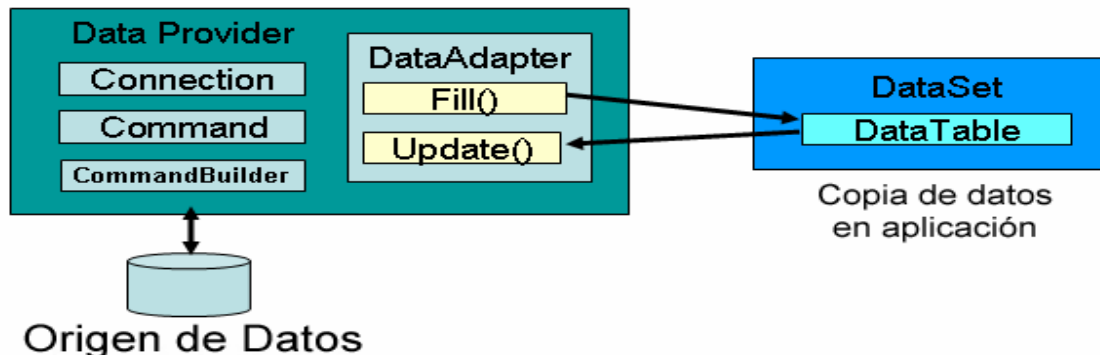
Cada proveedor de datos cuenta con su propia clase DataAdapter:

- *SqlDataAdapter*
- *OracleDataAdapter*
- *OleDbDataAdapter*
- *OdbcDataAdapter*

Todas las clases comparten un conjunto común de métodos y propiedades derivados de la clase padre DbDataAdapter. Sus funciones principales son:

- Rellenar una instancia DataSet con los datos provenientes del origen de datos.
- Actualizar los cambios realizados en un DataSet en el origen de datos.

La clase DataAdapter dispone de los métodos Fill y Update para llevar a cabo las operaciones de relleno y actualización de instancias de DataSet.



La clase DataAdapter dispone de cuatro propiedades con instancias DataCommand:

- **SelectCommand:** Contiene el comando de consulta que permite cargar un DataSet.
- **UpdateCommand:** Contiene el comando de actualización para los registros modificados.
- **InsertCommand:** Contiene el comando de inserción para el alta de registros creados.
- **DeleteCommand:** Contiene el comando de eliminación para los registros eliminados en el DataSet.

La propiedad *SelectCommand* se encarga de recuperar la información del origen de datos para cargar un DataSet. Las propiedad *InsertCommand*, *DeleteCommand*, y *UpdateCommand* administran la actualización de la información en el origen de datos para reflejar las modificaciones del DataSet.

## 1.10.1.- Creación de un DataAdapter

El adaptador de datos se obtiene creando una instancia de la clase correspondiente al origen de datos contra el que vayamos a trabajar. Estas clases ofrecen todas múltiples constructores con diferentes parámetros:

Nombre	Descripción
<a href="#">SqlDataAdapter ()</a>	Inicializa una nueva instancia de la clase <b>SqlDataAdapter</b> . Compatible con .NET Compact Framework.
<a href="#">SqlDataAdapter (SqlCommand)</a>	Inicializa una nueva instancia de la clase <b>SqlDataAdapter</b> con el objeto <a href="#">SqlCommand</a> especificado como propiedad <a href="#">SelectCommand</a> . Compatible con .NET Compact Framework.
<a href="#">SqlDataAdapter (String, SqlConnection)</a>	Inicializa una nueva instancia de la clase <b>SqlDataAdapter</b> con una propiedad <b>SelectCommand</b> y un objeto <a href="#">SqlConnection</a> . Compatible con .NET Compact Framework.
<a href="#">SqlDataAdapter (String, String)</a>	Inicializa una nueva instancia de la clase <b>SqlDataAdapter</b> con una propiedad <b>SelectCommand</b> y una cadena de conexión. Compatible con .NET Compact Framework.

*Lista de constructores de la clase adaptador de datos para SQL Server*

## 1.10.2.- Recuperación de datos de la base de datos mediante DataAdapter

El método **Fill** miembro de DbDataAdapter rellena un DataSet con los resultados de la ejecución de SelectCommand. El objeto DataSet debe ser inicializado previamente y pasado como parámetro.

```
[C#]
Using System.Data;
Using System.Data.SqlClient;

// cadena de texto con consulta parametrizada
string selectSQL =
    "SELECT CustomerID, CompanyName FROM Customers " +
    "WHERE CountryRegion = @CountryRegion AND City = @City";
// Se supone la existencia de una instancia SqlConnection ya configurada
SqlDataAdapter adapter = new SqlDataAdapter();
// Instanciación de SqlCommand para propiedad SelectCommand del adaptador
SqlCommand selectCMD = new SqlCommand(selectSQL, connection);
adapter.SelectCommand = selectCMD;
// Añadido de parámetros para comando consulta
selectCMD.Parameters.Add(
    "@CountryRegion", SqlDbType.NVarChar, 15).Value = "UK";
selectCMD.Parameters.Add(
    "@City", SqlDbType.NVarChar, 15).Value = "London";
// Instanciación de DataSet
DataSet customers = new DataSet();
// Rellenado de datos
adapter.Fill(customers, "Customers");
```

Para la realización de una consulta es necesario que la instancia del adaptador de datos tenga definida previamente la propiedad SelectCommand con un comando de consulta para el origen de datos.

## 1.10.3.- Actualización de la base de datos mediante DataAdapter.

El método **Update** miembro de las clases adaptadores de datos permite enviar las modificaciones realizadas en un DataSet al origen de datos para actualizar su información. Para ello son necesarios los comandos de las propiedades UpdateCommand, InsertCommand y DeleteCommand.

El proceso de actualización de un DataSet consiste en recorrer todos los registros ( DataRow ) de cada una de sus tablas ( DataTable ) comprobando su propiedad RowState. En función del estado del DataRow, el Adaptador de datos emplea el comando adecuado para insertar, modificar o eliminar el registro en la base de datos:

- **Registro Creado:** ( *RowState -> DataRowState.Added* ); se emplea el comando en InsertCommand para darlo de alta en la base de datos.
- **Registro Modificado** ( *RowState = DataRowState.Modified* ); se emplea el comando en UpdateCommand para actualizar sus campos en la base de datos.
- **Registro Eliminado** ( *RowState = DataRowState.Deleted* ); se emplea el comando DeleteCommand para eliminarlo en la base de datos definitivamente.
- **Registro No Modificado** ( *RowState = DataRowState.Unchanged* ). Estos registros no han sido modificados y no se actualizan.

Ejemplo de consulta, modificación, eliminación e inserción de registros a través de un adaptador de datos de SQL Server.

El método *Update* posee varias sobrecargas que es conveniente conocer:

- **Update( DataTable tabla )** : Actualiza la base de datos con la información del DataTable especificado.
- **Update( DataSet data, String nombreTabla )** : Actualiza la base de datos con la información de la tabla contenida en el DataSet especificado, cuyo nombre se indica como segundo parámetro.

Una vez realizado el proceso de actualización, es necesario que todos los registros cambien su estado a no modificado ( *RowState = DataRowState.Unchanged* ). De esta manera, se evitan actualizaciones redundantes. Para ello se debe invocar el método **AcceptChanges** del objeto DataSet empleado.

Por supuesto, este método debe ser utilizado **siempre después** de invocar al método Update del adaptador de datos utilizado, o de lo contrario la actualización es imposible.

## ACCESO A DATOS CON ADO.NET

La instancia `DataAdapter` puede abrir y cerrar por si misma la conexión al origen de datos si no está abierta. La conexión se establece entonces automáticamente al invocar el método `Fill` y es cerrada acto seguido. Lo mismo ocurre con el método `Update`.

Esto puede simplificar el código en operaciones sencillas de lectura y actualización. Sin embargo, en el caso de que se estén realicen varias operaciones que necesiten de una conexión abierta, puede mejorar el rendimiento si se abre y cierra la conexión manualmente mediante los métodos `Open` y `Close`.

Es muy importante siempre mantener la conexión con el origen de datos abierta el menor tiempo posible para liberar recursos en el servidor de base de datos.



### 1.11.- EL GENERADOR DE COMANDOS

Como ya se ha mencionado, las clases adaptadores de datos poseen cuatro propiedades con los comandos de consulta, alta, bajas y modificaciones necesarios para obtener y actualizar la información de una base de datos desde una instancia de DataSet.

El comando de consulta ( propiedad `SelectCommand` ) es definido normalmente por el usuario, no obstante; el resto de los comandos de actualización pueden ser generados automáticamente en base al comando de consulta. Para ello se emplean las clases `CommandBuilder`. Existe una clase específica para cada origen de datos:

- *SqlCommandBuilder*
- *OleDbCommandBuilder*
- *OdbcCommandBuilder*
- *OracleCommandBuilder*

Para obtener una instancia del generador de comandos se invoca a su constructor pasando como parámetro el adaptador de datos que queremos emplear. De esta manera, la instancia `CommandBuilder` resultante generará comandos de las propiedades `UpdateCommand`, `DeleteCommand` e `InsertCommand` en función del comando asignado a la propiedad `SelectCommand`,

Obviamente, es necesario tener declarada la propiedad `SelectCommand` en el adaptador de datos. El comando de consulta debe recuperar una única tabla e incluir una clave primaria o una columna con restricción única. En caso contrario, se lanza una excepción ***InvalidOperationException..***

```
[C#]
// Creación de instancia comando a partir de instancia conexion
SqlCommand selectCommand = conn.CreateCommand();
// Declaración de la sentencia de consulta. ( tipo : SQL )
selectCommand.CommandType = CommandType.Text;
selectCommand.CommandText = "SELECT * FROM dbo.usuarios";

// Declaración e inicialización de la instancia DataSet
DataSet dset = new DataSet();

// Declaración e inicialización de la instancia DataAdapter
SqlDataAdapter adapter = new SqlDataAdapter(selectCommand);

// Declaración e inicialización del generador de comandos para el adaptador de datos
SqlCommandBuilder comBuilder = new SqlCommandBuilder(adapter);
```

La estructura de la tabla recuperada por el comando de *SelectCommand* determina la sintaxis de los comandos de actualización, inserción y eliminación.

En este caso, el generador de comandos ejecuta la consulta de `SelectCommand` dos veces para obtener los metadatos necesarios para generar los comandos SQL INSERT, UPDATE y DELETE. Si durante la ejecución se modifica el comando de la propiedad `SelectCommand` del adaptador de datos deben volverse a generar los comandos de actualización. Para ello debe invocarse el método `RefreshSchema` del objeto generador de comandos.

## ACCESO A DATOS CON ADO.NET

En una implementación normal de varios niveles, los pasos de creación y actualización de un **DataSet** y, a su vez, de actualización de los datos originales, son los siguientes:

1. Construir y llenar cada **DataTable** de un **DataSet** con datos desde un origen de datos mediante **DataAdapter** empleando el método [Fill](#).
2. Cambiar los datos de los objetos **DataTable** individuales mediante la adición, actualización o eliminación de objetos [DataRow](#).
3. Llamar al método [Update](#) de **DataAdapter**, pasando el segundo **DataSet** como argumento.
4. Invocar al método [AcceptChanges](#) de **DataSet**. O bien, invocar al método [RejectChanges](#) para cancelar los cambios.

### 1.11.1.- Cuestiones sobre el uso del generador de comandos

Los adaptadores de datos presentan una serie de limitaciones a la hora de generar automáticamente las consultas de modificación, inserción y eliminación de registros necesarias para la actualización posterior de los objetos **DataSet**.

El requisito mínimo para que la generación automática de comandos funcione correctamente consiste en establecer la propiedad **SelectCommand**. El esquema de tabla que recupera la propiedad **SelectCommand** determina la sintaxis de las instrucciones **INSERT**, **UPDATE** y **DELETE** generadas automáticamente. La consulta indicada en la propiedad **SelectCommand** debe además devolver al menos una clave principal o columna única. Si no hay ninguna, se inicia una excepción **InvalidOperationException** y no se genera ningún comando.

Cuando se asocia con un **DataAdapter**, a un objeto **CommandBuilder** genera automáticamente las propiedades **InsertCommand**, **UpdateCommand** y **DeleteCommand** del **DataAdapter** si no se indican manualmente las consultas correspondientes para la inserción, eliminación y actualización de los datos.

En la siguiente tabla se muestra cómo se aplican cada uno de los comandos del adaptador de datos en función de las modificaciones realizadas a la información contenida en un **DataSet**.

Comando	Regla
<b>InsertCommand</b>	Inserta una fila en el origen de datos para todas las filas de la tabla con una <b>RowState</b> con el valor <a href="#">Added</a> . Inserta valores para todas las columnas actualizables, pero no para determinadas columnas como identidades, expresiones o marcas de hora.
<b>UpdateCommand</b>	Actualiza filas en el origen de datos para todas las filas de la tabla con un valor <b>RowState</b> <a href="#">Modified</a> . Actualiza los valores de todas las columnas, con excepción de las que no son actualizables, como identidades o expresiones. Actualiza todas las filas en las que los valores de columna en el origen de datos coinciden con los valores de la columna de clave principal de la fila, siempre que las restantes columnas del origen de datos coincidan con los valores originales de la fila. Para obtener más información, vea la sección "Modelo de simultaneidad optimista para actualizaciones y eliminaciones" de este mismo tema.
<b>DeleteCommand</b>	Elimina filas en el origen de datos para todas las filas de la tabla con un valor <b>RowState</b> <a href="#">Deleted</a> . Elimina todas las filas en las que los valores de columna coinciden con los valores de la columna de clave principal de la fila, siempre que las restantes columnas del origen de datos coincidan con los valores originales de la fila. Para obtener más información, vea la sección "Modelo de simultaneidad optimista para actualizaciones y eliminaciones" de este mismo tema.

## 1.12.- ALMACENAMIENTO DE LA CADENA DE CONEXIÓN EN Web.Config

Como ya se ha comentado, la cadena de conexión es básicamente una línea de texto que contiene información de un origen de datos, dirección del servidor, y nombre y contraseña de cuenta con permisos para iniciar sesión en el servidor de base de datos al que se desea acceder.

El fichero Web.Config permite almacenar tantas cadenas de conexión como sean necesarias en una aplicación Web, asignando a cada una un nombre identificativo dentro del bloque `<connectionStrings>`

Cada cadena de conexión se añade empleando una etiqueta `<add>` seguida de los atributos: *name*, *providerName* y *connectionString*.

```
<connectionStrings>
  <!-- Primera cadena de conexion -->
  <add name="Conexion_base"
        connectionString="Data Source=WEBSERVER;Initial Catalog=libreria;User
ID=user;Password=0000"
        providerName="System.Data.SqlClient" />
  <!-- Segunda cadena de conexion -->
  <add name="Conexion_aux"
        connectionString="Data Source=miServer\SQLEXPRESS;Initial Catalog=CSR;User
ID=miUsuario;Password=miContraseña"
        providerName="System.Data.SqlClient" />
</connectionStrings>
```

Para recuperar el valor de las cadenas de conexión almacenadas en el Web.Config, se emplea la clase `System.Configuration.ConfigurationManager`. Esta clase posee la propiedad `ConnectionStrings` que retorna una colección de objetos `ConnectionStringSettings`. La colección posee un indizador que permite identificar la cadena de conexión que deseamos recuperar empleando la cadena identificativa asignada al atributo *name* en el Web.Config:

```
ConnectionStringSettings conexion =
    ConfigurationManager.ConnectionStrings["Conexion_base"];

SqlConnection con = new SqlConnection(conexion.ConnectionString);
```

A partir de la instancia `ConnectionStringSettings`, basta invocar la propiedad `ConnectionString` para obtener el String con la cadena de conexión a emplear.

Ejemplo:

```
// Recuperación de la cadena de conexión "Conexion_base" del web.config.
ConnectionStringSettings conexion =
    ConfigurationManager.ConnectionStrings["Conexion_base"];
// Instanciación del objeto conexión con la cadena de conexión recuperada.
SqlConnection con = new SqlConnection(conexion.ConnectionString);
// Instanciación del Adaptador de datos.
SqlDataAdapter adapter = new SqlDataAdapter("SELECT * FROM CLIENTES", con);
// Declaración y descarga de datos en instancia DataSet desconectada del origen de datos
DataSet dt = new DataSet();
adapter.Fill(dt);
```

CLIPSA

# Vínculo a Datos

La vinculación de controles a datos consiste en poder enlazar ciertos controles Web a orígenes de datos.

La vinculación puede llevarse acabo de dos maneras:

1.- **Expresiones de vínculo a datos:** Algunos controles enlazan el valor de sus propiedades al valor de un campo de un registro concreto proveniente de una base de datos. Para ello se emplean expresiones de vínculo a datos en el código de diseño de la página.

2.- **Vinculación de controles de Lista:** Algunos controles de tipo Lista, tales como RadioButtonList, CheckBoxList, DropDownList, o ListBox; se vinculan a un origen de datos representando los datos de una determinada columna.

3.- **Vinculación a controles de Tabla:** Algunos controles como GridView, permiten el vínculo a una tabla completa de un origen de datos.

## 2.1.- VINCULO DE PROPIEDADES

La vinculación individual de las propiedades de un control a datos provenientes de una base de datos no es posible empleando controles origen de datos, como por ejemplo SqlDataSource. En su lugar pueden emplearse objetos DataTable, DataSet, DataView... etc.

Para realizar este tipo de vinculación es necesario emplear expresiones en línea insertadas en el diseño de la página. Estas expresiones designan un vínculo o enlace entre un objeto origen de datos ( p.ej; un *DataTable*, o un *DataView* ), y un propiedad del control a la que se aplica el valor.

Esta situación se da por ejemplo cuando se desea enlazar el contenido de una caja de texto en una página Web con un dato determinado. Para ello basta vincular a la propiedad Text del control TextBox, el valor proveniente de la base de datos que debe mostrar:

```
<asp:TextBox id="myTextBox" runat="server"
  Text='<%= dt[0]["ID"] %>'>
</asp:TextBox>
```

En el ejemplo se muestra la declaración de un control TextBox en el diseño de página. Su propiedad Text está vinculada a la valor que contenga la columna "ID" del primer registro de un objeto DataTable. Es muy importante fijarse que la expresión de vinculación está delimitada siempre por los caracteres: **<%= .. #>**.

## ACCESO A DATOS CON ADO.NET

El mismo ejemplo se le puede aplicar a un control Web de imagen cuya URL se obtenga de un campo desde una base de datos:

```
<asp:Image id="myImage" runat="server"
    ImageUrl='<%= dt[0]["foroURL"]%>'>
```

En este caso es la propiedad `ImageUrl` del control `Image` la que queda enlazada al valor del campo "foroURL" del primer registro del objeto `DataTable` indicado. Este tipo de vinculación puede emplearse con cualquier propiedad de un control siempre que el valor recuperado sea coherente a la propiedad.

Las expresiones de vínculo de datos deben resolverse en tiempo de ejecución para mostrar los valores a los que están vinculados los controles. Para llevar a cabo esta operación es necesario invocar el método `DataBind` de la página.

El método `DataBind` resuelve los vínculos con datos bien a nivel de control o a nivel de página. Esto quiere decir que se puede invocar el método `DataBind` para cada control, o de modo más eficiente; invocararlo a nivel de página. Si se invoca el método `DataBind` de la página, automáticamente se produce una llamada en cascada a los métodos `DataBind` de todos los controles de la misma para que resuelvan sus vínculos a datos.

Este método debe invocarse en las siguientes situaciones:

1.- La primera vez que se invoca a la página, pero justo después de haber rellenado los orígenes de datos. P.ej; una vez rellenado el objeto `DataSet` al que está vinculado uno o varios controles de la página.

```
private void Page_Load ( object src, EventArgs e ) {
    // El adaptador de datos rellena el objeto DataSet: DsAuthors
    myAdapter.Fill ( DsAuthors, "usuarios" );
    if ( !this.IsPostBack ) {
        // Si es la primera llamada a la página → DataBind()
        this.DataBind ( );
    }
}
```

2.- Tras cada modificación en los datos, por ejemplo; una modificación... etc.

No es necesario llamar al método `DataBind` cada vez que se ejecuta la página, puesto que si los datos no han sido modificados en el origen de datos, los propios controles se ocupan de mantenerlos y no es necesario refrescar los vínculos.

## 2.2.- ORIGENES DE DATOS PARA VINCULOS

Puede utilizarse como origen de datos vinculable a un control cualquier objeto cuya clase correspondiente implemente la interfaz *ICollection*, o herede de alguna clase que lo implementa.

- Cualquier colección de datos en memoria como *ArrayList*, *HashTable*, *Dictionary*... etc.

```
protected void Page_Load(object sender, System.EventArgs e)
{
    if (!Page.IsPostBack)
    {
        // Create the data source.
        Hashtable ht = new Hashtable();
        ht.Add("Lasagna", "key1");
        ht.Add("Spaghetti", "key2");
        ht.Add("Pizza", "key3");
        // Set the DataSource property for the controls.
        Select1.DataSource = ht;
        Select2.DataSource = ht;
        Listbox1.DataSource = ht;
        DropDownList1.DataSource = ht;
        CheckList1.DataSource = ht;
        OptionList1.DataSource = ht;
        // Bind the controls.
        this.DataBind();
    }
}
```

- Objetos de la clase *DataReader* que provee acceso conectado a base de datos. Estos objetos sólo permiten la navegación hacia delante y de sólo lectura.

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!Page.IsPostBack)
    {
        // Creación de cadena de conexión y comando de consulta.
        string connectionString =
            WebConfigurationManager.ConnectionStrings[
                "Northwind"].ConnectionString;
        string sql = "SELECT EmployeeID, TitleOfCourtesy + ' ' + " +
            "FirstName + ' ' + LastName AS FullName FROM Employees";
        SqlConnection con = new SqlConnection(connectionString);
        SqlCommand cmd = new SqlCommand(sql, con);
        try
        {
            // Apertura de la conexión y obtención del objeto
            DataReader.
            con.Open();
            SqlDataReader reader = cmd.ExecuteReader();
            // Vinculado del objeto DataReader al control Lista.
            lstNames.DataSource = reader;
            lstNames.DataBind();
            reader.Close();
        }
        finally
        {
            // Cierre de la conexión
            con.Close();
        }
    }
}
```

- Objetos de la clase *DataRow*, *DataSet*, *DataTable* que proveen acceso no conectado a datos.

## 2.3.- VINCULO A CONTROLES DE LISTA

Los controles de tipo Lista ( *ListBox*, *ComboBox* ) requieren de la configuración de tres propiedades para completar correctamente el vínculo a datos.

- **DataSource:** Esta propiedad se emplea para asignar el objeto origen de datos como un ArrayList, HashTable, DataView, DataReader, DataSet.
- **DataSourceID:** Esta propiedad se emplea para asignar un objeto origen de datos ( DataSource ) que se ocupe de obtener los datos desde la base de datos. Esta propiedad y DataSource son excluyentes y no pueden usarse al mismo tiempo.
- **DataTextField:** Esta propiedad indica el nombre de un campo o propiedad cuyo valor se visualizará el control.
- **DataValueField:** Esta propiedad es semejante a DataTextField, pero el valor que se vincula se almacena en el campo *Value* de cada elemento del control lista en vez de visualizarse. El uso más común es almacenar la clave primaria del registro al que se corresponde el valor mostrado para después recuperar el resto de los datos del registro correspondiente al valor seleccionado.
- **DataTextFormatString:** Permite declarar una cadena de texto para formatear cada dato a visualizar antes de mostrarse.

### 2.3.1.- Creación de vínculo desde código.

En el ejemplo siguiente se vincula desde código un control Web ListBox a objeto DataTable cargado previamente.

```
protected void Page_Load(object sender, EventArgs e)
{
   ConnectionStringSettings conexion =
ConfigurationManager.ConnectionStrings["libreriaConnectionString"];
SqlConnection con = new SqlConnection(conexion.ConnectionString);

SqlDataAdapter adapter = new SqlDataAdapter("SELECT * FROM CLIENTES", con);
adapter.Fill(dt);
this.ListBox1.DataSource = dt;
this.ListBox1.DataTextField = "USUARIO";
this.ListBox1.DataValueField = "TELEFONO";
if (!Page.IsPostBack)
{
    DataBind();
}
}
```

En el código anterior se conecta con una base de datos empleando la cadena de conexión definida en el fichero Web.Config con el identificador "libreriaConnectionString". A continuación se crea un objeto Adaptador de Datos y se lanza una consulta cuyos datos se recuperan en un objeto DataTable:

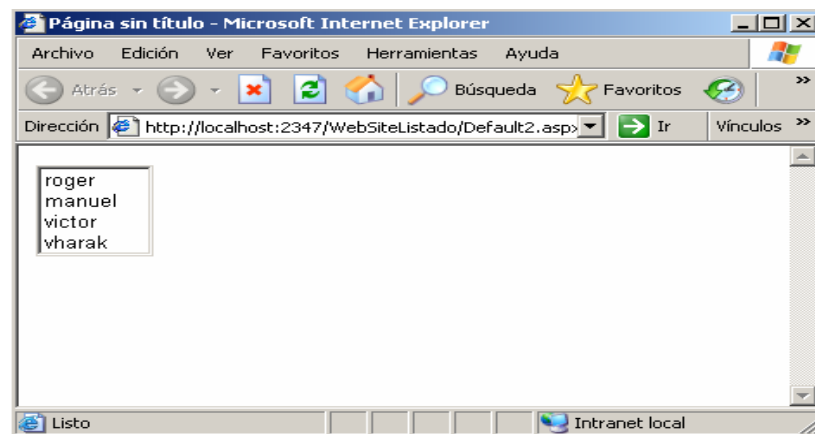
	usuario	password	email	telefono
▶	roger	petrov	none@none.com	5550309
	manuel	rodriguez	manu@cipsa.com	5550493
	victor	krawler	victor@cipsa.com	5554930
	vharak	wolfgang	wolf@cipsa.com	5550492
*	NULL	NULL	NULL	NULL



La propiedad `DataSource` se asocia al objeto `DataTable` que contiene la tabla de datos indicada, producto de la consulta lanzada contra la BD. La columna "usuario" se emplea para el texto de las opciones que se muestran en el `ListBox`, y el valor de la columna "telefono" como los valores asociados a cada uno de los elementos.

Una vez realizada la carga de los datos, y el establecimiento de las propiedades para el vínculo de datos, se invoca al método `DataBind()` de la página para resolverlo.

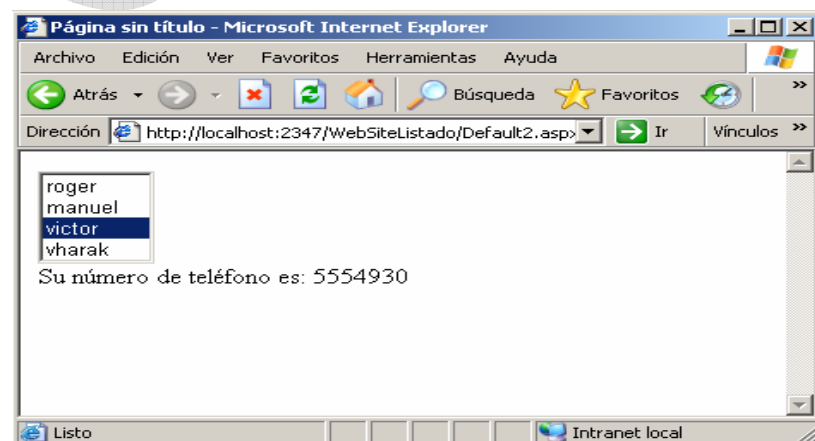
El resultado al solicitar la página sería el siguiente:



Para comprobar el funcionamiento podemos añadir un `Label` a la página, de modo que cuando el usuario seleccione el nombre de un usuario, se muestre a continuación su número de teléfono.

La selección de un elemento en el control `ListBox` provoca el evento `SelectedIndexChanged` del control. Pero para que este evento se ejecute en el servidor justo inmediatamente después de la selección, es preciso que produzca la recarga automática de la página. Para ello debe ajustar la propiedad `AutoPostBack` del control al valor 'true'.

```
protected void ListBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    this.Label1.Text = "Su número de teléfono es: " + this.ListBox1.SelectedValue.ToString();
}
```



### 2.4.- VINCULACION A CONTROLES DE TABLA

Además de los controles de tipo lista, ASP.NET incluye algunos controles que soportan también la vinculación múltiple. Estos controles se diferencian de los expuestos hasta en que están diseñados específicamente para representar datos mediante vinculación a datos.

Estos controles tienen la capacidad de mostrar múltiples campos de varios registros al estilo de una tabla siguiendo el diseño de objetos plantilla (*templates*), que permiten además definir capacidades como paginado, edición, ordenación de los datos mostrados ..etc.

Los controles específicos para datos en ASP.NET son los siguientes:

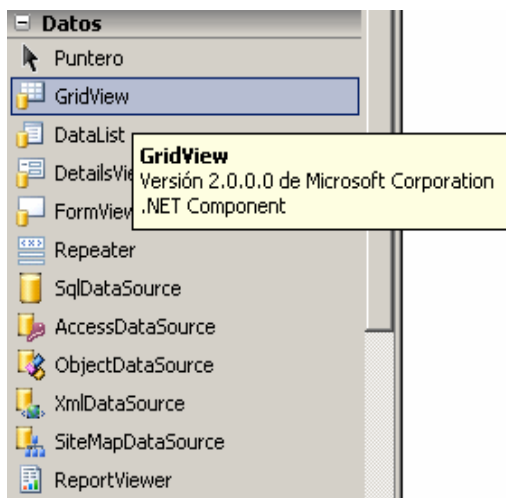
- **GridView:** Es un control multipropósito con un diseño en rejilla que permite mostrar grandes cantidades de datos. Soporta funciones de selección, edición, ordenación y paginamiento de datos. Sustituye al control DataGrid de versiones anteriores.
- **ListView:** Es un control especializado en la visualización de datos basado en plantillas con las mismas funciones que el control GridView. Sustituye al control Repeater de versiones anteriores.
- **DetailsView:** Es un control especializado en la visualización de datos de un único registro en cada momento en una tabla de un único registro por campo.
- **FormView:** Es un control muy semejante al DetailsView que muestra los datos de un único registro en cada momento. Soporta también la edición, paginación y navegación a través de un conjunto de registros. La diferencia con DetailsView es que éste se basa en plantillas que permite mostrar los datos de diferentes formas.

Al igual que los controles de tipo Lista, éstos controles poseen una propiedad *DataSource* y el método *DataBind()* que ejecuta la lectura de todos los registros del origen de datos y su visualización. Sin embargo; no poseen propiedades como *DataTextField* y *DataValueField* dado que estos controles son capaces una columna por cada propiedad ( si se representan objetos ), o por cada campo (si se representan registros de un *DataTable*, *DataRowView*, o *DataReader*).

## 2.5.- EL CONTROL TABLA GRIDVIEW

El control GridView se deriva de la clase BaseDataBoundControl al igual que otros muchos controles Web tales como: AdRotator, DetailsView, FormView, CheckBoxList y RadioButtonList.

El control GridView representa una tabla de datos que puede añadirse a una página Web y vincularse a un origen de datos para mostrar la información de forma ordenada. Permite paginar los datos, ordenarlos, y también mostrar controles asociados a cada fila ( como por ejemplo botones ), para añadirle funciones de selección sobre la información mostrada.



El control GridView es una versión modernizada del control DataList presente en ASP.NET 1.x.

Para empezar a utilizar un control GridView lo seleccionamos bajo la etiqueta "datos" dentro del cuadro de herramientas y lo arrastramos a la ventana de diseño de la página.

Al igual que con los controles de tipo lista, el control GridView puede asociarse a un control Origen de Datos, o a objetos DataSet, DataTable, DataView.

### 2.5.1.- Vínculo a datos por código

Al igual que los controles de tipo Lista, también es posible vincular un control GridView mediante código en tiempo de ejecución. Para ello basta asignar un objeto origen de datos a la propiedad DataSource del control, y llamar al método DataBind() de la página para resolver el vínculo en la primera llamada, o tras modificaciones.

```
protected void Page_Load(object sender, EventArgs e)
{
    SqlConnectionSettings conexion =
    ConfigurationManager.ConnectionStrings["libreriaConnectionString"];
    SqlConnection con = new SqlConnection(conexion.ConnectionString);

    SqlDataAdapter adapter = new SqlDataAdapter("SELECT * FROM CLIENTES", con);
    adapter.Fill(dt);
    if (!Page.IsPostBack)
    {
        this.GridView1.DataSource = dt;
        DataBind();
    }
}
```

El control GridView de forma predeterminada se autoconfigura mostrando tantas columnas como campos devuelva la consulta con sus respectivos nombres. Este comportamiento puede modificarse definiendo que columnas deben mostrarse y el aspecto mediante la edición de plantillas ( *templates* ).

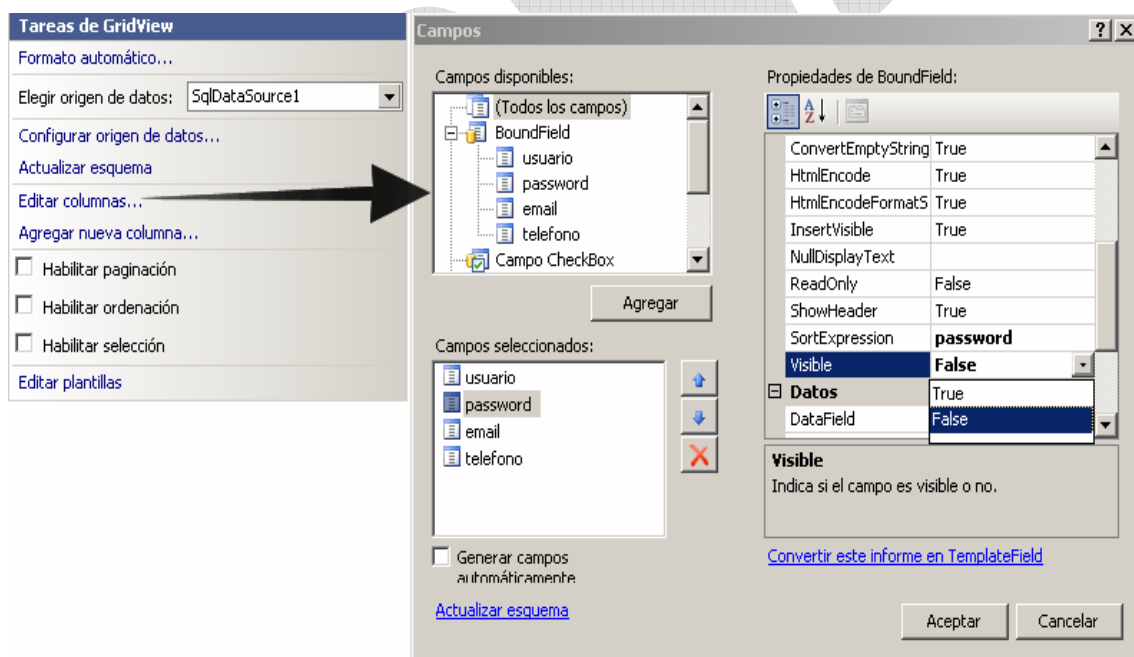
## 2.5.2.- Edición de columnas y plantillas.

Al vincular el control GridView a un objeto origen de datos, puede permitirse la generación automática de las correspondientes columnas fijando la propiedad *AllowGenerateColumns* a True. Esta propiedad es útil cuando se vincula mediante diseño un control GridView a un objeto SqlDataSource.

En el caso de querer vincular el control GridView a un DataSet, una matriz o una colección de datos, es necesario indicar manualmente las columnas del GridView y que datos va a representar cada una. Esto puede llevarse acabo alterando el código del diseño del control, o mediante la opción “Editar Columnas” del cuadro flotante “Tareas del GridView”.

Supóngase que se desea añadir cuatro columnas a un control GridView para mostrar los datos de la consulta a la tabla CLIENTES del ejemplo anterior. Dicha consulta devuelve cuatro columnas: usuario, password, email y teléfono.

Deberíamos añadir cuatro columnas de tipo *BoundField* ( campo vinculado ), indicando en cada una el nombre del campo que deben mostrar cada una en su propiedad *DataField*.






















El código resultante sería:

```
<asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="False"
DataSourceID="SqlDataSource1">
  <Columns>
    <asp:BoundField DataField="usuario" HeaderText="usuario" SortExpression="usuario" />
    <asp:BoundField DataField="password" HeaderText="password" SortExpression="password"/>
    <asp:BoundField DataField="email" HeaderText="email" SortExpression="email" />
    <asp:BoundField DataField="telefono" HeaderText="telefono" SortExpression="telefono"/>
  </Columns>
</asp:GridView>
```

Cada objeto **BoundField** representa una columna del **GridView** vinculada a los valores de un campo de un origen de datos.

Las propiedades de esta clase determinan las características de cada una de las columnas de datos contenidas en el **GridView**:

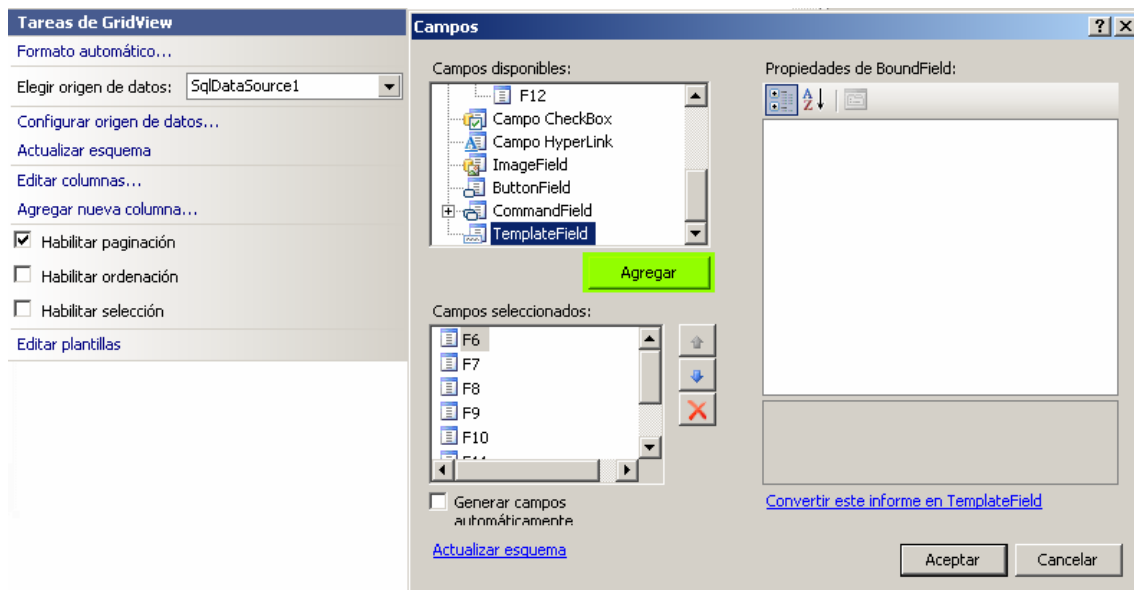
Nombre	Descripción
 <a href="#">AccessibleHeaderText</a>	Obtiene o establece el texto que se representa como el valor de la propiedad <b>AbbreviatedText</b> en algunos controles. (Se hereda de <a href="#">DataControlField</a> ).
 <a href="#">ApplyFormatInEditMode</a>	Obtiene o establece un valor que indica si la cadena de formato especificada por la propiedad <a href="#">DataFormatString</a> se aplica a los valores de campo cuando el control enlazado a datos que contiene el objeto <a href="#">BoundField</a> está en modo de edición.
 <a href="#">ControlStyle</a>	Obtiene o establece el estilo de cualquier control de servidor Web contenido en el objeto <a href="#">DataControlField</a> . (Se hereda de <a href="#">DataControlField</a> ).
 <a href="#">ConvertEmptyStringToNull</a>	Obtiene o establece un valor que indica si los valores de cadena vacía ("") se convierten automáticamente en valores nulos cuando se actualiza el campo de datos en el origen de datos.
 <a href="#">DataField</a>	Obtiene o establece el nombre del campo de datos que se enlaza al objeto <b>BoundField</b> .
 <a href="#">DataFormatString</a>	Obtiene o establece la cadena que especifica el formato de presentación del valor del campo.
 <a href="#">FooterStyle</a>	Obtiene o establece el estilo del pie de página del campo del control de datos. (Se hereda de <a href="#">DataControlField</a> ).
 <a href="#">FooterText</a>	Obtiene o establece el texto que se muestra en el elemento de pie de página de un campo de control de datos. (Se hereda de <a href="#">DataControlField</a> ).
 <a href="#">HeaderImageUrl</a>	Obtiene o establece la dirección URL de una imagen que se muestra en el elemento de encabezado de un campo de control de datos. (Se hereda de <a href="#">DataControlField</a> ).
 <a href="#">HeaderStyle</a>	Obtiene o establece el estilo del encabezado del campo del control de datos. (Se hereda de <a href="#">DataControlField</a> ).
 <a href="#">HeaderText</a>	Obtiene o establece el texto que se muestra en el encabezado de un control de datos.
 <a href="#">HtmlEncode</a>	Obtiene o establece un valor que indica si los valores de campo se codifican en HTML antes de mostrarlos en un objeto <b>BoundField</b> .
 <a href="#">InsertVisible</a>	Obtiene un valor que indica si el objeto <b>DataControlField</b> está visible cuando su control enlazado a datos primario está en modo de inserción. (Se hereda de <a href="#">DataControlField</a> ).
 <a href="#">ItemStyle</a>	Obtiene el estilo de cualquier contenido basado en texto mostrado por un campo de control de datos. (Se hereda de <a href="#">DataControlField</a> ).
 <a href="#">NullDisplayText</a>	Obtiene o establece el título mostrado para un campo cuando el valor del campo es nulo.
 <a href="#">ReadOnly</a>	Obtiene o establece un valor que indica si el valor del campo se puede modificar en modo de edición.
 <a href="#">ShowHeader</a>	Obtiene o establece un valor que indica si se representa el elemento de encabezado de un campo de control de datos. (Se hereda de <a href="#">DataControlField</a> ).
 <a href="#">SortExpression</a>	Obtiene o establece una expresión de ordenación utilizada por un control de origen de datos para ordenar los datos. (Se hereda de <a href="#">DataControlField</a> ).
 <a href="#">Visible</a>	Obtiene o establece un valor que indica si se representa un campo de control de datos. (Se hereda de <a href="#">DataControlField</a> ).

Entre las propiedades más destacables pueden encontrarse *HeaderText* que determina el nombre de la columna en la cabecera de la tabla.

## ACCESO A DATOS CON ADO.NET

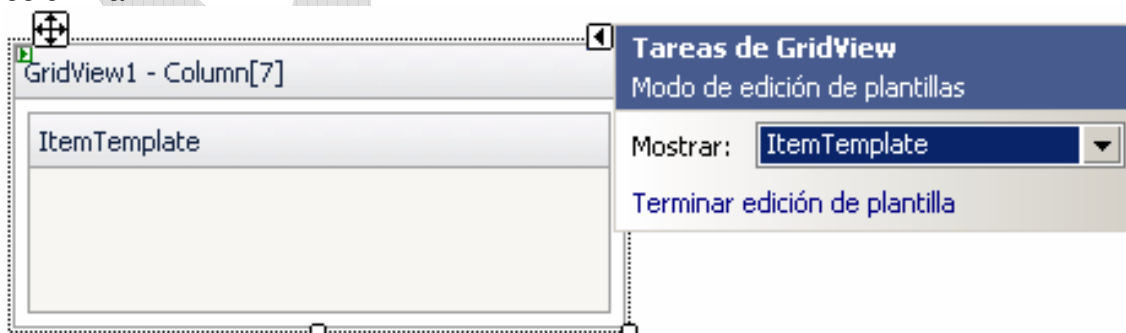
Las columnas de un control GridView también puede venir representadas por otros agrupaciones de diferentes de controles, tales como botones, cajas de texto, casillas de verificación... etc. Para ello hay que crear primero una nueva columna de tipo *TemplateField*. ( o columna plantilla ).

Para crear este tipo de columna debemos seleccionar la opción “Editar Columnas” en la etiqueta inteligente “Tareas de GridView”. A continuación, en la ventana de edición de columnas, seleccionamos en la lista de campos el control “TemplateField” y pulsamos el botón agregar.



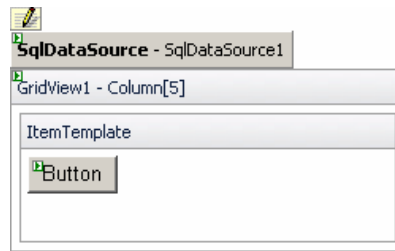
Una vez añadida la nueva columna de tipo TemplateField, debemos indicar el o los controles Web que deseamos se representen en ella. Para ello debemos seleccionar la opción “Editar Plantillas” del cuadro flotante del GridView.

En ese momento aparecerá un pequeño recuadro que nos mostrará el contenido del TemplateField. Se trata de un espacio de edición donde podemos colocar los controles que deseamos aparezcan por cada registro en esa columna.



En la imagen se muestra el diseño del objeto *ItemTemplate* correspondiente a la séptima columna del control GridView1. El objeto ItemTemplate representa el diseño de la columna al mostrar datos. Ahora es posible seleccionar los controles que deseamos en el cuadro de herramientas y arrastrarlos al interior de la superficie de edición del *ItemTemplate* para que aparezcan como parte de la columna.

Si añadimos por ejemplo un campo de texto y un botón, obtendríamos el siguiente resultado:



Estos controles pueden generar eventos y vincular el valor de sus propiedades a campos de datos. Una práctica común es emplear una columna plantilla con un botón que provoque un evento indicando la clave primaria del registro correspondiente al ser pulsado.

Una vez añadido el botón al ItemTemplate cerramos la edición seleccionando la opción "Terminar edición de plantilla" en la etiqueta inteligente. El control GridView queda entonces así:

```
<asp:GridView ID="GridView1" runat="server" DataSourceID="SqlDataSource1">
  <PagerSettings Mode="NumericFirstLast" />
  <Columns>
    <asp:BoundField DataField="usuario" HeaderText="usuario" SortExpression="usuario" />
    <asp:BoundField DataField="password" HeaderText="password" SortExpression="password" />
    <asp:BoundField DataField="email" HeaderText="email" SortExpression="email" />
    <asp:BoundField DataField="telefono" HeaderText="telefono" SortExpression="telefono" />
    <asp:BoundField DataField="id_usuario" HeaderText="id_usuario" ReadOnly="True"
SortExpression="id_usuario" />
    <asp:TemplateField>
      <ItemTemplate>
        <asp:Button ID="Button2" runat="server" Text="Selección"
CommandArgument="{%# Eval('id_usuario')}" oncommand="Button1_Command1" />
      </ItemTemplate>
    </asp:TemplateField>
  </Columns>
</asp:GridView>
```

Cada columna de tipo TemplateField, posee cinco elementos distintos:

- **ItemTemplate:** Plantilla para filas de datos.
- **AlternatingItemTemplate:** Plantilla para filas alternas de datos.
- **EditItemTemplate:** Plantilla para filas en modo de edición.
- **HeaderTemplate:** Plantilla para filas de cabecera de tabla.
- **FooterTemplate:** Plantilla para filas de pie de tabla.

La activación del modo de edición para una fila se lleva acabo añadiendo columnas de tipo CommandField:





## 2.5.3.- Paginación de los datos

El control GridView permite la paginación de los datos. La paginación consiste en mostrar la información fragmentada en páginas dependiendo del número total de registros en el origen de datos, y el número de registros visualizados por página.

Para habilitar la paginación se emplea la propiedad *AllowPaging*. También se puede habilitar marcando la casilla de verificación “Habilitar Paginación” en la etiqueta inteligente del control. El número de filas o registros a mostrar por página se puede ajustar empleando la propiedad *PageSize*.

F8	F6	F7	F9	F12	F11	F10
18	5	16	29	2	44	43
29	10	14	33	30	48	46
14	2	9	19	49	43	34
34	19	22	36	38	43	37
35	6	15	44	38	48	46
20	9	17	34	11	47	41
23	4	14	27	12	40	30
7	2	5	43	11	49	48
30	5	28	34	15	44	39
38	16	18	43	39	47	44
1	2	3	4	5	6	7
8	9	10	...			

La propiedad *PageIndex* permite obtener y establecer la página de datos mostrada en un momento dado:

La propiedad *PagerSettings* permite personalizar el interfaz de paginación asociado al control cuando ésta es habilitada.

Esta propiedad hace referencia a una instancia de la clase *PagerSettings*, que posee la propiedad *Mode* para definir cuatro tipos de interfaz de paginación:

Paginación	
AllowPaging	True
PageIndex	0
PagerSettings	
FirstPageImageUrl	
FirstPageText	&lt;&lt;
LastPageImageUrl	
LastPageText	&gt;&gt;
Mode	Numeric
NextPageImageUrl	NextPrevious
NextPageText	Numeric
PageButtonCou	NextPreviousFirstLast
Position	NumericFirstLast
PreviousPageImage	
PreviousPageText	&lt;
Visible	True
PageSize	10

- **NextPrevious:** Muestra un par de enlaces para navegar a la página anterior y a la siguiente.
- **Numeric:** Muestra únicamente enlaces numéricos para navegar a las páginas existentes.
- **NextPreviousFirstLast:** Muestra cuatro enlaces para navegar a la primera página, la anterior, la siguiente, y la última.
- **NumericFirstLast:** Muestra enlaces numéricos acompañados de dos más para navegar a la primera y última página.

Los diferentes enlaces pueden también personalizarse indicando un texto o la URL de una imagen a ser mostrada en su lugar. Para ello se emplean las propiedades *FirstPageText/ImageUri*, *PreviousPageText/ImageUri*, *NextPageText/ImageUri*, *LastPageText/ImageUri*.

El control GridView desencadena dos eventos cuando cambia la página de datos. El evento *PageIndexChanging* se produce antes de cambiar la página, y el evento *PageIndexChanged*, se produce justo después. Ambos eventos pueden aprovecharse para implementar acciones que deben llevarse acabo al cambiar de página de datos.



## 2.5.4.- Ordenación de datos

La ordenación permite seleccionar un campo para que los datos se muestren ordenados en base a sus valores.

Para habilitar la ordenación puede emplearse la propiedad `AllowSorting`, o también marcarse la casilla de verificación “Habilitar ordenación” de la etiqueta inteligente del control `GridView`.

<a href="#">usuario</a>	<a href="#">password</a>	<a href="#">email</a>	<a href="#">telefono</a>	<a href="#">id usuario</a>
roger	petrov	none@none.com	5550309	1
manuel	rodriguez	manu@cipsa.com	5550493	2
victor	krawler	victor@cipsa.com	5554930	3
vharak	wolfgang	wolf@cipsa.com	5550492	4

Al habilitarse la ordenación, los títulos de los campos en la tabla se sustituyen por hipervínculos, que al ser seleccionados; provocan la recarga de la página mostrando los datos ordenados por los valores de la columna correspondiente.

## 2.5.5.- Columna de selección de datos

La columna de selección es una columna funcional que no muestra realmente información, si no que provee una funcionalidad ligada al registro mostrado en la fila a la que pertenece. En el este caso, la funcionalidad es permitir al usuario seleccionar un registro para una labor.

Esta columna se compone de hipervínculos, y su generación se controla mediante la propiedad `AutoGenerateSelectButton`, o también marcando la casilla de verificación “Habilitar selección” de la etiqueta inteligente.

	<a href="#">usuario</a>	<a href="#">password</a>	<a href="#">email</a>	<a href="#">telefono</a>	<a href="#">id usuario</a>
<a href="#">Seleccionar</a>	roger	petrov	none@none.com	5550309	1
<a href="#">Seleccionar</a>	manuel	rodriguez	manu@cipsa.com	5550493	2
<a href="#">Seleccionar</a>	victor	krawler	victor@cipsa.com	5554930	3
<a href="#">Seleccionar</a>	vharak	wolfgang	wolf@cipsa.com	5550492	4

Si deseamos que aparezcan botones, o imágenes en vez de enlaces como controles en la columna de selección debemos modificar las propiedades de la columna. Para ello debemos seleccionar la opción “editar columnas” en la etiqueta inteligente del control.

Cuando se pulsa uno de los enlaces se produce la recarga de la página y se ejecuta el evento `SelectedIndexChanged`. Entonces se pueden emplear múltiples propiedades para obtener la posición y valores del registro que se ha seleccionado.

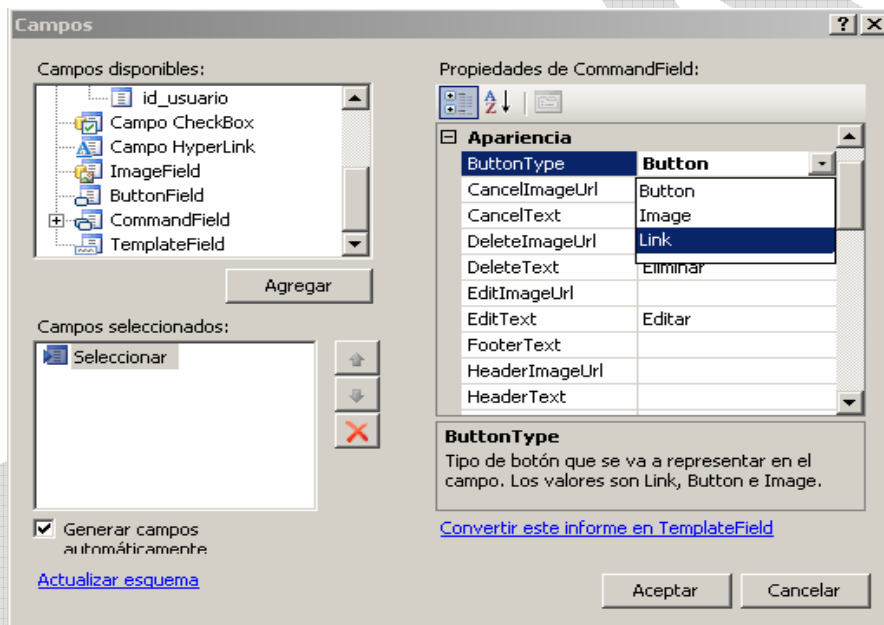
## ACCESO A DATOS CON ADO.NET

Para acceder a los datos inscritos en las celdas de un determinado registro del control GridView debe emplearse los métodos Rows y Cells:

```
protected void GridView1_SelectedIndexChanged(object sender, EventArgs e)
{
    int index = this.GridView1.SelectedIndex;
    this.LabelNombre.Text = this.GridView1.Rows[index].Cells[1].Text;
    this.LabelPassword.Text = this.GridView1.Rows[index].Cells[2].Text;
    this.LabelMail.Text = this.GridView1.Rows[index].Cells[3].Text;
}
```

En el código anterior se muestran en tres etiquetas los valores de los campos nombre, password y email seleccionados en el GridView. Para ello se accede al registro seleccionado por el usuario indicado por la propiedad SelectedIndex, y después a la propiedad Text de sus celdas 1, 2 y 3.

La columna de selección puede configurarse para mostrar un texto diferente, o una imagen. También se puede hacer que en vez de mostrar un enlace por fila, muestre un botón. Para ello debe seleccionarse la opción “Editar columnas” en la etiqueta inteligente del control GridView.



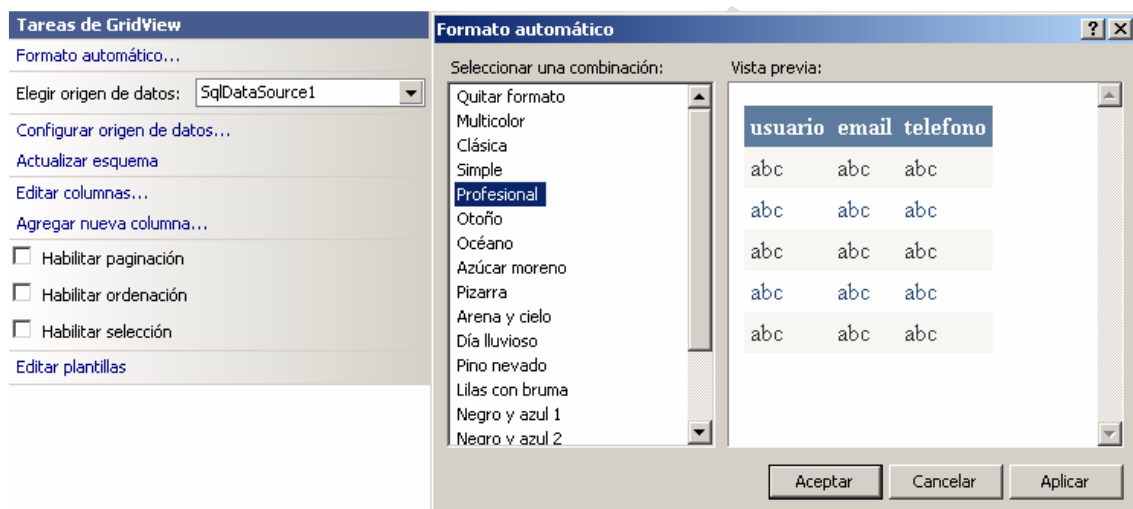
Dentro de la lista “Campos seleccionados”, podemos marcar la columna “Seleccionar” y acceder a todas sus propiedades de configuración. La propiedad ButtonType permite escoger entre mostrar un enlace ( la opción por defecto ), una imagen, o un botón.

Si deseamos alterar el texto para que ponga “Ver detalles” en vez de “Seleccionar” podemos hacerlo cambiando la cadena de texto de la propiedad SelectText.

### 2.5.6.- Formatos de representación de datos

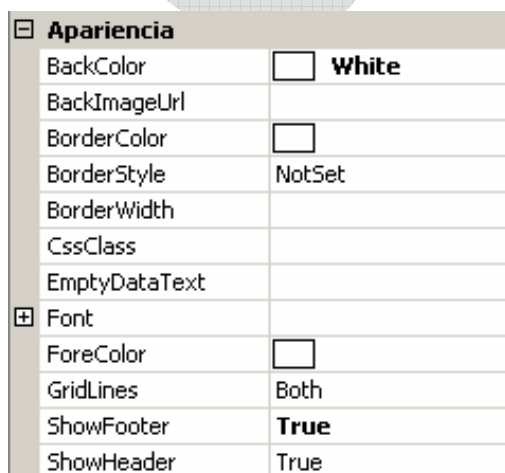
El control GridView puede ser personalizado para que muestre la información de forma más vistosa o clara mediante la modificación de colores, tipos de letras, líneas de separación... etc. Esto recibe el nombre de *formato*.

La definición del formato de un GridView puede hacerse modificando sus propiedades una a una ( que son bastantes ), o seleccionando un formato prediseñado de los que acompañan el control. Estos formatos pueden seleccionarse a través de la opción “Formato Automático” del cuadro flotante “Tareas del GridView”:



Como puede observarse; el formato determina cómo se muestran los datos en el GridView, no qué datos son mostrados.

Si deseamos crear nuestro propio formato o personalizar uno ya existente, existen una serie de propiedades sobre las que podemos actuar. Estas propiedades se encuentran dentro de la ventana de propiedades del propio control GridView bajo las etiquetas: *Apariencia*, *Diseño* y *Estilos*.



En el apartado apariencia se muestran las propiedades principales relativas al color y tipo de fuentes de todas las filas y columnas que aparecen en el control GridView, así como de las línea divisorias.

Es de destacar la propiedad *ShowHeader* que permiten que la tabla muestre una cabecera con el nombre de las columnas de datos.

## ACCESO A DATOS CON ADO.NET

Diseño	
CellPadding	-1
CellSpacing	0
Height	
HorizontalAlign	NotSet
Width	

En el apartado Diseño se muestra propiedades relativas al espaciado entre las celdas y los bordes ( *CellPadding* ), y entre las propiedades celdas ( *CellSpacing* ), así como las dimensiones deseadas del control.

Estilos	
AlternatingRowStyle	
EditRowStyle	
EmptyDataRowStyle	
FooterStyle	
HeaderStyle	
PagerStyle	
RowStyle	
SelectedRowStyle	

En el apartado Estilos es donde se concentran la mayoría de las propiedades que determinan el formato de un GridView.

Cada propiedad hace referencia a un tipo de fila en el GridView. El formato de cada fila viene determinado por una instancia de la clase *TableItemStyle*.

- **AlternatingRowStyle:** Determina el estilo para las filas alternativas de datos.
- **EditRowStyle:** Determina el estilo para las filas de datos en modo edición.
- **EmptyRowStyle:** Determina el estilo para una fila de datos que está vacía.
- **FooterStyle:** Determina el estilo para la fila pie del GridView.
- **HeaderStyle:** Determina el estilo para la fila cabecera del GridView.
- **PagerStyle:** Determina el estilo para la fila con el control de paginación.
- **RowStyle:** Determina el estilo para las filas de datos normales.
- **SelectedRowStyle:** Determina el estilo para la fila de datos que es seleccionada.

Como se ha mencionado, el formato de cada tipo de fila viene determinado por una instancia de la clase *System.Web.UI.WebControls.TableItemStyle*:

También es posible modificar el estilo mediante código en tiempo de ejecución. Esto puede resultar útil si deseamos que una determinada celda posea un color de fondo distinto si tiene un valor determinado. Para ello, debe capturarse el evento *RowDataBound*.

Supóngase que deseamos modificar en tiempo de ejecución el diseño del GridView, de modo que se tiña de verde el fondo de aquella fila cuyo nombre de usuario sea “roger”. El siguiente código lo haría:

```
protected void GridView1_RowDataBound(object sender, GridViewRowEventArgs e)
{
    if (e.Row.RowType == DataControlRowType.DataRow)
    {
        if (e.Row.Cells[1].Text.ToUpper() == "ROGER")
        {
            e.Row.BackColor = System.Drawing.Color.YellowGreen;
        }
    }
}
```

El código anterior se ejecuta por cada fila del GridView al resolverse el vínculo a datos correspondiente. Si la fila es de datos ( no es la cabecera, ni el pie de la tabla ), se comprueba si posee el valor “ROGER” en la celda 1. Si es así, se altera el color de fondo del estilo de fila mediante la propiedad *BackColor* del objeto *Row* que la representa.

A continuación se muestra la lista de propiedades de la clase `TableItemStyle`, que permiten definir el formato de cada tipo de fila.

Estas propiedades pueden configurarse en diseño empleando la ventana de edición de columnas, o también por código en tiempo de ejecución.

Nombre	Descripción
 <code>BackColor</code>	Obtiene o establece el color de fondo del control de servidor Web. (Se hereda de <a href="#">Style</a> ).
 <code>BorderColor</code>	Obtiene o establece el color del borde del control de servidor Web. (Se hereda de <a href="#">Style</a> ).
 <code>BorderStyle</code>	Obtiene o establece el estilo del borde del control de servidor Web. (Se hereda de <a href="#">Style</a> ).
 <code>BorderWidth</code>	Obtiene o establece el ancho del borde del control de servidor Web. (Se hereda de <a href="#">Style</a> ).
 <code>CanRaiseEvents</code>	Obtiene un valor que indica si el componente puede provocar un evento. (Se hereda de <a href="#">Component</a> ).
 <code>Container</code>	Obtiene <a href="#">IContainer</a> que contiene <a href="#">Component</a> . (Se hereda de <a href="#">Component</a> ).
 <code>CssClass</code>	Obtiene o establece la clase de hoja de estilo en cascada (CSS) representada por el control de servidor Web en el cliente. (Se hereda de <a href="#">Style</a> ).
 <code>DesignMode</code>	Obtiene un valor que indica si <a href="#">Component</a> está actualmente en modo de diseño. (Se hereda de <a href="#">Component</a> ).
 <code>Events</code>	Obtiene la lista de controladores de eventos asociados a <a href="#">Component</a> . (Se hereda de <a href="#">Component</a> ).
 <code>Font</code>	Obtiene las propiedades de fuente asociadas al control de servidor Web. (Se hereda de <a href="#">Style</a> ).
 <code>ForeColor</code>	Obtiene o establece el color de primer plano (normalmente el color del texto) del control de servidor Web. (Se hereda de <a href="#">Style</a> ).
 <code>Height</code>	Obtiene o establece el alto del control de servidor Web. (Se hereda de <a href="#">Style</a> ).
 <code>HorizontalAlign</code>	Obtiene o establece la alineación horizontal del contenido de una celda.
 <code>IsEmpty</code>	Infraestructura. Propiedad protegida. Obtiene un valor que indica si se han definido elementos de estilo en la bolsa de estados. (Se hereda de <a href="#">Style</a> ).
 <code>IsTrackingViewState</code>	Devuelve un valor que indica si se han definido elementos de estilo en la bolsa de estados. (Se hereda de <a href="#">Style</a> ).
 <code>RegisteredCssClass</code>	Obtiene la clase de hoja de estilo en cascada (CSS) que se registra con el control. (Se hereda de <a href="#">Style</a> ).
 <code>Site</code>	Obtiene o establece <a href="#">ISite</a> de <a href="#">Component</a> . (Se hereda de <a href="#">Component</a> ).
 <code>VerticalAlign</code>	Obtiene o establece la alineación vertical del contenido de una celda.
 <code>ViewState</code>	Infraestructura. Obtiene la bolsa de estados que contiene los elementos de estilo. (Se hereda de <a href="#">Style</a> ).
 <code>Width</code>	Obtiene o establece el ancho del control de servidor Web. (Se hereda de <a href="#">Style</a> ).
 <code>Wrap</code>	Obtiene o establece un valor que indica si el contenido de una celda se ajusta dentro de la misma.

### 2.6.- EL CONTROL TABLA LISTVIEW

El control ListView es un control Web de nueva aparición en el .NET Framework 3.5. Está diseñado para reemplazar al control Repeater anterior.

El Control ListView es un control específico para vinculación a datos cuyo diseño se basa en el uso de plantillas personalizadas. Estas plantillas permiten añadir capacidades como la ordenación, filtrado, selección y edición de los datos vinculados al control de igual manera que el control DataGrid.

El Control ListView admite los siguientes tipos de plantillas para representar los datos:

- **ItemTemplate:** Determina el diseño para la visualización de los datos de cada registro excepto en el caso de que se use la plantilla AlternatingItemTemplate.
- **AlternatingItemTemplate:** Determina el diseño para la visualización de los datos de los registros en posiciones pares en conjunto con la ItemTemplate. Esto permite una variación de estilos entre registros.
- **ItemSeparatorTemplate:** Determina el diseño del separador que se muestra entre la visualización de los datos de cada registro.
- **SelectedItemTemplate:** Determina el diseño del registro que se visualiza como seleccionado. Se puede emplear para mostrar información adicional de los registros al ser seleccionados.
- **EditItemTemplate:** Determina los controles usados al editar un registro.
- **InsertItemTemplate:** Determina los controles usados al insertar un nuevo registro.
- **LayoutTemplate:** Determina el diseño que contiene la lista de registros.
- **GroupTemplate:** Determina el diseño que contiene los distintos grupos de registros si se está utilizando la característica de agrupamiento.
- **EmptyItemTemplate:** Determina el diseño para rellenar huecos vacíos en el diseño al mostrar agrupamientos. Si por ejemplo se fija un agrupamiento de 5 registros, y la consulta retorna 13 registros; deben mostrarse dos huecos vacíos en el diseño del último grupo de registros.
- **EmptyTemplate:** Fija el diseño usado cuando los datos vinculados están vacíos y no poseen ningún registro.

Para emplar un ListView deben crearse primero las plantillas que organizan el diseño de los datos a visualizar. En el más simple de los casos es necesario definir un ItemTemplate, que represente el diseño de cada registro a mostrar, y un LayoutTemplate que determine el diseño del resto del control.

Cuando el control ListView se convierte en HTML, recorre todos los registros vinculados y los visualiza siguiendo el diseño establecido en el objeto ItemTemplate asignado. Finalmente, sitúa todo el contenido generado en el interior del diseño del LayoutTemplate asignado.

Cuando se diseña un LayoutTemplate es necesario indicar donde se van a colocar los contenidos de cada registro. Para ello se debe añadir un espacio que se duplicará por cada registro vinculado que deba mostrarse. Este espacio se define como un control de servidor (*runat="server"*) cuyo atributo ID debe ser "itemPlaceholder".

Ejemplo:

```
<form id="form1" runat="server">
  <div>
    <asp:SqlDataSource ID="SqlDataSource1" runat="server"
      ConnectionString="<%"$ ConnectionStrings:CIPSAConnectionString %>"
      SelectCommand="SELECT [cod_empl], [nom_empl], [ape_empl], [cod_dept] FROM
[Empleados]"></asp:SqlDataSource>
  </div>
  <asp:ListView ID="ListView1" runat="server" DataSourceID="SqlDataSource1">
    <LayoutTemplate>
      <table border="1"><span id="itemPlaceholder" runat="server"></span></table>
    </LayoutTemplate>
    <ItemTemplate>
      <tr><td><%"#Eval("cod_empl")%></td><td><%"#Eval("nom_empl")%></td></tr>
    </ItemTemplate>
  </asp:ListView>
</form>
```

El código HTML que generará el control será el siguiente:

```
<table border="1">
<tr><td>001</td><td>Emiliano</td></tr>
<tr><td>002</td><td>María Victoria</td></tr>
<tr><td>003</td><td>Eugenia</td></tr>
<tr><td>004</td><td>José</td></tr>
<tr><td>005</td><td>Raúl</td></tr>
<tr><td>006</td><td>Juan</td></tr>
<tr><td>007</td><td>Eva</td></tr>
<tr><td>008</td><td>Luisa</td></tr>
<tr><td>009</td><td>Pelayo</td></tr>
<tr><td>010</td><td>Santos</td></tr>
</table>
```

## 2.6.1.- Agrupamientos

El agrupamiento es una capacidad que permite mostrar los registros agrupados en grupos de un número determinado de registros. El número de registros por grupo es determinado por el valor de la propiedad *GroupItemCount*.

Para determinar el diseño por grupo y por elemento deben emplearse una nueva plantilla: *GroupTemplate* que contendrá el espacio de visualización de cada registro dentro del grupo determinado por un control de servidor con su atributo ID = "itemPlaceholder". A su vez, la plantilla *LayoutTemplate* pasa a contener el espacio de visualización de cada grupo ( y no de los registros ) determinado por un control de servidor con su atributo ID = "groupPlaceholder".

```
<asp:ListView ID="ListView1" runat="server" DataSourceID="SqlDataSource1"
GroupItemCount="3">
  <GroupTemplate><tr><td>GRUPO</td>
    <span id="itemPlaceholder" runat="server"></span></tr>
  </GroupTemplate>
  <LayoutTemplate>
    <table border="1"><span id="groupPlaceholder" runat="server"></span></table>
  </LayoutTemplate>
  <ItemTemplate>
    <td><%"#Eval("cod_empl")%></td><td><%"#Eval("nom_empl")%></td>
  </ItemTemplate>
</asp:ListView>
</form>
```

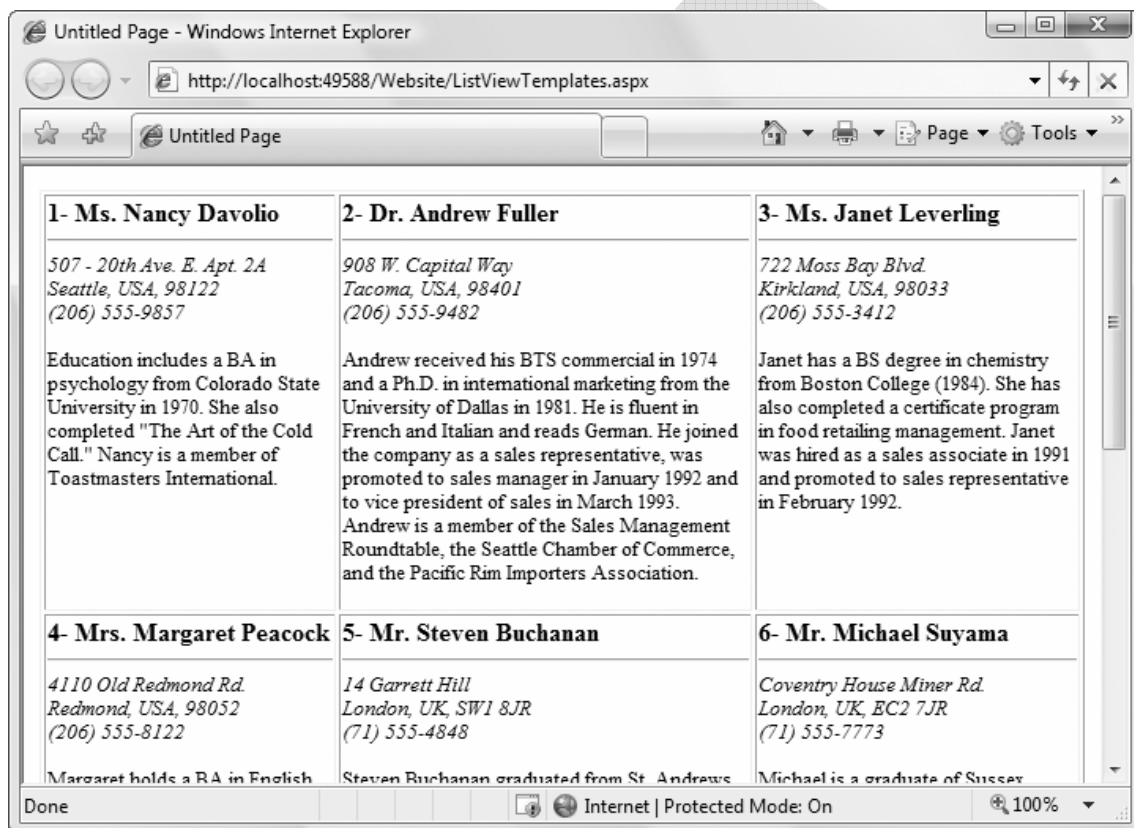


## ACCESO A DATOS CON ADO.NET

El código anterior generaría la siguiente representación en HTML:

```
<table border="1"><tr><td>GRUPO</td>
<td>001</td><td>Emiliano</td>
<td>002</td><td>María Victoria</td>
<td>003</td><td>Eugenia</td>
</tr><tr><td>GRUPO</td>
<td>004</td><td>José</td>
<td>005</td><td>Raúl</td>
<td>006</td><td>Juan</td>
</tr><tr><td>GRUPO</td>
<td>007</td><td>Eva</td>
<td>008</td><td>Luisa</td>
<td>009</td><td>Pelayo</td>
</tr><tr><td>GRUPO</td>
<td>010</td><td>Santos</td>
</tr></table>
```

Mediante el agrupamiento se pueden conseguir diseños de datos como el siguiente:



### 2.6.2.- Paginación

A diferencia de otros controles como el GridView, el control ListView no soporta directamente la paginación de datos. No obstante; se puede emplear un control DataPager para ello.

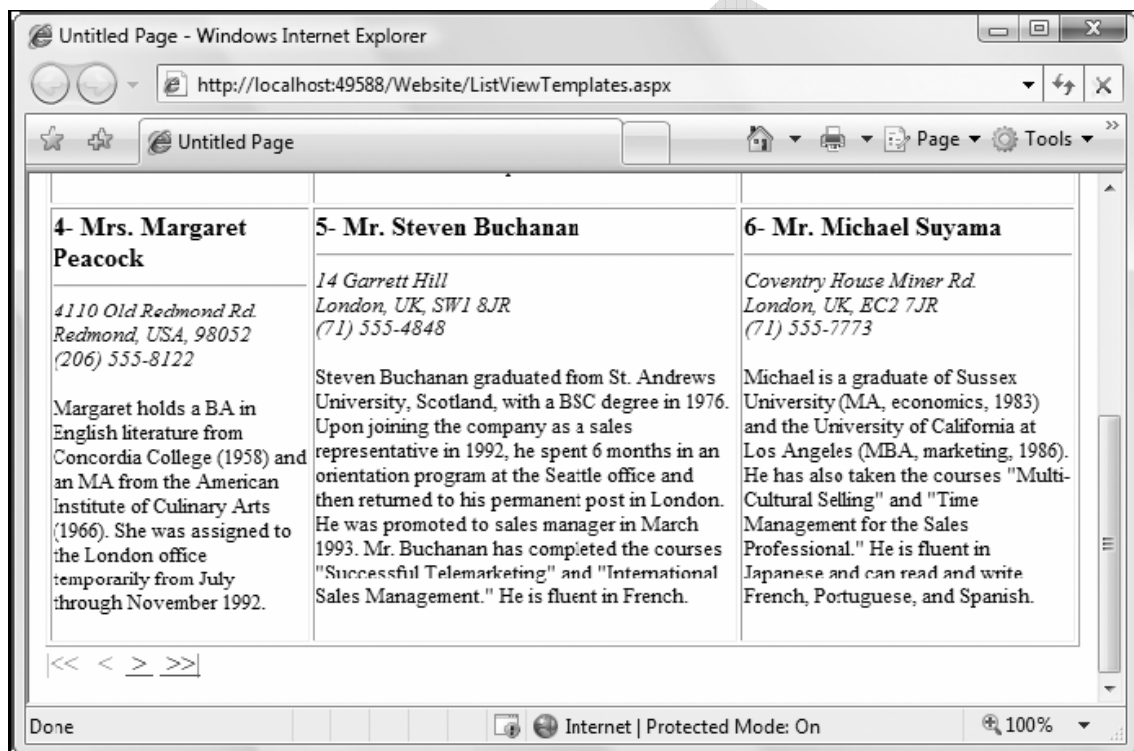
El control DataPager permite definir un mecanismo único para emplear la paginación con multitud de controles aunque por el momento sólo es operativo con ListView.



El uso más típico del control DataPager es colocarlo en la parte inferior del ListView mostrando los típicos enlaces para ir a la primera, la última, la siguiente y la anterior página de datos.

```
<LayoutTemplate>
  <table id="groupPlaceholder" runat="server">
  </table>
  <asp:DataPager runat="server" ID="ContactsDataPager" PageSize="6">
    <Fields>
      <asp:NextPreviousPagerField
        ShowFirstPageButton="true" ShowLastPageButton="true"
        FirstPageText="|<<<" LastPageText=">>>|"
        NextPageText=">" PreviousPageText="<" />
    </Fields>
  </asp:DataPager>
</LayoutTemplate>
```

El resultado del uso del control sería



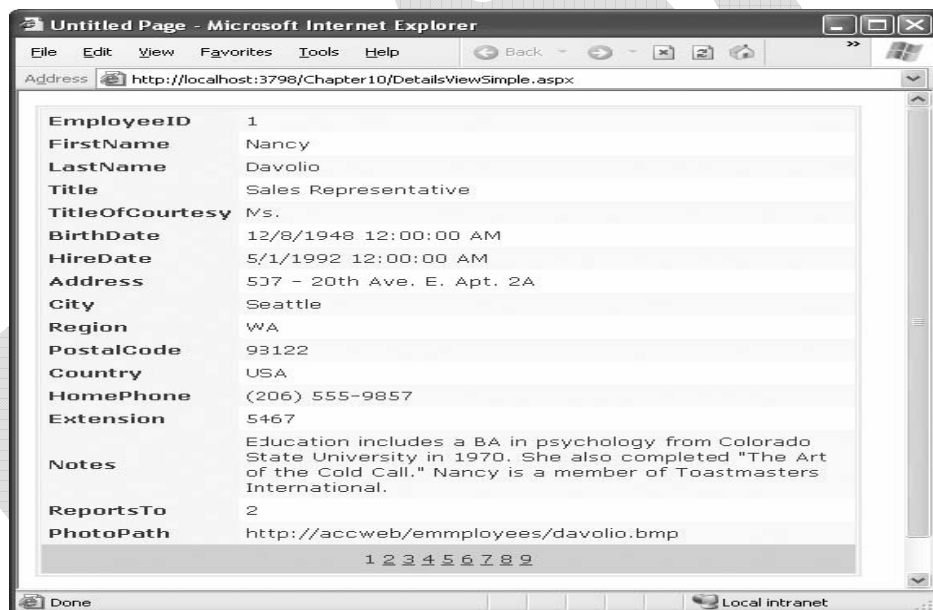
### 2.7.-CONTROLES VINCULADOS A DATOS CON DETALLES

Los controles como el *GridView* y el *ListView* permiten mostrar grandes cantidades de datos estructurados en forma de tablas. No obstante; en muchas ocasiones puede ser necesario mostrar los detalles de un determinado registro, en vez de datos generales de un conjunto de ellos.

ASP.NET incluye dos controles específicos para este propósito; *DetailsView* y *FormView*. Ambos son capaces de mostrar detalles de un único registro en cada momento permitiendo incluir navegar por los registros de uno en uno. Ambos controles soportan el uso de plantillas de diseño.

La principal diferencia entre ambos controles es que *DetailsView* muestra sus datos en el interior de una tabla pudiendo o no emplear plantillas de diseño, mientras que el control *FormView* requiere el uso de plantillas (*Templates*).

El control *DetailsView* está diseñado para mostrar los campos de un determinado registro de datos en cada momento. Cada campo se muestra en una fila de una tabla.



Un *DetailsView* puede vincularse a un origen de datos, a una colección de elementos, o a un control *DataSource*. En ese caso se mostrará el primer registro, y se mostrarán los controles de paginación si la propiedad *AllowPaging* tiene asignado el valor *True*. El diseño de estos controles puede configurarse mediante la propiedad *PagingStyle* y *PagingSettings*.

Debe tenerse presente que a pesar de la paginación; siempre se recuperan la totalidad de los registros del origen de datos con cada recarga de la página, aunque sólo se muestre uno de ellos. Esto puede reducir el rendimiento si empleamos el control como un navegador de registros.

Lo más recomendable es emplear este control en conjunto con un control de tipo Lista, o GridView que permita seleccionar un determinado registro. Una vez seleccionado un registro, se recupera mediante otro control DataSource vinculado a un DetailsView encargado de mostrar los detalles del registro seleccionado.

## 2.7.1.- Definición de campos

Los campos que muestra un DetailsView pueden generarse en base a los campos presentes en el origen de datos automáticamente, o bien ser declarados explícitamente asignando el valor False a la propiedad *AutoGenerateRows*.

Cada campo se declara mediante un control *BoundField*. La propiedad *DataField* indica el nombre de la columna cuyo dato va a ser mostrado en el campo. La propiedad *HeaderText* indica el texto identificativo que el campo mostrará.

```
<asp:DetailsView ID="DetailsView1" runat="server" DataSourceID="sourceEmployees"
AutoGenerateRows="False">
  <Fields>
    <asp:BoundField DataField="EmployeeID" HeaderText="EmployeeID" />
    <asp:BoundField DataField="FirstName" HeaderText="FirstName" />
    <asp:BoundField DataField="LastName" HeaderText="LastName" />
    <asp:BoundField DataField="Title" HeaderText="Title" />
    <asp:BoundField DataField="TitleOfCourtesy" HeaderText="TitleOfCourtesy"
    />
    <asp:BoundField DataField="BirthDate" HeaderText="BirthDate" />
  </Fields>
</asp:DetailsView>
```

A parte de las propiedades indicadas, se pueden emplear otras muchas propiedades con los controles BoundField para modificar su estilo y la forma en la que representa los datos vinculados.

```
<asp:BoundField DataField="EmployeeID" HeaderText="ID">
  <ItemStyle Font-Bold="True" BorderWidth="1" />
</asp:BoundField>
<asp:BoundField DataField="FirstName" HeaderText="First Name" />
<asp:BoundField DataField="LastName" HeaderText="Last Name" />
<asp:BoundField DataField="City" HeaderText="City">
  <ItemStyle BackColor="LightSteelBlue" />
</asp:BoundField>
<asp:BoundField DataField="Country" HeaderText="Country">
  <ItemStyle BackColor="LightSteelBlue" />
</asp:BoundField>
<asp:BoundField DataField="BirthDate" HeaderText="Birth Date"
DataFormatString="{0:MM/dd/yyyy}" />
<asp:BoundField DataField="Notes" HeaderText="Notes">
  <ItemStyle Wrap="True" width="400" />
</asp:BoundField>
```

El control DetailsView soporta la edición de los datos vinculados posibilitando la edición, creación y eliminación de registros de datos. Para ello debe asignarse el valor True a las propiedades *AutoGenerateInsertButton*, *AutoGenerateUpdateButton*, y *AutoGenerateDeleteButton*.

La operación de borrado se lleva acabo inmediatamente, pero la de modificación e inserción cambian el estado de control al modo de edición o inserción. En estos modos, los campos del DetailsView se sustituyen por cajas de texto convencionales vinculadas a los datos.

### 2.7.2.- El Control FormView

El control *FormView* realiza una función semejante al *DetailsView*, pero empleando plantillas para organizar el diseño de los datos a visualizar de modo semejante a como lo hace el control *GridView*.

```
<asp:FormView ID="FormView1" runat="server" DataSourceID="sourceEmployees">
  <ItemTemplate>
    <b>
      <## Eval("EmployeeID") %> -
      <## Eval("TitleOfCourtesy") %> <## Eval("FirstName") %>
      <## Eval("LastName") %>
    </b>
    <hr />
    <small><i>
      <## Eval("Address") %><br />
      <## Eval("City") %>, <## Eval("Country") %>,
      <## Eval("PostalCode") %><br />
      <## Eval("HomePhone") %></i>
    </small>
    <br /><br />
    <## Eval("Notes") %>
    <br /><br />
  </ItemTemplate>
</asp:FormView>
```

Los tipos de plantillas admitidos por el *FormView* son los siguientes:

- **ItemTemplate:** Diseño de visualización convencional.
- **EditItemTemplate:** Diseño para edición de datos.
- **InsertItemTemplate:** Diseño para inserción de datos nuevos.
- **FooterTemplate:** Diseño para el pie del formulario
- **HeaderTemplate:** Diseño para la cabecera de formulario
- **EmptyDataTemplate:** Diseño para registro sin datos.
- **PagerTemplate:** Diseño para fila de paginación.

A diferencia del control *DetailsView*, el control *FormView* no genera de forma automática los botones de edición; hay que añadirlos manualmente. Para ello basta con añadir controles *Button* o *LinkButton* y asignar a su propiedad *CommandName* el valor correspondiente a la función que deseamos que desempeñe:

- **Edit :** Cambia el control *FormView* a modo de edición utilizando la plantilla *EditItemTemplate*.
- **Cancel:** Cancela la edición o inserción de nuevos datos y retorna al modo de sólo lectura utilizando la plantilla *ItemTemplate*.
- **Update:** Aplica las modificaciones realizadas en el origen de datos y lanza el evento *ItemUpdating* y *ItemUpdated*.
- **New:** Cambia el control *FormView* al modo de inserción. El control visualiza entonces un registro nuevo con los campos vacíos aplicando la plantilla *InsertItemTemplate*.
- **Insert:** Inserta los datos nuevos en el origen de datos generando los eventos *ItemInserting* y *ItemInserted*.
- **Delete:** Elimina el registro actual visualizado. Se lanzan entonces los eventos *ItemDeleting* y *ItemDeleted*. No provoca cambios en el modo de visualización del control.

## ***El Control Web DataSource***

Los controles orígenes de datos se derivan de la clase *System.Web.UI.DataSourceControl*, y proporcionan información sobre la conexión y la consulta de unos datos, para poder vincularlos a otros controles Web dentro de una página. Existen múltiples controles de orígenes de datos especializados en acceder a datos a través de servidores SQL Server, controladores ODBC, o OLEDB, archivos XML... etc.

Es importante destacar que todos los controles orígenes de datos muestran los mismos métodos y propiedades y son vinculables de igual manera a otros controles Web tales como el control GridView. Esto permite independizar los datos mostrados y manipulados en una página ASP.NET mediante sus controles Web, de los orígenes de datos de donde proceden.

Entre los controles de origen de datos incluidos en ASP.NET se encuentran:

- ObjectDataSource
- SqlDataSource
- AccessDataSource
- XMLDataSource

Los controles DataSource tienen dos tareas:

1. Obtiene los datos deseados desde un origen de datos externo como una base de datos, un fichero de Access, o XML, y envía recuperados a los datos en los controles vinculados.
- 2.
3. Actualiza los datos en el origen de datos cuando éstos son modificados en los controles vinculados.

Las tareas de vinculación de datos se dan en el siguiente orden:

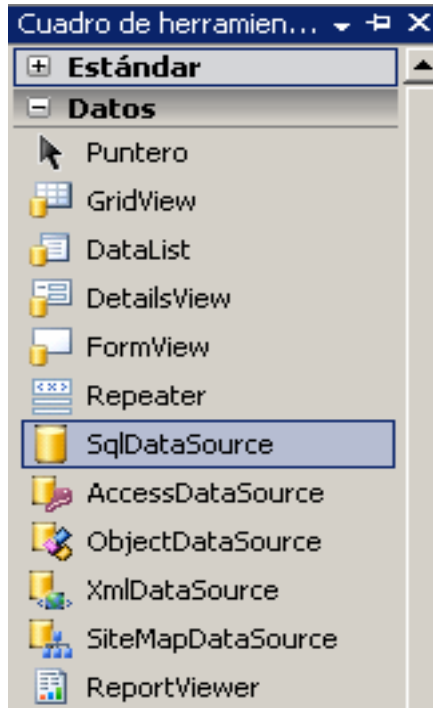
1. **Evento Init**
2. **Evento Load**
3. Eventos de Controles de página.
4. ***Eventos de actualización de DataSource.***
5. **Evento PreRender**
6. ***Ejecución de consulta del control origen de datos, e inserción de los datos recuperados en los controles vinculados.***
7. La página es renderizada.
8. **Evento Unload.**

El evento de consulta se provoca la primera vez que se carga la página con el consiguiente acceso a la base de datos, no obstante, si no hay modificaciones en los datos o en la consulta, no se establece conexión con la base de datos cada vez que se recarga la página.

El evento de actualización se dispara cuando se practican modificaciones de los datos en algún control vinculado al control DataSource, y genera acto seguido realiza una consulta para recargar la información actualizada desde la base de datos.

### 3.1.- CONTROL ORIGEN DE DATOS A SQL SERVER ( *SqlDataSource* )

Lo primero que necesitamos es añadir un objeto origen de datos a la página ASP.NET. Para ello debemos emplear la vista de diseño.



En la ventana del *Cuadro de herramientas* del Visual Studio, podemos encontrar en la categoría *Datos*, todos los controles Web ASP relacionados con el acceso a bases de datos, y entre ellos; los *DataSource*.

En el caso de querer realizar una consulta contra una base de datos de SQL Server, deberemos emplear el objeto *SqlDataSource*.

Este control debemos seleccionarlo y arrastrarlo a la ventana de diseño de la página ASP.NET en la que deseemos acceder a la base de datos.

El objeto *SqlDataSource* es un objeto de datos. Esto quiere decir, que a pesar de que aparezca en el diseño de la página; no genera ningún elemento visible para el cliente que reciba la página.

Para que el control *SqlDataSource* pueda funcionar requiere la configuración de un proveedor de datos, una cadena de conexión y una consulta.

El proveedor de datos a emplear se indica mediante una cadena de texto en la propiedad *ProviderName* del control. Ej:

```
<asp:SqlDataSource ProviderName="System.Data.SqlClient" ... />
```

La cadena de conexión se configura en la propiedad *ConnectionString*. Esta propiedad puede fijarse directamente en el código, o declararla como parte del fichero web.config de la aplicación Web en la sección <connectionStrings> de la sección <configuration>.

```
<configuration>
  <connectionStrings>
    <add name="Northwind"
      connectionString="Data Source=localhost;Initial Catalog=Northwind;
      Integrated Security=SSPI"/>
  </connectionStrings>
</configuration>
```

El código de diseño del control para emplear la cadena de conexión "Northwind" definida en el web.config sería:

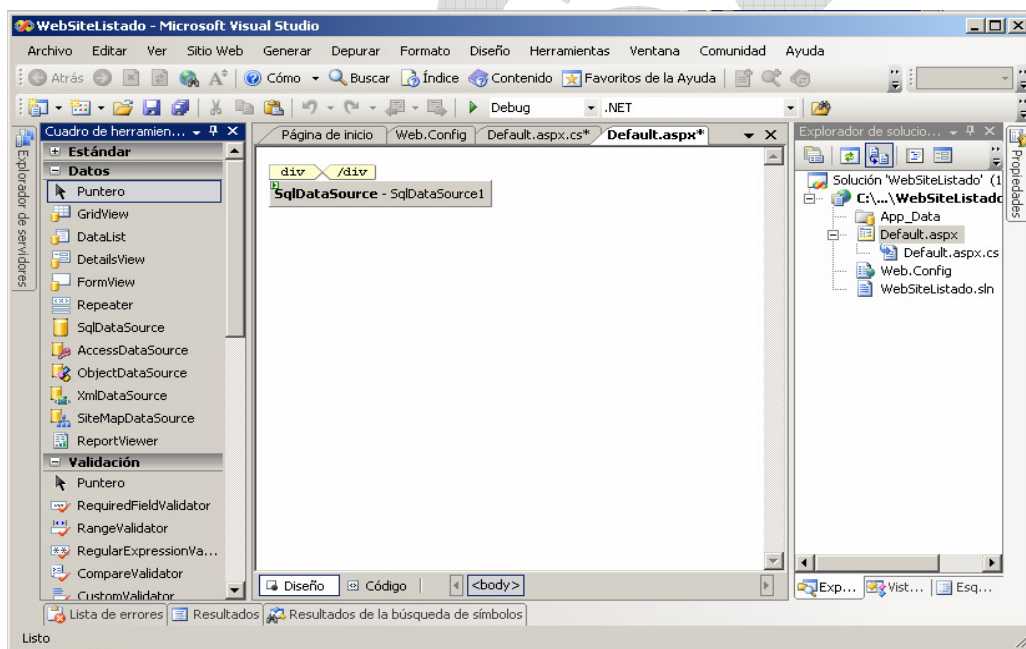
```
<asp:SqlDataSource ConnectionString="<%$ ConnectionStrings:Northwind %%" ... />
```

Un control `SqlDataSource` es capaz de recuperar los datos de una consulta. La propiedad `SelectCommand` puede recibir como valor dos tipos de datos:

- Una consulta SQL de tipo SELECT. En este caso el valor de la propiedad `SelectCommandType` deberá fijarse a "Text". Este es el valor por defecto.
- El nombre de un procedimiento almacenado presente en la base de datos que ejecutará la correspondiente consulta internamente. En este caso el valor de la propiedad `SelectCommandType` deberá fijarse a "StoredProcedure".

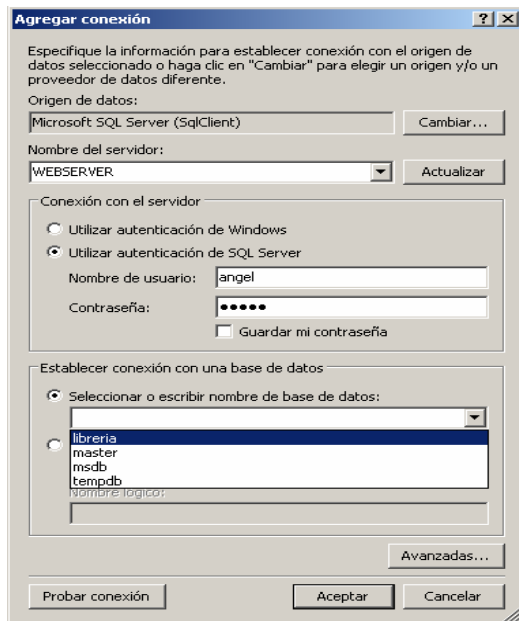
```
<asp:SqlDataSource ID="sourceEmployees" runat="server"
    ProviderName="System.Data.SqlClient"
    ConnectionString="<%%$ ConnectionStrings:Northwind %>"
    SelectCommand=
    "SELECT EmployeeID, FirstName, LastName, Title, City FROM Employees"/>
```

El control `SqlDataSource` también puede encargarse de actualizar en el origen de datos las modificaciones realizadas por el usuario en los valores de los campos vinculados. Deben entonces asignarse también los comandos correspondientes de modificación, inserción y eliminación basados en la misma consulta en las propiedades `InsertCommand`, `UpdateCommand`, y `DeleteCommand`.



Una vez incluido el control `SqlDataSource` en el diseño de la página ASP.NET, sólo nos basta configurarlo para poder empezar a emplearlo. Si hacemos clic en la etiqueta inteligente que en el extremo superior derecho del control; nos aparecerá el cuadro flotante: "Tareas de `SqlDataSource`", donde deberemos seleccionar la opción "Configurar origen de datos".





Nos aparecerá entonces la ventana “Agregar Conexión”.

En esta ventana deberemos seleccionar el origen de datos, y el nombre del servidor al que queremos acceder.

En el caso de SQL Server, debemos seleccionar:

### ***Microsoft SQL Server ( SqlClient )***

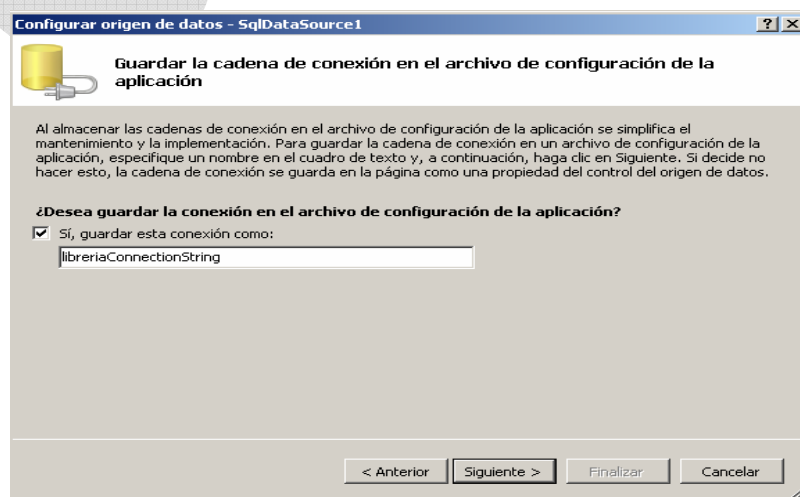
El nombre del equipo se corresponde con la dirección IP o nombre del equipo que posee el servicio de bases de datos instalado en la red.

En el caso de emplear una cuenta propia de SQL Server con permiso de inicio de sesión debemos marcar el opción “Utilizar Autenticación de SQL Server”, e introducir el nombre y contraseña correspondientes.

Finalmente, si los datos de conexión son correctos; debemos seleccionar en la lista desplegable el nombre de la base de datos contenida en el servidor a la que queremos conectarnos. Para comprobar que la conexión con la base de datos es completamente correcta, es recomendable pulsar el botón “Comprobar conexión” situado en la parte inferior del formulario.

Una vez completa la configuración de la conexión, Visual Studio ofrece la posibilidad de almacenar la cadena de conexión resultante con el nombre que deseemos.

Para ello debemos dejar marcada la casilla “Sí, guardar esta conexión como:”. La cadena de conexión se almacenará entonces en el fichero Web.Config dentro del bloque `<connectionStrings>` con el nombre indicado en su atributo `name`.





A continuación se muestra el cuadro “Configurar Origen de Datos”.

La finalidad de esta ventana es definir la consulta que provea de resultados al objeto DataSource que estamos creando. Para ello seleccionaremos la tabla, y los campos que deseamos mostrar en los resultados.

El botón “Where” permite definir condiciones para filtrar los resultados de la consulta. El botón “GroupBy” permite definir una orden de agrupamiento para los resultados en base a los valores de una o varias columnas devueltas.

En el cuadro “Instrucción Select” muestra la sentencia SQL resultante que se enviará al origen de datos para obtener la información. A continuación se mostrará la ventana “Consulta de Prueba”.

usuario	password	email	telefono
roger	petrov	none@none.com	5550309
manuel	rodriguez	manu@cipsa.com	5550493
victor	krawler	victor@cipsa.com	5554930
vharak	wolfgang	wolf@cipsa.com	5550492

Esta ventana permite realizar una simulación ejecutando la consulta SQL creada en la ventana anterior, para comprobar la información que se obtiene de la base de datos. Para todo ello basta pulsar el botón “Consulta de Prueba”.

## ACCESO A DATOS CON ADO.NET

Si los resultados son los que deseamos pulsamos el botón “Finalizar” para completar la configuración del objeto. La cadena de conexión creada al principio se inscribirá entonces en el fichero Web.Config para poder ser reutilizada por otros controles SqlDataSource:

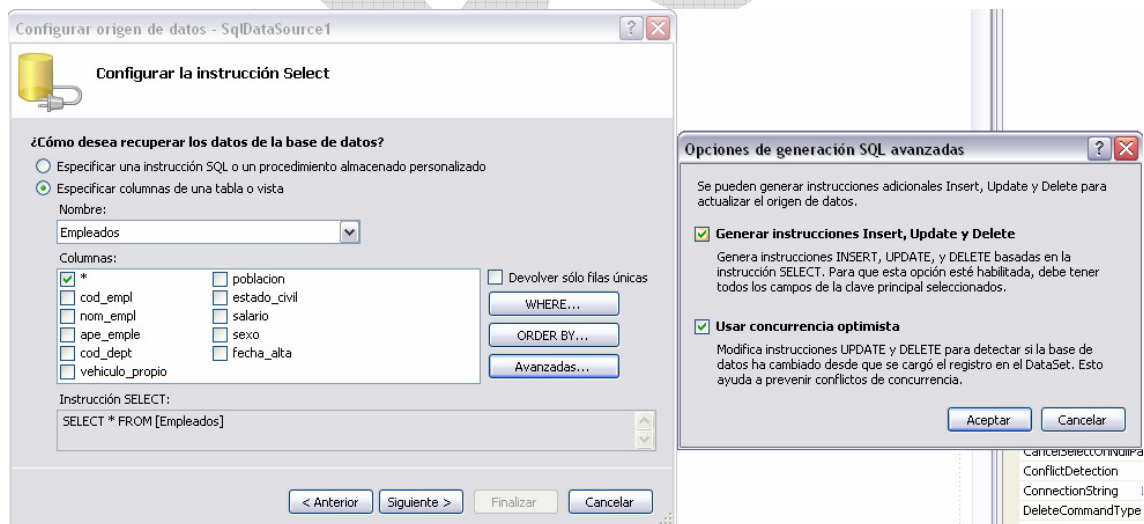
```
<connectionStrings>
  <add name="libreriaConnectionString" connectionString="Data Source=WEBSERVER;Initial
catalog=libreria;User ID=angel;Password=00000"
  providerName="System.Data.SqlClient" />
</connectionStrings>
```

Al mismo tiempo, en la vista de código de la página puede verse el código correspondiente al control origen de datos definido.

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
  ConnectionString="<%$ ConnectionStrings:libreriaConnectionString %%"
  SelectCommand="SELECT [usuario], [password], [email], [telefono] FROM [clientes]">
</asp:SqlDataSource>
```

El control DataSource también permite la modificación, inserción y eliminación de registros en la base de datos. Para ello es necesario definir los comandos correspondientes en sus propiedades *UpdateCommand*, *DeleteCommand*, e *InsertCommand*. Esto puede hacerse de dos maneras:

1.- Seleccionando la opción “Generar Instrucciones Insert, Update y Delete” en el cuadro de dialogo “Opciones de generación SQL avanzadas”. Dicho cuadro de dialogo es accesible desde la ventana de configuración de origen de datos del control DataSource pulsando el botón “Avanzadas...”.



2.- Empleando los diseñadores de consultas seleccionando las propiedades *UpdateQuery*, *InsertQuery* y *DeleteQuery* del cuadro de propiedades del control DataSource.

Esto permite que controles vinculados al control DataSource como el GridView permitan la edición, inserción y eliminación de registros.

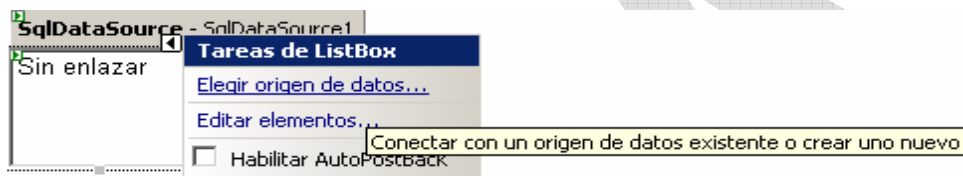
## 3.2.- VINCULOS CON CONTROL ORIGEN DE DATOS EN DISEÑO

El vínculo a datos también puede establecerse mediante diseño empleando un control origen de datos tipo `SqlDataSource`.

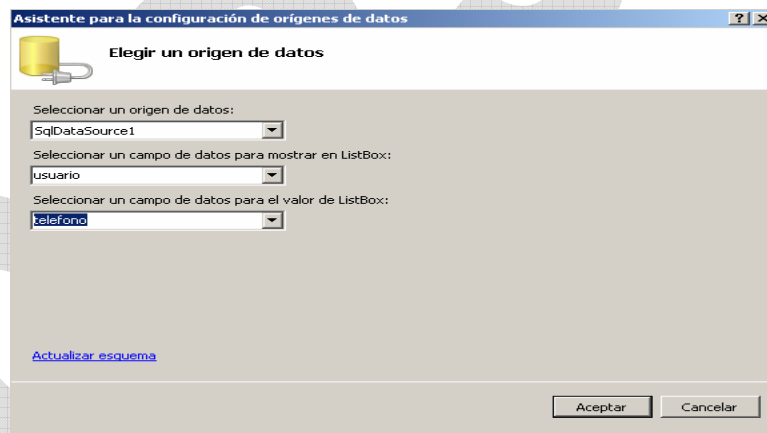
Supóngase una página a la que añadimos un control origen de datos para conectarnos a la misma base de datos de SQL Server del ejemplo anterior. El control sería un `SqlDataSource`, el cual debería configurarse para emplear la cadena de conexión “`LibreriaConnectionString`” realizando la misma consulta:

### 3.2.1.- Vínculo de control lista a origen de datos en diseño

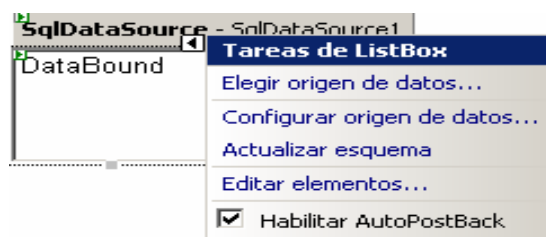
Una vez configurado el control origen de datos, se vincula al control `ListBox` en diseño haciendo clic sobre la etiqueta inteligente:



A continuación seleccionamos la opción “Elegir origen de datos” y se muestra la ventana del Asistente para la configuración de orígenes de datos:



En la lista bajo el título “Seleccionar un origen de datos”, aparecerán todos los controles orígenes de datos presentes en la página. En el mismo momento en que se selecciona éste, se rellenan automáticamente las demás listas con los nombres de las columnas presentes en el origen de datos seleccionado. Finalmente, marcamos la casilla de verificación del *AutoPostBack*:



## ACCESO A DATOS CON ADO.NET

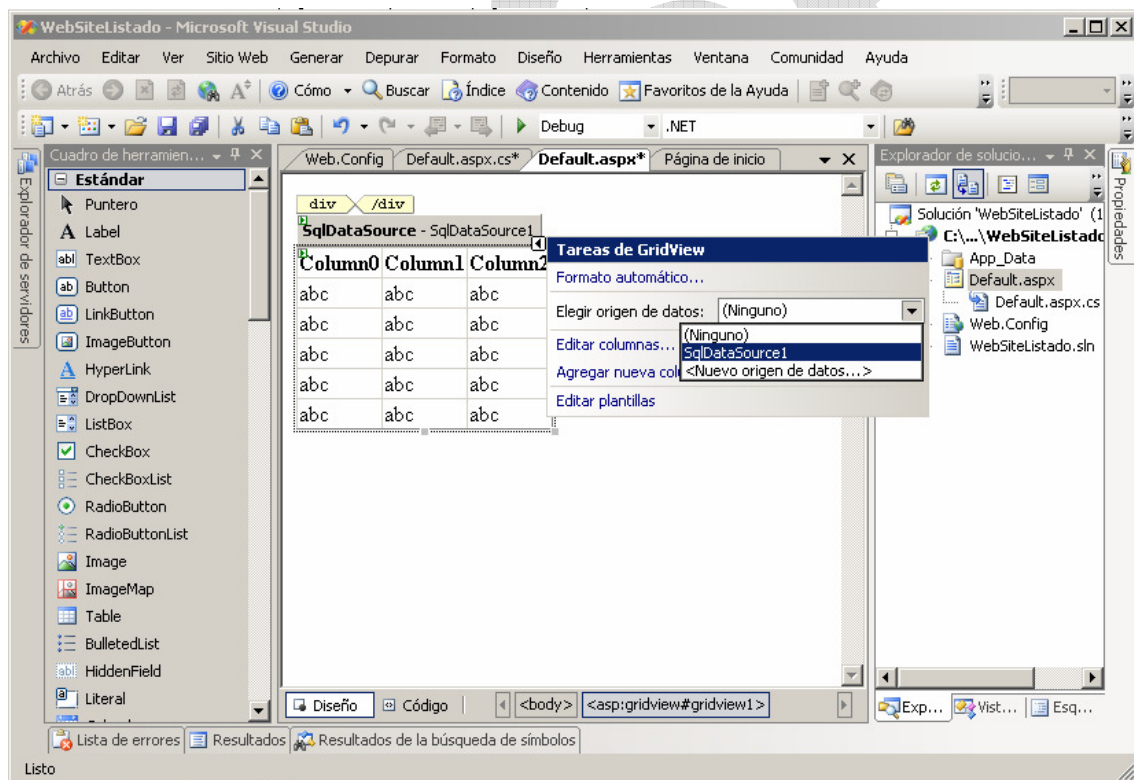
Con estas operaciones el vínculo a datos queda completamente definido en el código de diseño de la propia página ASP.NET:

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server" ConnectionString="<%=  
connectionStrings:libreriaConnectionString %>"  
SelectCommand="SELECT [usuario], [password], [email], [telefono] FROM  
[clientes]"></asp:SqlDataSource>  
<asp:ListBox ID="ListBox2" runat="server" AutoPostBack="True" DataSourceID="SqlDataSource1"  
DataTextField="usuario" DataValueField="telefono"  
onSelectedIndexChanged="ListBox2_SelectedIndexChanged">  
</asp:ListBox>
```

El resultado al ejecutar la página es exactamente el mismo al del ejercicio anterior. Tan sólo quedaría añadir el código correspondiente al evento SelectedIndexChanged del control ListBox de igual manera que en el ejemplo anterior.

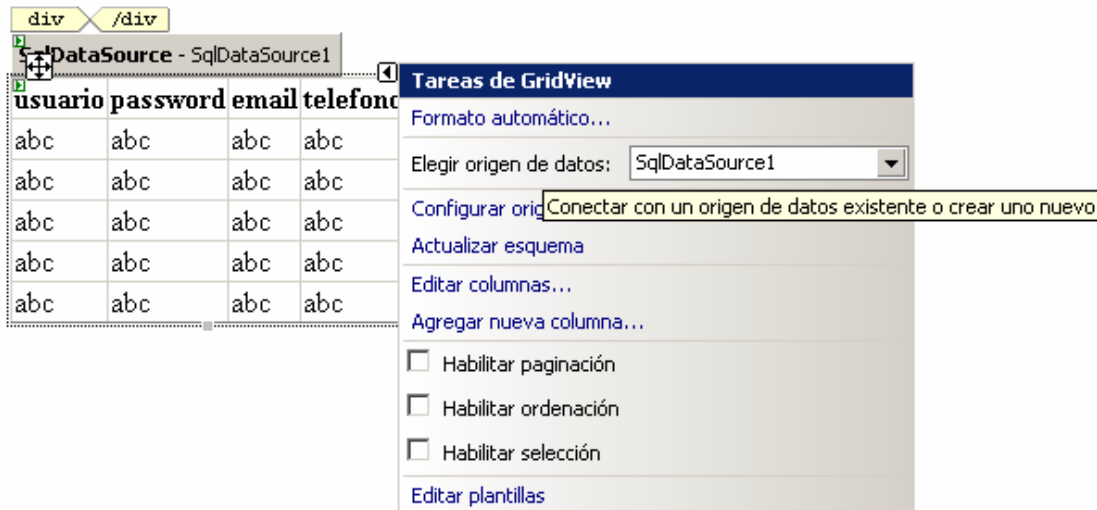
### 3.2.2.- Vinculo de control tabla a control origen de datos en diseño

Para vincular en diseño un control GridView a un control origen de datos previamente definido pulsamos sobre la etiqueta inteligente que aparece en el margen superior derecho del diseño del control:



Dentro del cuadro flotante “Tareas de GridView” podemos seleccionar cualquier control origen de datos ya presente en la página, para que sirva de fuente de datos para el control GridView.

Al seleccionarlo; el control GridView se reconfigura automáticamente mostrando las columnas de los datos que provienen del control origen de datos seleccionado:



Si pasamos a la vista de código se puede ver en el código del control, la declaración de las nuevas columnas acordes a los resultados del control de origen de datos:

```
<asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="False"
DataSourceID="SqlDataSource1">
  <Columns>
    <asp:BoundField DataField="usuario" HeaderText="usuario" SortExpression="usuario" />
    <asp:BoundField DataField="password" HeaderText="password" SortExpression="password"/>
    <asp:BoundField DataField="email" HeaderText="email" SortExpression="email" />
    <asp:BoundField DataField="telefono" HeaderText="telefono" SortExpression="telefono"/>
  </Columns>
</asp:GridView>
```

En este caso Visual Studio ha rediseñado el control a través de su código definiendo las columnas necesarias para mostrar todos los campos presentes en los resultados del control origen de datos.

### 3.2.3.- Manipulación de datos vinculados en control tabla

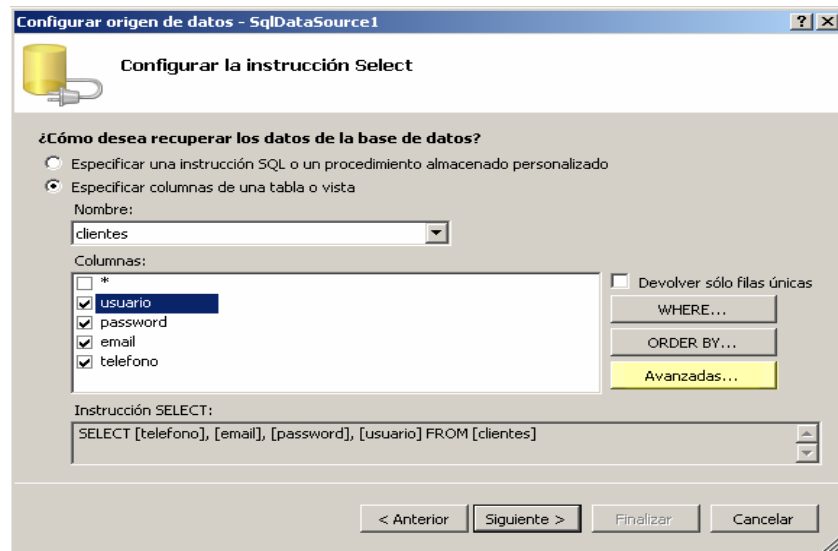
El Control GridView también permite operaciones de modificación y actualización de los datos vinculados al mismo. Pero para ello, el control origen de datos vinculado al GridView debe permitir estas operaciones sobre los datos.

Cuando creamos un control de origen de datos, éste sólo cuenta inicialmente con la sentencia SQL de consulta definida en el atributo SelectCommand:

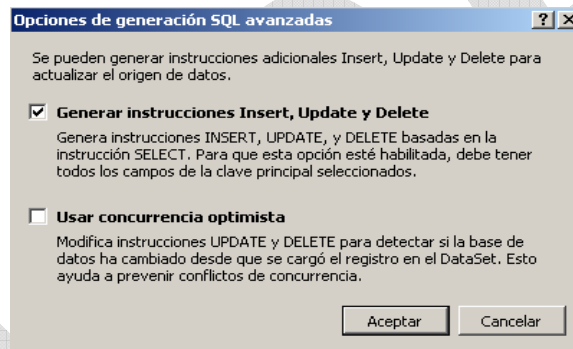
```
<asp:SqlDataSource ID="SqlDataSource1"
Runat="server"
ConnectionString="<?%$ ConnectionStrings:loteriaConnectionString %>"
SelectCommand="SELECT * FROM DATOS">
</asp:SqlDataSource>
```

Sin embargo, si deseamos que el control permita la actualización de los datos debemos añadir las sentencias de inserción, modificación y borrado. Esta operación puede hacerse en diseño haciendo clic sobre la etiqueta inteligente del control origen de datos y seleccionando la opción "Configurar origen de datos". Se abrirá entonces la ventana de Configuración de origen de datos.

## ACCESO A DATOS CON ADO.NET



Una vez seleccionada la consulta, debemos pulsar el botón Avanzadas...



Se abre entonces la ventana de opciones de generación SQL avanzadas. Esta ventana está provista de dos opciones:

La casilla de verificación "Generar instrucciones Insert, Update, y Delete", permite que Visual Studio genere automáticamente las sentencias SQL necesarias para las operaciones de inserción, modificación y eliminación de registros para la información recuperada por el control origen de datos.

Debe tenerse presente que todas sentencias se generan en función de los campos recuperados por la sentencia de consulta, por lo cual es necesario que previamente la hayamos definido correctamente.

El código de diseño del control origen de datos queda entonces así:

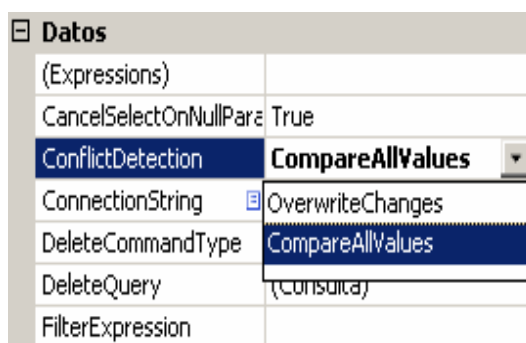
```
<asp:SqlDataSource ID="SqlDataSource1"
    runat="server"
    ConnectionString="<%$ ConnectionStrings:libreriaConnectionString %>"
    SelectCommand="SELECT [usuario], [password], [email], [telefono], [id_usuario] FROM
[cientes]"
    DeleteCommand="DELETE FROM [clientes] WHERE [id_usuario] = @id_usuario"
    InsertCommand="INSERT INTO [clientes] ([usuario], [password], [email], [telefono],
[id_usuario]) VALUES (@usuario, @password, @email, @telefono, @id_usuario)"
    UpdateCommand="UPDATE [clientes] SET [usuario] = @usuario, [password] = @password,
[email] = @email, [telefono] = @telefono WHERE [id_usuario] = @id_usuario">
</asp:SqlDataSource>
```



La principal diferencia en el código es la aparición de los atributos *DeleteCommand*, *InsertCommand*, y *UpdateCommand* con las sentencias SQL generadas por el Visual Studio.

La segunda casilla de verificación “Usar concurrencia optimista”, habilita el control de concurrencia optimista. Esto consiste en añadir a las sentencias de modificación de la base de datos los valores originales de los registros que van a modificarse. Esto permite detectar si un registro ha sido modificado por otro usuario antes de actualizarlo.

En la mayoría de los casos la actualización se realizará con éxito. Si se detecta una modificación posterior a la consulta, se puede decidir no continuar con la modificación, o sobrescribir la información.

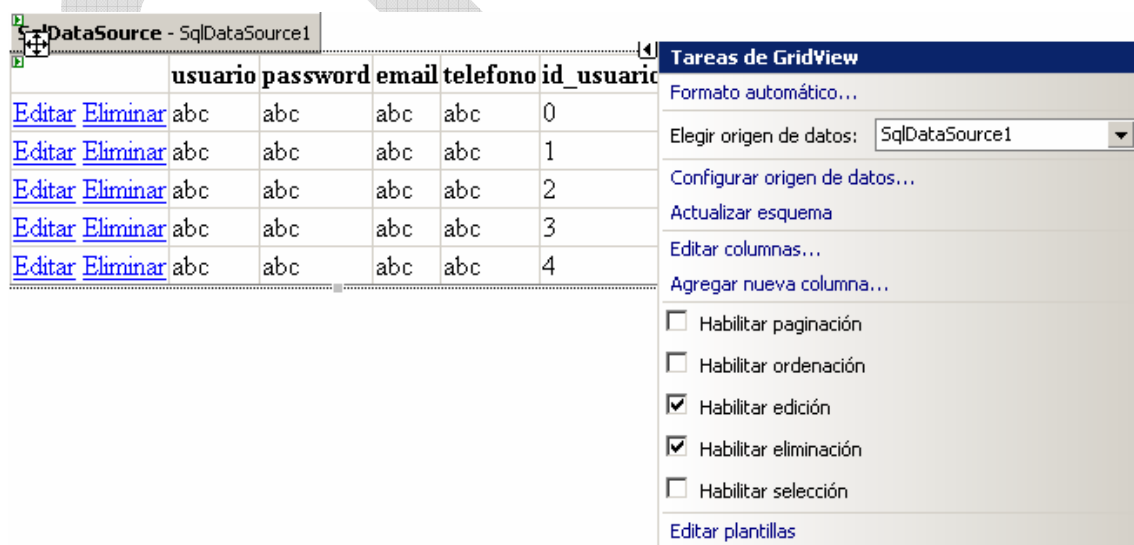


Este comportamiento se establece mediante la propiedad *ConflictDetection*.

Si se le asigna el valor *CompareAllValues*, la actualización no se lleva a cabo si se detecta una modificación tras la consulta. Si se le asigna el valor *OverwriteChanges* la modificación se lleva a cabo de todos modos.

### 3.2.4.- Habiliar columnas de edición, actualización y eliminación.

Cuando se vincula un control GridView a un control origen de datos configurado para permitir la actualización de los datos, aparecen nuevas opciones en etiqueta inteligente:



Vemos que aparecen dos casillas de verificación nuevas: “Habilitar edición” y “Habilitar eliminación”. Al marcar estos controles, aparecen dos nuevas columnas en el control GridView.

## ACCESO A DATOS CON ADO.NET

La columna de edición permite que el usuario pueda editar directamente en el GridView los campos de la fila seleccionada. Al hacer clic sobre el enlace “Editar” se recarga la página automáticamente, pero en esta ocasión los datos de la fila aparecen en cajas de texto preparadas para ser modificadas.

	usuario	password	email	telefono	id_usuario
<a href="#">Editar</a> <a href="#">Eliminar</a>	roger	petrov	none@none.com	5550309	1
<a href="#">Actualizar</a> <a href="#">Cancelar</a>	manuel	rodriguez	manu@cipsa.com	5550493	2
<a href="#">Editar</a> <a href="#">Eliminar</a>	victor	krawler	victor@cipsa.com	5554930	3
<a href="#">Editar</a> <a href="#">Eliminar</a>	vharak	wolfgang	wolf@cipsa.com	5550492	4

Al mismo tiempo, en la misma fila aparecen dos enlaces “Actualizar” y “Cancelar” que permiten aceptar los cambios e ingresarlos en la base de datos, o bien descartarlos y recuperar los valores originales.

Si hacemos clic sobre el enlace “Eliminar”, el registro de la fila correspondiente es eliminado automáticamente de la tabla y borrado de la base de datos.

De manera semejante a como se hizo con la columna de selección, es posible configurar las columnas de edición y eliminación de registros para que muestren imágenes o botones en vez de enlaces.

### 3.2.5.- Eventos de actualización de control GridView

Los controles de ASP.NET en general, y los controles de datos en particular, proporcionan múltiples eventos para controlar desde código las acciones sobre los datos. Por ejemplo; existe el evento *RowUpdating* que se dispara justo tras editar una fila y pulsar el enlace “Actualizar”. A su vez; existe otro evento *RowUpdated* que se dispara justo tras la actualización y que permite comprobar si ha habido errores.

A continuación se muestra una lista de los eventos más importantes:

	<a href="#">RowCancelingEdit</a>	Se produce cuando se hace clic en el botón Cancelar de una fila en modo de edición, pero antes de que la fila salga del modo de edición.
	<a href="#">RowCommand</a>	Se produce cuando se hace clic en un botón de un control <b>GridView</b> .
	<a href="#">RowCreated</a>	Se produce cuando se crea una fila en un control <b>GridView</b> .
	<a href="#">RowDataBound</a>	Se produce cuando una fila de datos se enlaza a los datos de un control <b>GridView</b> .
	<a href="#">RowDeleted</a>	Se produce cuando se hace clic en el botón Eliminar de una fila, pero después de que el control <b>GridView</b> elimine la fila.
	<a href="#">RowDeleting</a>	Se produce cuando se hace clic en el botón Eliminar de una fila, pero antes de que el control <b>GridView</b> elimine la fila.
	<a href="#">RowEditing</a>	Se produce cuando se hace clic en el botón Editar de una fila, pero antes de que el control <b>GridView</b> entre en el modo de edición.
	<a href="#">RowUpdated</a>	Se produce cuando se hace clic en el botón Actualizar de una fila, pero después de que el control <b>GridView</b> actualice la fila.
	<a href="#">RowUpdating</a>	Se produce cuando se hace clic en el botón Actualizar de una fila, pero antes de que el control <b>GridView</b> actualice la fila.
	<a href="#">SelectedIndexChanged</a>	Se produce cuando se hace clic en el botón Seleccionar de una fila, pero después de que el control <b>GridView</b> se ocupe de la operación de selección.
	<a href="#">SelectedIndexChanging</a>	Se produce cuando se hace clic en el botón Seleccionar de una fila, pero antes de que el control <b>GridView</b> se ocupe de la operación de selección.



Una de las características más importantes de los eventos como *RowUpdated*, o *RowDeleted*, es que permiten conocer si la operación de modificación se completó satisfactoriamente.

```
protected void GridView1_RowUpdated(object sender, GridViewUpdatedEventArgs e)
{
    if (e.Exception != null)
    {
        e.ExceptionHandled = true;
        Response.Write("Se ha producido un error al actualizar los datos.");
    }
}
```

En el caso del evento *RowUpdated*, el segundo parámetro es un objeto de tipo *GridViewUpdatedEventArgs*, que posee las propiedades *Exception* y *ExceptionHandled*.

La propiedad *Exception* hace referencia a la excepción que se haya podido producir al actualizar, y de no ser así su valor es *null*. Por otro lado, la propiedad *ExceptionHandled* indica si la excepción es atendida por el método asignándole el valor *True*.

CLIPSA

# Autenticación de Usuarios

La autenticación es el proceso por el que se comprueba la identificación de un usuario con el fin de concederle unos permisos en el uso de una aplicación Web. Estos permisos pueden permitir al usuario realizar ciertas operaciones en la aplicación Web, o acceder a páginas restringidas en la misma.

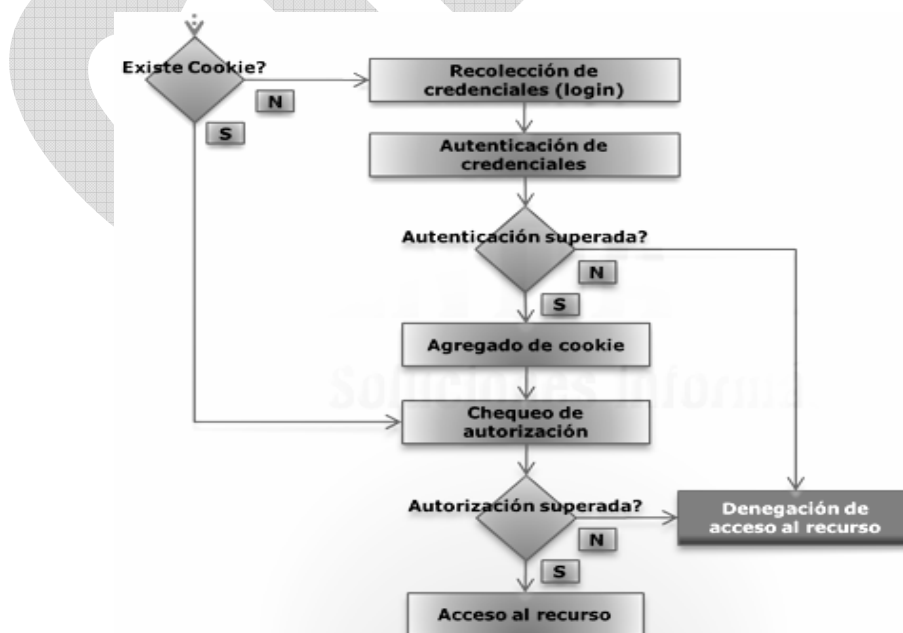
ASP.NET soporta dos modos fundamentales de autenticación:

- *Windows* : Este modo se basa en cuentas de usuario de dominio que se emplea cuando los usuarios se identifican utilizando los datos de cuentas definidas en un dominio de red.
- *Forms*: Este modo se basa en cuentas propias de la aplicación Web independientes de ningún dominio. Es el modo más común.

El modo de autenticación predeterminado es inexistente con lo que todos los usuarios pueden acceder a todas las páginas y tienen los mismos permisos. Para añadir un modo de autenticación debe incluirse la directiva *authentication* dentro del archivo de configuración *web.config*.

```
<authentication mode = "Forms">
```

El modo de autenticación mediante formulario se basa en cookies. Cuando el usuario se autentica se envía automáticamente una cookie con el nombre *.ASPXAUTH* al navegador con sus credenciales. El sistema de autenticación comprueba de forma automática la existencia y datos de dicha cookie en cada petición que recibe para comprobar la identidad del usuario:



Algoritmo de autenticación mediante formulario en ASP.NET

## ACCESO A DATOS CON ADO.NET

Con el valor “forms” asignado al atributo *mode* de la directiva *authentication* se activa la autenticación, pero falta indicar la página que debe ocuparse de mostrar el formulario correspondiente y comprobar las credenciales dadas por el usuario. Por defecto; ASP.NET utiliza como página de autenticación *login.aspx*. Si dicha página no existe y se desea utilizar otra para la autenticación; es necesario indicarlo añadiendo la directiva *forms* a la directiva *authentication*. Esta directiva contiene posee el atributo *loginURL* indicando en su valor asociado la URL de la página con el formulario de autenticación correspondiente:

```
<authentication mode = “Forms”>
  <forms loginURL = “autorización.aspx” />
</authentication>
```

Adicionalmente se pueden añadir otros atributos tales como *defaultUrl* que indica la URL de la página a la que debe ser reconducido el usuario si este accede directamente a la página de autenticación e introduce un nombre de usuario y una contraseña correctos.

La autenticación sirve para comprobar la identidad de un determinado usuario pero no asocia ningún permiso que le permita o restrinja el acceso a ciertas páginas de la aplicación Web. Para ello es necesario definir *reglas de acceso*.

### 4.1.- REGLAS DE ACCESO

Las reglas de acceso permiten indicar que usuarios tienen acceso a las páginas contenidas la aplicación Web. Las reglas se definen dentro de la sección *authorization* del archivo *web.config* mediante etiquetas *<deny>* que deniegan el acceso, y *<allow>* que conceden acceso, siguiendo el siguiente esquema:

```
<authorization>
  <[allow|deny] users roles verbs />
</authorization>
```

La siguiente regla de acceso confiere permiso únicamente al usuario “Jhon” y se lo niega a todos los demás usuarios. ( \*)

```
<authorization>
  <allow users=“John”/>
  <deny users=“*”/>
</authorization>
```

La siguiente regla niega el acceso a usuarios no autenticados ( ? ), o lo que es lo mismo; se lo confiere a todos los que están autenticados.

```
<authroization>
  <deny users=“?”/>
</authorization>
```

Cualquier acceso no permitido provoca la redirección de la petición del usuario a la página de autenticación que indique las credenciales de una cuenta con permisos para acceder.

Las reglas de acceso se definen dentro del web.config y su efecto se aplica por tanto a todos los archivos y subcarpetas contenidas en la aplicación Web. Cualquier solicitud a un archivo contenido en la carpeta principal de la aplicación Web o cualquiera de sus subcarpetas estará sometida a las reglas de acceso.

## 4.2.- CONTROL DE AUTENTICACION

El control de autenticación consiste en tomar las credenciales dadas por el usuario y compararlas con las de los usuarios reconocidos para aceptar o rechazar su identificación. Esta operación puede realizarse manualmente mediante código, o delegarlo al sistema de autenticación de ASP.NET.

### 4.2.1.- Comprobación de autenticación manual.

La comprobación manual consiste en tomar el nombre de usuario y contraseña dada por el usuario y decidir si el usuario es o no reconocido. Sea la siguiente página ASP.NET:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="auth.aspx.cs"
Inherits="WebApplication1.autorizacion" %>

<!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>
      <asp:TextBox ID="txtUsuario" runat="server"></asp:TextBox>
      <asp:TextBox ID="txtClave" runat="server"></asp:TextBox>
      <asp:Button ID="Button1" runat="server" Text="Button" onclick="Button1_Click" />
      <asp:Label ID="lblError" runat="server" Text="Label"></asp:Label>
    </div>
  </form>
</body>
</html>
```

Se trata de un formulario provisto de dos cajas de texto para introducir un nombre de usuario y una contraseña, un botón para enviar los datos, y una etiqueta para mostrar un mensaje de error si los datos no son reconocidos:

Suponiendo que se realiza la autenticación manualmente, y que el único usuario reconocido es "alumno" con clave "cipsa"; el código correspondiente al evento click del botón sería el siguiente:

```
protected void Button1_Click(object sender, EventArgs e)
{
    if (this.txtUsuario.Text == "alumno" && this.txtClave.Text == "cipsa")
    {
        FormsAuthentication.RedirectFromLoginPage(this.txtClave.Text, false);
    } else {
        lblError.Text = "Usuario y/o contraseña incorrectos.";
    }
}
```

## ACCESO A DATOS CON ADO.NET

La clase *FormAuthentication* está definida dentro del espacio de nombres *System.Web.Security*. El método estático *RedirectFromLoginPage* permite considerar al usuario autenticado con el nombre indicado y lo redirige a la página original solicitada, o bien a la página indicada por el atributo *defaultURL* en la directiva *authentication* del fichero *web.config*.

### 4.2.2.- Comprobación de autenticación automática.

La comprobación automática consiste en dejar que sea ASP.NET quien compruebe el nombre de usuario y contraseña empleando el método estático *Authenticate* de la clase *FormAuthentication*.

```
public static bool Authenticate(  
    string name,  
    string password  
)
```

El método recibe dos parámetros de tipo cadena con el nombre de usuario y contraseña introducidos por el usuario, y retorna un valor lógico indicando si se corresponden con un usuario o no.

En este caso los nombres de usuario y contraseña deben introducirse en el archivo *web.config* dentro de la directiva *forms* de *authentication*.

```
<authentication mode="Forms">  
  <forms loginUrl="ingreso1.aspx" >  
    <credentials passwordFormat="Clear">  
      <user name ="alumno" password ="cipsa"/>  
    </credentials>  
  </forms>  
</authentication>
```

Cada nombre de usuario y contraseña se inserta mediante la directiva *credentials* utilizando la etiqueta *user* y sus atributos *name* y *password*.

El código de validación del ejemplo anterior debería reescribirse de la siguiente manera:

```
protected void Button1_Click(object sender, EventArgs e)  
{  
    if (FormsAuthentication.Authenticate( this.txtUsuario.Text,  
                                           this.txtClave.Text) )  
    {  
        FormsAuthentication.RedirectFromLoginPage(this.txtClave.Text, false);  
    } else {  
        lblError.Text = "Usuario y/o contraseña incorrectos.";  
    }  
}
```

Las cuentas de usuario definidas en la directiva *authentication* pueden emplearse en las reglas de acceso para restringir y conceder permisos de acceso a usuarios concretos.

### **EJERCICIOS RECOMENDADOS**

Crear una BD en ACCESS con una tabla de clientes que conste de los siguientes campos.

DNI  
NOMBRE  
APELLIDOS  
TELEFONO  
NACIMIENTO

El fichero debe llamarse DATOS.MDB

Crear una aplicación que cumpla las siguientes especificaciones:

1. Crea una aplicación Web con una pequeña página que muestre la cadena de conexión a la base de datos de Access almacenada en el archivo *web.config*.
2. Modificar la página para que muestre un control DataGridView con todos los datos de la tabla Clientes.
3. Modificar la página añadiendo dos cajas de texto y dos botones que permitan filtrar los registros mostrados bien por DNI o por nºTelefono. Emplea para ello comandos de consulta parámetros.
4. Añade a la página un enlace a otra página ( *altas.aspx* ) con un formulario que permita dar de alta nuevos clientes. No debe permitirse dos clientes con el mismo DNI.
5. Modificar la página añadiendo una opción de modificación a cada registro de cliente mostrado en el GridView con un enlace a una página con un formulario que muestre los datos del cliente seleccionado y permite modificarlos. Este formulario debe mostrar en cajas de texto todos los campos de un registro previamente seleccionado en el GridView.
6. Modifica la página añadiendo una opción de eliminación a cada registro de cliente mostrado en el GridView con un botón que permita eliminar el registro correspondiente de la base de datos.

Debe emplearse en todo momento una clase que encapsule todas las operaciones contra base de datos. ( Véase como ejemplo clase planteada en el apartado “Modelo de Aplicación en 3 Capas”. )

### **PRÁCTICA**

Se desea crear una aplicación Web que permite consultar desde Internet el precio de los productos existentes en un pequeño comercio de material informático. Dicho comercio vende diferentes tipos de productos tales como discos duros, procesadores, memorias, placas bases... etc. De cada tipo se almacenan diferentes datos tales como el nombre del producto, el precio, el stock disponible del mismo... etc.

#### **Funcionamiento público:**

La aplicación Web consta de una página principal ( *inicio.aspx* ) provista de una lista de categorías de productos existentes. Esta página debe mostrar también un pequeño enlace "Administración" cuya funcionalidad se describe más adelante.

Al seleccionarse la categoría se pasa a la página de detalles ( *detalles.aspx* ) donde se debe mostrar una tabla con el conjunto de todos los productos de la categoría seleccionada previamente. Por cada producto debe indicarse la *descripción*, el *precio*, y un indicador de la disponibilidad del producto con los siguientes colores:

- Verde → Existen más de 10 unidades.
- Amarillo → Existen entre 10 y 3 unidades.
- Rojo → Existen 3 o menos unidades.
- "SIN EXISTENCIAS" → No hay ninguna unidad disponible.

Esta página debe tener también un enlace *Volver a Inicio* que permite retornar a la página de inicio para poder consultar productos de otras categorías.

El enlace "Administración" de la página principal permite activar la gestión de la aplicación Web reservada sólo para el administrador. Al pulsarse el enlace se accede a una página de autenticación ( *login.aspx* ) provista de un formulario para introducir un nombre de usuario y una contraseña. Estos deben almacenarse en el fichero de configuración de la aplicación: *web.config*.

#### **Funcionamiento para administrador:**

Si el usuario indica el nombre de usuario y contraseña correspondiente se retorna a la página de inicio. En caso contrario se le envía a una página de error.

Para el administrador la página de inicio ( *inicio.aspx* ) debe mostrar además de la lista de categorías ya mencionada, una caja de texto y un botón *Nueva Categoría* para crear nuevas categorías. Debe verificarse que el nombre no sea nulo ni coincida con el de ninguna categoría ya existente, en cuyo caso se mostrará un mensaje de error.

Para el administrador la página de detalles ( *detalles.aspx* ) debe mostrar por cada producto un botón *Eliminar* y otro *Modificar*. Al final de la lista de artículos debe mostrarse un botón *Insertar Nuevo Artículo*.

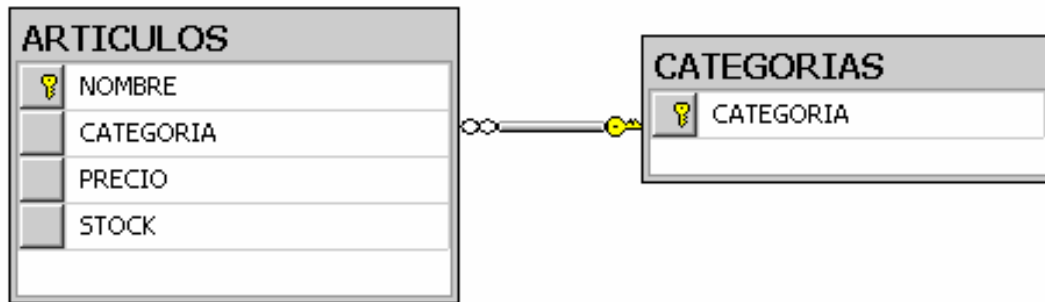
El botón *Eliminar* borra el artículo correspondiente de la lista. Los botones *Insertar* y *Modificar* permiten pasar a una página de detalles ( *articulo.aspx* ) provista de un formulario con tres campos de texto: *Descripción*, *Precio* y *Stock*, y dos botones *Guardar* y *Cancelar*. En el caso del botón *Modificar* debe mostrarse en cada campo del formulario el valor del artículo correspondiente para permitir su modificación. En caso del botón *Insertar* los campos deberán aparecer vacíos.

El botón *Guardar* modifica o inserta el nuevo artículo en la base de datos. Debe en ambos casos validarse la descripción no coincida con la de ningún otro artículo, en cuyo caso se recarga la página mostrando un mensaje de error descriptivo. Si los datos son correctos se retorna a la página detalles que debe volver a mostrar todos los artículos de la categoría seleccionada anteriormente.



## ACCESO A DATOS CON ADO.NET

La Base de datos a emplear ya ha sido diseñada y creada en un servidor de SQL Server 2005 con la siguiente estructura.



La definición de la tabla CATEGORIAS es la siguiente:

CATEGORIA → NVARCHAR(20) NOT NULL

PrimaryKey → CATEGORIA.

La definición de ARTICULOS es la siguiente:

NOMBRE → VARCHAR(50) NOT NULL

CATEGORIA → VARCHAR(20) NOT NULL

PRECIO → SMALLMONEY NOT NULL

STOCK → INT NOT NULL

Primary Key → NOMBRE.

El campo CATEGORIA de ARTICULOS es una clave externa relacionada con el campo de mismo nombre de la tabla CATEGORIAS.

Los datos de conexión al servidor SQL Server 2005 desde .NET son los siguientes:

**Tipo de proveedor:**

Microsoft SQL Server ( SqlClient )

**Espacio de nombres:**

System.Data.SqlClient

**Nombre del servidor:**

WEBSERVER

**Nombre de base de datos:**

informatica

**Usuario:**

alumno

**Contraseña:**

alumno

Crear la aplicación siguiendo una estructura en capas constituida por:

- *Proyecto Librería: Base Datos* → Contiene la/las clases de acceso a cualquier base de datos dada una cadena de conexión. Con métodos/clases necesarias para enviar y recoger los datos de cualquier consulta.
- *Proyecto Librería: Logica* → Contiene clases correspondientes a las operaciones de consulta y modificación de datos correspondientes a cada página de la aplicación Web.
- *Proyecto Web: Aplicación* → Contiene todas las páginas estáticas y dinámicas que conforman la aplicación Web.