



Acceso a Bases de Datos
ADO.NET



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

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

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

1.- Fundamentos de ADO.NET

ADO.NET es el nombre que recibe una parte del framework .NET dedicada a la conexión y manipulación de bases de datos desde cualquier aplicación escrita en Visual Basic, o C#.NET

1.1.- Modelos de acceso Conectado / Desconectado

La tecnología ADO.NET introduce dos modalidades de trabajo con las bases de datos:

- **Modo Conectado** → El modo conectado se basa en una conexión constante con la BD durante toda la ejecución de la aplicación. La información se maneja directamente contra la BD.
- **Modo Desconectado** → El modo desconectado se basa en una conexión a la BD el tiempo mínimo necesario para recuperar los datos. Estos se copian en la memoria de la aplicación donde se manipulan sin conexión a la BD.

Los modelos 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, y es lo que se denomina comúnmente modelo conectado.

Este modelo tiene como inconveniente que puede provocar la caída del servidor de datos cuando se conecten muchos usuarios al mismo tiempo, lo cual es especialmente grave si se trata de conexiones a través de Internet.

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.

Este modelo tiene como inconveniente el que los datos con los que trabaja la aplicación son una copia y no reflejan el estado actual de la base de datos.

1.2.- Modelo de Objetos de ADO.NET

ADO.NET se compone de un conjunto de clases integradas en el Framework .NET que permite la conexión y manipulación de bases de datos. Las clases básicas principales se encuentran definidas en el espacio de nombres *System.Data.Common*.

- **Conexión** (*System.Data.Common.DbConnection*) → Representa la conexión a una base de datos concreta empleando un nombre y contraseña determinados.
- **Comando** (*System.Data.Common.DbCommand*) → Representa una sentencia SQL que se desea ejecutar contra la base de datos. La sentencia puede ser una consulta, alta, baja o actualización.
- **DataReader** (*System.Data.Common.DbDataReader*) → Representa un conjunto de datos conectados obtenidos a partir de una consulta. Cada dato se obtiene de la base de datos manteniendo la conexión abierta.
- **DataAdapter** (*System.Data.Common.DbDataAdapter*) → Representa un acceso en modo desconectado a los datos resultantes de una consulta. Los datos se copian en la memoria de la aplicación en un objeto *DataSet*.

1.3.- Proveedores de Datos

ADO.NET permite la conexión a diferentes sistemas de bases de datos empleando diferentes *proveedores de datos*. Un proveedor de datos es una librería que permite la intercomunicación entre una aplicación y un determinado sistema de base de datos.

ADO.NET es soporta de manera predeterminada los siguientes proveedores:

- **SQL Server** → Proveedor específico para bases de datos Microsoft SQL Server
- **Oracle** → Proveedor específico para bases de datos Oracle.
- **OLE DB, ODBC** → Proveedores genéricos que permiten la conexión a sistemas de bases de datos que no cuentan con proveedores específicos, p.ej: Microsoft Access.

La jerarquía de clases ADO.NET está diseñada de manera que la forma de trabajar con diferentes proveedores es la misma. Para ello, ADO.NET define un conjunto de clases específicas para cada proveedor de datos extendiendo las clases bases definidas en el espacio de nombres *System.Data.Common*.

En función del proveedor de datos que vaya a emplearse para acceder a una determinada base de datos, deberán emplearse las clases de un determinado espacio de nombres:

Las clases específicas para cada proveedor se almacenan en espacios de nombres propios:

- **SQL Server** → *System.Data.SqlClient*.
(*SqlConnection, SqlCommand, SqlDataReader, SqlDataAdapter*)
- **Oracle** → *System.Data.OracleClient*
(*OracleConnection, OracleCommand, OracleDataReader, OracleDataAdapter*)
- **OLE DB** → *System.Data.OleDb*
(*OleDbConnection, OleDbCommand, OleDbDataReader, OleDbDataAdapter*)
- **ODBC** → *System.Data.Odbc*
(*OdbcConnection, OdbcCommand, OdbcDataReader, OdbcDataAdapter*)

2.- Conexión a Base de Datos

La conexión a la base de datos se lleva a cabo empleando un objeto *Connection*. En función del proveedor de datos, debe emplearse la clase correspondiente:

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

Todas estas poseen un conjunto común de métodos y propiedades heredados de la clase base *DbConnection*, que puede emplearse para referenciar una instancia de cualquiera de ellas.

2.1.- La cadena de conexión

Un objeto conexión requiere de una *Cadena de Conexión*. Una Cadena de conexión es una simple cadena de texto que contiene la información necesaria para conectarse a una base de datos empleando un determinado proveedor.

La sintaxis de una cadena de conexión depende del proveedor de datos al que vaya destinada:

Ejemplo: Cadena de conexión para base de datos Microsoft Access 2007 con proveedor OleDb.

```
Provider=Microsoft.ACE.OLEDB.12.0;Data Source="C:\Producto.accdb"
```

Ejemplo: Cadena de conexión para base de datos SQL Server 2008 mediante proveedor específico para SQL Server.

```
Data Source=192.168.1.100;Initial Catalog=MULTAS;UserID=usuario;Password=cipsa
```

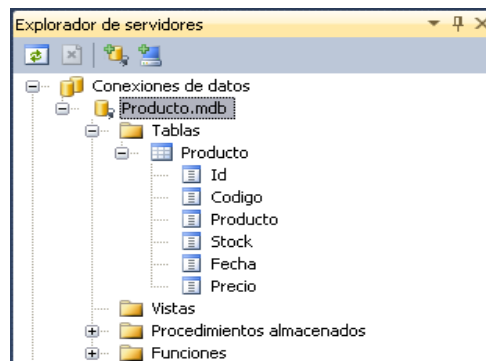
2.2.- Obtención de la cadena de conexión

El entorno de desarrollo de Visual Studio permite conectarse directamente a una base de datos para observar su información y estructura como ayuda para el desarrollo de aplicaciones mediante la ventana del **Explorador de Servidores**.

2.2.1.- Uso del Explorador de Servidores

El Explorador de servidores permite establecer conexión con bases de datos desde el IDE de Visual Studio para examinar su estructura y datos mediante se desarrolla la aplicación.

Ver → Explorador de Servidores

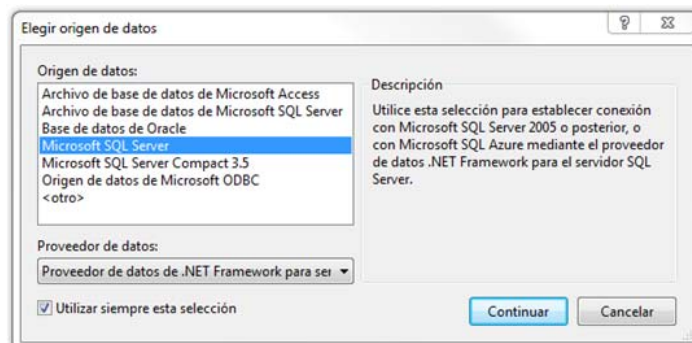


La herramienta permite observar la estructura de la base de datos así como sus datos de manera visual.

Para conectarse desde Visual Studio a una base de datos debe seleccionarse:

Herramientas → Conectar con base de datos...

Primero debe indicarse el proveedor de datos a emplear para la conexión a la base de datos:



- **Para SQL Server** → Microsoft SQL Server
- **Para MS Access** → Archivo de bases de datos de Microsoft Access

A continuación debe indicarse los parámetros de conexión necesarios. Los parámetros requeridos varían en función del proveedor seleccionado previamente:

Ejemplo: Conexión con SQL Server

Nombre del servidor → Nombre de red del servidor o dirección IP en la red.

Conexión con el servidor → Autenticación de usuario para conexión al servidor:

- **Autenticación de Windows:** Usuario local de Windows, o usuario del dominio con permisos de acceso a SQL Server.
- **Autenticación de SQL Server:** Usuario y contraseña de una cuenta con permisos de acceso definida en SQL Server.

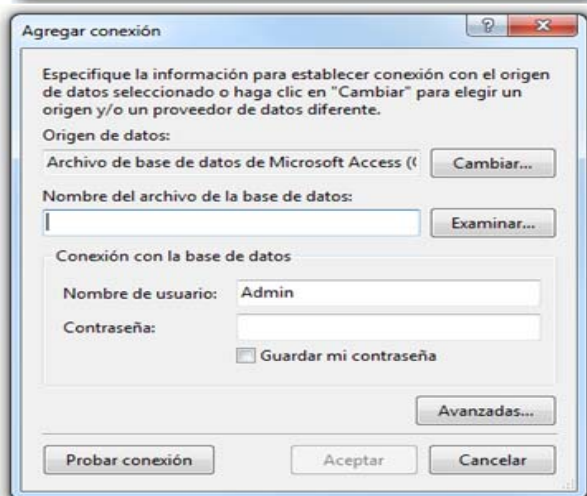
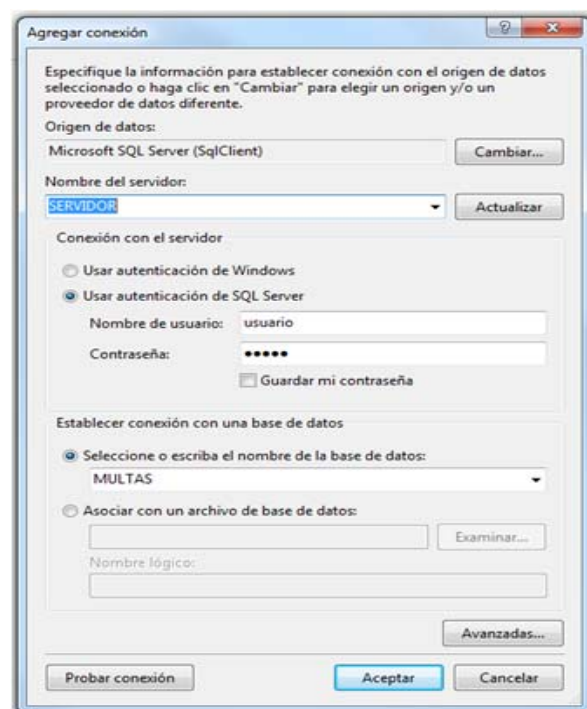
Nombre de la base de datos → Nombre de la base de datos a la que se establece conexión.

Ejemplo: Conexión con Microsoft Access

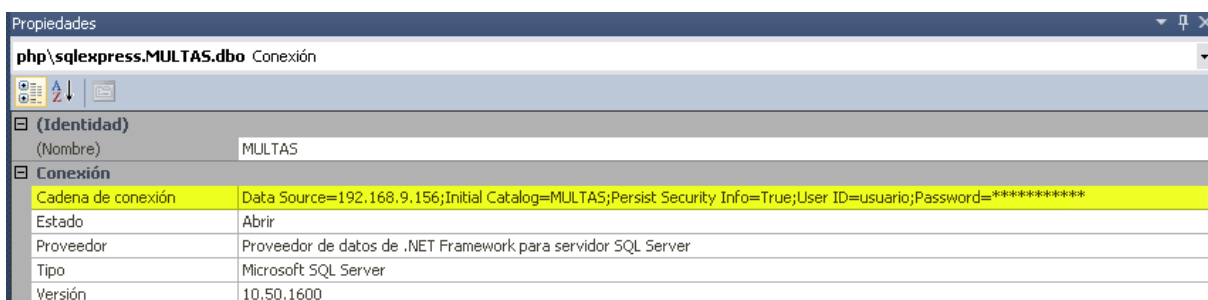
Nombre del archivo de la base de datos → Nombre y ruta al archivo Access con la base de datos.

Conexión con el servidor → Autenticación de usuario para apertura de la base de datos.

Guardar mi contraseña → Indica si la contraseña de acceso se guarda como parte de la cadena de conexión para no tener que introducirla.



Para cada base de datos presente en el Explorador de Servidores es posible obtener la cadena de conexión generada por el Visual Studio en el campo "Cadena de Conexión" presente en la ventana *Propiedades*:



2.2.2.- Consulta y manipulación de datos.

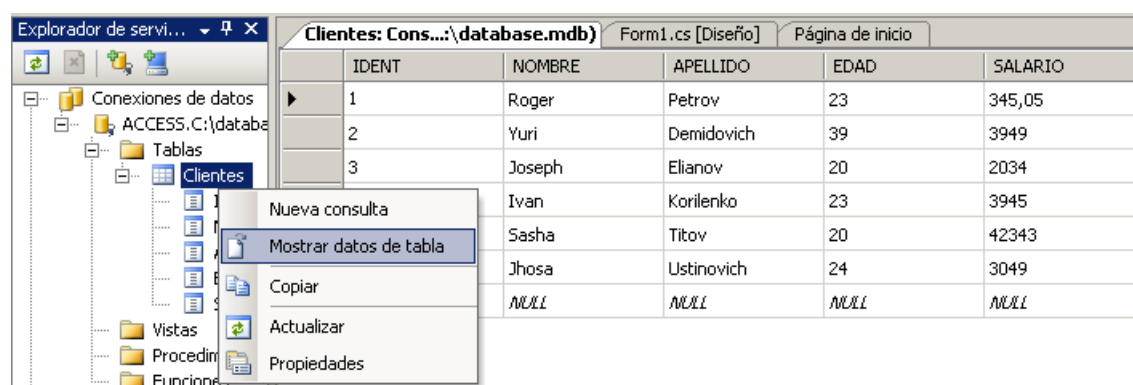
Desde la ventana del explorador de servidores es posible consultar y manipular directamente los datos contenidos en un origen de datos partiendo de una conexión ya establecida. Para ello basta con expandir el nodo de la conexión correspondiente:



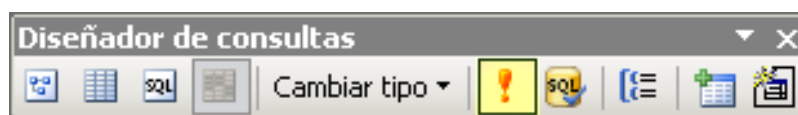
Expandiendo el nodo de conexión observaremos cuatro carpetas que contienen las tablas, vistas, funciones y procedimientos almacenados que haya en el origen de datos.

Si expandimos la carpeta Tablas obtendremos todas las tablas de la base de datos con sus respectivas columnas.

Para consultar y manipular directamente los datos de una tabla hacemos clic con el botón derecho sobre su nodo y seleccionamos la opción *Mostrar Datos de Tabla*.



Con la tabla ya abierta podemos ver los datos de los registros contenidos en la misma, además de modificarlos, eliminarlos y añadir otros. Para confirmar las modificaciones que hagamos debemos pulsar el botón “Ejecutar SQL” presente en la barra de herramientas del Diseñador de Consultas.



El botón con icono de exclamación confirma las modificaciones que hagamos en una tabla

Mediante estas herramientas podemos preparar y conocer la información sobre la vamos a trabajar con nuestra aplicación.

2.3.- Conexión con la base de datos

La conexión con la base de datos se realiza a través de un objeto conexión. Este objeto debe crearse de la clase correspondiente al proveedor de datos que se use:

- ***SqlConnection***
- ***OdbcConection***
- ***OleDbConnection***
- ***OracleConnection***

Todas las clases comparten métodos y propiedades comunes heredados de la clase base ***DbConnection***. En todos los casos, la conexión se instancia pasando como parámetro al constructor la cadena de conexión.

2.3.1.- Apertura de conexión a la base de datos

La apertura de conexión a la base de datos se lleva acabo empleando el método ***Open()***.

Ejemplo: El siguiente código muestra la apertura de una conexión a una base de datos empleando el proveedor para SQL Server. La variable *cadenaConexion* lleva la cadeba de conexión a la base de datos. La variable booleana *ok* recibe el valor lógico cierto si la conexión se establece sin problemas, o falso en caso contrario.

VB
<pre> Dim ok As Boolean Dim cadenaConexion As String = "Data Source=192.168.9.158; Initial Catalog=MULTAS; Persist Security Info=True; UserID=usuario; Password=cipsa" Dim conexion As SqlConnection = Nothing Try conexion = New SqlConnection(cadenaConexion) conexion.Open() ok = True Catch ex As SqlException ok = False End Try </pre>
C#
<pre> bool ok; String cadenaConexion = "Data Source=192.168.9.158;Initial Catalog=MULTAS;Persist Security Info=True;UserID=usuario;Password=cipsa"; SqlConnection conexion = null; try { conexion = new SqlConnection(cadenaConexion); conexion.Open(); ok = true; } catch(SqlException ex) { ok = false; } </pre>

Debe recordarse que debe importarse el espacio de nombres *System.Data.SqlClient* para poder instanciar directamente la clase *SqlConnection*.

La apertura de la conexión es una operación delicada que puede provocar una excepción por multitud de causas:

- La cadena de conexión está mal formada.
- El servidor no está presente en la red.
- El usuario y clave indicados no tienen permisos de acceso al servidor.
- La base de datos indicada no existe o no se tiene permiso de acceso a la misma.

Por ello, es necesario encapsular siempre la apertura de la conexión en un bloque de TRY-CATCH para capturar las excepciones que podrían darse. Los métodos miembros de las clases de ADO.NET definen excepciones propias en función del proveedor de datos que emplean. De este modo, los métodos de la clase *SqlConnection*, genera excepciones *SqlException*, definidas también dentro de *System.Data.SqlClient*.

2.3.2.- Cierre de conexión a la base de datos

Tan importante como abrir la conexión a la base de datos es cerrarla convenientemente una vez que ya no es necesario. Para ello se emplea el método **Close()**.

Es de vital importancia asegurarse de que cada conexión que se abre sea cerrada tras su uso pase lo que pase. De no ser así, la conexión quedaría abandonada ocupando recursos de manera innecesaria. Hay dos formas de garantizar esto.

Ejemplo 1 .- Uso de estructuras TRY-CATCH

VB
<pre> Dim conexion As SqlConnection = Nothing Try conexion = New SqlConnection(cadenaConexion) conexion.Open() ' Apertura de la conexión ' OPERACIONES CON LA BASE DE DATOS Catch ex As SqlException ' DETECCION E INFORME DEL ERROR. Finally If Not conexion Is Nothing Then conexion.Close() ' Cierre de la conexión End If End Try </pre>

C#

```

String cadenaConexion = "Data Source=192.168.9.158;Initial Catalog=MULTAS;Persist Security Info=True;User ID=usuario;Password=cipsa";

SqlConnection conexion = null;
try {
    conexion = new SqlConnection( cadenaConexion );
    conexion.Open();
    // OPERACIONES CON LA BASE DE DATOS
} catch( SqlException ex ) {
    // DETECCIÓN E INFORME DE ERROR
} finally {
    if ( conexion != null ) conexion.Close();
}

```

La cláusula FINALLY asegura la ejecución de la sentencia Close sobre el objeto conexión, tanto si las operaciones sobre la base de datos se completan con éxito, como si se produce una excepción recogida por el CATCH. En cualquiera de los casos, la conexión es cerrada.

Ejemplo 2.- Uso de estructura USING**VB**

```

Using objConexion As New SqlConnection(cadenaConexion)
    objConexion.Open()           ' Apertura de conexion
    ' OPERACIONES CON LA BASE DE DATOS
objConexion.Close()           ' Cierre de conexion
End Using

```

C#

```

using ( SqlConnection conexion = new SqlConnection( cadenaConexion ) ) {
    conexion.Open();                // Apertura de la base de datos.
    // OPERACIONES CON LA BASE DE DATOS
    conexion.Close();              // Cierre de la base de datos.
}

```

El bloque USING asegura que el objeto conexión es eliminado (y por tanto; cerrada la conexión que representa) cuando la ejecución sale del código definido dentro del bloque. Esto puede suceder tanto si se produce una excepción, como si se ejecuta correctamente todo el código contenido en el USING.

El bloque USING sólo puede usarse con clases que implementan el interfaz *IDisposable*.

2.3.3.- Agrupamiento de conexiones (*Pooling de Conexiones*)

El establecimiento de conexión a una base de datos es una operación compleja y lenta. Para mejorar la ejecución fluida del código ADO.NET emplea de manera automática el agrupamiento de conexiones.

En la práctica, la mayoría de las aplicaciones se conectan a una o dos bases de datos aunque pueden emplear múltiples conexiones al tiempo. Esto significa que durante la ejecución de la aplicación, muchas conexiones idénticas se abrirán y cerrarán de forma repetida con el consiguiente coste en tiempo para establecerlas. Para reducir el costo de la apertura de conexiones, ADO.NET emplea la agrupación de conexiones.

Para cada cadena de conexión única se crea un grupo de conexión. Cuando se crea un grupo, se crean y agregan tantos objetos conexión como el mínimo especificado (por defecto 0). Las conexiones se agregan al grupo cuando es necesario, hasta el tamaño máximo del grupo especificado (100 es el valor predeterminado), y se liberan de nuevo en el grupo cuando se cierran o eliminan. Cuando se solicita un objeto conexión, se obtiene del grupo correspondiente si se encuentra disponible una conexión que se pueda utilizar. Una conexión de este tipo debe estar sin utilizar, tener un contexto de transacción coincidente o no estar asociada con ningún contexto de transacción y tener un vínculo válido al servidor. Cuando se llama al método Close de la conexión, ésta retorna al grupo de conexiones activas en lugar de ser cerrada. Una vez que la conexión vuelve al grupo está preparada para volverse a utilizar en la siguiente llamada a Open.







La agrupación de conexiones puede mejorar de forma significativa el rendimiento y la escalabilidad de la aplicación. De forma predeterminada, la agrupación de conexiones está habilitada en ADO.NET. A menos que la deshabilite explícitamente, el agrupador optimiza las conexiones a medida que se abren y cierran en la aplicación. El agrupamiento de conexiones puede configurarse mediante ciertas cláusulas indicadas en la cadena de conexión. En el caso de SQL Server los parámetros relacionados con el agrupamiento de conexiones indicables como parte de la cadena de conexión son:

- **Pooling** → (por defecto: TRUE) → Habilita el uso de pooling de conexiones.
- **Min Pool Size** (por defecto: 0) → Indica el número mínimo de conexiones que se mantienen abiertas para su reutilización.
- **Max Pool Size** (por defecto: 100) → Indica el número máximo de conexiones que se permiten.

Conclusión: El pool de conexiones es un sistema que engloba las conexiones a la base de datos de una aplicación para que se realicen más rápidamente. No obstante, ello provoca que las conexiones permanezcan abiertas consumiendo recursos en el servidor.

2.3.4.- Estado de la conexión

La propiedad *State* del objeto conexión es de sólo lectura y permite conocer el estado de la conexión en un momento dado. La propiedad devuelve un valor del tipo enumerado *System.Data.ConnectionState*. Los posibles valores son los siguientes:

	Nombre de miembro	Descripción
	Broken	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.
	Closed	La conexión está cerrada.
	Connecting	El objeto de conexión está conectando con el origen de datos. Este valor se reserva para versiones futuras del producto.
	Executing	El objeto de conexión está ejecutando un comando. Este valor se reserva para versiones futuras del producto.
	Fetching	El objeto de conexión está recuperando datos. Este valor se reserva para versiones futuras del producto.
	Open	La conexión está abierta.

Valores de estados de conexión a la base de datos

Esta propiedad puede utilizarse para conocer el estado de la conexión a la base de datos.

2.3.5.- Eventos de conexión

Los objetos conexión definen dos eventos que pueden capturarse:

- **StateChange** → Se dispara cuando cambia el estado de la conexión y da información del estado anterior y actual de la conexión. La sintaxis de la función manejadora es:

VB
<pre>Shared Sub EventoConexi on(sender As Object, e As System.Data.StateChangeEventArgs) End Sub</pre>
C#
<pre>void conexi on_StateChange(object sender, StateChangeEventArgs e) { ... }</pre>

- **InfoMessage** → Se dispara cuando el servidor envía un mensaje o colección de errores a la aplicación cliente. La sintaxis de la función manejadora es:

VB
<pre>Shared Sub EventoMensaj e(sender As Object, e As System.Data.SqlClient.Sql I nfoMessageEventArgs) End Sub</pre>
C#
<pre>void conexi on_StateChange(object sender, Sql I nfoMessageEventArgs e) { ... }</pre>

Estos eventos pueden emplearse para recoger mensajes de error o detectar cambios inesperados en el estado de la conexión. Para asociar funciones manejadoras a eventos de un objeto mediante código debe emplearse la estructura:

addhandler <obj>.<evento>, addressof <funcion_manejadora>

En C# se emplea la estructura

<obj>.<evento> += new <delegado_evento>(<función_manejadora>);

Ejemplo: El siguiente código de consola muestra la apertura y cierre de conexión a una base de datos SQL Server. La función EventoConexión está asociada al objeto conexión creado y muestra por pantalla todos los cambios que se producen en su estado:

VB

```
' FUNCION MANEJADORA DE EVENTO STATECHANGED
Shared Sub EventoConexi on(sender As Object, e As System.Data.StateChangeEventArgs)
    Console.WriteLine(e.OriginalState.ToString() & " - " & e.CurrentState.ToString())
End Sub

Shared Sub Main()
    Dim cadenaConexi on = "Data Source=192.168.9.158;Initial Catalog=MULTAS;Persist
Security Info=True;User ID=usuario;Password=cipsa"
    Dim conexi on As SqlConnection = Nothing
    Try
        ' Instanciacion
        conexi on = New SqlConnection(cadenaConexi on)
        ' Asociacion de evento
        AddHandler conexi on.StateChange, AddressOf EventoConexi on
        Console.WriteLine("LLAMANDO A OPEN...")
        conexi on.Open()
        ' Apertura de la conexión
        ' OPERACIONES CON LA BASE DE DATOS
    Catch ex As SqlException
        ' DETECCION E INFORME DEL ERROR.
    Finally
        If Not conexi on Is Nothing Then
            Console.WriteLine("LLAMANDO A CLOSE...")
            conexi on.Close()
            ' Cierre de la conexión
        End If
    End Try

    Console.ReadKey()

End Sub
```

C#

```

static void Main( String[] args ) {

    String cadenaConexion = "Data Source=192.168.9.158;Initial Catalog=MULTAS;Persist
Security Info=True;User ID=usuario;Password=cipsa";

    SqlConnection conexion = null;
    try {
        // Instanciación
        conexion = new SqlConnection( cadenaConexion );
        // Asociación de evento
        conexion.StateChange+=new StateChangeEventHandler(conexion_StateChange);
        Console.WriteLine("LLamando a OPEN....");
        conexion.Open();                                     // Apertura de conexión

        // OPERACIONES CON LA BASE DE DATOS

    } catch( SqlException ex ) {
        // DETECCIÓN E INFORME DE ERROR
    } finally {
        Console.WriteLine("LLamando a CLOSE....");
        if ( conexion != null ) conexion.Close();
    }

    // Función manejadora del evento StateChange del objeto conexion
    public static void conexion_StateChange(object sender, StateChangeEventArgs e)
    {
        Console.WriteLine(e.OriginalState.ToString()+"-"+e.CurrentState.ToString());
    }
}

```

El resultado mostrado por pantalla es:



```

file:///C:/Documents and Settings/angel/Escritorio/ExamenFinal BBDD con Estructura Muy B...
LLAMANDO A OPEN...
Closed - Open
LLAMANDO A CLOSE...
Open - Closed

```


2.4.- Almacenado de cadena de conexión en fichero de configuración (App.Config)

La cadena de conexión empleada en un objeto conexión puede dejarse especificada directamente en el código:

VB
<code>Dim cadenaConexi on = "Data Source=192. 168. 9. 158; Ini ti al Catalog=MULTAS; Persi st Securi ty Info=True; User ID=usuari o; Password=ci psa"</code>
C#
<code>String cadenaConexi on = "Data Source=192.168. 9. 158; Ini ti al Catalog=MULTAS; Persi st Securi ty Info=True; User ID=usuari o; Password=ci psa";</code>

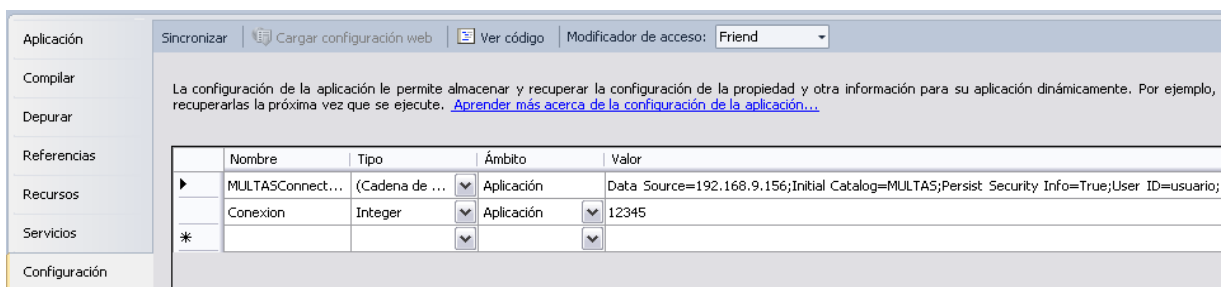
Esto es lo más sencillo pero exige modificar el código de la aplicación si al servidor de base de datos cambia de IP, o cambian el usuario y/o contraseña de la cuenta empleada para conectar con el servidor de base de datos.

Otra opción mucho más conveniente es incluir la cadena de conexión como un **parámetro de configuración**. Un parámetro de configuración es un par valor - identificador que se almacena en un archivo especial llamado *App.Config* en formato XML:

```
<?xml version="1.0" encoding="utf-8" ?>
<configurati on>
  <connectionStrings>
    <add name="ExamenFi nal . My. MySettings. MULTASConnecti onString"
      connectionString="Data Source=192. 168. 9. 158; Ini ti al Catalog=MULTAS; Persi st
      Securi ty Info=True; User ID=usuari o; Password=ci psa"
      providerName="System. Data. Sql Cl i ent" />
    </connectionStrings>
  <startup>
    <supportedRuntime version="v4. 0" sku=". NETFramework, Versi on=v4. 0, Profi le=Cl i ent" />
  </startup>
  <applicati onSettings>
    <ExamenFi nal . My. MySettings>
      <setting name="Conexi on" serial izeAs="String">
        <val ue>12345</val ue>
      </setting>
    </ExamenFi nal . My. MySettings>
  </applicati onSettings>
</configurati on>
```

Los valores pueden añadirse y modificarse desde Visual Studio a través de la pestaña *Configuración de las Propiedades del Proyecto*:

Proyecto → Propiedades de <nombre_proyecto>




Vista de la pestaña Configuración.

2.4.1.- Añadir cadena de conexión al archivo de configuración

Para añadir una cadena de conexión al archivo de configuración se inserta una fila nueva y se indica un **Nombre** y se selecciona como **Tipo** la opción: *(Cadena de Conexión)*.

	Nombre	Tipo	Ámbito	Valor
▶	MULTASConnect...	(Cadena de conexión)	Aplicación	Data Source=192.168.9.156;Initial Catalog=MULTAS;Persist Security...
	Conexion	Integer	Aplicación	12345
*				

Para establecer la cadena de conexión puede escribirse directamente, o bien pulsar el botón punteado  y generarla a través del Asistente de Conexión. Una vez configurada y probada la conexión el propio asistente la inserta en el archivo de configuración.

2.4.2.- Obtener cadena de conexión desde el código.

Para obtener desde código la cadena de conexión almacenada en el archivo de configuración se la llama por el identificador indicado en el campo Nombre contra **My.Setting**:

```
Dim cadenaConexion As String = My.Settings.MULTASConnectionString
```

En C# se utilizan las propiedades estáticas de la clase *Settings* para facilitar el acceso a los valores almacenados en el archivo de configuración *app.config*:

```
String cadenaConexion = Properties.Settings.Default.MULTASConnectionString;
```

La propiedad *MULTASConnectionString* devuelve el valor de la cadena de conexión insertada con el identificador *MULTASConnectionString*.

NOTA IMPORTANTE:

Cuando se establece conexión con bases de datos Access el Visual Studio ofrece la opción de incluir el fichero de base de datos dentro del proyecto. Esto permite que la base de datos se copie junto con el ejecutable cada vez que se compila la aplicación.

Esto implica que cada vez que la aplicación se compila, el fichero de base de datos utilizado por la aplicación es sobrescrito por la copia que forma parte del proyecto. Por lo tanto; todos los cambios realizados por la aplicación sobre la base de datos se pierden al recompilar y ejecutar la aplicación desde el Visual Studio.

Para alterar este comportamiento debe seleccionarse el fichero de base de datos dentro del explorador de soluciones y modificar su propiedad "Copiar en el directorio resultados" a "Copiar si es posterior."

Para más información consultar MSDN:

<http://msdn.microsoft.com/es-es/library/ms233817%28VS.80%29.aspx>

3.- Ejecución de un Comando

Una vez establecida una conexión con el origen de datos es posible ejecutar una consulta para recuperar, modificar, insertar o eliminar datos empleando dicha conexión. Para ello se emplea un objeto comando utilizando clase correspondiente al proveedor utilizado:

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

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

Puede obtenerse un objeto comando llamando al método *CreateCommand* del objeto conexión que se esté empleando. También es posible crear un objeto comando a partir de su propio constructor. Existen varias sobrecargas:

- Public Sub **New**(cmdText As [String](#),
connection As [System.Data.SqlClient.SqlConnection](#))
- Public Sub **New**(cmdText As [String](#),
connection As [System.Data.SqlClient.SqlConnection](#),
transaction As [System.Data.SqlClient.SqlTransaction](#))

En ambos casos, debe indicarse como parámetro el objeto conexión en uso, y una cadena de texto con el comando SQL que se quiere ejecutar contra la base de datos.

Ejemplo: El siguiente código muestra la obtención de un objeto comando sobre el proveedor de SQL Server a partir del método *CreateCommand* del objeto conexión.

VB
<pre> Shared Sub Main() Dim cadenaConexion = "Data Source=192.168.9.158;Initial Catalog=MULTAS;Persist Security Info=True;User ID=usuario;Password=ci psa" Dim conexion As SqlConnection = Nothing Dim comando As SqlCommand = Nothing Try ' Instanciación de la conexión conexion = New SqlConnection(cadenaConexion) conexion.Open() ' Apertura de la conexión ' Instanciación del comando comando = conexion.CreateCommand() Catch ex As SqlException Finally If Not conexion Is Nothing Then conexion.Close() ' Cierre de la conexión End If End Try End Sub </pre>

C#

```
static void Main(string[] args)
{
    String cadenaConexion = "Data Source=192.168.9.158;Initial Catalog=MULTAS;Persist Security Info=True;User ID=usuario;Password=cipsa";
    SqlConnection conexion = null;
    SqlCommand comando = null;
    try
    {
        // Instanciación de la conexión
        conexion = new SqlConnection(cadenaConexion);
        conexion.Open(); // Apertura de conexión
        // Instanciación del comando
        comando = conexion.CreateCommand();

    }
    catch (SqlException exc)
    {
        // Excepción
    }
    finally
    {
        if (conexion != null)
        {
            conexion.Close(); // Cierre de conexión
        }
    }
}
```




3.1.- Tipos de Comandos

Un comando permite enviar a la base de datos tres tipos distintos de órdenes.

- **Sentencias SQL** → Son órdenes en forma de sentencias SQL que permiten operaciones tanto de consulta como de manipulación tanto de los datos como de la estructura de la propia base de datos.
- **Procedimientos Almacenados** → Son órdenes de ejecución de pequeños programas escritos en SQL que llevan acabo operaciones más complejas de lo que es capaz una simple sentencia SQL.
- **Tabla** → Es una orden simple que solicita a la base de datos la información de todos los registros de una tabla con todos sus campos.

En función del tipo de orden debe asignarse un valor a la propiedad **CommandType** del objeto comando. Esta propiedad admite únicamente valores del tipo enumerado *System.Data.CommandType*.

Los valores posibles son los siguientes:

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

3.2.- Comandos con Sentencias SQL

Los comandos con sentencias SQL son útiles para operaciones de consulta o modificación de datos relativamente sencillas.

La sentencia SQL se indica en forma de cadena de caracteres que se asigna a la propiedad **CommandText** del objeto comando:

```
' Instanciación del comando
comando = conexion.CreateCommand()
comando.CommandText = "SELECT COUNT(DNI) FROM CONDUCTOR"
```

No es necesario indicar ningún valor a la propiedad **CommandType**, ya que de manera predeterminada tiene el valor *Text* que permite el envío de sentencias SQL.

3.3.- Comandos Parametrizados

Los comandos parametrizados incluyen una serie de parámetros que permiten reutilizar sentencias SQL con parámetros anidados. El uso de los parámetros permite reutilizar una misma sentencia indicando valores distintos a los mismos.

Los parámetros se declaran en la propia sentencia SQL como identificadores precedidos de una '@':

```
SELECT COUNT(DNI) FROM CONDUCTOR WHERE EDAD > @edad
```

La sentencia SQL indicada devuelve la cantidad de personas registradas en la tabla CONDUCTOR cuya edad es superior a la que se indique como valor del parámetro @edad.

Antes de ejecutar un comando parametrizado es preciso dar valor a todos y cada uno de los parámetros presentes en la sentencia SQL. Para ello se emplean objetos parámetro creados a partir de la clase correspondiente al proveedor utilizado:

SqlParameter
OracleParameter
OdbcParameter
OleDbParameter

Todas estas clases tienen métodos y propiedades comunes heredadas de la clase *System.Data.Common.DbParameter*.

Cada parámetro posee tres propiedades fundamentales:

- El **identificador** del parámetro al que representa en la sentencia SQL.
- El **tipo de dato** en la base de datos asociado al parámetro
- El **valor** asociado al parámetro, que debe ser de un tipo compatible con el tipo correspondiente a la base de datos.

El identificador y el tipo pueden indicarse a través del constructor:

```
Public Sub New(parameterName As String, dbType As System.Data.DbType)
```

```
Public Sub New(parameterName As String, dbType As System.Data.DbType, size As Integer)
```

El primer parámetro (*parameterName*) permite indicar el identificador del parámetro en la sentencia. El segundo parámetro (*dbType*) indica el tipo de dato en la base de datos de destino asociado al parámetro. En el caso de parámetros asociados a cadenas de caracteres, el tercer parámetro (*size*) indica la longitud en caracteres.

Los tipos de datos propios de cada proveedor están recogidos en los tipos enumerados *OleDbType*, *SqlDbType*, *OdbcDbType*, *OracleDbType* definidos en sus respectivos espacios de nombres.

El valor asociado al parámetro para la consulta se asigna a la propiedad Value. El valor asignado al parámetro debe ser de un tipo .NET compatible con el tipo de la base de datos al que va dirigido el parámetro.

Para terminar, todos y cada uno de los parámetros que participan en la sentencia deben registrarse en la colección de parámetros del objeto comando empleando la propiedad *Parameters* y el método miembro *Add*.

Ejemplo:

```
SELECT COUNT(DNI) FROM CONDUCTOR WHERE EDAD > @edad
```

Si la columna EDAD está declarada en la base de datos como tipo numérico simple (INT); el parámetro @edad debe definirse como:

VB
<pre>' Instanciación del comando comando = conexion.CreateCommand() comando.CommandText = "SELECT COUNT(DNI) FROM CONDUCTOR WHERE EDAD > @edad" ' Instanciación del parámetro para la consulta Dim parametro As New SqlParameter("@edad", SqlDbType.Int) parametro.Value = 25 comando.Parameters.Add(parametro)</pre>

C#

```
// Instanciación del comando
comando = conexion.CreateCommand();
comando.CommandText = "SELECT COUNT( DNI ) FROM CONDUCTOR WHERE EDAD > @edad";

// Instanciación del parámetro para la consulta
SqlParameter parametro = new SqlParameter("@edad", System.Data.SqlDbType.Int)
parametro.Value = 25;
// Registro del parámetro
comando.Parameters.Add( parametro );
```

Según este ejemplo, la consulta devolvería el total de conductores registrados en la tabla CONDUCTOR cuya edad supere los 25 años.

Ejemplo:

```
SELECT * FROM CONDUCTOR WHERE DNI = @dni
```

Si la columna DNI está declarada en la base de datos como tipo CHAR(9), el parámetro correspondiente a @dni debe declararse como:

VB

```
' Instanciación del comando
comando = conexion.CreateCommand()
comando.CommandText = "SELECT * FROM CONDUCTOR WHERE DNI = @dni"

' Instanciación del parámetro para la consulta
Dim parametro As New SqlParameter("@dni", SqlDbType.Char, 9)
parametro.Value = "39040945-K"
comando.Parameters.Add(parametro)
```

C#

```
// Instanciación del comando
comando = conexion.CreateCommand();
comando.CommandText = "SELECT COUNT( DNI ) FROM CONDUCTOR WHERE DNI = @dni";

// Instanciación del parámetro para la consulta
SqlParameter parametro = new SqlParameter("@dni", System.Data.SqlDbType.NChar, 9);
parametro.Value = "39040945-K";
comando.Parameters.Add( parametro );
```

Según este ejemplo, la consulta devolvería todos los campos del conductor registrado en la tabla CONDUCTOR cuyo DNI es 39040945-K.

3.3.1.- Tablas de correlación de tipos

Tabla de equiparación de tipos de SQL Server con tipos de .NET. Los tipos de datos para el proveedor de SQL Server están enumerados en el tipo *SqlDbType* requerido por los objetos *SqlParameter*.

SQL Server Database Engine type	.NET Framework type	SqDbType enumeration
bigint	Int64	BigInt
binary	Byte[]	VarBinary
bit	Boolean	Bit
char	String Char[]	Char
date (SQL Server 2008 only)	DateTime	Date
datetime	DateTime	DateTime
datetime (SQL Server 2008 only)	DateTime	DateTime2
datetimeoffset (SQL Server 2008 only)	DateTimeOffset	DateTimeOffset
decimal	Decimal	Decimal
FILESTREAM attribute (varbinary(max))	Byte[]	VarBinary
float	Double	Float
image	Byte[]	Binary
int	Int32	Int
money	Decimal	Money
nchar	String Char[]	NChar
ntext	String Char[]	NText
numeric	Decimal	Decimal
nvarchar	String Char[]	NVarChar
real	Single	Real
rowversion	Byte[]	Timestamp
smalldatetime	DateTime	DateTime
smallint	Int16	SmallInt
smallmoney	Decimal	SmallMoney
sql_variant	Object *	Variant
text	String Char[]	Text
time (SQL Server 2008 only)	TimeSpan	Time
timestamp	Byte[]	Timestamp
tinyint	Byte	TinyInt
uniqueidentifier	Guid	UniqueIdentifier
varbinary	Byte[]	VarBinary
varchar	String Char[]	VarChar
xml	Xml	Xml

Tabla con la correspondencia de tipos de datos entre .NET y SQL Server

Tabla de equiparación de tipos de datos de proveedor OleDb para bases de datos Microsoft Access, con tipos de datos de .NET. Los tipos de datos para proveedores OleDb están enumerados en el tipo *OleDbType* requerido por los objetos *OleDbParameter*.

nombre de tipo de acceso	tipo de datos de base de datos	Tipo OLE DB	tipo de .NET framework	nombre de miembro
Texto	VarChar	DBTYPE_WSTR	System.String	OleDbType.VarChar
Memorando	LongVarChar	DBTYPE_WSTR	System.String	OleDbType.LongVarChar
Número: Byte	UnsignedTinyInt	DBTYPE_UI1	System.Byte	OleDbType.UnsignedTinyInt
Sí/no	booleano	DBTYPE_BOOL	System.Boolean	OleDbType.Boolean
Fecha y hora	DateTime	DBTYPE_DATE	System.DateTime	OleDbType.Date
Moneda	decimal	DBTYPE_NUMERIC	System.Decimal	OleDbType.Numeric
Número: decimal	decimal	DBTYPE_NUMERIC	System.Decimal	OleDbType.Numeric
Número: Double	doble	DBTYPE_R8	System.Double	OleDbType.Double
Autonumérico (ID. de réplica)	GUID	DBTYPE_GUID	System.GUID	OleDbType.Guid
Número: (ID. de réplica)	GUID	DBTYPE_GUID	System.GUID	OleDbType.Guid
Autonumérico (entero largo)	entero	DBTYPE_I4	System.Int32	OleDbType.Integer
Número: (Entero largo)	entero	DBTYPE_I4	System.Int32	OleDbType.Integer
Objeto OLE	LongVarBinary	DBTYPE_BYTES	Matriz de System.Byte	OleDbType.LongVarBinary
Número: único	único	DBTYPE_R4	System.Single	OleDbType.Single
Número: entero	SmallInt	DBTYPE_I2	System.Int16	OleDbType.SmallInt
Binario	VarBinary *	DBTYPE_BYTES	Matriz de System.Byte	OleDbType.Binary

Tabla con la correspondencia de tipos de datos entre .NET y Proveedor OleDb

3.4.- Comandos de llamada a procedimientos almacenados.

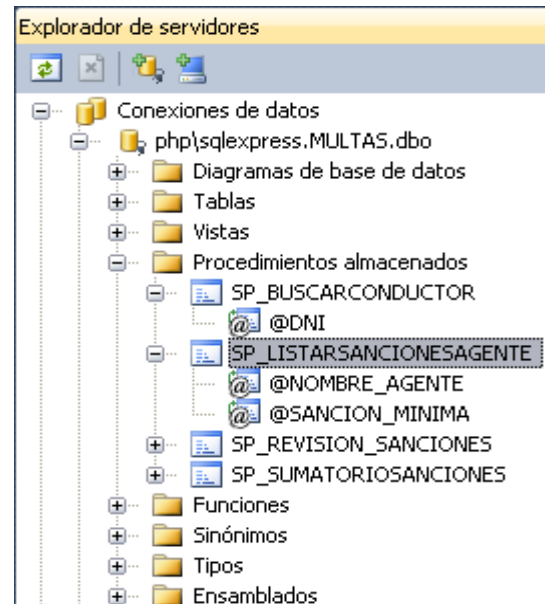
Un procedimiento almacenado es un conjunto de sentencias SQL almacenadas en el servidor de base de datos que realizan una tarea compleja. Cada procedimiento almacenado consta de un identificador que permite llamarlo, y un conjunto opcional de parámetros necesarios para realizar su tarea.

3.4.1.- Examen de procedimientos almacenados

Los procedimientos almacenados existentes en una la base de datos pueden examinarse desde la ventana del *Explorador de Servidores* bajo la carpeta "Procedimientos almacenados."

Por cada procedimiento almacenado se indica su nombre y el conjunto de parámetros que requieren para su funcionamiento:

Puede consultarse el código de un procedimiento almacenado seleccionando la opción "Abrir" del menú contextual que aparece al hacer clic con el botón derecho sobre su identificador:



Ejemplo: Código del procedimiento almacenado SP_LISTARSANCIONAGENTE que retorna los datos del agente (tabla AGENTE) con el nombre indicado en el parámetro @NOMBRE_AGENTE, junto con de las sanciones que ha impuesto (tabla SANCION) con coste superior al indicado por el parámetro @SANCION_MINIMA.

```
-- =====
-- Author:          <Author, , Name>
-- Create date:    <Create Date, , >
-- Description:    <Description, , >
-- =====
ALTER PROCEDURE SP_LISTARSANCIONESAGENTE
    @NOMBRE_AGENTE VARCHAR(30),
    @SANCION_MINIMA MONEY
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for procedure here
    SELECT * FROM SANCION INNER JOIN AGENTE
        ON SANCION.CODIGO_AGENTE = AGENTE.CODIGO
        WHERE AGENTE.NOMBRE = @NOMBRE_AGENTE
        AND
            SANCION.COSTE_SANCION > @SANCION_MINIMA
END
```

Código de procedimiento almacenado escrito en Transact-SQL para Microsoft SQL Server 2008

3.4.2.- Prueba de procedimientos almacenados.

Un procedimiento almacenado puede ejecutarse manualmente desde la ventana del Explorador de Servidores para probar su funcionamiento y los resultados que devuelve. Para ello debe hacerse clic con el botón derecho sobre su identificador y seleccionando a continuación la opción “Ejecutar” del menú contextual:

Visual Studio muestra entonces una ventana solicitando tantos valores como parámetros declara el procedimiento almacenado. Por cada parámetro se indica su identificador, el tipo de dato que requiere y si es de entrada o salida.



Los resultados devueltos por la ejecución del procedimiento almacenado pueden verse en la ventana de resultados del Visual Studio.

3.4.3.- Comando llamada a procedimiento almacenado

Para ejecutar un procedimiento almacenado a través de un objeto conexión debe asignarse su identificador a la propiedad **CommandText**, y asignar el valor *CommandType.StoredProcedure* a la propiedad *CommandType*:

```
' Instanciación del comando
comando = conexion.CreateCommand()

' Configuración llamada a procedimiento almacenado
comando.CommandType = CommandType.StoredProcedure
comando.CommandText = "SP_LISTARSANCIONESAGENTE"
```

A continuación deben registrarse todos los parámetros necesarios indicando los mismos identificadores que los declarados por el procedimiento almacenado.

```
' Creación de parámetros para el procedimiento almacenado
Dim paramNombreAgente As New SqlParameter("@NOMBRE_AGENTE", SqlDbType.VarChar, 30)
paramNombreAgente.Value = "Manolo Escobar" ← Valor de tipo String

Dim paramSancionMinima As New SqlParameter("@SANCION_MINIMA", SqlDbType.Money)
paramSancionMinima.Value = 100.00 ← Valor de tipo Decimal

' Registro de parámetros.
comando.Parameters.Add(paramNombreAgente)
comando.Parameters.Add(paramSancionMinima)
```

El código completo en C# es:

```
// Instanciación del comando
comando = conexion.CreateCommand();
comando.CommandType = System.Data.CommandType.StoredProcedure;
comando.CommandText = "SP_LI STARSANCI ONEAGENTE";

// Declaración y registro de parámetros de llamada
SqlParameter param_nombreAgente = new SqlParameter("@NOMBRE_AGENTE",
                                                    System.Data.SqlDbType.VarChar, 30);
param_nombreAgente.Value = "Manolo Escobar";

SqlParameter param_sanccionMulta = new SqlParameter("@SANCIÓN_MULTA",
                                                    System.Data.SqlDbType.Money);
param_sanccionMulta.Value = 100.00;

// Registro de parámetros
comando.Parameters.Add(param_nombreAgente);
comando.Parameters.Add(param_sanccionMulta);
```

3.5.- Ejecución de comandos

La ejecución de un objeto comando se lleva a cabo teniendo en cuenta el tipo de sentencia SQL indicada y los resultados que produce.

3.5.1.- Ejecución de sentencias escalares

Las operaciones escalares son sentencias SQL que devuelven un único valor. Este valor suele obtenerse como resultado de recuentos u operaciones matemáticas efectuadas sobre la totalidad o parte de los valores de una o varias tablas en la base de datos:

Ejemplo: Devuelve el total de conductores registrados en la tabla CONDUCTOR

```
SELECT COUNT( DNI ) FROM CONDUCTOR
```

Ejemplo: Devuelve el sumatorio de todas las sanciones registradas en la tabla SANCION

```
SELECT SUM( COSTE_SANCION ) FROM SANCION
```

Ejemplo: Devuelve la media de edad de todos los conductores en la tabla CONDUCTOR

```
SELECT AVG( EDAD ) FROM CONDUCTOR
```

Los comandos con este tipo de sentencias se ejecutan mediante el método **ExecuteScalar** del objeto comando:

```
Public Overrides Function ExecuteScalar() As Object
```

El método retorna el valor devuelto por la ejecución de la sentencia. El tipo del valor de retorno es **Object**, dado que puede retornarse un valor cualquier tipo desde la base de datos.

Ejemplo: El siguiente código obtiene y muestra el recuento de conductores registrados en la tabla CONDUCTOR cuya edad es superior a 25 años:

VB

```
Shared Sub Main()
    Dim cadenaConexion = ""
    Dim conexion As SqlConnection = Nothing
    Dim comando As SqlCommand = Nothing
    Try
        ' Instanciación de la conexión
        conexion = New SqlConnection(cadenaConexion)
        conexion.Open() ' Apertura de la conexión

        ' Instanciación del comando
        comando = conexion.CreateCommand()
        comando.CommandText = "SELECT COUNT(DNI) FROM CONDUCTOR WHERE EDAD > @edad"

        ' Instanciación del parámetro para la consulta
        Dim parametro As New SqlParameter("@edad", SqlDbType.Int)
        parametro.Value = 25
        comando.Parameters.Add(parametro)

        ' Obtención del valor de retorno
        Dim valor As Integer = CType(comando.ExecuteScalar(), Integer)
        Console.WriteLine("El número de conductores mayores de 25 es " + valor)

    Catch ex As SqlException
        ' DETECCIÓN E INFORME DEL ERROR.
    Finally
        If Not conexion Is Nothing Then
            conexion.Close() ' Cierre de la conexión
        End If
    End Try
End Sub
```

C#

```
static void Main(string[] args)
{
    String cadenaConexion = "";
    SqlConnection conexion = null;
    SqlCommand comando = null;
    try
    {
        // Instanciación de la conexión
        conexion = new SqlConnection(cadenaConexion);
        conexion.Open(); // Apertura de conexión
        // Instanciación del comando
        comando = conexion.CreateCommand();
        comando.CommandText = "SELECT COUNT( DNI ) FROM CONDUCTOR WHERE EDAD > @edad";

        // Instanciación del parámetro para la consulta
        SqlParameter parametro = new SqlParameter("@edad", System.Data.SqlDbType.Int);
        parametro.Value = 25;
        // Registro del parámetro
        comando.Parameters.Add( parametro );

        // Ejecución de la consulta y obtención del valor resultante
        int valor = (int)comando.ExecuteScalar();
        Console.WriteLine("El número de conductores mayores de 25 es " + valor);

    }
    catch (SqlException exc)
    {
        // Control de errores
    }
    finally
    {
        if (conexion != null)
        {
            conexion.Close(); // Cierre de conexión y liberación de memoria.
            conexion.Dispose();
        }
    }
}
```

3.5.2.- Ejecución de sentencias sin retorno de datos

Las sentencias SQL de modificación de datos (*INSERT*, *UPDATE*, *DELETE*), conforman un caso especial de comando puesto que no retornan realmente ningún valor. Para la ejecución de estas sentencias se emplea el método ***ExecuteNonQuery*** del objeto comando:

Public Overrides Function ***ExecuteNonQuery()*** As [*Integer*](#)

El método retorna un valor entero que indica el número total de registros que han sido modificados por la sentencia ejecutada. Si se trata de sentencias de modificación (*UPDATE*), o eliminación (*DELETE*), el valor retornado indica cuántos registros han sido modificados o eliminados por la sentencia ejecutada.

Ejemplo: El siguiente código ejecuta una sentencia de eliminación de todos los conductores presentes en la tabla CONDUCTOR cuyo campo edad es inferior a 24 años. La ejecución del comando con el método *ExecuteNonQuery* devuelve el número de registros eliminados.

```
VB
Shared Sub Main()
    Dim cadenaConexion = "Data Source=192.168.9.158;Initial Catalog=MULTAS;Persist
Security Info=True;User ID=usuario;Password=cipsa"
    Dim conexion As SqlConnection = Nothing
    Dim comando As SqlCommand = Nothing
    Try
        ' Instanciación de la conexión
        conexion = New SqlConnection(cadenaConexion)
        conexion.Open() ' Apertura de la conexión

        comando = conexion.CreateCommand()
        comando.CommandText = "DELETE CONDUCTOR WHERE EDAD < @edad"
        ' Creación y registro de parámetros.
        comando.Parameters.Add(New SqlParameter("@edad", 24))

        ' Ejecución y obtención de num. de registros modificados
        Dim regs As Integer = comando.ExecuteNonQuery()
        Console.WriteLine("Se han eliminado " + regs.ToString() + " registros.")
    Catch ex As SqlException
        ' DETECCION E INFORME DEL ERROR.
    Finally
        If Not conexion Is Nothing Then
            conexion.Close() ' Cierre de la conexión
        End If
    End Try
    Console.ReadKey()
End Sub
```

C#

```

static void Main(string[] args)
{
    String cadenaConexion = "Data Source=192.168.9.158;Initial Catalog=MULTAS;Persist
Security Info=True;User ID=usuario;Password=cipsa";
    SqlConnection conexion = null;
    SqlCommand comando = null;
    try
    {
        // Instanciación de la conexión
        conexion = new SqlConnection(cadenaConexion);
        conexion.Open(); // Apertura de conexión
        // Instanciación del comando
        comando = conexion.CreateCommand();
        comando.CommandText = "DELETE CONDUCTOR WHERE EDAD < @edad";

        // Instanciación del parámetro para la consulta
        SqlParameter parametro = new SqlParameter("@edad", 24);
        // Registro del parámetro
        comando.Parameters.Add( parametro );

        // Ejecución y obtención del número de registros modificados.
        Int regs = comando.ExecuteNonQuery();
        Console.WriteLine("Se han eliminado " + regs.ToString() + " registros.");
    }
    catch (SqlException exc)
    {
        // Control de errores
        Console.WriteLine("Se ha producido el error {0} - {1}",
exc.GetType().ToString(), exc.Message);
    }
    finally
    {
        if (conexion != null)
        {
            // Liberación de memoria.
            conexion.Close(); // Cierre de conexión y
            conexion.Dispose();
        }
    }
}

```

4.- Modo Conectado

4.1.- El objeto cursor (*DataReader*)

La obtención de datos desde la base de datos en modo conectado emplea *Cursores*. El acceso en modo conectado se caracteriza por requerir una conexión constantemente abierta con la base de datos durante su uso. Se emplea en consultas que retornan un gran número de registros que sólo se desean recorrer sin introducir modificación alguna.

ADO.NET representa un cursor mediante clases específicas para cada proveedor de datos en su respectivo espacio de nombres. Todas ellas comparten métodos y propiedades comunes de la clase padre base *System.Data.Common.DbDataReader*.

- ***SqlDataReader***
- ***OracleDataReader***
- ***OdbcDataReader***
- ***OleDbDataReader***

4.2.- Obtención del DataReader.

Un objeto *DataReader* representa un *cursor* que apunta en cada instante a un registro dentro del conjunto de resultados devueltos por una consulta SQL o procedimiento almacenado de consulta. Este objeto se obtiene indistintamente al ejecutar el método ***ExecuteReader*** sobre un objeto comando.

VB	
	<pre> ' Instanciación de la conexión conexión = New SqlConnection(cadenaConexión) conexión.Open() ' Apertura de la conexión ' Instanciación del comando comando = conexión.CreateCommand() comando.CommandText = "SELECT * FROM CONDUCTOR WHERE EDAD > @edad" ' Instanciación del parámetro para la consulta Dim parametro As New SqlParameter("@edad", SqlDbType.Int) parametro.Value = 25 comando.Parameters.Add(parametro) ' Obtención del DataReader Dim reader As SqlDataReader = comando.ExecuteReader()</pre>
C#	
	<pre> // Instanciación de la conexión conexión = new SqlConnection(cadenaConexión); conexión.Open(); // Apertura de conexión // Instanciación del comando comando = conexión.CreateCommand(); comando.CommandText = "SELECT * FROM CONDUCTOR WHERE EDAD > @edad"; // Instanciación del parámetro para la consulta SqlParameter parametro = new SqlParameter("@edad", System.Data.SqlDbType.Int); parametro.Value = 25; // Registro del parámetro comando.Parameters.Add(parametro); // Obtención del DataReader SqlDataReader reader = comando.ExecuteReader();</pre>

El cursor solo posee en cada momento acceso a un único registro del conjunto total de resultados. Para pasar al siguiente registro es necesario llamar al método **Read** del objeto `DataReader`:

Public Overrides Function **Read**() As [Boolean](#)

Este método retorna un valor lógico cierto en caso de existir un registro, o falso en caso no haber más registros.

El objeto `DataReader` también dispone del método **HasRows** que retorna un valor lógico cierto si la consulta ha devuelto uno o varios resultados, o falso en caso de no retornar ningún resultado:

VB

```
' Obtencion del DataReader
Dim reader As SqlDataReader = comando.ExecuteReader()
' Comprobacion de existencia de resultados
If reader.HasRows Then
    ' Bucle de recorrido de resultados
    Do While reader.Read
        ' Lectura de campos de registro apuntado.
        Console.WriteLine("resultado")
    Loop
Else
    Console.WriteLine("No hay resultados.")
End If
```

C#

```
// Obtencion del DataReader
SqlDataReader reader = comando.ExecuteReader();
// Comprobacion de existencia de resultados
if (reader.HasRows)
{
    // Bucle de recorrido de resultados
    while (reader.Read())
    {
        Console.WriteLine("resultado");
    }
}
else
{
    Console.WriteLine("No hay resultados.");
}
```

4.3.- Lectura de campos

Los valores de los campos del registro apuntado por el cursor en cada momento pueden obtenerse empleando los métodos *GetXXXX* del objeto *DataReader*. Cada método retorna el valor de un campo del registro apuntado por el *datareader*.

El método debe elegirse en función del tipo de dato que se sabe contenido en el campo. Los métodos requieren como parámetro el índice de base 0 del campo a recuperar.

	GetBoolean	Obtiene el valor de la columna especificada como tipo Boolean.
	GetByte	Obtiene el valor de la columna especificada como byte.
	GetBytes	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> .
	GetChar	Obtiene el valor de la columna especificada como un único carácter.
	GetChars	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> .
	GetData	Devuelve un objeto DbDataReader para el ordinal de columna solicitado.
	GetDataTypeName	Obtiene el nombre del tipo de datos de la columna especificada.
	GetDateTime	Obtiene el valor de la columna especificada como un objeto DateTime .
	GetDecimal	Obtiene el valor de la columna especificada como un objeto Decimal .
	GetDouble	Obtiene el valor de la columna especificada como un número de punto flotante de precisión doble.
	GetEnumerator	Devuelve una interfaz IEnumerator que se puede utilizar para recorrer en iteración las filas en el lector de datos.
	GetFieldType	Obtiene el tipo de datos de la columna especificada.
	GetFloat	Obtiene el valor de la columna especificada como un número de punto flotante de precisión simple.
	GetGuid	Obtiene el valor de la columna especificada como un identificador global único (GUID).
	GetHashCode	Sirve como función hash para un tipo concreto. GetHashCode es apropiado para su utilización en algoritmos de hash y en estructuras de datos como las tablas hash. (Se hereda de Object).
	GetInt16	Obtiene el valor de la columna especificada como un entero de 16 bits con signo.
	GetInt32	Obtiene el valor de la columna especificada como un entero de 32 bits con signo.
	GetInt64	Obtiene el valor de la columna especificada como un entero de 64 bits con signo.

Muestra de métodos principales de obtención de datos miembros de los objetos DataReader

Ejemplo: Sea una tabla CONDUCTOR cuyos campos están declarados con los siguientes tipos de datos observables mediante la ventana del Explorador de Servidores del Visual Studio:

Nombre de columna	Tipo de datos	Permitir valores NULL
DNI	char(9)	<input type="checkbox"/>
NOMBRE	varchar(10)	<input checked="" type="checkbox"/>
APELLIDOS	varchar(20)	<input checked="" type="checkbox"/>
DIRCCION	varchar(50)	<input checked="" type="checkbox"/>
MATRICULA	char(8)	<input type="checkbox"/>
EDAD	int	<input checked="" type="checkbox"/>
LICENCIA	char(1)	<input checked="" type="checkbox"/>

El siguiente código recupera los campos DNI, NOMBRE, APELLIDOS y EDAD de los conductores cuya edad sea superior a 25 años, mostrando los resultados obtenidos por pantalla en modo conectado.

VB

```

Dim cadenaConexion = "Data Source=192.168.9.158;Initial Catalog=MULTAS;Persist
Security Info=True;User ID=usuario;Password=clpsa"
Dim conexion As SqlConnection = Nothing
Dim comando As SqlCommand = Nothing
Dim reader As SqlDataReader = Nothing
Try
    ' Instanciación de la conexión
    conexion = New SqlConnection(cadenaConexion)
    conexion.Open() ' Apertura de la conexión

    ' Instanciación del comando
    comando = conexion.CreateCommand()
    comando.CommandText = "SELECT DNI, NOMBRE, APELLIDOS, EDAD FROM CONDUCTOR WHERE
EDAD > @edad"

    ' Instanciación del parámetro para la consulta
    Dim parametro As New SqlParameter("@edad", SqlDbType.Int)
    parametro.Value = 25
    comando.Parameters.Add(parametro)

    ' Obtención del DataReader
    reader = comando.ExecuteReader()
    ' Comprobación de existencia de resultados
    If reader.HasRows Then
        ' Bucle de recorrido de resultados
        Dim dni As String
        Dim nombre As String
        Dim apellidos As String
        Dim edad As Integer
        Do While reader.Read
            ' Lectura de campos de registro apuntado.
            dni = reader.GetString(0) ' Índice 0 -> campo DNI
            nombre = reader.GetString(1) ' Índice 1 -> campo NOMBRE
            apellidos = reader.GetString(2) ' Índice 2 -> campo APELLIDOS
            edad = reader.GetInt32(3) ' Índice 3 -> campo EDAD
            ' Visualización de datos.
            Console.WriteLine("{0}" + vbTab + "{1}" + vbTab + "{2}" + vbTab + "{3}",
-
                                dni, nombre, apellidos, edad)
        Loop
    Else
        Console.WriteLine("No hay resultados.")
    End If
Catch ex As SqlException
    ' DETECCIÓN E INFORME DEL ERROR.
Finally
    If Not reader Is Nothing Then ' Cierre de cursor
        reader.Close()
    End If
    If Not conexion Is Nothing Then ' Cierre de la conexión
        conexion.Close()
    End If
End Try

```

C#

```

static void Main(string[] args)
{
    String cadenaConexion = "Data Source=192.168.9.158;Initial Catalog=MULTAS;Persist Security Info=True;User ID=usuario;Password=cipsa";
    SqlConnection conexion = null;
    SqlCommand comando = null;
    SqlDataReader reader = null;
    try
    {
        // Instanciación de la conexión
        conexion = new SqlConnection(cadenaConexion);
        conexion.Open(); // Apertura de conexión
        // Instanciación del comando
        comando = conexion.CreateCommand();
        comando.CommandText = "SELECT * FROM CONDUCTOR WHERE EDAD > @edad";

        // Instanciación del parámetro para la consulta
        SqlParameter parametro = new SqlParameter("@edad", System.Data.SqlDbType.Int);
        parametro.Value = 25;
        // Registro del parámetro
        comando.Parameters.Add(parametro);
        // Obtención del DataReader
        reader = comando.ExecuteReader();
        // Comprobación de existencia de resultados
        if (reader.HasRows)
        {
            String dni;
            String nombre;
            String apellidos;
            Int32? edad; // Int32? -> Tipo entero que admite valores NULL
            while (reader.Read())
            {
                // Lectura del registro
                dni = reader["DNI"] as String;
                nombre = reader["NOMBRE"] as String;
                apellidos = reader["APELLIDOS"] as String;
                edad = reader["EDAD"] as Int32?;
                Console.WriteLine("{0}\t{1}\t{2}\t{3}", dni, nombre, apellidos, edad);
            }
        }
        else
        {
            Console.WriteLine("No hay resultados.");
        }
    }
    catch (SqlException exc)
    {
        // Control de errores
        Console.WriteLine("Se ha producido el error {0} - {1}",
            exc.GetType().ToString(), exc.Message);
    }
    finally
    {
        if (reader != null) reader.Close(); // Cerrado del cursor
        if (conexion != null)
        {
            conexion.Close(); // Cierre de conexión y liberación de memoria.
            conexion.Dispose();
        }
    }
}

```

Otra manera alternativa de obtener el valor de cada campo es utilizarlo el objeto *DataReader* como si se tratase de una matriz de valores.

El valor de cada campo se obtiene indicando el nombre de la columna indicado en la sentencia SQL correspondiente como índice. El tipo retornado es Object, por lo que debe realizarse la conversión al tipo de dato correspondiente mediante la función **Ctype**:

```
' Obtencion del DataReader
reader = comando.ExecuteReader()
' Comprobacion de existencia de resultados
If reader.HasRows Then
' Bucle de recorrido de resultados
Dim dni As String
Dim nombre As String
Dim apellidos As String
Dim edad As Integer
Do While reader.Read
' Lectura de campos de registro apuntado.
dni = CType(reader("DNI"), String)
nombre = CType(reader("NOMBRE"), String)
apellidos = CType(reader("APELLIDOS"), String)
edad = CType(reader("EDAD"), Integer)
Console.WriteLine("{0}" + vbTab + "{1}" + vbTab + "{2}" + vbTab + "{3}", _
dni, nombre, apellidos, edad)
Loop
Else
Console.WriteLine("No hay resultados.")
End If
```

En el caso de C# se realiza la conversión empleando el operador **as**:

```
String dni;
String nombre;
String apellidos;
Int32? edad; // Int32? -> Tipo entero que admite valores NULL
while (reader.Read())
{
// Lectura del registro
dni = reader["DNI"] as String;
nombre = reader["NOMBRE"] as String;
apellidos = reader["APELLIDOS"] as String;
edad = reader["EDAD"] as Int32?;
Console.WriteLine("{0}\t{1}\t{2}\t{3}", dni, nombre, apellidos, edad);
}
```

El resultado mostrado en ambos casos sería:

34567543N	VANESSA	GONZÁLEZ PRIETO	32
34890234S	Mª CARMEN	LÓPEZ HERRERA	35
45123456C	DAVID	ESTÉVEZ MOLINA	39
45345220B	JESÚS	GONZÁLEZ UÁZQUEZ	41
54234000W	JESÚS	GÓMEZ BAENA	53
56900001M	ÓSCAR	ALCÁNTARA BAENA	48
65123111L	ANA	CRUZ RIAÑO	45
90111234K	Mª CARMEN	BENITO REODRIGUEZ	37

4.4.- Cerrado del cursor

El cursor usa de forma exclusiva el objeto conexión no permitiendo la ejecución de ningún otro comando mientras esté abierto. Esto es debido a que el objeto DataReader recupera directamente desde la base de datos cada registro al llamarse a su método Read, obteniendo así siempre los datos más recientes.

Cuando se emplea un cursor, la conexión no debe cerrarse mientras se utiliza el objeto DataReader correspondiente, ya que imposibilitaría recuperar más registros. De igual modo; una vez finalizado el uso del cursor, debe cerrarse llamando al método **Close()**.

El cerrado del cursor debe asegurarse incluso en caso de producirse errores. Para ello puede emplearse una estructura TRY-CATCH-FINALLY, o bien una estructura USING:

```
' Obtencion del DataReader
Using reader As SqlDataReader = comando.ExecuteReader()
    ' Comprobacion de existencia de resultados
    If reader.HasRows Then
        ' Bucle de recorrido de resultados
        Dim dni As String
        Dim nombre As String
        Dim apellidos As String
        Dim edad As Integer
        Do While reader.Read
            ' Lectura de campos de registro apuntado.
            dni = CType(reader("DNI"), String)
            nombre = CType(reader("NOMBRE"), String)
            apellidos = CType(reader("APELLIDOS"), String)
            edad = CType(reader("EDAD"), Integer)
            Console.WriteLine("{0}" + vbTab + "{1}" + vbTab + "{2}" + vbTab +
"{3}", dni, nombre, apellidos, edad)
        Loop
    Else
        Console.WriteLine("No hay resultados.")
    End If
End Using
```

El uso de la estructura Using asegura el cerrado y eliminación del cursor en el momento en que la ejecución abandone (bien sea de manera correcta, o a causa de una excepción), el bloque de código comprendido entre las sentencias **Using** y **End Using**

En C# el uso de la sentencia *using* es idéntico.

```
using( SqlDataReader reader = comando.ExecuteReader() {
    // Comprobacion de existencia de resultados
    if (reader.HasRows) {
        String dni;
        String nombre;
        String apellidos;
        Int32? edad; // Int32? -> Tipo entero que admite valores NULL
        while (reader.Read()) {
            // Lectura del registro
            dni = reader["DNI"] as String;
            nombre = reader["NOMBRE"] as String;
            apellidos = reader["APELLIDOS"] as String;
            edad = reader["EDAD"] as Int32?;
            Console.WriteLine("{0}\t{1}\t{2}\t{3}", dni, nombre, apellidos, edad);
        }
    }
    else {
        Console.WriteLine("No hay resultados.");
    }
}
```

5.- Modo Desconectado

El modo desconectado se basa en el manejo de la base de datos manteniendo la conexión con la base de datos abierta el menor tiempo posible. Para conseguirlo, los resultados de una consulta se transfieren a la memoria de la aplicación para cerrar la conexión tan pronto es posible. Una vez realizada la transferencia de datos, éstos se manejan en la memoria de la aplicación ajenos a la base de datos.

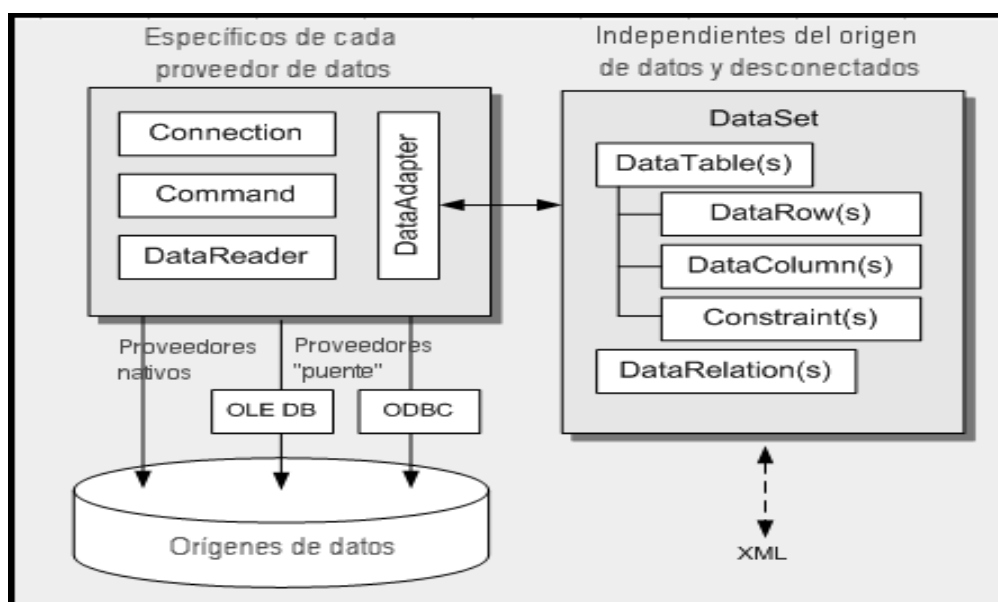
Para almacenar los resultados se emplean Conjuntos de Datos. Un conjunto de datos se corresponde con un objeto de la clase **System.Data.DataSet**.

5.1.- La clase DataSet.

La clase **DataSet** permite recuperar información de una base de datos sin necesidad de mantener la conexión abierta. Cada objeto DataSet constituye una *copia desconectada* de la información en la base de datos.

La clase DataSet no depende de ningún proveedor de datos ya que almacena la información procedente de una consulta independiente de la base de datos y el proveedor utilizado. Solo existe por tanto una única clase DataSet definida en el espacio de nombres **System.Data**.

El siguiente esquema muestra la arquitectura general de ADO.NET, y su relación con otros elementos de .NET tales como formularios de windows, páginas web... etc. Las clases en azul son aquellas que dependen del proveedor de datos empleado para acceder a la base de datos, y de las que se tienen versiones específicas. La clase DataSet representa por el contrario un conjunto de datos completamente independiente del proveedor utilizado.



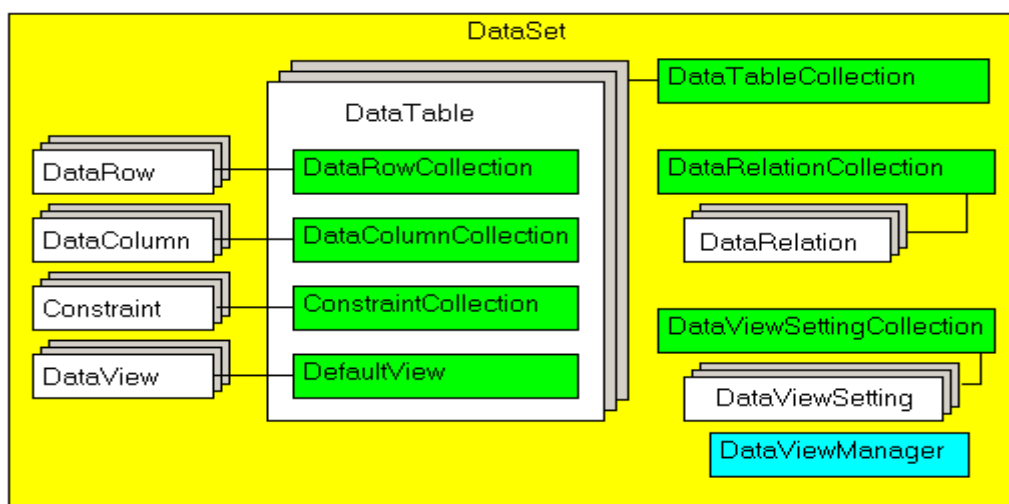
Estructura general de clases de ADO.NET

5.2.- Estructura de un conjunto de datos

Un objeto **DataSet** es en realidad una colección de diferentes objetos de distintos tipos que permite almacenar parte de los datos de una base de datos así como sus restricciones y relaciones. El fin es poder manipular los datos de igual modo que en la base de datos de donde proceden respetando su integridad.

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

- **DataTable:** Representa una tabla almacenadas en el **DataSet**. Son accesibles a través de la propiedad *Tables* de un objeto **DataSet**.
- **DataRow:** Representa un registro contenido en una tabla. Son accesibles a través de la propiedad *Rows* de un objeto **DataTable**.
- **DataColumn:** Representa una columnas que conforma una tabla. Son accesibles a través de la propiedad *Columns* de un objeto **DataTable**.
- **DataRelation:** Representa una relaciones entre dos tablas. Son accesibles a través de la propiedad *Relations* de un objeto **DataSet**.
- **UniqueConstraint, ForeignContrain:** Representan restricciones de clave primaria y clave externa vigentes en una tabla. Son accesibles a través de la propiedad *Constraints* de un objeto **DataTable**.



Jerarquía de clases asociadas a la clase DataSet

5.3.- El Adaptador de datos

El Adaptador de datos es un objeto dependiente del proveedor de datos que hace de intermediario entre la copia desconectada de datos almacenados en un objeto DataSet, y la base de datos de la que proceden. Las principales funciones de un adaptador de datos son:

- Obtener todos los datos resultantes de una consulta y almacenarlos en un objeto DataSet.
- Actualizar la información de la base de datos con las modificaciones realizadas en los datos desconectados de un objeto DataSet.

Cada proveedor de datos posee un tipo específico de adaptador de datos implementado en las correspondientes clases:

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

Todas las clases comparten un conjunto común de métodos y propiedades derivados de la clase padre ***System.Data.Common.DbDataAdapter***.

5.3.1.- Obtención de un Adaptador de datos

Un objeto adaptador de datos requiere para funcionar una conexión a la base de datos (o la cadena de conexión correspondiente), y la sentencia SQL a ejecutar (o objeto comando correspondiente). Algunos de los constructores definidos son:

Constructor 1 → Requiere dos cadenas de caracteres, una con la sentencia SQL y otra con la cadena de conexión correspondiente.

```
Public Sub New(selectCommandText As String, selectConnectionString As String)
```

Constructor 2 → Requiere una cadena de caracteres con la sentencia SQL a ejecutar y un objeto conexión previamente instanciado.

```
Public Sub New(selectCommandText As String,  
selectConnection As System.Data.SqlClient.SqlConnection)
```

Constructor 3 → Requiere un objeto comando previamente instanciado. La conexión va anidada dentro del objeto comando.

```
Public Sub New(selectCommand As System.Data.SqlClient.SqlCommand)
```

5.3.2.- Ejecución de una consulta mediante Adaptador de datos

La ejecución del objeto comando a través de un objeto adaptador de datos tiene como fin recoger los resultados en un objeto DataSet. Para ello se emplea el método **Fill**.

El método Fill ejecuta la sentencia o procedimiento almacenado del objeto comando, obtiene todos los resultados de la base de datos, y los almacena en el objeto *DataSet* o *DataTable* indicado. Las principales sobrecargas del método son:

Sobrecarga1 → El método *Fill* recibe como parámetro un objeto *DataTable* que será rellenado con las columnas y registros devueltos por la consulta:

```
Public Function Fill(dataTable As System.Data.DataTable) As Integer
```

Ejemplo: El siguiente código realiza una consulta contra la base de datos y almacena los resultados en un objeto *DataTable*.

VB	
C#	<pre> conexi on = New SqlConnection(cadenaConexi on) comando = conexi on.CreateCommand() comando.CommandText = "SELECT DNI, NOMBRE, APELLIDOS, EDAD FROM CONDUCTOR WHERE EDAD > @edad" Dim parametro As New SqlParameter("@edad", SqlDbType.Int) parametro.Value = 25 comando.Parameters.Add(parametro) ' Obtención de Adaptador de datos Dim adaptador As New SqlDataAdapter(comando) ' Creacion y relleno del conjunto de datos. Dim dt As New DataTable() Dim resul tado As Integer = adaptador.Fill(dt) </pre>

Sobrecarga2 → El método *Fill* admite otra sobrecarga que recibe como parámetro un objeto DataSet acompañado de una cadena de texto. Los resultados se almacenan en un objeto DataTable que se almacena en la colección de tablas del objeto DataSet.

La tabla generada recibe en su propiedad *TableName* la cadena del segundo parámetro para diferenciarla del resto de tablas en el DataSet.

```
Public Function Fill(dataSet As System.Data.DataSet, srcTable As String) As Integer
```

Ejemplo: El siguiente código realiza una consulta contra la base de datos y almacena los resultados dentro de un objeto DataTable con su propiedad *TableName* = "conductores", almacenada en el objeto DataSet pasado como parámetro.

VB
<pre>' Obtención de Adaptador de datos Dim adaptador As New SqlDataAdapter(comando) ' Creación y relleno del conjunto de datos. Dim ds As New DataSet() Dim resultado As Integer = adaptador.Fill(ds, "conductores")</pre>
C#
<pre>// Obtención del adaptador de datos adaptador = new SqlDataAdapter(comando); // Creación y relleno del conjunto de datos DataSet ds = new DataSet(); int regs = adaptador.Fill(ds, "conductores");</pre>

5.3.3.- Manejo de la conexión con Adaptador de datos

Un objeto DataAdapter puede abrir y cerrar la conexión con la base de datos al ejecutarse el método **Fill**. En ese caso, el adaptador de datos abre la conexión, recupera los datos devueltos por la consulta y cierra la conexión.

Ejemplo: La siguiente función **Query** encapsula la ejecución de una consulta SQL pasada como parámetro y devuelve los resultados almacenados en un objeto conjunto de datos (DataSet).

VB
<pre>Public Shared Function Query(ByVal SQL As String, ByVal source As String) As DataSet Dim cadenaConexion = "..." Dim comando As SqlCommand = Nothing Dim ds As New DataSet() ' Instanciación de la conexión Using conexion As New SqlConnection(cadenaConexion) ' Instanciación del comando comando = conexion.CreateCommand() comando.CommandText = SQL ' Obtención de Adaptador de datos Dim adaptador As New SqlDataAdapter(comando) ' Creación y relleno del conjunto de datos. adaptador.Fill(ds, source) End Using Return ds End Function</pre>

C#

```
public static DataSet Query(String SQL, String source)
{
    DataSet ds = new DataSet();
    String cadenaConexion = "Data Source=192.168.9.156;Initial Catalog=MULTAS;Persist
Security Info=True;User ID=usuario;Password=cipsa";
    using (SqlConnection conexion = new SqlConnection(cadenaConexion))
    {
        // Instanciación del comando
        SqlCommand comando = conexion.CreateCommand();
        comando.CommandText = SQL;
        // Instanciación del adaptador
        SqlDataAdapter adaptador = new SqlDataAdapter(comando);
        // Creación y relleno del conjunto de datos
        adaptador.Fill(ds, source);
    }
    return ds;
}
```

La estructura Using asegura el cierre y eliminación del objeto conexión en el momento que la ejecución abandone la sección de código con o sin errores. Puesto que la conexión no se abre manualmente (no hay llamadas al método Open), es el adaptador de datos el que abre y cierra la conexión al ejecutarse el método Fill.

5.4.-Lectura de datos de un objeto DataSet.

Los resultados de una consulta se almacenan siempre en un objeto DataTable. Para recuperar los datos debe obtenerse primero el objeto *DataTable* destinatario de los resultados empleando el identificador indicado en el método **Fill**.

Ejemplo:

VB

```
' Instanciación de la conexión
conexion = New SqlConnection(cadenaConexion)

' Instanciación del comando
comando = conexion.CreateCommand()
comando.CommandText = "SELECT DNI, NOMBRE, APELLIDOS, EDAD FROM CONDUCTOR WHERE
EDAD > @edad"
' Instanciación del parámetro para la consulta
Dim parametro As New SqlParameter("@edad", SqlDbType.Int)
parametro.Value = 25
comando.Parameters.Add(parametro)
' Obtención de Adaptador de datos
Dim adaptador As New SqlDataAdapter(comando)
' Creación y relleno del conjunto de datos.
Dim ds As New DataSet()
Dim resultado As Integer = adaptador.Fill(ds, "conductores")
```

C#

```
// Instanciación de la conexión
conexion = new SqlConnection(cadenaConexion);
comando = conexion.CreateCommand();
comando.CommandText = "SELECT DNI, NOMBRE, APELLIDOS, EDAD FROM CONDUCTOR
WHERE EDAD > @edad";
SqlParameter parametro = new SqlParameter("@edad",
System.Data.SqlDbType.Int);
parametro.Value = 25;
comando.Parameters.Add(parametro);
// Obtención del adaptador de datos
adaptador = new SqlDataAdapter(comando);
// Creación y relleno del conjunto de datos
DataSet ds = new DataSet();
Int regs = adaptador.Fill(ds, "conductores");
```

El objeto *DataTable* con los resultados de la consulta se obtiene a través de la colección **Tables** del objeto *DataSet*. La propiedad *Item* permite recuperar el objeto *DataTable* almacenado empleando como índice el identificador indicado en el método **Fill**.

```
' Obtencion del objeto DataTable con los resultados de dentro
' del objeto DataSet
Dim dt As DataTable = ds.Tables.Item("conductores")
```

Los registros almacenados en el objeto *DataTable* son accesibles a través de la propiedad **Rows**. Esta propiedad devuelve una colección de objetos *DataRow*, donde cada objeto se corresponde con un registro devuelto. La colección puede recorrerse empleando una estructura *foreach*.

```
For Each registro As DataRow In dt.Rows
Next
```

Cada objeto *DataRow* representa un registro. Los valores de los campos pueden recuperarse empleando la propiedad **Item** indicando como índice el nombre de la columna correspondiente. Hay que tener en cuenta, que el método retorna un tipo *Object* por lo que es necesario realiza conversión de tipos mediante la función *CType*.

VB

```
For Each registro As DataRow In dt.Rows
    dni = CType(registro.Item("DNI"), String)
    nombre = CType(registro.Item("NOMBRE"), String)
    apellidos = CType(registro.Item("APELLIDOS"), String)
    edad = CType(registro.Item("EDAD"), Integer)
    ' Visualización de datos.
    Console.WriteLine("{0}" + vbTab + "{1}" + vbTab + "{2}" + vbTab + "{3}",
        dni, nombre, apellidos, edad)
Next
```

En el caso de C# puede emplearse el operador **as** para convertir el tipo de dato retornado *Object* al deseado.

C#

```
int regs = adaptador.Fill(ds, "conductores");

String dni;
String nombre;
String apellidos;
Int32? edad;

DataTable dt = ds.Tables["conductores"];
foreach (DataRow reg in dt.Rows)
{
    dni = reg["DNI"] as String;
    nombre = reg["NOMBRE"] as String;
    apellidos = reg["APELLIDOS"] as String;
    edad = reg["DNI"] as Int32?;
    Console.WriteLine("{0}\t{1}\t{2}\t{3}", dni, nombre, apellidos, edad);
}
```

(*) Los tipos de datos nullable (*int32?*), se emplean en sustitución de los tipos primitivos por valor para recoger aquellos campos que admiten el valor nulo. (*null*)

5.5.- Búsqueda de datos en un objeto DataSet

Es posible realizar búsquedas sobre el conjunto de objetos DataRow obtenidos en un objeto DataTable mediante el uso de su método miembro **Select**:

```
Public Function [Select](filterExpression As String) As System.Data.DataRow()
```

El método recibe como parámetro una cadena de texto con una expresión condicional que identifica qué registros desean obtenerse. El resultado es una matriz de objetos DataRow con todos los que cumplen la condición.

Ejemplo: El siguiente código recupera empleando la función Query todos los conductores registrados en la tabla CONDUCTOR en la base de datos. Después se realiza una búsqueda sobre los datos devueltos para obtener y mostrar los datos de aquellos mayores de 25 años.

```
VB
Dim ds As DataSet
Try
    ' Recuperacion de datos
    ds = Query("SELECT DNI, NOMBRE, APELLIDOS, EDAD FROM CONDUCTOR", "conductores")
    ' Obtención del DataTable con los resultados
    Dim dt As DataTable = ds.Tables.Item("conductores")

    Dim dni As String
    Dim nombre As String
    Dim apellidos As String
    Dim edad As Integer

    ' Seleccion de aquellos registros con edad superior a 25
    Dim rows As DataRow() = dt.Select("EDAD > 25")

    ' Hay resultados?
    If rows.Length > 0 Then
        ' Recorrido de la matriz de resultados
        For Each registro As DataRow In rows
            dni = CType(registro.Item("DNI"), String)
            nombre = CType(registro.Item("NOMBRE"), String)
            apellidos = CType(registro.Item("APELLIDOS"), String)
            edad = CType(registro.Item("EDAD"), Integer)
            ' Visualización de datos.
            Console.WriteLine("{0}" + vbTab + "{1}" + vbTab + "{2}" + vbTab + "{3}",
dni, nombre, apellidos, edad)
        Next
    Else
        Console.WriteLine("No hay ningun conductor mayor de 25 años.")
    End If

Catch ex As SqlException
    Console.WriteLine("Error {0}", ex.Message)
End Try
Console.ReadKey()
```

C#

```

try
{
    DataSet ds = Query("SELECT DNI, NOMBRE, APELLIDOS, EDAD FROM CONDUCTOR",
"conductores");

    String dni;
    String nombre;
    String apellidos;
    Int32? edad;

    DataTable dt = ds.Tables["conductores"];

    // Seleccíon de aquellos registros mayores de 25 años
    DataRow[] regs = dt.Select("EDAD > 25");

    // Comprobación ¿Hay registros?
    if (regs.Count() > 0)
    {
        // Recorrido de matriz de registros resultados.
        foreach (DataRow reg in regs)
        {
            dni = reg["DNI"] as String;
            nombre = reg["NOMBRE"] as String;
            apellidos = reg["APELLIDOS"] as String;
            edad = reg["DNI"] as Int32?;
            Console.WriteLine("{0}\t{1}\t{2}\t{3}", dni, nombre, apellidos,
edad);
        }
    }
    else
    {
        Console.WriteLine("No hay resultados.");
    }
}
catch (SqlException exc)
{
    // Control de errores
    Console.WriteLine("Se ha producido el error {0} - {1}",
exc.GetType().ToString(), exc.Message);
}
Console.ReadKey();

```

5.6.- Uso de Vistas

Una vista es una representación de los datos de una tabla en el que se muestran parte de ellos y/o organizados de diferente modo. La clase ***DataView*** representa una vista de los datos almacenados en un objeto ***DataTable***.

Los objetos ***DataView*** pueden obtenerse llamando a la propiedad ***DefaultView*** de una ***DataTable***. Esta propiedad devuelve una instancia de ***DataView*** que muestra por defecto todos los registros de la ***DataTable*** sin aplicar filtro alguno:

```
Public ReadOnly Property DefaultView As System.Data.DataView
```

También es posible crear una vista mediante el constructor de la clase ***DataView***:

Sobrecarga 1 → Crea un objeto ***DataView*** que muestra todos los datos contenidos en el objeto ***DataTable*** indicado como argumento:

```
Public Sub New(table As System.Data.DataTable)
```









Sobrecarga 2 → Crea un objeto ***DataView*** que muestra los datos contenidos en el objeto ***DataTable*** en un orden determinado, aquellos objetos ***DataRow*** que cumplen una determinada condición y/o están en un determinado estado.

```
Public Sub New(table As System.Data.DataTable, RowFilter As String,  
Sort As String, RowState As System.Data.DataViewRowState)
```

La descripción de los parámetros es la siguiente:

- ***Table***: Hace referencia a la instancia de la tabla en la que se basa el ***DataView***.
- ***RowFilter***: Filtro de registros a mostrar.
- ***RowStateFilter***: Filtra los registros a mostrar según su estado de modificación.
- ***Sort***: Criterio de ordenación ascendente o descendente de los registros a mostrar.

El parámetro ***RowStateFilter*** admite un valor del tipo enumerado ***DataViewRowState***. En función del valor asignado a esta propiedad se mostrarán unos registros u otros en función de su estado de modificación:

Nombre de miembro	Descripción
 Added	Fila nueva.
 CurrentRows	Filas actuales, incluidas las filas sin modificar, las nuevas y las modificadas.
 Deleted	Fila eliminada.
 ModifiedCurrent	Versión actual, que es una versión modificada de los datos originales (vea ModifiedOriginal).
 ModifiedOriginal	Versión original (aunque se haya modificado y esté disponible como ModifiedCurrent).
 None	Ninguno.
 OriginalRows	Filas originales, incluidas las filas sin modificar y las eliminadas.
 Unchanged	Fila sin modificar.

La propiedad *Sort* referencia una cadena de texto que contiene el nombre de una o varias columnas de la *DataTable* seguidas de “ASC” o “DESC” indicando que el orden de los registros debe hacerse en función de las columnas citadas en orden ascendente o descendente respectivamente.

Ejemplo: El siguiente código recupera empleando la función *Query* de ejemplos anteriores todos los conductores registrados en la tabla *CONDUCTOR*. Se crea a continuación un objeto *DataView* que represente a los conductores superiores de 25 años ordenados por edad ascendente:

```
VB
Shared Sub Main()
    Dim ds As DataSet
    Try
        ' Recuperacion de datos
        ds = Query("SELECT DNI, NOMBRE, APELLIDOS, EDAD FROM CONDUCTOR", "conductores")
        ' Obtención del DataTable con los resultados
        Dim dt As DataTable = ds.Tables.Item("conductores")

        Dim dni As String
        Dim nombre As String
        Dim apellidos As String
        Dim edad As Integer

        ' Obtención de la vista filtrada
        Dim dv As DataView = New DataView(dt, "EDAD>25", "EDAD ASC",
DataViewRowState.Unchanged)

        ' Hay resultados?
        If dv.Count > 0 Then

            ' Objeto Vista Registro
            Dim vistaRegistro As DataRowView
            ' Bucle de recorrido y mostrado de los registros en la vista
            For indx As Integer = 0 To dv.Count - 1
                ' Obtención del objeto vistaRegistro correspondiente al registro a
mostrar

                vistaRegistro = dv.Item(indx)
                dni = CType(vistaRegistro.Item("DNI"), String)
                nombre = CType(vistaRegistro.Item("NOMBRE"), String)
                apellidos = CType(vistaRegistro.Item("APELLIDOS"), String)
                edad = CType(vistaRegistro.Item("EDAD"), Integer)
                ' Visualización de datos.
                Console.WriteLine("{0}" + vbTab + "{1}" + vbTab + "{2}" + vbTab + "{3}",
dni, nombre, apellidos, edad)
            Next
        Else
            Console.WriteLine("No hay ningun conductor mayor de 25 años.")
        End If

        Catch ex As SqlException
            Console.WriteLine("Error {0}", ex.Message)
        End Try
        Console.ReadKey()
    End Sub
```

Los registros representados por una vista se definen como objetos *DataRowView*. La propiedad **Count** de *DataView* indica el número de registros mostrados. Para obtener cada registro de la vista se emplea la propiedad **Item** indicando su posición empezando por 0.

C#

```

        try
        {
            DataSet ds = Query("SELECT DNI, NOMBRE, APELLIDOS, EDAD FROM CONDUCTOR",
"conductores");






            String dni;
            String nombre;
            String apellidos;
            Int32? edad;
            // Obtencion del DataTable con los resultados.
            DataTable dt = ds.Tables["conductores"];
            // Obtencion de la vista
            DataView dv = new DataView(dt, "EDAD > 25", "EDAD ASC",
DataViewRowState.Unchanged);
            // Comprobacion ¿Hay registros?
            if (dv.Count > 0)
            {
                // Recorrido de matriz de registros resultados.
                foreach (DataRowView reg in dv)
                {
                    dni = reg["DNI"] as String;
                    nombre = reg["NOMBRE"] as String;
                    apellidos = reg["APELLIDOS"] as String;
                    edad = reg["DNI"] as Int32?;
                    Console.WriteLine("{0}\t{1}\t{2}\t{3}", dni, nombre, apellidos,
edad);
                }
            }
            else
            {
                Console.WriteLine("No hay resultados.");
            }
        }
        catch (SqlException exc)
        {
            // Control de errores
            Console.WriteLine("Se ha producido el error {0} - {1}",
exc.GetType().ToString(), exc.Message);
        }
    }

```

5.5.- Modificación de datos de un objeto DataSet.

Los registros de las tablas de un DataSet pueden manipularse, ser eliminados y creados. Para ello se trabaja principalmente con los objetos DataRow contenidos en la colección *DataRowCollection* de cada DataTable accesible mediante la propiedad *Rows*.

Todas las operaciones de modificación, eliminación o alta de nuevos objetos DataRow son registradas mediante el valor de la propiedad **RowState** de cada objeto DataRow. Esta propiedad indica si un objeto DataRow ha sido modificado, creado, eliminado, o permanece intacto desde que se recuperó de la base de datos. La propiedad admite los valores del tipo enumerado: *System.Data.DataRowState*:

Nombre de miembro	Descripción
 Added	La fila se ha agregado a DataRowCollection y no se ha llamado a AcceptChanges .
 Deleted	La fila se ha eliminado mediante el método Delete del DataRow .
 Detached	Se ha creado la fila, pero no forma parte de ningún DataRowCollection . DataRow 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.
 Modified	La fila se ha modificado y no se ha llamado a AcceptChanges .
 Unchanged	La fila no ha cambiado desde que se llamó a AcceptChanges por última vez.

El valor de la propiedad *RowState* se emplea posteriormente para actualizar en la base de datos las modificaciones sobre los objetos DataRow. Por cada objeto modificado, insertado, o eliminado (en función de su RowState), se genera una sentencia SQL del tipo INSERT, UPDATE o DELETE.

5.5.1.- Modificación de valores de un DataRow

Los valores de los campos de un objeto DataRow se modifican sobrescribiendo sus valores mediante asignación. Ha de respetarse los tipos de datos de manera que los nuevos valores asignados se correspondan con el tipo de dato de la columna:

Ejemplo: El siguiente código emplea la función *Query* del ejemplo anterior para recuperar los datos de un conductor y modificar el valor del campo NOMBRE.

```

VB
Dim ds As DataSet
Try
    ' Recuperacion de datos
    ds = Query("SELECT DNI, NOMBRE, APELLIDOS, EDAD FROM CONDUCTOR WHERE DNI = '23670002V' " & "conductores")
    ' Obtención del DataTable con los resultados
    Dim dt As DataTable = ds.Tables.Item("conductores")
    ' Comprobación.. Hay resultados?
    If dt.Rows.Count <> 0 Then
        ' Obtención del objeto DataRow y modificación del valor del campo NOMBRE
        Dim dr As DataRow = dt.Rows.Item(0)          ← OBTENCION DEL 1ER REGISTRO
        dr.Item("NOMBRE") = "Manolo"
    End If
Catch ex As SqlException
    ' DETECCION E INFORME DEL ERROR.
End Try

```

C#

```

try
{
    // Recuperacion de datos.
    DataSet ds = Query("SELECT DNI, NOMBRE, APELLIDOS, EDAD FROM CONDUCTOR WHERE
DNI = '23670002V'", "conductores");
    // Obtencion del DataTable con los datos.
    DataTable dt = ds.Tables["conductores"];
    // Comprobacion de resultados.
    if (dt.Rows.Count != 0)
    {
        // Obtencion del PRIMER registro
        DataRow dr = dt.Rows[0];
        // Modificacion del valor del campo "NOMBRE"
        dr["NOMBRE"] = "Manolo";
    }
}
catch (SqlException exc)
{
    // Control de errores
    Console.WriteLine("Se ha producido el error {0} - {1}",
exc.GetType().ToString(), exc.Message);
}

```

Nota: Esta operación sólo modifica el objeto DataRow. No la base de datos.

5.5.2.- Inserción de nuevos registros (DataRow)

Para dar de alta un nuevo registro debe crearse un nuevo objeto DataRow. Para obtener un nuevo objeto DataRow debe invocarse el método **NewRow** del objeto DataTable en el que se quiere insertar el nuevo registro. No debe emplearse el constructor de la clase DataRow, ya que esto daría lugar a un objeto DataRow sin los campos correspondientes a las columnas de la tabla a la que ha de pertenecer. Una vez creado el objeto DataRow y asignados valores a sus campos debe insertarse en la colección Rows del objeto DataTable con el método **Add**.

Ejemplo: El siguiente código emplea la función *Query* para obtener un DataSet sin registros pero con los datos de las columnas y tipos de la tabla AGENTE. Al objeto DataTable recibido se le añade un nuevo objeto DataRow con los datos de un agente nuevo

VB

```

Dim ds As DataSet
Try
    ' Recuperacion de datos
    ds = Query("SELECT CODIGO, NOMBRE, TELEFONO FROM AGENTE WHERE 1 = 0", "agentes")
    ' Obtencion del DataTable con los resultados
    Dim dt As DataTable = ds.Tables.Item("agentes")
    Dim nuevoRegistro As DataRow = dt.NewRow()
    nuevoRegistro.Item("CODIGO") = "0005"
    nuevoRegistro.Item("NOMBRE") = "Roger Petrov"
    nuevoRegistro.Item("TELEFONO") = 123456789
    dt.Rows.Add(nuevoRegistro)
Catch ex As SqlException
    ' DETECCION E INFORME DEL ERROR.
End Try

```

C#

```

    try
    {
        // Recuperacion de datos.
        DataSet ds = Query("SELECT DNI, NOMBRE, APELLIDOS, EDAD FROM CONDUCTOR WHERE
1 = 0", "conductores");
        // Obtencion del DataTable con los datos.
        DataTable dt = ds.Tables["conductores"];
        // Obtención del nuevo registro
        DataRow nuevoRegistro = dt.NewRow();
        // Insercion de nuevos valores
        nuevoRegistro["CODIGO"] = "0005";
        nuevoRegistro["NOMBRE"] = "Roger Petrov";
        nuevoRegistro["TELEFONO"] = 123456789;
        // Insercion de nuevo registro en la tabla de resultados.
        dt.Rows.Add(nuevoRegistro);
    }
    catch (SqlException exc)
    {
        // Control de errores
        Console.WriteLine("Se ha producido el error {0} - {1}",
exc.GetType().ToString(), exc.Message);
    }
    Console.ReadKey();

```

Nota: Esta operación añade un nuevo objeto DataRow al DataTable. No la base de datos.

5.5.3.- Eliminación de registros

Para eliminar un registro debemos llamarse al método **Delete** del objeto DataRow para que su propiedad RowState pase al valor *DataRowState.Deleted*. Esto marca al objeto DataRow para ser eliminado.

Ejemplo: El siguiente código emplea la función Query para obtener los datos de un determinado conductor y marca para eliminación sus datos.

VB

```

Dim ds As DataSet
Try
    ' Recuperacion de datos
    ds = Query("SELECT DNI, NOMBRE, APELLIDOS, EDAD FROM CONDUCTOR WHERE DNI =
' 23670002V' ", "conductores")
    ' Obtención del DataTable con los resultados
    Dim dt As DataTable = ds.Tables.Item("conductores")
    ' Comprobación. Hay resultados?
    If dt.Rows.Count <> 0 Then
        ' Obtencion del objeto DataRow y marcación para eliminación.
        Dim dr As DataRow = dt.Rows.Item(0)      ← OBTENCION DEL 1ER REGISTRO
        dr.Delete()
    End If
Catch ex As SqlException
    ' DETECCION E INFORME DEL ERROR.
End Try

```

C#

```

try
{
    // Recuperacion de datos.
    DataSet ds = Query("SELECT DNI, NOMBRE, APELLIDOS, EDAD FROM CONDUCTOR WHERE
DNI = '23670002V' ", "conductores");
    DataTable dt = ds.Tables["conductores"];
    // Comprobacion de registro
    if (dt.Rows.Count != 0)
    {
        // Obtencion del registro a eliminar
        DataRow registro = dt.Rows[0];
        // Registro marcado para eliminacion
        registro.Delete();
    }
}
catch (SqlException exc)
{
    // Control de errores
    Console.WriteLine("Se ha producido el error {0} - {1}",
exc.GetType().ToString(), exc.Message);
}

```

Nunca debe eliminarse el objeto DataRow de la colección Rows. Si se elimina el objeto no será posible su borrado de la base de datos al realizar la actualización posteriormente.

Nota: Esta operación marca como eliminado el objeto DataRow en el DataTable. No lo elimina de la base de datos.

Conclusión:

Todas las operaciones vistas tienen como fin modificar los datos almacenados en el conjunto de resultados. Sin embargo, éstos cambios no afectan por si mismos a la base de datos de origen.

Para reflejar las modificaciones realizadas en la base de datos es necesario actualizarla a través del objeto adaptador de datos utilizado para la obtención de los datos originales.

5.6.- Actualización de la base de datos

Los objetos adaptadores de datos también permiten actualizar en la base de datos las modificaciones realizadas en un objeto *DataSet*. La clase *DataAdapter* dispone para ello de cuatro propiedades con objetos comando:

- **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* contiene la referencia al objeto comando que retorna los datos para el objeto *DataSet* al usar el método *Fill*. El resto de propiedades *InsertCommand*, *DeleteCommand*, y *UpdateCommand* contienen objetos comandos para actualizar en la base de datos cada objeto *DataRow* según haya sido insertado, modificado, o marcado para eliminación.

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 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.

5.6.1.- El generador de comandos

Los comandos de las propiedades *InsertCommand*, *UpdateCommand* y *DeleteCommand* pueden ser asignados manualmente, o ser generados en base al comando *SelectCommand* empleando un objeto generador de comandos.

ADO.NET define como las siguientes clases generadoras de comandos para los diferentes proveedores. Todas ellas heredan de la clase base *DbCommandBuilder*.

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

Para generar los comandos para las propiedades *InsertCommand*, *UpdateCommand* y *DeleteCommand* de un adaptador de datos se crea un objeto *CommabdBuilder* pasando a su constructor como parámetro el adaptador de datos a emplear.

El objeto generador de comandos resultante elabora los comandos y los asigna a las propiedades del adaptador de datos en función del comando de la propiedad *SelectCommand* que debe estar establecido de antemano.

5.6.2.- Actualización de datos

El método **Update** de un objeto adaptador de datos actualiza en la base de datos las modificaciones realizadas en un *DataSet*. El método posee varias sobrecargas:

Sobrecarga1 : Actualiza la base de datos con la información del *DataTable* especificado

Public Function **Update**(dataTable As [System.Data.DataTable](#)) As [Integer](#)

Sobrecarga2 : Actualiza la base de datos con la información de la tabla contenida en el *DataSet* especificado, cuyo nombre se indica como segundo parámetro.

Public Function **Update**(dataSet As [System.Data.DataSet](#), srcTable As [String](#)) As [Integer](#)

El método devuelve un valor entero que indica el número de registros que han sido actualizados en la base de datos.

5.6.3.- Confirmación / Rechazo de los datos

Si la actualización de los datos se realiza sin problemas pueden confirmarse las modificaciones en el objeto *DataSet* o *DataTable* llamando al método **AcceptChanges**. El método reinicia la propiedad *RowState* de todos los objetos *DataRow* al valor *DataRowState.Unchanged*. Este método debe llamarse siempre después de haber actualizado en la base de datos las modificaciones en el DataSet.

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.

5.6.4.- Requisitos para la generación automática de comandos

La creación automática de los comandos de inserción, actualización y borrado por parte del generador de comandos requiere ciertas condiciones:

- El adaptador de datos debe tener asignado el comando de consulta, ya que se emplea como patrón para generar el resto.
- El comando de consulta debe devolver obligatoriamente un campo clave primaria.
- El comando de consulta debe devolver datos procedentes de una única tabla.

Ejemplo: La siguiente clase define dos métodos Query y Update que implementan el código básico necesario para relizar consultas y actualizaciones a través de un adaptador de datos en modo desconectado:

```
VB
Imports System.Data.SqlClient

Public Class AccesoBD
    ' Cadena de conexion
    Public Shared cadenaConexion As String = ""

    ' Adaptador de datos
    Private adaptador As SqlDataAdapter = Nothing

    ' función de consulta
    Public Function Query(ByVal SQL As String, ByVal source As String) As DataSet
        Dim comando As SqlCommand = Nothing
        Dim ds As New DataSet()

        ' Instanciacion de conexion a la base de datos
        Dim conexion As New SqlConnection(cadenaConexion)

        ' Instanciacion del comando
        comando = conexion.CreateCommand()
        comando.CommandText = SQL

        ' Obtención de Adaptador de datos
        adaptador = New SqlDataAdapter(comando)
        ' Generador de comandos
        Dim generador As New SqlCommandBuilder(adaptador)

        ' Creacion y relleno del conjunto de datos.
        adaptador.Fill(ds, source)

        ' Retorno de conjunto de datos
        Return ds
    End Function

    ' Funcion de actualizacion
    Public Function Update(ByRef dataset As DataSet, ByVal source As String) As Integer
        Dim regsModificados As Integer

        ' Actualizacion de registros de conjunto de datos
        regsModificados = adaptador.Update(dataset, source)

        ' Retorno registros modificados
        Return regsModificados
    End Function
End Class
```

C#

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

namespace Consol eAppl i cati on1
{
    public class AccesoBD
    {
        // Cadena de conexion
        public static String cadenaConexi on = "...";

        // Adaptador de datos
        private SqlDataAdapter adaptador = null;

        // Método de consulta
        public DataSet Query(String SQL, String source)
        {
            DataSet ds = new DataSet();
            using (SqlConnection conexi on = new SqlConnection(cadenaConexi on))
            {
                // Instanciacion del comando
                SqlCommand comando = conexi on.CreateCommand();
                comando.CommandText = SQL;
                // Instanciacion del adaptador
                adaptador = new SqlDataAdapter(comando);
                // Instanciacion del generador de comandos
                SqlCommandBuilder generadorComandos = new SqlCommandBuilder(adaptador);
                // Creacion y relleno del conjunto de datos
                adaptador.Fill(ds, source);
            }
            return ds;
        }

        // Metodo de actualizacion
        public int Update(DataSet ds, String source)
        {
            int regsModi fi cados;
            // Actualizacion de registros en la base de datos
            regsModi fi cados = adaptador.Update(ds, source);
            return regsModi fi cados;
        }
    }
}
```

Ejemplo de Uso: El siguiente código utiliza un objeto de la clase AccesoBD para recuperar los datos de un registro de la tabla CONDUCTOR, modificar su campo NOMBRE y actualizarlo en la base de datos:

VB

```
Sub main()

    Dim ds As DataSet
    Dim acceso As New AccesoBD()

    ' Obtencion de datos
    ds = acceso.Query("SELECT DNI, NOMBRE, APELLIDOS FROM CONDUCTOR WHERE DNI = '23670002V' ", "conductor")

    ' Comprobacion de obtención de datos
    If ds.Tables("conductor").Rows.Count = 1 Then

        ' Obtencion del registro
        Dim registro As DataRow = ds.Tables("conductor").Rows(0)
        Console.WriteLine("CONDUCTOR {0} {1} {2}", registro("DNI").ToString(),
            registro("NOMBRE").ToString(), registro("APELLIDOS").ToString())

        ' Modificación valor del campo NOMBRE
        registro("NOMBRE") = "LUI SA"

        ' Actualizacion
        Try
            acceso.Update(ds, "conductor")
            ' Confirmacion de modificaciones en DataSet
            ds.AcceptChanges()
        Catch ex As Exception
            Console.WriteLine("ERROR: {0}", ex.Message)
            ' Cancelacion de modificaciones en DataSet
            ds.RejectChanges()
        End Try

    Else
        Console.WriteLine("No existe conductor.")
    End If

    Console.ReadKey()

End Sub
```

C#

```
static void Main(string[] args)
{
    DataSet ds = null;

    AccesoBD bd = new AccesoBD();

    // Obtencion de datos
    ds = bd.Query("SELECT DNI, NOMBRE, APELLIDOS FROM CONDUCTOR WHERE DNI = '23670002V' ", "conductor");

    // Comprobacion de obtencion de datos
    if ( ds.Tables["conductor"].Rows.Count == 1 ) {
        // Obtencion del registro
        DataRow registro = ds.Tables["conductor"].Rows[0];
        Console.WriteLine("CONDUCTOR {0} {1} {2}",
            registro["DNI"].ToString(),
            registro["NOMBRE"].ToString(),
            registro["APELLIDOS"].ToString());

        // Modificación del valor del campo NOMBRE
        registro["NOMBRE"] = "Lui sa";

        // Actualizacion
        try {
            bd.Update( ds, "conductor");
            // Confirmacion de modificaciones en conjunto de datos.
            ds.AcceptChanges();
        } catch ( Exception ex ) {
            Console.WriteLine("ERROR {0}", ex.Message );
            // Cancelacion de modificaciones en conjunto de datos.
            ds.RejectChanges();
        }
    }
    else {
        Console.WriteLine("No existe el conductor.");
    }
}
```

5.7.- Problemas de Actualización y Concurrency

Uno de los problemas que plantea el acceso desconectado a una base de datos es la concurrencia que se produce cuando varios usuarios o aplicaciones leen y modifican la misma información en una base de datos.

Ejemplo de problema de concurrencia:

Supóngase una aplicación de ventas en la que se gestionan las compras en un almacén. Un cliente A solicita 10 unidades de un producto del que quedan en stock 15 unidades. La aplicación carga el número de unidades que quedan de ese producto, y viendo que hay más unidades de las solicitadas se autoriza la compra. Entre tanto, otro cliente B realiza una solicitud idéntica que también se autoriza puesto que el cliente A no ha modificado los datos.

El cliente A termina la operación y actualiza el stock a 5 unidades sin saber que otro cliente está haciendo la misma operación. El cliente B termina también la operación actualiza el stock también a 5 unidades, sin saber que otro cliente ha vendido esas mismas unidades entre tanto.

El control de concurrencia detecta al actualizar una información en la base de datos si otro usuario lo ha modificado entre medias. El adaptador de datos detecta esta situación al actualizar llamando al método **Update** un objeto DataSet a partir de los comandos creados por el generador de comandos.

La sentencia **UPDATE** creadas por el generador de comandos comprueba que el registro a modificar tenga los mismos valores que cuando se obtuvo con la consulta. Si no coinciden, se asume que ha sido modificado entretanto y se produce una excepción **DbConcurrencyException**.

5.7.1.- Captura de errores de actualización

Por defecto, al darse un error al actualizar un registro de un DataSet se produce la excepción correspondiente y se detiene la actualización. La excepción puede capturarse encapsulando la llamada al método Update del adaptador de datos en un bloque TRY-CATCH. Si se trata de una excepción de concurrencia debida a que otro usuario ha modificado un registro antes que nosotros, puede recargarse el DataSet con los valores actuales mediante volviendo a llamar al método Fill.

VB

```
Try
    ' Actualización
    adaptador.Update(ds, "mul tas")
    ds.AcceptChanges()
Catch ex As DbConcurrencyException
    Console.WriteLine("Error de concurrencia")
    ds.Tables("mul tas").Rows.Clear();
    adaptador.Fill(ds, "mul tas")
Catch ex As Exception
    Console.WriteLine("ERROR TIPO {0} ", ex.GetType().ToString())
    Console.WriteLine("MENSAJE: ", ex.Message)
End Try
```

C#

```

try
{
    adaptador.Update(ds, "mul tas");
    ds.AcceptChanges();
}
catch (Exception ex)
{
    Console.WriteLine("ERROR {0} : {1}", ex.GetType().ToString(), ex.Message);
    ds.Tables["mul tas"].Rows.Clear();
    adaptador.Fill(ds, "mul tas");
}

```

En el caso de que la actualización se realice satisfactoriamente debe invocarse al método **AcceptChanges()** del objeto DataSet. Este método modifica el valor de la propiedad *RowState* de todos los objetos *Row* a *Unchanged*, de manera que se consideran ya actualizados.

En caso de error puede obtenerse por rechazar los cambios realizados en el conjunto de datos llamando al método **RejectChanges()** que recupera los valores originales para todos los objetos *Row*. Otra opción es recargar el objeto *DataSet* ejecutando nuevamente el método *Fill* contra el adaptador de datos obteniendo así una nueva copia actualizada de los datos existentes en la base de datos.

5.7.2.- Manejo de errores controlando evento *RowUpdated*

Si la modificación afecta a varios registros puede emplearse el evento **RowUpdated** para controlar la actualización de cada registro del DataSet.

El evento **RowUpdated** es un evento perteneciente al adaptador de datos. Este evento se ejecuta tras la actualización de cada objeto DataRow que ha sido insertado, modificado o marcado para eliminación en un objeto DataSet. La función manejadora para el evento tiene la siguiente declaración:

```
Private Shared Sub Manejador(sender As Object, e As SqlRowUpdatedEventArgs)
```

El parámetro de entrada 'e' de tipo **SqlRowUpdatedEventArgs** devuelve información del registro que se ha actualizado y el resultado de la actualización mediante sus propiedades:

Las propiedades interesantes del objeto 'e' son:

- **Row** → Devuelve el objeto DataRow actualizado.
- **Error** → Devuelve la excepción en caso de error al actualizar el registro. Si no retorna Nothing.
- **RowCount** → Devuelve el índice del registro actualizado.
- **StatementType** → Indica el tipo de sentencia ejecutada para la actualización.

Ejemplo: El siguiente código obtiene un objeto DataSet con el código de identificación y la cuantía de todas las sanciones registradas en la tabla SANCION. Se recorre a continuación el conjunto de registros devueltos incrementando en 100 el coste de la sanción para actualizarlos en la base de datos empleando el adaptador de datos.

El evento *RowUpdated* del objeto adaptador de datos se asocia a la función *Manejador*.

```
Dim comando As SqlCommand = Nothing
Dim adaptador As SqlDataAdapter = Nothing
Dim generador As SqlCommandBuilder = Nothing
Dim ds As New DataSet()

' Instanciación de conexión a la base de datos
Dim conexion As New SqlConnection("Data Source=192.168.9.158;Initial
Catalog=MULTAS;Persist Security Info=True;User ID=usuario;Password=ci psa")
' Instanciación del comando
comando = conexion.CreateCommand()
comando.CommandText = "SELECT NUMERO_SANCION, COSTE_SANCION FROM SANCION"

' Obtención de Adaptador de datos
adaptador = New SqlDataAdapter(comando)
' Asociación de función manejadora del evento RowUpdated
AddHandler adaptador.RowUpdated, AddressOf Manejador

' Generador de comandos
generador = New SqlCommandBuilder(adaptador)

' Creación y relleno del conjunto de datos.
adaptador.Fill(ds, "mul tas")

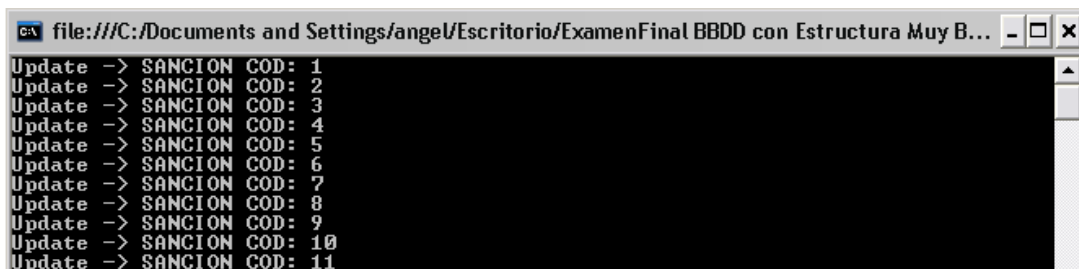
' Modificación de registros
For Each reg As DataRow In ds.Tables("mul tas").Rows
    reg.Item("COSTE_SANCION") = CType(reg.Item("COSTE_SANCION"), Single) + 100
Next

Try
    ' Actualización
    adaptador.Update(ds, "mul tas")
Catch ex As Exception
    Console.WriteLine("ERROR TIPO {0} ", ex.GetType().ToString())
    Console.WriteLine("MENSAJE: ", ex.Message)
End Try
```

La función manejadora asociada al evento *RowUpdated* podría tener el siguiente código:

```
Private Shared Sub Manejador(sender As Object, e As SqlRowUpdatedEventArgs)
    If e.Errors Is Nothing Then
        ' Muestra el tipo de comando utilizado.
        Console.WriteLine(e.StatementType.ToString())
        ' Muestra información del registro ya actualizado
        Console.WriteLine("-> SANCION COD: {0}", e.Row.Item("NUMERO_SANCION").ToString())
    Else
        ' Muestra el tipo de excepción y el mensaje asociado
        Console.WriteLine(e.Errors.GetType().ToString() + " -> " + e.Errors.Message)
        ' Muestra la información del registro no actualizado.
        Console.WriteLine("ERROR -> SANCION COD: {0}",
            e.Row.Item("NUMERO_SANCION").ToString())
    End If
End Sub
```

Resultados mostrados si todo es correcto:



```
C:\ file:///C:/Documents and Settings/angel/Escritorio/ExamenFinal BBDD con Estructura Muy B...
Update -> SANCION COD: 1
Update -> SANCION COD: 2
Update -> SANCION COD: 3
Update -> SANCION COD: 4
Update -> SANCION COD: 5
Update -> SANCION COD: 6
Update -> SANCION COD: 7
Update -> SANCION COD: 8
Update -> SANCION COD: 9
Update -> SANCION COD: 10
Update -> SANCION COD: 11
```

Todos los registros han sido actualizados correctamente.

Resultados mostrados en caso de error:

```

C:\ file:///C:/Documents and Settings/angel/Escritorio/ExamenFinal BBDD con Estructura Muy B...
Update -> SANCION COD: 1
Update -> SANCION COD: 2
Update -> SANCION COD: 3
Update -> SANCION COD: 4
Update -> SANCION COD: 5
Update -> SANCION COD: 6
Update -> SANCION COD: 7
System.Data.DBConcurrencyException -> Infracción de simultaneidad: UpdateCommand
afectó a 0 de los 1 registros esperados.ERROR -> SANCION COD: 8
Error de concurrencia
  
```

La actualización se ha producido hasta que ha fallado un registro.

C#

```

static void Main(string[] args)
{
    SqlCommand comando = null;
    SqlDataAdapter adaptador = null;
    DataSet ds = new DataSet();

    // Instanciación conexión a la base de datos.
    SqlConnection conexion = new SqlConnection("...");

    // Instanciación comando
    comando = conexion.CreateCommand();
    comando.CommandText = "SELECT NUMERO_SANCION, COSTE_SANCION FROM SANCION";

    // Instanciación del adaptador de datos
    adaptador = new SqlDataAdapter(comando);

    // Asociación de función manejadora para evento RowUpdated
    adaptador.RowUpdated += new SqlRowUpdatedEventHandler(adaptador_RowUpdated);
    SqlCommandBuilder generadorComandos = new SqlCommandBuilder(adaptador);

    // Obtención de resultados
    adaptador.Fill(ds, "mul tas");

    // Modificación de sanciones
    foreach (DataRow reg in ds.Tables["mul tas"].Rows)
    {
        float? valor = reg["COSTE_SANCION"] as float?;
        if (valor.HasValue) {
            reg["COSTE_SANCION"] = valor.Value + 1000.0f;
        }
    }
    // Actualización de la base de datos.
    try
    {
        adaptador.Update(ds, "mul tas");
        ds.AcceptChanges();
    }
    catch (Exception ex)
    {
        Console.WriteLine("ERROR {0} : {1}", ex.GetType().ToString(), ex.Message);
        ds.Tables["mul tas"].Rows.Clear();
        adaptador.Fill(ds, "mul tas");
    }
}

static void adaptador_RowUpdated(object sender, SqlRowUpdatedEventArgs e)
{
    // Comprobación de errores.
    if (e.Errors == null) {
        // SIN ERRORES
        // Muestra operación de actualización practicada sobre registro
        Console.WriteLine(e.StatementType.ToString());
        // Muestra clave primaria del registro actualizado.
        Console.WriteLine("Row -> {0}", e.Row["NUMERO_SANCION"].ToString());
    }
    else {
        // CON ERRORES
        // Muestra el tipo de excepción y mensaje de la actualización del registro
        Console.WriteLine(e.Errors.GetType().ToString() + " -> " + e.Errors.Message);
        // Muestra la información del registro no actualizado.
        Console.WriteLine("ERROR -> SANCION COD: {0}", e.Row["NUMERO_SANCION"].ToString());
    }
}
  
```

La función manejadora se activa tras intentar actualizar cada DataRow modificado, insertado o borrado en el objeto DataSet. Si la propiedad *Error* del objeto 'e' vale *Nothing* indica que la actualización se ha realizado sin problemas. En caso de producirse un error, la propiedad *Error* devuelve el objeto *Exception* que indica el error producido.

De manera predeterminada, esto implica además que el método *Update* lanza una excepción y se detiene el proceso de actualización para el resto de registros pendientes. Sin embargo; es posible modificar este comportamiento asignando diferentes valores a la propiedad *Status* del objeto 'e'. Esta propiedad admite valores del tipo enumerado **System.Data.UpdateStatus**:

- **UpdateStatus.ErrorOcurrred** → (valor predeterminado). El método **Update** lanza excepción y se detiene el proceso de actualización. No se actualizan el resto de registros pendientes, pero los anteriores quedan actualizados en la base de datos.
- **UpdateStatus.Continue** → Fuerza la actualización del registro a sabiendas que ha sido modificado antes por otro usuario. No se lanza excepción alguna. (*para casos de errores de concurrencia*)
- **UpdateStatus.SkipAllRemainingRows** → El método *Update* no lanza ninguna excepción, pero no se continúa la actualización de los registros pendientes. Los registros anteriores quedan actualizados en la base de datos.
- **UpdateStatus.SkipCurrentRow** → No se lanza excepción y se continúa con la actualización del resto de registros pendientes. Sólo quedan sin actualizar los que han fallado.

Supóngase que se modifica la función manejadora del ejemplo anterior añadiendo una sentencia que asigna el valor *UpdateStatus.SkipCurrentRow* a la propiedad *Status* del objeto 'e' en caso de producirse un error:

```
Private Shared Sub Manejador(sender As Object, e As SqlRowUpdatedEventArgs)
    If e.Errors Is Nothing Then
        ' Muestra el tipo de comando utilizado.
        Console.WriteLine(e.StatementType.ToString())
        ' Muestra información del registro ya actualizado
        Console.WriteLine("-> SANCION COD: {0}", e.Row.Item("NUMERO_SANCION").ToString())
    Else
        ' Muestra el tipo de excepción y el mensaje asociado
        Console.WriteLine(e.Errors.GetType().ToString() + " -> " + e.Errors.Message)
        ' Muestra la información del registro no actualizado.
        Console.WriteLine("ERROR -> SANCION COD: {0}",
e.Row.Item("NUMERO_SANCION").ToString())
        ' Orden de continuar con el resto de actualizaciones
        e.Status = UpdateStatus.SkipCurrentRow
    End If
End Sub
```


El resultado mostrado ahora en caso de producirse un error al actualizar uno de los resultados sería el siguiente:

```

C:\ file:///C:/Documents and Settings/ange/Escritorio/ExamenFinal BBDD con Estructura Muy B...
Update -> SANCION COD: 1
Update -> SANCION COD: 2
Update -> SANCION COD: 3
Update -> SANCION COD: 4
Update -> SANCION COD: 5
Update -> SANCION COD: 6
Update -> SANCION COD: 7
System.Data.DBConcurrencyException -> Infracción de simultaneidad: UpdateCommand
afectó a 0 de los 1 registros esperados.ERROR -> SANCION COD: 8
Update -> SANCION COD: 9
Update -> SANCION COD: 10
Update -> SANCION COD: 11
  
```

El proceso de actualización continúa más allá del registro que ha sufrido el error y se completa sin que el método `Update` lance excepción alguna.

5.7.3.- Manejo de errores sin controlar evento *RowUpdated*

También es posible controlar la actualización de los registros en caso de error mediante la propiedad ***ContinueUpdateOnError*** del adaptador de datos. Esta propiedad hace que al producirse un error al actualizar un registro, no se lance excepción y se continúe actualizando el resto de los registros. La información del error sucedido se almacena en la propiedad ***RowError*** del objeto `DataRow`.

Una vez ejecutado el método `Update`, puede comprobarse si ha habido algún error durante la actualización mediante la propiedad ***HasErrors*** del objeto `DataSet`. Esta propiedad devuelve un valor lógico cierto si algún objeto `DataTable` contenido tiene su propiedad `HasError` a cierto. De igual modo; esta propiedad devuelve cierto si alguno de los objetos `DataRow` contenidos en la tabla ha sufrido un error de actualización.

Ejemplo: El siguiente código realiza la modificación del valor de todas las sanciones registradas en la tabla `SANCION`. En esta ocasión no se captura el evento `RowUpdated` del adaptador de datos. Se asigna el valor cierto a la propiedad `ContinueUpdateOnError` al adaptador de manera que si se producen errores en la actualización, ni se produzca excepción alguna, ni se interrumpa la actualización del resto de registros.

VB

```

Dim comando As SqlCommand = Nothing
Dim adaptador As SqlDataAdapter = Nothing
Dim generador As SqlCommandBuilder = Nothing
Dim ds As New DataSet()

' Instanciación de conexión a la base de datos
Dim conexion As New SqlConnection("Data Source=192.168.9.158;Initial
Catalog=MULTAS;Persist Security Info=True;User ID=usuario;Password=ci psa")

' Instanciación del comando
comando = conexion.CreateCommand()
comando.CommandText = "SELECT NUMERO_SANCION, COSTE_SANCION FROM SANCION"

' Obtención de Adaptador de datos
adaptador = New SqlDataAdapter(comando)
' Generador de comandos
  
```

```

generador = New SqlCommandBuilder(adaptador)

' Creacion y relleno del conjunto de datos.
adaptador.Fill(ds, "mul tas")

' Modificacion de registros
For Each reg As DataRow In ds.Tables("mul tas").Rows
    reg.Item("COSTE_SANCION") = CType(reg.Item("COSTE_SANCION"), Single) + 100
Next

' Actualizacion
adaptador.ContinueUpdateOnError = True
adaptador.Update(ds, "mul tas")

```

C#

```

static void Main(string[] args)
{
    SqlCommand comando = null;
    SqlDataAdapter adaptador = null;
    DataSet ds = new DataSet();

    // Instanciacion conexion a la base de datos.
    SqlConnection conexion = new SqlConnection("Data Source=192.168.9.156;Initial
Catalog=MULTAS;Persist Security Info=True;User ID=usuario;Password=ci psa");

    // Instanciacion comando
    comando = conexion.CreateCommand();
    comando.CommandText = "SELECT NUMERO_SANCION, COSTE_SANCION FROM SANCION";

    // Instanciacion del adaptador de datos
    adaptador = new SqlDataAdapter(comando);
    // Asociacion de función manejadora para evento RowUpdated
    adaptador.RowUpdated += new SqlRowUpdatedEventHandler(adaptador_RowUpdated);
    SqlCommandBuilder generadorComandos = new SqlCommandBuilder(adaptador);

    // Obtencion de resultados
    adaptador.Fill(ds, "mul tas");

    // Modificacion de sanciones
    foreach (DataRow reg in ds.Tables["mul tas"].Rows)
    {
        float? valor = reg["COSTE_SANCION"] as float?;
        if (valor.HasValue) {
            reg["COSTE_SANCION"] = valor.Value + 1000.0f;
        }
    }
    // La actualización continúa en caso de error al actualizar un registro.
    adaptador.ContinueUpdateOnError = true;
    adaptador.Update(ds, "mul tas");

    // Actualizacion de la base de datos.
    try
    {
        adaptador.Update(ds, "mul tas");
        ds.AcceptChanges();
    }
    catch (Exception ex)
    {
        Console.WriteLine("ERROR {0} : {1}", ex.GetType().ToString(), ex.Message);
        ds.Tables["mul tas"].Rows.Clear();
        adaptador.Fill(ds, "mul tas");
    }
}

```

Dado que el método **Update** ya no producirá excepciones no es necesario encapsularlo en una estructura TRY-CATCH. No obstante, Sí debe comprobarse después si los registros han sido correctamente actualizados consultando el valor lógico de la propiedad **HasErrors** del objeto DataSet.

El siguiente código muestra como recorrer los registros del DataSet buscando aquellos con errores de actualización. Cada objeto DataRow dispone de una propiedad **HasErrors** que indica si tiene errores pendientes. En tal caso; su propiedad *RowError* devuelve una cadena con el mensaje explicativo del error:

VB
<pre> ' Comprobacion de errores If ds.Tables("multas").HasErrors() Then ' Bucle de búsqueda de registros no actualizados por error For Each registro As DataRow In ds.Tables("multas").Rows ' Comprobacion de errores en registro If registro.HasErrors = True Then Console.WriteLine("ERROR ACTUALIZANDO SANCION -> COD: {0} SANCION: {1}", registro.Item("NUMERO_SANCION"), registro.Item("COSTE_SANCION")) ' Mostrar causa de error Console.WriteLine("CAUSA: {0}", registro.RowError) End If Next End If </pre>
C#
<pre> // Comprobacion de errores de actualización en el DataSet if (ds.Tables["multas"].HasErrors) { // Bucle de recorrido de registros. foreach (DataRow reg in ds.Tables["multas"].Rows) { // Deteccion de error de actualización en registro if (reg.HasErrors) { // Mostrar código de registro sanción no actualizado Console.WriteLine("ERROR ACTUALIZANDO REG. SANCION COD : {0}", reg["CODIGO_SANCION"]); // Muestra información del error de actualización del registro Console.WriteLine("ERROR {0}", reg.RowError); } } } </pre>

5.8.- Control de Transacciones

La modificación de varios registros de una o varias tablas en una base de datos es una operación necesario en muchas ocasiones.

Ejemplo: Una entidad bancaria almacena en una base de datos los datos de sus clientes y sus cuentas. La base de datos dispone de dos tablas: CUENTAS y AHORROS. Cuando un usuario realiza un traspaso de su cuenta corriente corriente a su cuenta de ahorros, se modifica su registro en la tabla CUENTAS eliminando la cantidad traspasada, y se crea o modifica el registro correspondiente en la tabla AHORROS añadiendo la cantidad.

Supóngase al realizar un traspaso, se modifica el registro de la tabla CUENTAS, pero debido a un error, NO se actualiza la tabla AHORROS. Esto produciría un gravísimo error. Para evitar este tipo de situaciones se emplean las transacciones.

Una *transacción* es un agrupamiento de sentencias SQL que deben ejecutarse al completo, de manera que si se produce algún error, todos los cambios se deshacen. Se evita de este modo que las modificaciones puedan quedar a medias y se asegura la integridad y coherencia de los datos

Las transacciones tienen tres acciones asociadas:

- **Apertura de la transacción:** A partir de ese momento todas las sentencias de modificación realizadas (INSERT, UPDATE, y DELETE), se encuadran en la transacción. Las sentencias se aplican 'temporalmente' de manera que pueden deshacerse si es necesario y volver a los datos originales.
- **Confirmación de la transacción:** Las modificaciones realizadas por las sentencias encuadradas en la transacción se aplican de manera definitiva en la base de datos.
- **Cancelación de la transacción:** Todos los cambios realizados por las sentencias desde el inicio de la transacción se deshacen devolviendo la base de datos al estado original.

Las transacciones están representadas en ADO.NET por un conjunto de clases específicas para cada proveedor de datos.

- ***SqlTransaction***
- ***OracleTransaction***
- ***OleDbTransaction***
- ***OdbcTransaction***.

Todas estas clases comparten métodos y propiedades comunes heredados de la clase base ***System.Data.Common.DbTransaction***.

5.8.1.- Apertura y asignación de transacción a comandos.

Las transacciones se definen a partir de una conexión. La apertura de transacción se realiza llamando al método ***BeginTransaction*** del objeto conexión en uso, el cual devuelve como retorno un objeto de tipo transacción del proveedor en uso:

```
Public Function BeginTransaction() As System.Data.SqlClient.SqlTransaction
```

```
Dim comando As SqlCommand = Nothing
Dim transaccion As SqlTransaction = Nothing

' Instanciación de conexión a la base de datos
Dim conexión As New SqlConnection("...")

Try
    ' Apertura manual de la conexión
    conexión.Open()
    ' Obtención de la transacción
    transaccion = conexión.BeginTransaction()
```

El objeto *transaccion* debe asociarse a cada objeto comando que se quiera ejecutar como parte de la transacción. Para ello, debe asignarse el objeto transacción a la propiedad **Transaction** del objeto comando.

```
' Obtencion del comando
comando = conexion.CreateCommand()
comando.CommandText = "UPDATE SANCION SET COSTE_SANCION = COSTE_SANCION + 100"
' Asociación del comando a la transaccion
comando.Transaction = transaccion
```

5.8.2.- Confirmación y Cancelación de transacciones

Si la ejecución del comando o comandos se realiza satisfactoriamente debe confirmarse el objeto transacción llamando a su método **Confirm()** :

```
' Ejecucion del comando
comando.ExecuteNonQuery()
' Confirmación de transaccion y cambios en el DataSet
transaccion.Commit()
```

Habitualmente se llama al método **RollBack()** desde el bloque *Catch* de una estructura TRY-CATCH dispuesta para capturar las excepciones que puedan producirse al realizar las modificaciones.

```
Catch ex As Exception
' Comprobar si conexion existe y está abierta.
If Not conexion Is Nothing AndAlso conexion.State = ConnectionState.Open Then
' Rechazo de transaccion si existe.
If Not transaccion Is Nothing Then transaccion.Rollback()
End If
' Mostrar error.
Console.WriteLine("ERROR: {0} - {1}", ex.GetType.ToString(), ex.Message)
Finally
' Cierre de conexion
If Not conexion Is Nothing Then
conexion.Close()
End If
End Try
```

El manejo de transacciones requiere de una conexión abierta desde su inicio hasta su confirmación o cancelación. Es necesario comprobar si la conexión existe y está abierta antes de cancelar la transacción

C#
<pre>SqlConnection conexion = new SqlConnection("..."); SqlCommand comando = null; SqlTransaction transaccion = null; try { // Apertura de conexion conexion.Open(); // Obtencion de la transaccion transaccion = conexion.BeginTransaction(); // Obtencion del comando comando = conexion.CreateCommand(); comando.CommandText = "UPDATE SANCION SET COSTE_SANCION = COSTE_SANCION + 100"; // Asociación de la transacción al comando. comando.Transaction = transaccion; // Ejecucion del comando comando.ExecuteNonQuery(); // Confirmacion de la transaccion si no hay excepciones transaccion.Commit(); }</pre>

```

catch (Exception ex)
{
    if (conexion != null && conexion.State == ConnectionState.Open)
    {
        // Cancelacion de la transaccion
        if (transaccion != null) transaccion.Rollback();
    }
    // Visualizacion del error
    Console.WriteLine("ERROR {0} - {1}", ex.GetType().ToString(), ex.Message);
}
finally
{
    if (conexion != null) conexion.Close();
}

```

5.8.2.- Uso de transacciones con el adaptador de datos

Para emplear transacciones con adaptadores de datos es preciso asignar la transacción a los comandos de inserción, modificación y borrado creados por el generador de comandos :

Ejemplo : El código siguiente incrementa en 100€ el coste de todas las sanciones registradas en la tabla sanciones empleando un adaptador de datos. La actualización se realiza aplicando una transacción que permite deshacer todos los cambios en caso de error.

VB

```

Dim comando As SqlCommand = Nothing
Dim transaccion As SqlTransaction = Nothing
Dim adaptador As SqlDataAdapter = Nothing
Dim generador As SqlCommandBuilder = Nothing
Dim ds As New DataSet()
' Instanciacion de conexion a la base de datos
Dim conexion As New SqlConnection("...")
Try
    ' Apertura manual de la conexión
    conexion.Open()
    ' Obtencion de la transaccion
    transaccion = conexion.BeginTransaction()
    ' Obtencion del comando
    comando = conexion.CreateCommand()
    comando.CommandText = "SELECT NUMERO_SANCION, COSTE_SANCION FROM SANCION"
    ' Asociación del comando a la transaccion
    comando.Transaction = transaccion
    ' Obtención de Adaptador de datos
    adaptador = New SqlDataAdapter(comando)
    ' Generador de comandos
    generador = New SqlCommandBuilder(adaptador)
    ' Los comandos deben asignarse manualmente para poder asignarles
    ' el objeto transaccion en uso.
    adaptador.InsertCommand = generador.GetInsertCommand()
    adaptador.UpdateCommand = generador.GetUpdateCommand()
    adaptador.DeleteCommand = generador.GetDeleteCommand()
    adaptador.InsertCommand.Transaction = transaccion
    adaptador.UpdateCommand.Transaction = transaccion
    adaptador.DeleteCommand.Transaction = transaccion
    ' Creacion y relleno del conjunto de datos.
    adaptador.Fill(ds, "multas")
    ' Modificacion de registros
    For Each reg As DataRow In ds.Tables("multas").Rows
        reg.Item("COSTE_SANCION") = CType(reg.Item("COSTE_SANCION"), Single) + 100
    Next
    ' Actualizacion
    adaptador.Update(ds, "multas")
    ' Confirmación de transaccion y cambios en el DataSet
    transaccion.Commit()
    ds.AcceptChanges()
Catch ex As Exception
    ' Comprobar si conexion existe y está abierta.
    If Not conexion Is Nothing AndAlso conexion.State = ConnectionState.Open Then
        ' Rechazo de transaccion si existe.
        If Not transaccion Is Nothing Then transaccion.Rollback()
    End If
    ' Mostrar error.
    Console.WriteLine("ERROR: {0} - {1}", ex.GetType().ToString(), ex.Message)

```

```

Finally
    ' Cierre de conexión
    If Not conexion Is Nothing Then
        conexion.Close()
    End If
End Try

```

C#

```

SqlCommand comando = null;
SqlDataAdapter adaptador = null;
SqlTransaction transaccion = null;
DataSet ds = new DataSet();

// Instanciación conexión a la base de datos.
SqlConnection conexion = new SqlConnection("...");
try {
    // Apertura manual de la conexión
    conexion.Open();
    // Apertura de la transacción
    transaccion = conexion.BeginTransaction();
    // Instanciación comando
    comando = conexion.CreateCommand();
    comando.CommandText = "SELECT NUMERO_SANCION, COSTE_SANCION FROM SANCION";
    // Instanciación del adaptador de datos
    adaptador = new SqlDataAdapter(comando);
    // Generación de comandos
    SqlCommandBuilder generador = new SqlCommandBuilder(adaptador);
    // Asignación de transacción a los comandos de actualización
    // creados por el generador de comandos.
    adaptador.InsertCommand = generador.GetInsertCommand();
    adaptador.UpdateCommand = generador.GetUpdateCommand();
    adaptador.DeleteCommand = generador.GetDeleteCommand();
    adaptador.InsertCommand.Transaction = transaccion;
    adaptador.UpdateCommand.Transaction = transaccion;
    adaptador.DeleteCommand.Transaction = transaccion;
    // Obtención de resultados
    adaptador.Fill(ds, "multas");
    // Modificación de sanciones
    foreach (DataRow reg in ds.Tables["multas"].Rows)
    {
        float? valor = reg["COSTE_SANCION"] as float?;
        if (valor.HasValue) {
            reg["COSTE_SANCION"] = valor.Value + 1000.0f;
        }
    }
    // La actualización continúa en caso de error al actualizar un registro.
    adaptador.ContinueUpdateOnError = true;
    adaptador.Update(ds, "multas");
    // Confirmación de transacción y cambios en el conjunto de datos
    transaccion.Commit();
    ds.AcceptChanges();
}
catch (Exception ex)
{
    if (conexion != null && conexion.State == ConnectionState.Open)
    {
        // Cancelación de la transacción
        if (transaccion != null) transaccion.Rollback();
    }
    // Visualización del error
    Console.WriteLine("ERROR {0} - {1}", ex.GetType().ToString(), ex.Message);
}
finally
{
    // Cierre de conexión
    if (conexion != null) conexion.Close();
}

```

Cuando se emplean transacciones es preciso abrir y cerrar manualmente la conexión a pesar del uso del adaptador de datos. En este caso, el adaptador de datos no manipula la conexión, y la deja abierta tras la ejecución de sus métodos (*fill*, *update*), en vez de cerrarla.

De igual modo, en caso de error es preciso comprobar si la conexión sigue activa antes de invocar el método **Rollback** para deshacer la transacción. Si se invoca el método con la conexión cerrada se produce un error.

6.- Entity Framework

Los modelos conectado y desconectado enfrentan el modelo de datos relacional propio de las bases de datos (conformado por tablas, campos y relaciones), al modelo de objetos propio de .NET (conformado por clases, objetos y atributos).

En el modelo conectado, los datos se obtienen a través de un objeto *DataReader*, que accede en cada momento a la información de un registro y permite leer sus campos como una matriz de valores de tipo *Object*. En el modelo desconectado, los datos se obtienen en objetos *DataSet*, que aunque sí representan la estructura de los datos en forma de tablas, columnas, registros y relaciones; todos los datos son representados como objetos *DataRow* que contienen los valores en una matriz de valores de tipo *Object*. En ninguno de los casos se obtiene los datos en forma de objetos pertenecientes a clases que representen cada tipo de dato presente en la base de datos con atributos que simbolizen sus respectivos campos.

Por otro lado; tanto el modelo conectado como el desconectado requieren del uso de un lenguaje ajeno a Visual Basic, o C# para la definición de consultas: SQL. Esto implica una dependencia de la base de datos ya que no todas las bases de datos emplean totalmente el SQL estandarizado (ANSI-SQL). Por ejemplo: Microsoft SQL Server emplea Transact-SQL, mientras que Oracle emplea PL-SQL.

Entity Framework

El Entity Framework es una nueva plataforma basada en ADO.NET que permite el acceso, consulta y manipulación de bases de datos añadiendo dos características nuevas:

- **Los datos se obtienen y manipulan como objetos** → La información se obtiene directamente como objetos conforme al modelo orientado a objetos.
- **Las consultas y modificaciones integradas** → Se elimina la necesidad de utilizar SQL dependiendo de la base de datos a la que accede.

Entity Framework se basa por entero en el uso de *Modelos de Entidades*.

Un *modelo de entidades* es una representación que refleja el modelo relacional de una base de datos a la que se quiere acceder incluyendo todas las tablas, campos, relaciones y procedimientos almacenados que se desean utilizar. Este resulta la base sobre la que se crea el *proveedor de entidades*.

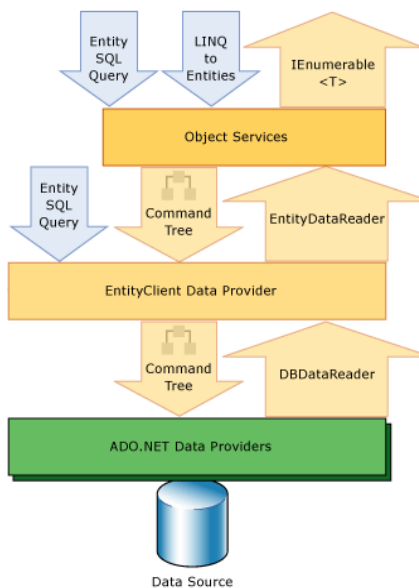


Izquierda.- Diagrama relacional de una base de datos desde SQL Server 2008.

Derecha .- Modelo de entidades generado en Visual Studio 2010 para Entity Framework

6.2.- El proveedor de entidades

ADO.NET emplea proveedores de datos en función de la base de datos que se emplea. *Entity Framework* emplea un único *proveedor de entidades* basado en el modelo de entidades e independiente de la base de datos.



Representación de la estructura funcional de Entity Framework

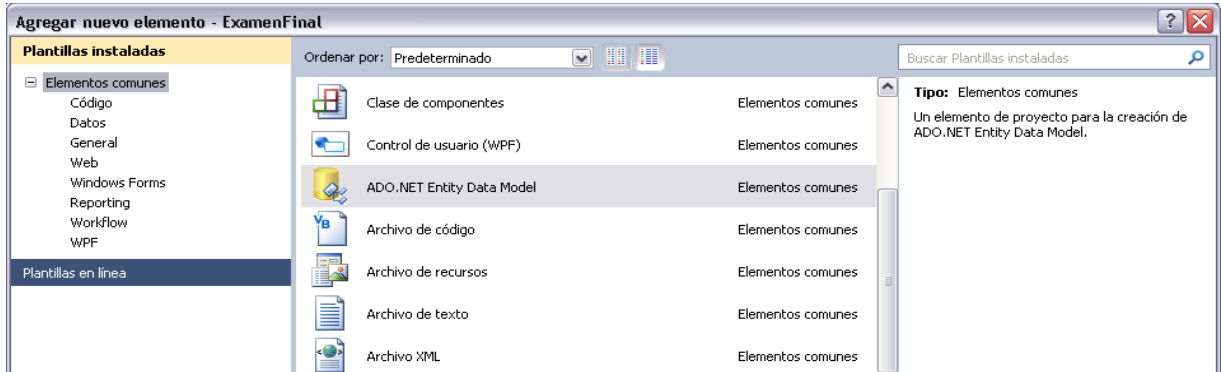
El proveedor de entidades (*Entity Client Data Provider*), se ocupa de administrar las conexiones a la base de datos, traducir las consultas integradas en consultas SQL y obtener los datos procedentes de la base de datos. ADO.NET define las clases específicas para este proveedor en el espacio de nombres **System.Data.EntityClient**.

El servicio de objetos (*Object Services*) permite la realización de consultas a la base de datos empleando *LINQ* o *EntitySQL*, y la recuperación de los datos en forma de objetos. Este también se ocupa de actualizar la base de datos con las modificaciones realizadas sobre los objetos obtenidos.

6.1.- Creación del modelo de entidades

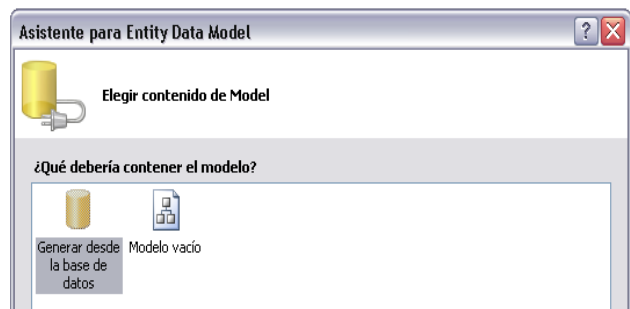
El modelo de entidad es un fichero con extensión *.edmx* que se incluye en el proyecto durante su diseño:

Proyecto → Agregar nuevo elemento...

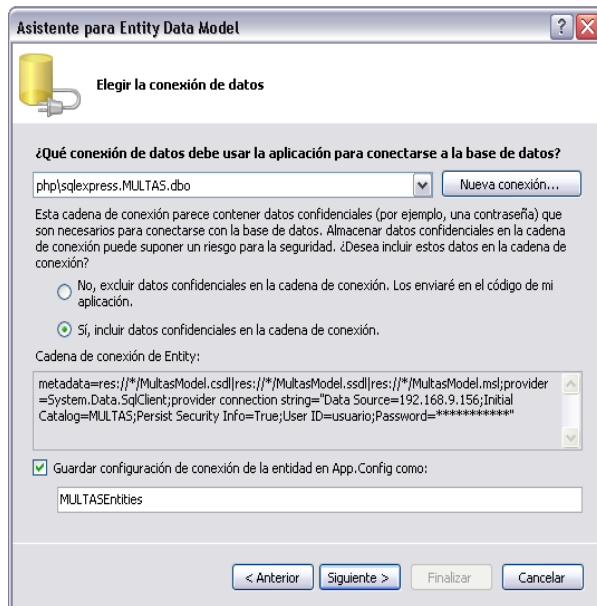


Podemos asignarle como nombre: *MultasModel.edmx*.

Indicar después que se va a crear el modelo en base a la estructura de una base de datos existente seleccionando la opción: **“Generar desde la base de datos.”**



Después hay que indicar la cadena de conexión a la base de datos que se va a utilizar. Puede crearse una nueva conexión o seleccionar la cadena de conexión de una base de datos conectada en el *Explorador de Servidores*.

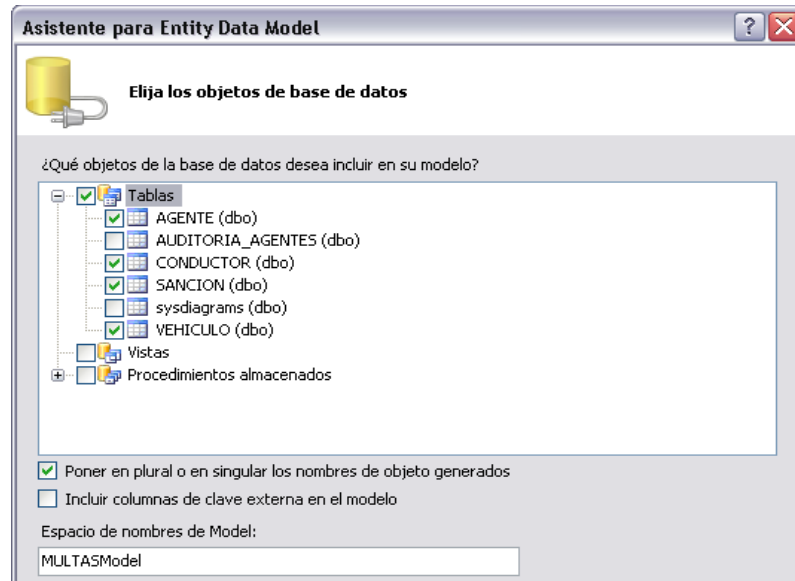


De manera opcional puede permitirse que se genere una cadena de conexión en el fichero *App.Config* con los parámetros necesarios para conectarse a la base de datos que se va a utilizar en la aplicación.

En este caso, guardamos la cadena de conexión con el nombre *MultasEntities*.

También es posible indicar si se desea que la clave de la cuenta de acceso a la base de datos se guarde junto con la cadena de conexión.

Una vez definida la conexión a la base de datos, hay que indicar los elementos de la base de datos (*tablas, relaciones, vistas, procedimientos almacenados*), que se desean añadir al modelo de entidades:



La casilla “Incluir columnas de clave externa en el modelo” permite indicar si se desean añadir como atributos los campos correspondientes a las claves externas.

La creación del modelo de entidades da lugar a un nuevo archivo con extensión .edmx visible en la ventana del Explorador de Soluciones.



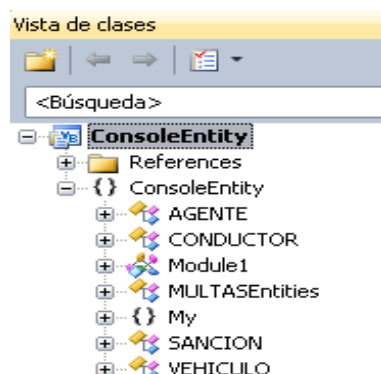
De igual modo, la creación del modelo también da lugar a una cadena de conexión a la base de datos almacenada en el archivo de configuración de la aplicación (*App.Config*):

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
  </configSections>
  <connectionStrings>
    <add name="MULTASEntities"
      connectionString="metadata=res://*/ModelMultas.csdl|res://*/ModelMultas.ssdl|res://*/ModelMultas.msl;provider=System.Data.SqlClient;provider_connection_string="data source=192.168.1.100;initial catalog=MULTAS;persist security info=True;user id=usuario;password=ci psa;multipleactiveresultsets=True;App=EntityFramework";
      providerName="System.Data.EntityClient" />
  </connectionStrings>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework, Version=v4.0, Profile=Client" />
  </startup>
</configuration>
```

Es destacable el hecho de que el proveedor de datos designado para la cadena de conexión no es un proveedor de datos convencional (*SqlClient, OracleClient, Odbc, OleDb*); sino el proveedor de entidades (*System.Data.EntityClient*) generado en base al modelo de entidades recién creado.

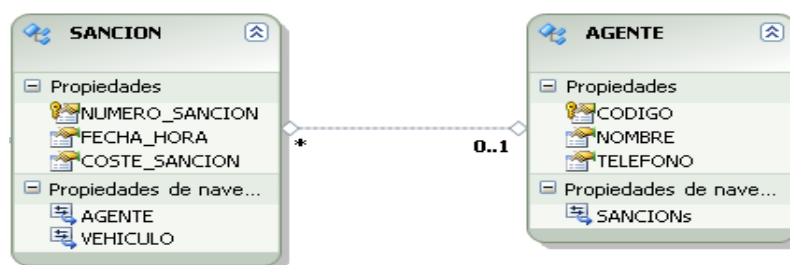
6.4.-El Contexto de Objetos y las Clases Entidad

Una vez creado el fichero *modelo de entidad* (.edmx) en el proyecto, aparecen varias clases nuevas en la *Vista de Clases*:



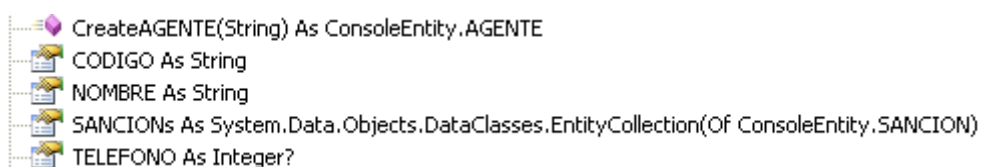
La clase *MULTASentities* constituye el *Contexto de Objetos*. Esta clase hereda de la clase base **ObjectContext** definida en el espacio de nombres **System.Data.Object**, y dispone de los métodos y propiedades necesarias para consultar y trabajar con las entidades del modelo como objetos.

Las clases *AGENTE*, *CONDUCTOR*, *SANCION* y *VEHICULO* son *Clases Entidad*. Todas las clases entidad heredan de la clase base **EntityObject** definida en el espacio de nombres **System.Data.Object**. Cada clase se corresponde con una entidad del modelo de entidades y define atributos para cada campo de la entidad correspondiente, así como *Propiedades de Navegación* por cada relación con otra entidad del modelo.



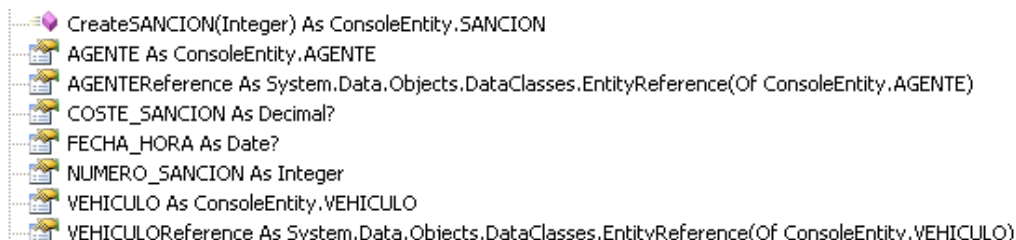
Propiedades de la clase entidad AGENTE:

La clase entidad *AGENTE* dispone de un atributo para cada campo de la entidad. La propiedad de navegación *SANCIONs* devuelve una colección de objetos *SANCION* que corresponde con todas las sanciones asociadas al agente (*aquellas que él interpuso*).



Propiedades de la clase entidad SANCION:

La clase entidad SANCION dispone de un atributo AGENTE que devuelve un objeto de la clase entidad AGENTE correspondiente al agente asociado con la sanción (*aquel que la interpuso*).



6.5.- Acceso a base de datos mediante Contexto de Objetos

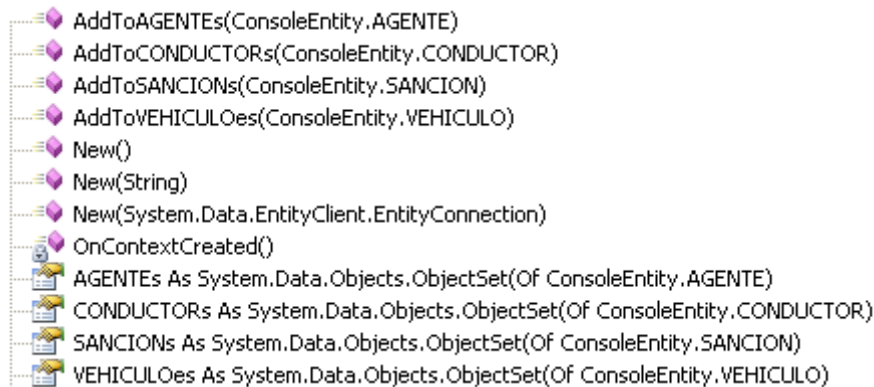
Para acceder a la base de datos empleando *Entity Framework* es necesario crear primero una instancia de la clase contexto de objetos. El constructor predeterminado emplea la cadena de conexión incluida en el archivo de configuración *app.config*.

```
' Instancia contexto empleando la cadena de conexion definida
' por el propio modelo de entidades.
Dim obj Contexto As New MULTASEntities()
```

También es posible crear el objeto indicando un objeto conexión de tipo *EntityConnection* asociado a una cadena de conexión diferente:

VB
<pre>' Creacion de objeto conexion de Entity Provider a base de datos. Dim conexion As New EntityConnection(My.Settings.CadenaConexion) ' Creacion de instancia del contexto de objetos Dim contexto As New MULTASEntities(conexion)</pre>
C# (código completo)
<pre>using System; using System.Collections.Generic; using System.Linq; using System.Text; using System.Data.EntityClient; using System.Configuration; namespace ConsoleApplication1 { class Program { public static void Main(String[] args) { EntityConnection conexion = new EntityConnection(ConfigurationManager.AppSettings["MULTASEntities"]); MULTASEntities contexto = new MULTASEntities(conexion); } } }</pre> <p>La obtención en C# de la cadena de conexión registrada en el archivo <i>app.config</i> con el identificador "MultasEntities" requiere la referencia de la librería <i>System.Configuration</i>.</p>

El *contexto de objetos* incluye además de los miembros heredados de **ObjectContext**, múltiples atributos y propiedades específicos:



Las propiedades AGENTEs, SANCIONs, CONDUCTORs Y VEHICULOes retornan colecciones genéricas del tipo **ObjectQuery(Of E)** con objetos de la clase entidad correspondiente. Así pues, la propiedad AGENTE, devuelve una colección *ObjectQuery(Of AGENTE)*.

6.6.- Consultas

Las consultas de datos pueden realizarse empleando dos lenguajes:

- **EntitySQL (eSQL)** → Lenguaje de SQL específico del proveedor de entidades que permite la llamada a procedimientos almacenados... etc . Para más información consultar la página de referencia de *Entity SQL* en el MSDN:

<http://msdn.microsoft.com/en-us/library/bb387118.aspx>

- **LINQ to Entities** → Lenguaje integrado de consultas LINQ.

El resultado de una consulta es un objeto de la colección genérica **System.Data.Objects.ObjectQuery(Of T)**, donde T puede ser una clase entidad o un tipo anónimo generado en base a los datos devueltos por resultado.

La colección *ObjectQuery* implementa el interfaz *IQueryable*, que hereda a su vez de la interfaz *IEnumerable*. Todo esto permite definir consulta con LINQ sobre la colección, así como recorrer su contenido empleando un bucle tipo FOR EACH.

Consultas empleando LINQ-Entity

Ejemplo 1: El siguiente código devuelve todos los agentes contenidos en la tabla AGENTES.

VB
<pre>Imports System.Data.Objects Module Module1 Sub Main() ' Obtencion del contexto Dim contexto As New MULTASEntities() ' Obtencion de la coleccion completa de la tabla AGENTE Dim resultados As ObjectQuery(Of AGENTE) = contexto.AGENTEs ' Visualizacion de conjunto de resultados For Each objAgente As AGENTE In resultados Console.WriteLine("CODIGO : " & objAgente.CODIGO) Console.WriteLine("NOMBRE : " & objAgente.NOMBRE) Console.WriteLine("TELEFONO : " & If(objAgente.TELEFONO.HasValue, objAgente.TELEFONO.ToString(), "SIN TELEFONO")) Next End Sub End Module</pre>
C#
<pre>using System.Data.Objects; namespace ConsoleApplication1 { class Program { public static void Main(String[] args) { MULTASEntities contexto = new MULTASEntities(); ObjectQuery<AGENTE> resultados = contexto.AGENTEs; foreach(AGENTE agente in resultados) { Console.WriteLine("CODIGO: " + agente.CODIGO); Console.WriteLine("NOMBRE: " + agente.NOMBRE); Console.WriteLine("TELEFONO: " + ((agente.TELEFONO.HasValue) ? "1" : "2")); } Console.ReadKey(); } } }</pre>

La propiedad AGENTE_s del objeto contexto devuelve una colección *ObjectQuery(of AGENTE)* con todos los objetos AGENTE correspondientes a los agentes registrados en la tabla AGENTES.

La propiedad TELEFONO de la clase entidad AGENTE es de tipo *Integer?* (*tipo nutable*), debido a que la columna en la tabla admite valores NULL. Debe por tanto comprobarse si la propiedad tiene un valor (*HasValue*) asignado antes de visualizarla.

Ejemplo 2: El siguiente código recupera los datos de todos los conductores registrados mayores de 25 años.

VB
<pre> ' Obtencion del contexto Dim contexto As New MULTASEntities() ' Obtencion de la coleccion completa de la tabla AGENTE Dim resultados As ObjectQuery(Of CONDUCTOR) = From conductor In contexto.CONDUCTORS Where conductor.EDAD > 25 Select conductor ' Visualizacion de conjunto de resultados For Each objConductor As CONDUCTOR In resultados With objConductor Console.WriteLine(.DNI) Console.WriteLine(.NOMBRE) Console.WriteLine(.EDAD.ToString()) End With Next </pre>
C#
<pre> MULTASEntities contexto = new MULTASEntities(); IEnumerable<CONDUCTOR> resultados = contexto.CONDUCTORS.Where(c => c.EDAD > 25); foreach(CONDUCTOR conductor in resultados) { Console.WriteLine("DNI: " + conductor.DNI); Console.WriteLine("NOMBRE: " + conductor.NOMBRE); Console.WriteLine("EDAD: " + conductor.EDAD.ToString()); } </pre>

Ejemplo 3: El siguiente código devuelve la fecha y cuantía de todas y cada una de las sanciones registradas en la tabla SANCIONES de la base de datos:

VB
<pre> ' Obtencion del contexto Dim contexto As New MULTASEntities() ' Obtencion de la coleccion completa de la tabla AGENTE Dim resultados = From sancion In contexto.SANCIONES Select sancion.FECHA_HORA, sancion.COSTE_SANCION ' Visualizacion de conjunto de resultados For Each objSancion In resultados Console.WriteLine(objSancion.FECHA_HORA.ToString()) Console.WriteLine(objSancion.COSTE_SANCION.ToString()) Next </pre>
C#
<pre> MULTASEntities contexto = new MULTASEntities(); var resultados = contexto.SANCIONES.Select(s => new { Fecha = s.FECHA_HORA, Coste = s.COSTE_SANCION }); foreach(var sancion in resultados) { Console.WriteLine("FECHA: " + sancion.Fecha.ToString()); Console.WriteLine("COSTE: " + sancion.Coste.ToString()); } </pre>

En este caso el conjunto de resultados es una colección *ObjectQuery* de tipo anónimo ya que se especifica un conjunto arbitrario de campos a devolver por cada resultado.

Ejemplo 4: El siguiente código muestra la fecha de imposición de todas las sanciones registradas con cuantías superiores a 1000€, indicando el nombre y telefono del agente que la impuso.

VB

```
' Obtencion del contexto
Dim contexto As New MULTASEntities()

' Obtencion de la coleccion completa de la tabla AGENTE
Dim resultados = From sancion In contexto.SANCIONES
                  Where sancion.COSTE_SANCION > 1000
                  Select sancion.FECHA_HORA, sancion.AGENTE

' Visualizacion de conjunto de resultados
For Each obj resultado In resultados
    ' Datos de la sancion
    Console.WriteLine(obj resultado.FECHA_HORA)
    ' Datos del agente.
    With obj resultado.AGENTE
        Console.WriteLine(.NOMBRE)
        Console.WriteLine(.TELEFONO)
    End With
Next
```

C#

```
MULTASEntities contexto = new MULTASEntities();
var resultados = contexto.SANCIONES.Where(s => s.COSTE_SANCION > 500)
    .Select(s => new
    {
        fecha = s.FECHA_HORA,
        agente = s.AGENTE
    });

AGENTE agente;
foreach (var resultado in resultados)
{
    Console.WriteLine("FECHA: " + resultado.fecha.Value.ToShortDateString());
    agente = resultado.agente;
    Console.WriteLine("NOMBRE AGENTE: " + agente.NOMBRE);
    Console.WriteLine("TELEFONO AGENTE: " + ((agente.TELEFONO.HasValue) ?
        agente.TELEFONO.Value.ToString() :
        "SIN TELEFONO"));
}
}
```

La consulta devuelve una colección con todos los objetos SANCION cuyo campo COSTE_SANCION supera el valor 1000. Para obtener la información del agente que impuso la sanción se utiliza la propiedad AGENTE, que devuelve el objeto AGENTE correspondiente.

Ejemplo 5: El siguiente código muestra la fecha y cuantía de todas las sanciones impuestas por el agente con código "0001":

VB

```
' Obtencion del contexto
Dim contexto As New MULTASEntities()

' Obtencion de la coleccion completa de la tabla AGENTE
Dim resultados As ObjectQuery(Of AGENTE) = From agente In contexto.AGENTES
Where agente.CODIGO.Equals("0001")
Select agente

If resultados.Count > 0 Then
    ' Visualizacion datos del AGENTE
    Dim objAgente As AGENTE = resultados.First()
    Console.WriteLine("NOMBRE AGENTE: " & objAgente.NOMBRE)
    ' Hay alguna sancion impuesta por el agente?
    If objAgente.SANCIIONS.Count > 0 Then
        ' Visualizacion de las SANCIIONES del agente.
        For Each objSancion As SANCION In objAgente.SANCIIONS
            With objSancion
                Console.WriteLine("Fecha sancion: " & .FECHA_HORA.ToString())
                Console.WriteLine("Coste: " & .COSTE_SANCION.ToString())
            End With
        Next
    Else
        Console.WriteLine("El agente no ha impuesto ninguna sanción.")
    End If
Else
    Console.WriteLine("No existe ningún agente con ese código.")
End If
```

C#

```
MULTASEntities contexto = new MULTASEntities();
IEnumerable<AGENTE> resultado = contexto.AGENTES
    .Where(a => a.CODIGO.Equals("0001"));

if (resultado.Count() > 0) {
    AGENTE agente = resultado.First();
    Console.WriteLine("NOMBRE: " + agente.NOMBRE);
    if (agente.SANCIIONS.Count > 0) {
        foreach (SANCION sancion in agente.SANCIIONS) {
            Console.WriteLine("Fecha sanción: " +
                sancion.FECHA_HORA.Value.ToShortDateString());
            Console.WriteLine("Coste sanción: " +
                sancion.COSTE_SANCION.ToString());
        }
    } else {
        Console.WriteLine("El agente no ha impuesto ninguna sanción.");
    }
} else {
    Console.WriteLine("No existe ningún agente con el código '0001'");
}
```

6.7.1.- Gestión de la conexión con la base de datos.

El manejo de la conexión está automatizado en Entity Framework por el proveedor de entidades, que recupera la información de la base de datos empleando objetos *DataReader*, por lo que la conexión permanece abierta únicamente mientras se utilizan los resultados de las consultas.

Ejemplo: El siguiente código recupera y muestra la información de todos los agentes registrados en la tabla AGENTE. Se han incluido líneas que muestran el estado de la conexión con la base de datos, antes, durante y después del recorrido de los datos:

```
' Obtencion del contexto
Using contexto As New MULTASentities()

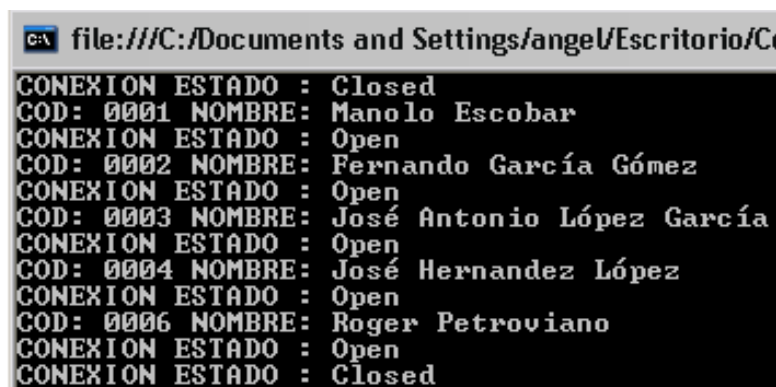
' Obtencion de la coleccion completa de la tabla AGENTE
Dim resultados As ObjectQuery(Of AGENTE) = contexto.AGENTES
Console.WriteLine("CONEXION ESTADO : " & contexto.Connection.State.ToString())

' Bucle de recorrido de agentes
For Each objAgente As AGENTE In resultados
    ' Visualizacion de datos de agente
    Console.WriteLine("COD: {0} NOMBRE: {1}", objAgente.CODIGO, objAgente.NOMBRE)
    Console.WriteLine("CONEXION ESTADO : " & contexto.Connection.State.ToString())
Next

Console.WriteLine("CONEXION ESTADO : " & contexto.Connection.State.ToString())
End Using
```

Cada iteración del bucle FOR-EACH provoca un avance del objeto *DataReader* empleado internamente por el proveedor de entidades (*Entity Provider*). Cada registro es a su vez convertido en un objeto con la ayuda del servicio de objetos (*Object Service*).

El resultado es que la conexión con la base de datos durante el tiempo que dure el recorrido de los datos:



```
file:///C:/Documents and Settings/angel/Escritorio/C
CONEXION ESTADO : Closed
COD: 0001 NOMBRE: Manolo Escobar
CONEXION ESTADO : Open
COD: 0002 NOMBRE: Fernando García Gómez
CONEXION ESTADO : Open
COD: 0003 NOMBRE: José Antonio López García
CONEXION ESTADO : Open
COD: 0004 NOMBRE: José Hernandez López
CONEXION ESTADO : Open
COD: 0006 NOMBRE: Roger Petroviano
CONEXION ESTADO : Open
CONEXION ESTADO : Closed
```

(*) El hecho de que los resultados del objeto *ObjectQuery* se basen en un objeto *DbDataReader*, implica que no es posible acceder directamente a un objeto del conjunto de resultados empleando métodos como **ElementAt** o el **indizador**. En su lugar es preciso recorrerlo, o emplear métodos como **First()**.

6.7.- Modificación de datos.

Entity Framework permite la modificación de los datos directamente creando, eliminando y modificando los objetos entidad del contexto de objetos. Sin embargo, estas modificaciones no provocan la actualización de la base de datos de manera automática.

Para actualizar la base de datos de modo que refleje las modificaciones, inserciones y eliminaciones realizadas es preciso el uso del método *SaveChanges* miembro del contexto de objetos. La llamada a este método provoca las siguientes operaciones:

- Se examinan los objetos recuperados buscando modificaciones. Para ello, se almacena un objeto **ObjectStateEntry** asociado con cada objeto entidad, que refleja si fue modificado, creado o eliminado.
- Se inicia una transacción para encapsular todas las modificaciones a realizar.
- Se generan y ejecutan sentencias SQL de actualización (INSERT, UPDATE, DELETE) para la base de datos en función del estado almacenado en el objeto *ObjectStateEntry* de cada objeto entidad. Estos objetos estado se almacenan en una colección accesible a través de la propiedad **ObjectStateManager** del contexto de objetos. Si todas las sentencias se ejecutan correctamente se confirma la transacción y pueden aceptarse los cambios mediante el método **AcceptAllChanges** miembro del contexto de objetos. En caso contrario se rechaza la transacción anulando todos los cambios realizados y dejando la base de datos en el estado original.

6.7.1.- Modificación.

La modificación de un campo se lleva a cabo modificando el valor de la propiedad correspondiente en el objeto entidad deseada. Las modificaciones pueden realizarse a nivel de un único registro o modificando varios al mismo tiempo.

Ejemplo: El siguiente código obtiene y modifica el nombre del agente cuyo código es 0001.

```
VB
' Obtencion del contexto
Using contexto As New MULTASentities()
    ' Obtencion de la coleccion completa de la tabla AGENTE
    Dim resultados As ObjectQuery(Of AGENTE) = From agente In contexto.AGENTEs
                                                Where agente.CODIGO.Equals("0001")
                                                Select agente

    ' Hay resultados?
    If resultados.Count > 0 Then
        ' Obtencion del agente
        Dim objAgente As AGENTE = resultados.First()
        objAgente.NOMBRE = "Ivan Petrovich"
    End If
    Try
        ' Actualizacion
        Dim regs As Integer = contexto.SaveChanges()
        Console.WriteLine("OK. {0} registros modificados", regs)
        ' Aceptación de cambios
        contexto.AcceptAllChanges()
    Catch ex As Exception
        Console.WriteLine("ERROR: {0} - {1}", ex.GetType().ToString(), ex.Message)
    End Try
    Console.ReadKey()
End Using
```

C#

```
MULTASentites contexto = new MULTASentites();
IEnumerable<AGENTE> resultados = contexto.AGENTES
    .Where(a => a.CODIGO.Equals("0001"));

if (resultados.Count() > 1)
{
    AGENTE agente = resultados.First();
    agente.NOMBRE = "Ivan Petrovich";
}
try
{
    int regs = contexto.SaveChanges();
    Console.WriteLine("OK. Registros modificados: {0}", regs.ToString());
    contexto.AcceptAllChanges();
}
catch (Exception ex)
{
    Console.WriteLine("ERROR: {0} - {1}", ex.GetType().ToString(), ex.Message);
}
```

Ejemplo 2: El siguiente código obtiene y aumenta en 100€ la cuantía de todas las sanciones impuestas por el agente “Ivan Petrovich”

VB

```
' Obtencion del contexto
Using contexto As New MULTASentites()

    ' Obtencion de la coleccion completa de la tabla AGENTE
    Dim resultados As ObjectQuery(Of SANCION) =
        From sancion In contexto.SANCIONS
        Where sancion.AGENTE.NOMBRE.Equals("Ivan Petrovich")
        Select sancion

    ' Hay resultados?
    If resultados.Count > 0 Then
        ' Actualizacion de sanciones seleccionadas
        For Each objSancion As SANCION In resultados
            objSancion.COSTE_SANCION += 100
        Next
    End If
    Try
        ' Actualizacion
        Dim regs As Integer = contexto.SaveChanges()
        Console.WriteLine("OK. {0} registros modificados", regs)
        ' Aceptación de cambios
        contexto.AcceptAllChanges()
    Catch ex As Exception
        Console.WriteLine("ERROR: {0} - {1}", ex.GetType().ToString(), ex.Message)
    End Try
    Console.ReadKey()
End Using
```

C#

```
using (MULTASentites contexto = new MULTASentites())
{
    IEnumerable<SANCION> resultados = contexto.SANCIONS
        .Where(s => s.AGENTE.NOMBRE.Equals("Ivan Petrovich"));
    foreach (SANCION sancion in resultados)
    {
        sancion.COSTE_SANCION += 100;
    }
    try
    {
        int regs = contexto.SaveChanges();
        Console.WriteLine("Registros modificados: {0}", regs.ToString());
        contexto.AcceptAllChanges();
    }
    catch (Exception ex)
    {
        Console.WriteLine("ERROR: {0} - {1}", ex.GetType().ToString(),
            ex.Message);
    }
}
```

6.7.2.- Inserción de registros

La inserción de nuevos registros pasa por la creación de nuevos objetos entidad. Para ello deben emplearse el método **CreateXXXX** de cada clase entidad. Este método funciona a modo de constructor recibiendo tantos parámetros como campos obligatorios requiere la entidad y devolviendo el nuevo objeto entidad.

Ejemplo: El siguiente código crea un nuevo agente de nombre "Roger Petrov", con código "0004" y teléfono "5558960":

La clase entidad AGENTE define el método miembro CreateAGENTE que recibe como parámetro el código del nuevo agente, puesto que es el único campo obligatorio exigido en la tabla AGENTES. (*los demás campos admiten valores NULL*).

```
' Obtencion del contexto
Using contexto As New MULTASEntities()

Dim nuevoAgente As AGENTE = AGENTE.CreateAGENTE("0007")
nuevoAgente.NOMBRE = "Roger Petrov"
nuevoAgente.TELEFONO = "5558690"
```

Una vez obtenido el nuevo objeto entidad es necesario añadirlo a la colección correspondiente en el contexto de objetos empleando el método **AddObject**. En este caso, el nuevo objeto entidad agente debe añadirse a la colección *AGENTES* del contexto, que es la que representa a los agentes almacenados en la base de datos.

```
' Inclusion de agente en la coleccion de agentes del contexto.
contexto.AGENTES.AddObject(nuevoAgente)
```

Finalmente debe llamarse al método **SaveChanges** para actualizar la base de datos.

```
Try
    ' Actualizacion
    Dim regs As Integer = contexto.SaveChanges()
    Console.WriteLine("OK. {0} registros modificados", regs)

    ' Aceptación de cambios
    contexto.AcceptAllChanges()

Catch ex As Exception

    Console.WriteLine("ERROR: {0} - {1}", ex.GetType().ToString(), ex.Message)

End Try
End Using
```

Debe tenerse en cuenta que si el nuevo registro no puede insertarse en la base de datos, el método *UpdateChanges* producirá una excepción de tipo **UpdateException**. En el caso de una inserción este error es común cuando se introduce un valor ya existente en el atributo que sirve de clave primaria.

6.7.3.- Inserción de asociaciones

La inserción de registros asociados a otros existentes en la base de datos requiere que las propiedades de navegación de los objetos entidad se configuren como corresponde.

Ejemplo: En la base de datos MULTAS, cada sanción se asocia de manera única con un vehículo (contra el que se impone la sanción), y un agente (quien la impone). En el modelo de entidad la relación se muestra como:



El siguiente código inserta una nueva sanción impuesta por el agente con código "0001" al vehículo matrícula "3258CGX".

Lo primero es obtener los objetos correspondientes al agente y al vehículo sancionado:

```
' Obtencion del contexto
Using contexto As New MULTASEntities()

' Seleccion de agente
Dim objAgente As AGENTE = (From agente In contexto.AGENTES
Where agente.CODIGO.Equals("0001")
Select agente).First()

' Seleccion de vehiculo
Dim objVehiculo As VEHICULO = (From vehiculo In contexto.VEHICULOes
Where vehiculo.MATRICULA.Equals("3258CGX")
Select vehiculo).First()
```

Después se crea el objeto entidad correspondiente a la nueva sanción y se añade a la colección de sanciones del contexto de objetos. Como cada sanción se relaciona con un vehículo y un agente, se asignan los objetos *objAgente* y *objSancion* a las propiedades *AGENTE* y *VEHICULO* del objeto sanción.

```
Dim nuevaSancion As SANCION = SANCION.CreateSANCION(20)
With nuevaSancion
    .AGENTE = objAgente
    .VEHICULO = objVehiculo
    .FECHA_HORA = DateTime.Now
    .COSTE_SANCION = 120.5
End With
contexto.SANCIONs.AddObject(nuevaSancion)
```

Una vez configurado el nuevo objeto sanción, se añade a la colección SANCIONs del contexto de objetos para ser insertado en la base de datos.

Finalmente se actualizan todos los cambios en la base de datos llamando al método *SaveChanges* del contexto de objetos:

```
Try
    ' Actualización
    Dim regs As Integer = contexto.SaveChanges()
    Console.WriteLine("OK. {0} registros modificados", regs)
    ' Aceptación de cambios
    contexto.AcceptAllChanges()
Catch ex As Exception
    Console.WriteLine("ERROR: {0} - {1}", ex.GetType().ToString(), ex.Message)
    Console.WriteLine(ex.InnerException.Message)
End Try
```

La ejecución del código dará lugar a dos sentencias de modificación (UPDATE), y una de inserción (INSERT) contra la base de datos

6.7.5.- Eliminación de registros

La eliminación de objetos provoca la eliminación de las filas correspondientes en la base de datos. Un objeto entidad se marca para eliminar pasándolo como argumento al método **DeleteObject()** del contexto de objetos.

Ejemplo: El siguiente código elimina de la base de datos la sanción con el campo número sancion = 4.

VB

```
' Obtención del contexto
Using contexto As New MULTASEntities()
    ' Selección de la sanción
    Dim obj Sancion As SANCION = (From sancion In contexto.SANCIONS
                                   Where sancion.NUMERO_SANCION = 4
                                   Select sancion).First()

    ' Marcado para eliminar.
    contexto.SANCIONS.DeleteObject(obj Sancion)
    Try
        ' Actualización
        Dim regs As Integer = contexto.SaveChanges()
        Console.WriteLine("OK. {0} registros modificados", regs)
        ' Aceptación de cambios
        contexto.AcceptAllChanges()
    Catch ex As Exception
        Console.WriteLine("ERROR: {0} - {1}", ex.GetType().ToString(), ex.Message)
        Console.WriteLine(ex.InnerException.Message)
    End Try
    Console.ReadKey()
End Using
```

C#

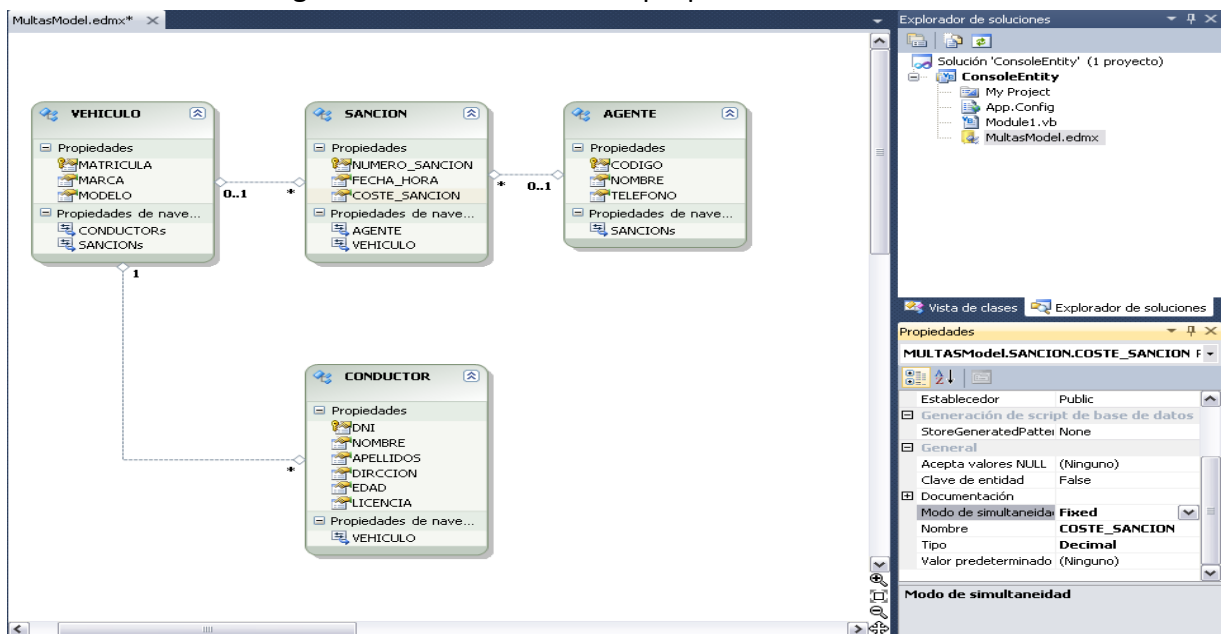
```
using (MULTASEntities contexto = new MULTASEntities())
{
    SANCION obj Sancion = contexto.SANCIONS
        .Where(s => s.NUMERO_SANCION == 4).First();
    contexto.SANCIONS.DeleteObject(obj Sancion);
    try
    {
        int regs = contexto.SaveChanges();
        Console.WriteLine("OK. Registros modificados: {0} ", regs);
        contexto.AcceptAllChanges();
    }
    catch (Exception ex)
    {
        Console.WriteLine("ERROR: {0} - {1}", ex.GetType().ToString(),
                           ex.Message);
        Console.WriteLine(ex.InnerException.Message);
    }
}
```


Nota: Entity Framework no admite eliminaciones en cascada. Si se desea eliminar un registro con registros asociados en otras tablas, es preciso eliminar primero los registros asociados antes de eliminar el principal.

6.8.- Problemas de Concurrencia

Entity Framework no comprueba de manera predeterminada la concurrencia. Esto permite la modificación de datos sin comprobar si han sido modificados posteriormente a su lectura. Sin embargo; puede habilitarse la comprobación de concurrencia en aquellas entidades susceptibles de sufrir problemas de concurrencia, es decir; aquellas que son leídas y modificadas por múltiples usuarios al mismo tiempo.

La detección de concurrencia para un determinado campo de una entidad se habilita en el modelo de entidad asignando el valor 'Fixed' a la propiedad "Modo de simultaneidad":



Modificación del modo de simultaneidad del campo COSTE_SANCION de la entidad SANCION a modo 'Fixed'.

El método *SaveChanged* del contexto de objetos informa del error de concurrencia lanzando una excepción de tipo **OptimisticConcurrencyException**. Una vez detectada puede llamarse al método **Refresh** para resolver el conflicto:

Public Sub **Refresh**(refreshMode As [System.Data.Objects.RefreshMode](#), collection As [System.Collections.IEnumerable](#))

El método `Refresh` recibe como segundo parámetro (*collection*) la colección de resultados con las modificaciones a actualizar. El primer parámetro (*refreshMode*) es un valor del tipo enumerado ***System.Data.Object.RefreshMode*** que determina como resolver el conflicto con uno de sus valores posibles:

- ***StoreWins*** : Determina que prevalecen los datos existentes en la base de datos. La colección de resultados se actualiza con los datos existentes en la base de datos.
- ***ClientWins*** : Determina que prevalecen las modificaciones a actualizar. En la siguiente llamada al método `SaveChanges` los datos son actualizados a la fuerza en la base de datos.

Ejemplo: El siguiente código incrementa en 100€ todas las sanciones registradas teniendo en cuenta los posibles problemas de concurrencia que pueden darse al modificar la cuantía de las sanciones por parte de otros usuarios.

```
VB
' Obtencion del contexto
Using contexto As New MULTASEntities()

    ' Obtencion de consulta + modificaciones
    Dim sanciones As ObjectQuery(Of SANCION) = contexto.SANCIONs
    For Each objSancion As SANCION In sanciones
        objSancion.COSTE_SANCION += 100
    Next
    Try
        ' Actualizacion
        Dim regs As Integer = contexto.SaveChanges()
        Console.WriteLine("OK. {0} registros modificados", regs)
        contexto.AcceptAllChanges()

        Catch ex As OptimisticConcurrencyException

            Console.WriteLine("Se ha producido una excepcion de concurrencia")
            Console.WriteLine("Sobrescribir datos? S/N")
            Dim respuesta As ConsoleKeyInfo = Console.ReadKey()
            If (respuesta.Key = ConsoleKey.S) Then

                ' Se confirman las modificaciones y se sobrescribe la base de datos.
                contexto.Refresh(RefreshMode.ClientWins, sanciones)
                ' Actualizacion
                Dim regs As Integer = contexto.SaveChanges()
                Console.WriteLine("OK. {0} registros modificados", regs)

            Else

                ' Se descartan los cambios y se recargan desde la base de datos.
                contexto.Refresh(RefreshMode.StoreWins, sanciones)
                Dim regs As Integer = contexto.SaveChanges()
                Console.WriteLine("OK. {0} registros modificados", regs)

            End If

        End Try
    End Using
```

Si en el tiempo transcurrido entre la obtención de los datos y la actualización de las sanciones, otro usuario o aplicación modifica en la base de datos la cuantía de alguna de las sanciones el método `SaveChanges()` lanza la excepción *OptimisticCurrencyException*. La excepción se captura encapsulando el método `SaveChange` en una estructura TRY-CATCH.

C#

```

using (MULTASEntities contexto = new MULTASEntities())
{
    IEnumerable<SANCION> resultados = contexto.SANCIONES
        .Where(s => s.AGENTE.NOMBRE.Equals("Ivan Petrovich"));
    foreach (SANCION sancion in resultados)
    {
        sancion.COSTE_SANCION += 100;
    }
    try
    {
        int regs = contexto.SaveChanges();
        Console.WriteLine("Registros modificados: {0}", regs.ToString());
        contexto.AcceptAllChanges();
    }
    catch (OptimisticConcurrencyException ex)
    {
        Console.WriteLine("Error de concurrencia detectado.");
        Console.WriteLine("Sobrescribir datos? S/N: ");
        ConsoleKeyInfo respuesta = Console.ReadKey();
        if (respuesta.Key == ConsoleKey.S) {
            // Se confirman las modificaciones y se sobrescribe la base de datos.
            contexto.Refresh(RefreshMode.ClientWins, resultados);
            // Actualización
            int regs = contexto.SaveChanges();
            Console.WriteLine("OK. {0} registros modificados", regs);
        } else {
            // Se descartan los cambios y se recargan desde la base de datos
            contexto.Refresh(RefreshMode.StoreWins, resultados);
            int regs = contexto.SaveChanges();
            Console.WriteLine("OK. {0} registros modificados", regs);
        }
    }
}

```

En caso de querer sobrescribir la base de datos a pesar del problema de concurrencia detectado; se llama al método *Refresh* indicando el modo *ClientWins*, y a continuación al método *SaveChanges()* para actualizar la base de datos a la fuerza. Para respetar las modificaciones realizadas por otras personas se llama al método *Refresh* indicando el modo *StoreWins*. Esto recarga los objetos de la colección sanciones con los datos actuales de la base de datos.

Nota: Debe tenerse en cuenta que para que se detecte el problema de concurrencia al modificar el campo *COSTE_SANCION* de la entidad *SANCION* es necesario haberle asignado el valor *Fixed* a su propiedad simultaneidad en el modelo de entidades.

7.- Vinculación de Controles

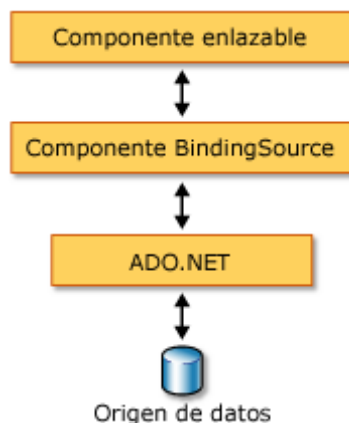
7.1.- Concepto de Vinculación.

La vinculación de datos (*DataBinding*) consiste en vincular controles de Windows tales como cajas de texto, listas, y tablas, con los resultados de una consulta contra una base de datos.

La vinculación de controles a datos permite que los éstos se muestren automáticamente en los controles sin necesidad de copiarlos mediante código. Del mismo modo, las modificaciones de los valores de los controles modifican automáticamente los datos vinculados

7.2.- Orígenes de Vinculación

Se llama *origen de vinculación* a un objeto de la clase **System.Windows.Forms.BindingSource** que hace de intermediario entre los controles de un formulario y los resultados procedentes de una base de datos encapsulados en un objeto DataSet, o una colección de objetos.



Para ello se asigna el objeto con el conjunto de resultados obtenidos de la base de datos al objeto origen de vinculación, y a continuación se vinculan los controles del formulario al objeto BindingSource que hace de origen de datos para ellos.

Una vez establecido el objeto origen de vinculación; todas las operaciones de interacción con los datos, incluido el desplazamiento, la ordenación, el filtrado y la actualización, se lleva a cabo mediante los métodos del objeto BindingSource

Las propiedades que deben configurarse para vincular un objeto origen de vinculación a un conjunto de resultados son:

- **DataSource:** Esta propiedad obtiene o establece el conjunto de resultados obtenidos de la base de datos. Puede ser un objeto DataSet, DataTable, una colección, o un conjunto de objetos entidad.
- **DataMember:** Esta propiedad obtiene o establece el nombre de un miembro del origen de datos; normalmente el nombre de una tabla contenida en el DataSet.

Ejemplo: El siguiente código muestra la recuperación de todos los registros presentes en la tabla CONDUCTOR de la base de datos MULTAS y su asociación a un objeto origen de vinculación.

En este código se emplea como conjunto de resultados un objeto DataSet obtenido a partir del método Fill de un adaptador de datos de SQL Server:

VB
<pre> ' Recuperación de datos de la base de datos Dim conexion As New SqlConnection(My.Settings.MULTASConexi on) Dim adaptador As New SqlDataAdapter("SELECT * FROM CONDUCTOR", conexion) Dim dset As New DataSet() adaptador.Fill(dset, "CONDUCTORES") ' Establecimiento de origen de vinculacion para controles Dim vinculo As New BindingSource() vinculo.DataSource = dset vinculo.DataMember = "CONDUCTORES" </pre>
C#
<pre> using System; using System.Collections.Generic; using System.ComponentModel; using System.Data; using System.Drawing; using System.Linq; using System.Text; using System.Windows.Forms; using System.Data.SqlClient; namespace WindowsFormsAplicacion2 { public partial class Form1 : Form { // Atributos private SqlDataAdapter adaptador; private BindingSource vinculo; private DataSet dset; public Form1() { InitializeComponent(); // Inicializacion con obtencion de datos SqlConnection conexion = new SqlConnection(Properties.Settings.Default.MultasConexi on); adaptador = new SqlDataAdapter("SELECT * FROM CONDUCTOR", conexion); dset = new DataSet(); adaptador.Fill(dset, "CONDUCTORES"); // Establecimiento del vinculo a datos con conjunto de datos 'dset' vinculo = new BindingSource(dset, "CONDUCTORES"); } } } </pre>

El mismo caso pero empleando como conjunto de resultados una colección de objetos entidad CONDUCTOR obtenidos mediante Entity Framework empleando un modelo de entidades basado en la base de datos MULTAS:

```

' Inicializacion del contexto de objetos
contexto = New MULTASEntities()
' Obtención del conjunto de resultados con todos los conductores
' registrados en la base de datos.
Dim todosConductores = contexto.CONDUCTORS
' Establecimiento de origen de vinculacion para controles
vinculo = New BindingSource()
vinculo.DataSource = todosConductores

```

En este caso no es necesaria la asignación de valor a la propiedad *DataMember*.

En ambos casos el objeto origen de vinculación se asocia a un conjunto de resultados procedentes de la base de datos. Una vez declarado y configurado el objeto `BindingSource`, éste se emplea de igual modo independientemente del tipo de conjunto de resultados que se utilice.

7.3.- Vinculación de controles a orígenes de vinculación

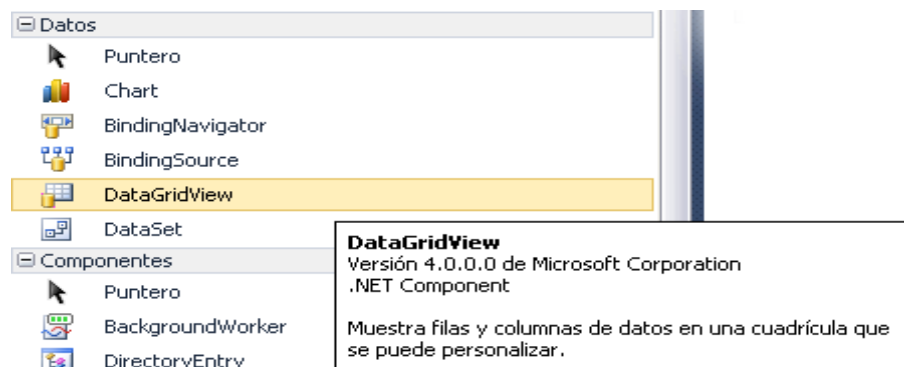
Para conectar los controles a un objeto origen de vinculación `BindingSource` existen tres mecanismos en función del tipo de control a vincular:

- **Vínculo a Propiedad:** Permite vincular una propiedad de un control a un valor obtenido desde la base de datos.
- **Vínculo a Listas:** Permite vincular un control de tipo Lista (*ListBox*, *ComboBox...* etc), a un columna de un conjunto de resultados obtenidos desde la base de datos.
- **Vínculo a Tablas:** Permite vincular un control de tipo Tabla (*GridView*), a un grupo de columnas de un conjunto de resultados obtenidos desde la tabla de base.

7.3.1.- Vinculación a Tablas

Este es el caso del control *DataGridView*. Este control permite mostrar múltiples columnas de un conjunto registros recuperados de la base de datos mediante una rejilla de celdas organizadas en filas y columnas.

El control `DataGridView` se encuentra situado en cuadro de herramientas, bajo la pestaña Datos:



Para vincular un control `DataGridView` a un origen de datos se emplean sus propiedades:

- **DataSource:** Objeto origen de vinculación (`DataBinding`). Opcionalmente también puede asignársele directamente el conjunto de resultados (objetos *DataTable*, *DataSet* de ADO.NET, listas y colecciones de objetos, y conjuntos de resultados de *Entity Framework*).
- **DataMember:** Propiedad que contiene el nombre de la tabla con los datos a mostrar en el caso de que se indice un objeto `DataSet` como origen de datos.

Ejemplo: El siguiente código recupera todos los campos de los registros almacenados en la tabla CONDUCTORES, y los vincula a un control *DataGridView* a través de un objeto *BindingSource*.

```

VB

Dim conexion As New SqlConnection(My.Settings.MULTASCConexion)
Dim adaptador As New SqlDataAdapter("SELECT DNI,
                                     NOMBRE,
                                     APELLIDOS,
                                     DIRECCION,
                                     MATRICULA,
                                     EDAD,
                                     LICENCIA
                                     FROM CONDUCTOR", conexion)

' Recuperación de datos de la base de datos
adaptador.Fill(dset, "CONDUCTORES")

' Establecimiento de origen de vinculación para controles
vinculo = New BindingSource()
vinculo.DataSource = dset
vinculo.DataMember = "CONDUCTORES"

' Vinculación de control DataGridView a objeto origen de vinculación
Me.DataGridView1.DataSource = vinculo

C#

// Inicialización con obtención de datos
SqlConnection conexion = new
    SqlConnection(Properties.Settings.Default.MULTASCConexion);
SqlDataAdapter adaptador = new SqlDataAdapter("SELECT
DNI, NOMBRE, APELLIDOS, DIRECCION, MATRICULA, EDAD, LICENCIA FROM CONDUCTOR", conexion);
DataSet dset = new DataSet();
adaptador.Fill(dset, "CONDUCTORES");

// Establecimiento del vínculo a datos con conjunto de datos 'dset'
vinculo = new BindingSource(dset, "CONDUCTORES");

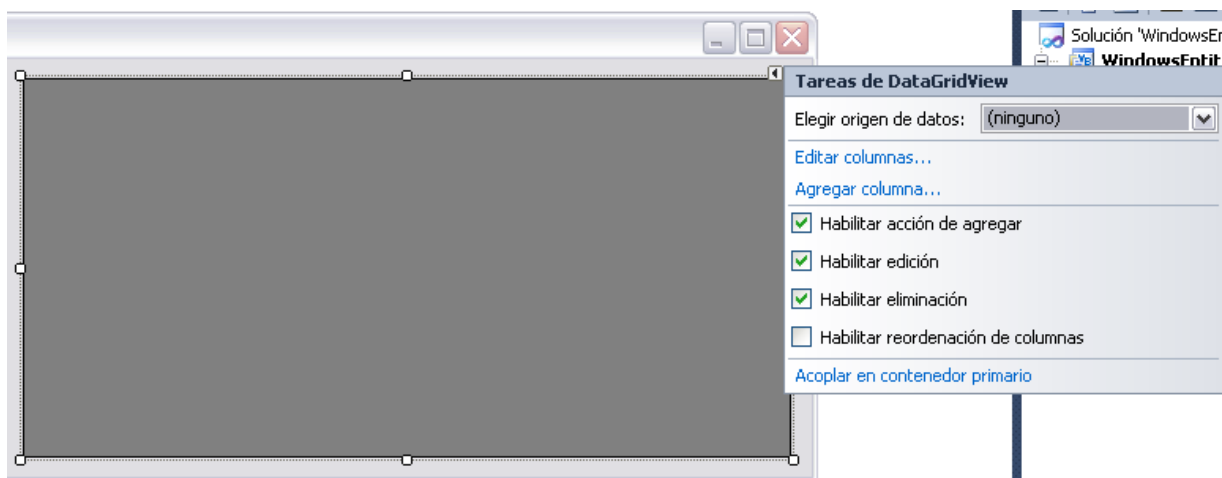
// Vinculación a control DataGridView
this.dataGridView1.DataSource = vinculo;
    
```

El control *DataGridView* una vez vinculado muestra por defecto todas las columnas de los registros contenidos en el objeto *DataTable* vinculado del siguiente modo:


	DNI	NOMBRE	APELLIDOS	DIRECCION	MATRICULA	EDAD	LICENCIA
▶	23670002V	ROSA	GARCIA MURCIA	C/ ALFONSO, 12...	4568CCX	23	A
	34567123P	ÓSCAR	GARCIA LÓPEZ	C/ ASTURIAS, 5...	6527DBC	24	B
	34567543N	VANESSA	GONZÁLEZ PRIETO...	C/ CARLOS I, 23...	B1548WS	32	B
	34890234S	Mª CARMEN	LÓPEZ HERRERA ...	AVDA. VIDAL, 3...	AB3698Z	35	B
	45123456C	DAVID	ESTÉVEZ MOLINA	C/ BUENOS AIR...	M1234ZY	39	C
	45345220B	JESÚS	GONZÁLEZ VÁZQU...	C/ BONAVISTA, ...	IB3678L	41	C
	51101001R	JAIME	GARCÍA DE LA CRU...	C/ BUENOS AIR...	5842FFV	22	D
	54234000W	JESÚS	GÓMEZ BAENA	C/ ALFONSO, 23...	3258CGX	53	B
	56900001M	ÓSCAR	ALCÁNTARA BAEN...	PSD. BUENA VE...	2548BBV	48	A

Aspecto de un control *DataGridView* mostrando el conjunto de resultados vinculado

El control DataGridView dispone de muchas propiedades que permiten indicar qué datos se verán en el control, cómo se verán y qué operaciones puede hacerse sobre ellos directamente desde el control.

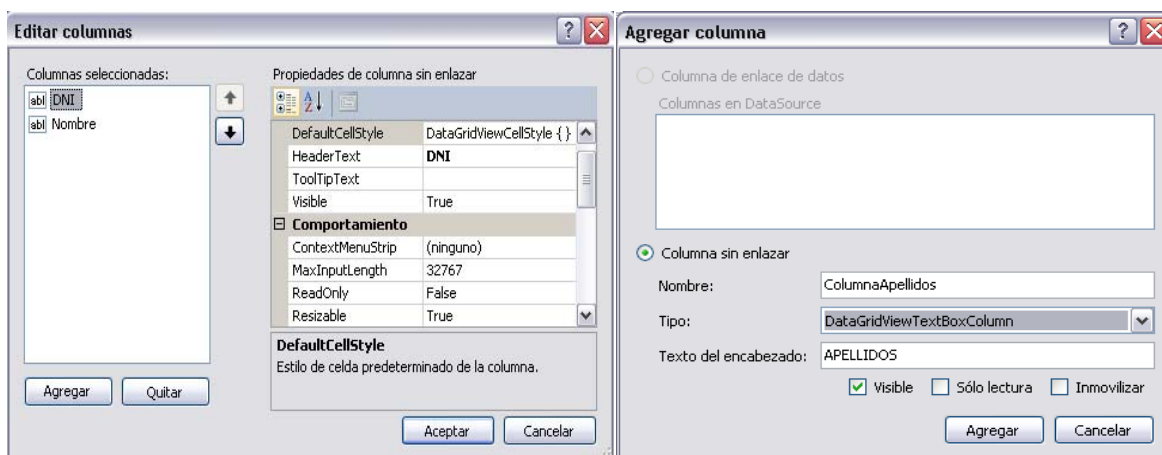


Vista del cuadro de Tareas del control DataGridView en diseño.

Para configurar las columnas mostradas por el control DataGridView puede emplearse el cuadro de “*Tareas de DataGridView*”. Este muestra al hacer clic sobre el ícono que aparece en el extremo superior derecho al seleccionar el control en diseño. 

La opción “*Editar Columnas....*”, permite indicar las columnas que tendrá el control. Por defecto, el control genera de manera automática las columnas necesarias para mostrar todos los campos del conjunto de resultados al que se vincula. La edición manual de las columnas permite indicar qué datos se verán, en qué columnas y de qué manera:

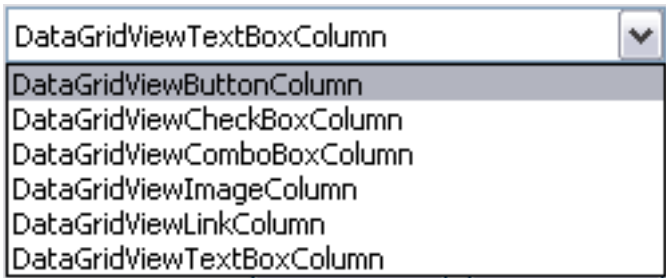
El cuadro “*Editar Columnas*” muestra las columnas definidas en el control junto con sus propiedades. Los botones “*Agregar*” y “*Quitar*” permiten añadir y eliminar columnas al control.



Izquierda.- Cuadro de diálogo de edición de columnas de un control DataGridView.

Derecha.- Cuadro de agregado de nueva columna.

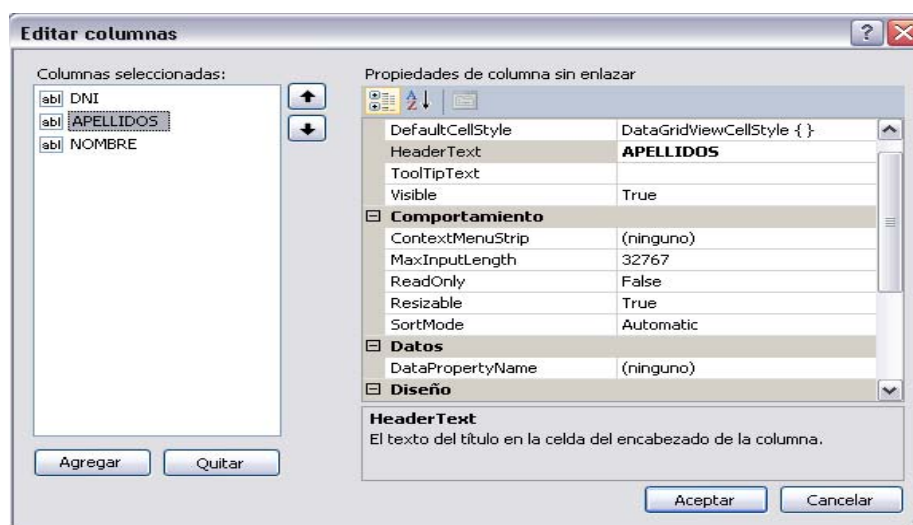
Al pulsar el botón “Agregar” se muestra el cuadro “Agregar Columna”. Este permite indicar varias características de la nueva columna:

Propiedad	Significado
Nombre	Identificador del objeto columna que permitirá la manipulación de la misma desde código.
Tipo	Indica el tipo de columna: 
Texto de encabezado	Texto que se muestra en la cabecera de la columna
Visible	Indica si la columna es o no visible.
Sólo lectura	Indica si los valores mostrados en la columna pueden ser modificados por el usuario.
Inmovilizar	Indica si la columna puede ser redimensionada o cambiada de sitio por el usuario.

En función del tipo de columna que se seleccione, la celda aparece representada por un control diferente según el tipo:

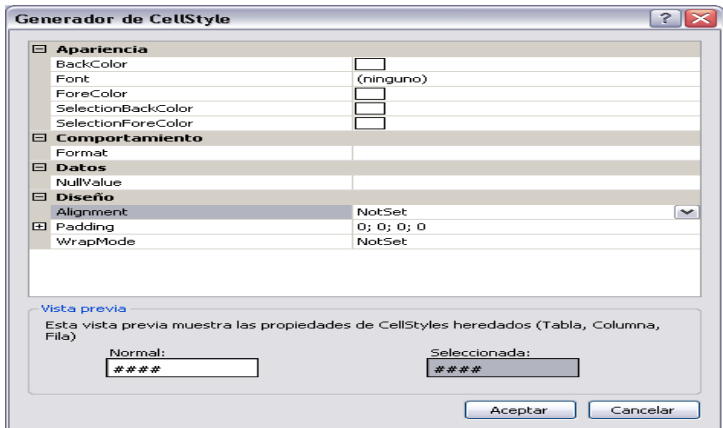
- **DataGridViewButtonColumn** → La celda aparece representando un control botón.
- **DataGridViewCheckBoxColumn** → La celda aparece representando una casilla de verificación.
- **DataGridViewComboBoxColumn** → La celda aparece conteniendo una lista desplegable.
- **DataGridViewImageColumn** → La celda aparece mostrando una imagen
- **DataGridViewLinkColumn** → La celda aparece mostrando un enlace web
- **DataGridViewTextBoxColumn** → La celda aparece mostrando una caja de texto editable.

Cada tipo de columna tiene sus propias propiedades. Estas se muestran en la parte derecha del cuadro “Editar Columnas”:



El tipo más común de columna es *DataGridViewTextBoxColumn* que permite mostrar cada celda como una caja de texto editable. El tipo de columna *DataGridViewCheckBoxColumn* suele emplearse para la representación de valores de carácter booleano.

Cada columna de un *DataGridView* es a su vez un objeto con su propio identificador y propiedades. Algunas de estas propiedades pueden configurarse en diseño desde la ventana de edición de columnas:

Propiedad	Significado
(Name)	Indica el identificador del objeto columna en el código para poder acceder a sus propiedades.
HeaderText	Indica el texto del encabezamiento de la columna
Visible	Indica si la columna es o no visible
ReadOnly	Indica si las celdas de la columna son editables
Resizable	Indica si las columnas son redimensionables por el usuario
AutoSizeMode	Indica si la columna se mantiene de un ancho fijo, o si se redimensiona para dar cabida a los valores a mostrar
DataPropertyName	Indica el nombre del campo o propiedad correspondiente al campo del conjunto de resultados cuyos valores se mostrarán en la columna. Si se emplea como conjunto de resultados un objeto de la clase <i>DataSet</i> , o <i>DataTable</i> ; el valor de esta propiedad debe coincidir con el nombre de una columna de la consulta <i>SELECT</i> . Si se emplea como conjunto de resultados una colección de objetos entidad; el valor debe coincidir con el nombre de una propiedad de los objetos
DefaultCellStyle	<p>Esta propiedad permite determinar características del estilo de celda para la columna tales como color de fondo, color de texto, tipo de letra, y colores de selección:</p>  <p>Ventana de edición de estilo de celdas → Propiedad DefaultCellStyle</p> <p>Las propiedades Format y NullValue permiten indicar el formato en que deben mostrarse los valores vinculados, y el valor en caso de un campo nulo.</p>

Los objetos columnas de un control DataGridView disponen además de muchas más propiedades que son accesibles y modificables desde código en tiempo de ejecución.

Ejemplo: El siguiente código agrega por código al control DataGridView tres columnas de tipo cajas de texto (*DataGridViewTextBoxColumn*). Para cada una de las columnas es preciso indicar como mínimo el identificador de cabecera (*HeaderText*), el nombre de la propiedad de vinculación (*DataPropertyName*):

VB

```
' Indicación de columnas para el control DataGridView
With Me.DataGridView1
    ' Columna vinculada al campo DNI con cabecera "DNI" y anchura = 100px
    .Columns.Add(New DataGridViewTextBoxColumn()
        With { .HeaderText = "DNI",
               .DataPropertyName = "DNI",
               .Width = 100}
    )
    ' Columna vinculada al campo NOMBRE con cabecera "NOMBRE" y anchura = 100px
    .Columns.Add(New DataGridViewTextBoxColumn()
        With { .HeaderText = "NOMBRE",
               .DataPropertyName = "NOMBRE",
               .Width = 100}
    )
    ' Columna vinculada al campo APELLIDOS con cabecera "APELLIDOS" y anchura = 200px
    .Columns.Add(New DataGridViewTextBoxColumn()
        With { .HeaderText = "APELLIDOS",
               .DataPropertyName = "APELLIDOS",
               .Width = 200}
    )
End With

' Desactivado de la generación automática de columnas para el DataGridView
Me.DataGridView1.AutoGenerateColumns = False
' Vinculación al objeto origen de vinculación BindingSource
Me.DataGridView1.DataSource = vinculo
```

C#

```
namespace WindowsFormsApplication2
{
    public partial class Form1 : Form
    {
        // Atributos
        private SqlDataAdapter adaptador;
        private BindingSource vinculo;
        private DataSet dset;

        public Form1()
        {
            InitializeComponent();

            // Inicialización con obtención de datos
            SqlConnection conexion = new
            SqlConnection(Properties.Settings.Default.MultasCConexion);
            adaptador = new SqlDataAdapter("SELECT DNI, NOMBRE, APELLIDOS FROM CONDUCTOR",
            conexion);
            dset = new DataSet();
            adaptador.Fill(dset, "CONDUCTORES");

            // Establecimiento del vínculo a datos con conjunto de datos 'dset'
            vinculo = new BindingSource(dset, "CONDUCTORES");

            // Declaración de columna 1.
            this.dataGridView1.Columns.Add(
                new DataGridViewTextBoxColumn()
                {
                    HeaderText = "DNI",
                    DataPropertyName = "DNI",
                    Width = 100
                }
            );
            // Declaración de columna 2.
            this.dataGridView1.Columns.Add(
                new DataGridViewTextBoxColumn()
                {
                    HeaderText = "NOMBRE",
                    DataPropertyName = "NOMBRE",
                    Width = 100
                }
            );
        }
    }
}
```

```

    });
    // Declaracion de columna 3
    this.dataGridView1.Columns.Add(
        new DataGridViewTextBoxColumn()
        {
            HeaderText = "APELLIDOS",
            DataPropertyName = "APELLIDOS",
            Width = 100
        }
    );

    // Desactivar generacion automatica de columnas.
    this.dataGridView1.AutoGenerateColumns = false;
    // Vinculacion a control DataGridView
    this.dataGridView1.DataSource = vinculo;
}
}
}

```

El resultado mostrado en el formulario será:



	DNI	NOMBRE	APELLIDOS	
▶	23670002V	ANA	GARCIA MURCI...	
	34567123P	ÓSCAR	GARCIA LÓPEZ ...	
	34567543N	VANESSA	GONZÁLEZ PRI...	
	34890234S	Mª CARMEN	LÓPEZ HERRE...	
	45123456C	DAVID	ESTÉVEZ MOLI...	
	45345220B	JESÚS	GONZÁLEZ VÁZ...	

7.3.2.- Vinculación a Listas

Este tipo de vinculación se emplea con controles de tipo Lista (`ListBox`, `ComboBox...`). Estos controles permiten mostrar los valores de una determinada columna de un conjunto de resultados.

Para vincular el control lista deben configurarse las siguientes propiedades:

- **DataSource:** Objeto origen de vinculación (`DataBinding`). Opcionalmente también puede asignársele directamente el conjunto de resultados (objetos `DataTable`, `DataSet` de ADO.NET, listas y colecciones de objetos, y conjuntos de resultados de *Entity Framework*).
- **DisplayMember:** Nombre de la columna cuyos valores serán mostrados al usuario como representación de cada dato.
- **ValueMember:** Nombre de la columna cuyos valores serán almacenados conjuntamente con cada valor mostrado en la lista para operaciones internas. (p.ej: el valor de *clave primaria*).

Ejemplo: El siguiente código todos los campos de los registros almacenados en la tabla CONDUCTORES, y los vincula a un control ListBox mostrando el nombre de cada conductor y almacenando como valor su DNI.

VB
<pre> ' Recuperación de datos de la base de datos conexion = New SqlConnection(My.Settings.MULTASCConexion) adaptador = New SqlDataAdapter("SELECT DNI, NOMBRE, APELLIDOS, DIRECCION, MATRICULA, EDAD, LICENCIA FROM CONDUCTOR", conexion) adaptador.Fill(dset, "CONDUCTORES") ' Establecimiento de origen de vinculacion para controles vinculo = New BindingSource() vinculo.DataSource = dset vinculo.DataMember = "CONDUCTORES" ' Vinculacion de control ListBox a objeto origen de vinculacion Me.ListBox.DataSource = vinculo Me.ListBox.DisplayMember = "NOMBRE" ' Nombre de la columna con los valores a mostrar Me.ListBox.ValueMember = "DNI" ' Nombre de la columna con valores a almacenar </pre>
C#
<pre> // Inicialización con obtención de datos SqlConnection conexion = new SqlConnection(Properties.Settings.Default.MULTASCConexion); adaptador = new SqlDataAdapter("SELECT DNI, NOMBRE, APELLIDOS FROM CONDUCTOR", conexion); dset = new DataSet(); adaptador.Fill(dset, "CONDUCTORES"); // Establecimiento del vínculo a datos con conjunto de datos 'dset' vinculo = new BindingSource(dset, "CONDUCTORES"); // Vínculo a lista this.ListBox1.DataSource = vinculo; this.ListBox1.DisplayMember = "NOMBRE"; this.ListBox1.ValueMember = "DNI"; </pre>

El control ListBox muestra los nombres de todos los conductores recuperados de la base de datos.

<div> <div>ROSA</div> <div>ÓSCAR</div> <div>VANESSA</div> <div>Mª CARMEN</div> <div>DAVID</div> <div>JESÚS</div> <div>JAIME</div> <div>JESÚS</div> <div>ÓSCAR</div> <div>ANA</div> <div>Mª CARMEN</div> </div>	<table> <tr> <th>DNI</th><th>NOMBRE</th></tr> <tr> <td>23670002V</td><td>ROSA</td></tr> <tr> <td>34567123P</td><td>ÓSCAR</td></tr> <tr> <td>34567543N</td><td>VANESSA</td></tr> <tr> <td>34890234S</td><td>Mª CARMEN</td></tr> <tr> <td>45123456C</td><td>DAVID</td></tr> <tr> <td>45345220B</td><td>JESÚS</td></tr> <tr> <td>51101001R</td><td>JAIME</td></tr> <tr> <td>54234000W</td><td>JESÚS</td></tr> <tr> <td>56900001M</td><td>ÓSCAR</td></tr> </table>	DNI	NOMBRE	23670002V	ROSA	34567123P	ÓSCAR	34567543N	VANESSA	34890234S	Mª CARMEN	45123456C	DAVID	45345220B	JESÚS	51101001R	JAIME	54234000W	JESÚS	56900001M	ÓSCAR
DNI	NOMBRE																				
23670002V	ROSA																				
34567123P	ÓSCAR																				
34567543N	VANESSA																				
34890234S	Mª CARMEN																				
45123456C	DAVID																				
45345220B	JESÚS																				
51101001R	JAIME																				
54234000W	JESÚS																				
56900001M	ÓSCAR																				

Aspecto del control ListBox vinculado a la columna Nombre.

Para obtener el valor asociado al elemento seleccionado en la lista por el usuario debe emplearse la propiedad *SelectedValue*.

VB
<pre> Private Sub ListBox_Click(sender As System.Object, e As System.EventArgs) Handles ListBox.Click Dim valorSeleccionado As Object = Me.ListBox.SelectedValue If Not valorSeleccionado Is Nothing Then MessageBox.Show("El DNI es: " & valorSeleccionado.ToString()) End If End Sub </pre>

C#

```
private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    // Algún elemento seleccionado?
    if (this.listBox1.SelectedValue != null)
    {
        // Obtención del valor asociado a la opción seleccionada en la lista.
        String DNI_Seleccionado = this.listBox1.SelectedValue as string;
        MessageBox.Show("El DNI seleccionado es : " + DNI_Seleccionado);
    }
}
```

7.3.3.- Vinculación Simple

Los controles Windows disponen todos de una propiedad *DataBindings*. Esta propiedad contiene una colección de objetos *Binding*, cada uno de los cuales representa un vínculo de una propiedad del control con un dato.

Un objeto vinculación se crea llamando al constructor de la clase *Binding*:

```
Public Sub New(propertyName As String, dataSource As Object, dataMember As String)
```

- **Nombre de la propiedad:** Propiedad del control que se vincula.
- **Origen de datos:** Objeto origen de vinculación (*DataBinding*). Opcionalmente también puede asignársele directamente el conjunto de resultados (objetos *DataTable*, *DataSet* de ADO.NET, listas y colecciones de objetos, y conjuntos de resultados de *Entity Framework*).
- **Miembro de datos:** El nombre de la columna a cuyo valor se vincula la propiedad.

El origen de datos debe ser un objeto con la colección de valores recuperados de la base de datos. Este puede ser un objeto *DataTable*, una matriz o lista de objetos, o también un objeto origen de vinculación de la clase *BindingSource*.

Una vez creado el objeto vínculo éste puede añadirse a la colección de vínculos del control empleando el método *Add* de la colección *DataBindings*.

Ejemplo: El siguiente código recupera de la base de datos MULTAS todos los conductores registrados en la tabla CONDUCTOR. Después se vincula la propiedad Text del control txtDNI a la columna DNI.

VB

```
' Recuperación de datos de la base de datos
conexion = New SqlConnection(My.Settings.MULTASConexion)
adaptador = New SqlDataAdapter("SELECT DNI, NOMBRE, APELLIDOS, DIRECCION, MATRICULA,
EDAD, LICENCIA FROM CONDUCTOR", conexion)
adaptador.Fill(dset, "CONDUCTORES")

' Establecimiento de origen de vinculación para controles
vinculo = New BindingSource()
vinculo.DataSource = dset
vinculo.DataMember = "CONDUCTORES"

' Vinculo de la propiedad Text del control, a la columna DNI del registro activo
Me.txtDni.DataBindings.Add(New Binding("Text", vinculo, "DNI"))
```

C#

```
// Inicialización con obtención de datos
SqlConnection conexion = new
    SqlConnection(Properties.Settings.Default.MultasCConexion);
adaptador = new SqlDataAdapter("SELECT DNI, NOMBRE, APELLIDOS FROM CONDUCTOR",
                                conexion);

dset = new DataSet();
adaptador.Fill(dset, "CONDUCTORES");

// Establecimiento del vínculo a datos con conjunto de datos 'dset'
vinculo = new BindingSource(dset, "CONDUCTORES");

// Vinculación del campo DNI del registro activo a la caja de texto 'TextBox1'
this.textBox1.DataBindings.Add(new Binding("Text", vinculo, "DNI"));
```

7.4.- Selección de registro activo

Los controles como cajas de texto que emplean vinculación simple son únicamente capaces de mostrar el valor de una columna de un determinado registro. El registro mostrado depende del registro activo en el origen de vinculación.

El registro activo es un registro del conjunto de resultados que es mostrado y puede ser modificado directamente desde los controles vinculados. El control sobre este registro se lleva a cabo empleando las siguientes propiedades y métodos de la clase *BindingSource*:

Miembro	Acción
Count	Devuelve un valor numérico con el total de registros recuperados contenidos en el conjunto de resultados asociado al objeto origen de vinculación.
Position	Devuelve el índice del registro activo sobre el total de registros recuperados, siendo 0 el primer registro.
MoveNext	Desplaza el registro activo al siguiente del registro actual.
MovePrevious	Desplaza el registro activo al anterior del registro actual.
MoveFirst	Desplaza el registro activo al primero del conjunto de resultados.
MoveLast	Desplaza el registro activo al último del conjunto de resultados.
Current	Devuelve el objeto que representa el registro activo.

Ejemplo: Sea el siguiente formulario:

Este formulario muestra 7 cajas de texto que están vinculadas a las columnas de la tabla CONDUCTORES de la base de datos MULTAS.

Las cajas de texto muestran el valor de cada campo del registro activo en cada momento. Los botones en la parte inferior del formulario permiten desplazar este registro a través del conjunto de resultados, hacia delante, atrás, al primero y al último. La etiqueta situada en el centro muestra la posición relativa del registro activo entre el total de registros contenidos en el conjunto de resultados vinculados.

```

VB
Imports System.Data.SqlClient
Public Class Form2

    Private vinculo As BindingSource
    Private conexion As SqlConnection
    Private adaptador As SqlDataAdapter
    Private dset As New DataSet()

    Private Sub Form2_Load(sender As System.Object, e As System.EventArgs) Handles MyBase.Load
        ' Recuperación de datos de la base de datos
        conexion = New SqlConnection(My.Settings.MULTASConexion)
        adaptador = New SqlDataAdapter("SELECT DNI, NOMBRE, APELLIDOS, DIRECCION, MATRICULA, EDAD, LICENCIA FROM CONDUCTOR", conexion)
        adaptador.Fill(dset, "CONDUCTORES")

        ' Establecimiento de origen de vinculacion para controles
        vinculo = New BindingSource()
        vinculo.DataSource = dset
        vinculo.DataMember = "CONDUCTORES"

        ' Vinculo de la propiedad Text del control, a la columna DNI del registro activo
        Me.txtDni.DataBindings.Add(New Binding("Text", vinculo, "DNI"))
        Me.txtNombre.DataBindings.Add(New Binding("Text", vinculo, "NOMBRE"))
        Me.txtApellidos.DataBindings.Add(New Binding("Text", vinculo, "APELLIDOS"))
        Me.txtDireccion.DataBindings.Add(New Binding("Text", vinculo, "DIRECCION"))
        Me.txtMatricula.DataBindings.Add(New Binding("Text", vinculo, "MATRICULA"))
        Me.txtEdad.DataBindings.Add(New Binding("Text", vinculo, "EDAD"))
        Me.txtLicencia.DataBindings.Add(New Binding("Text", vinculo, "LICENCIA"))
    End Sub

    ' Boton de avance al siguiente registro
    Private Sub btnSiguiente_Click(sender As System.Object, e As System.EventArgs) Handles btnSiguiente.Click
        ' Comprobacion de que no estamos en el último registro del conjunto de resultados.
        If Me.vinculo.Position < Me.vinculo.Count - 1 Then
            Me.vinculo.MoveNext()
        End If
        Me.lblPosicion.Text = vinculo.Position.ToString() & "/" & vinculo.Count
    End Sub

    ' Boton de retroceso al registro anterior
    Private Sub btnAnterior_Click(sender As System.Object, e As System.EventArgs) Handles btnAnterior.Click
        ' Comprobacion de que no estamos en el primer registro del conjunto de resultados.
        If Me.vinculo.Position > 0 Then
            Me.vinculo.MovePrevious()
        End If
        Me.lblPosicion.Text = vinculo.Position.ToString() & "/" & vinculo.Count
    End Sub

    ' Boton de salto al primer registro del conjunto de resultados
    Private Sub btnPrimero_Click(sender As System.Object, e As System.EventArgs) Handles btnPrimero.Click
        Me.vinculo.MoveFirst()
        Me.lblPosicion.Text = vinculo.Position.ToString() & "/" & vinculo.Count
    End Sub

    ' Boton de salto al último registro del conjunto de resultados
    Private Sub btnFinal_Click(sender As System.Object, e As System.EventArgs) Handles btnFinal.Click
        Me.vinculo.MoveLast()
        Me.lblPosicion.Text = vinculo.Position.ToString() & "/" & vinculo.Count
    End Sub
End Class

```


C#

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

using System.Data.SqlClient;

namespace WindowsFormsAplicacion2
{
    public partial class Form1 : Form
    {
        // Atributos
        private SqlDataAdapter adaptador;
        private BindingSource vinculo;
        private DataSet dset;

        public Form1()
        {
            InitializeComponent();

            // Inicializacion con obtencion de datos
            SqlConnection conexion = new
            SqlConnection(Properties.Settings.Default.MultasCConexion);
            adaptador = new SqlDataAdapter("SELECT DNI, NOMBRE, APELLIDOS, DIRCCION,
MATRICULA, EDAD, LICENCIA FROM CONDUCTOR", conexion);
            dset = new DataSet();
            adaptador.Fill(dset, "CONDUCTORES");

            // Establecimiento del vinculo a datos con conjunto de datos 'dset'
            vinculo = new BindingSource(dset, "CONDUCTORES");

            // Vinculacion del campo DNI del registro activo a la caja de texto 'TextBox1'
            this.txtDNI.DataBindings.Add(new Binding("Text", vinculo, "DNI"));
            this.txtNombre.DataBindings.Add(new Binding("Text", vinculo, "NOMBRE"));
            this.txtApellido.DataBindings.Add(new Binding("Text", vinculo, "APELLIDOS"));
            this.txtDireccion.DataBindings.Add(new Binding("Text", vinculo, "DIRCCION"));
            this.txtMatricula.DataBindings.Add(new Binding("Text", vinculo, "MATRICULA"));
            this.txtEdad.DataBindings.Add(new Binding("Text", vinculo, "EDAD"));
            this.txtLicencia.DataBindings.Add(new Binding("Text", vinculo, "LICENCIA"));
        }

        // Desplazamiento registro siguiente
        private void btnSiguiete_Click(object sender, EventArgs e)
        {
            if (this.vinculo.Position < this.vinculo.Count)
            {
                this.vinculo.MoveNext();
                this.lblPosicion.Text = vinculo.Position.ToString() + "/" +
vinculo.Count.ToString();
            }
        }

        // Desplazamiento registro anterior
        private void btnAnterior_Click(object sender, EventArgs e)
        {
            if (this.vinculo.Position > 0)
            {
                this.vinculo.MovePrevious();
                this.lblPosicion.Text = vinculo.Position.ToString() + "/" +
vinculo.Count.ToString();
            }
        }

        // Desplazamiento a primer registro
        private void btnPrimero_Click(object sender, EventArgs e)
        {
            this.vinculo.MoveFirst();
            this.lblPosicion.Text = vinculo.Position.ToString() + "/" +
vinculo.Count.ToString();
        }

        // Desplazamiento a ultimo registro
        private void btnUltimo_Click(object sender, EventArgs e)
        {
            this.vinculo.MoveLast();
            this.lblPosicion.Text = vinculo.Position.ToString() + "/" +
vinculo.Count.ToString();
        }
    }
}

```

El control del *registro activo* también puede llevarse a cabo mediante la selección en controles de tipo List y Tabla vinculados al objeto origen de vinculación.

Si un control de tipo Lista o Tabla está vinculado a un objeto origen de vinculación, la selección de un elemento o de una fila respectivamente establece el registro vinculado como el registro activo para todos los demás controles vinculados.

Ejemplo: Sea el siguiente formulario variante del anterior:

El control ListBox de la derecha está vinculado al mismo objeto origen de vinculación que el resto de cajas de texto:

```
Private Sub Form2_Load(sender As System.Object, e As System.EventArgs) Handles MyBase.Load
    ' Recuperación de datos de la base de datos
    conexion = New SqlConnection(My.Settings.MULTASCONEXION)
    adaptador = New SqlDataAdapter("SELECT DNI, NOMBRE, APELLIDOS, DIRECCION, MATRICULA,
EDAD, LICENCIA FROM CONDUCTOR", conexion)
    adaptador.Fill(dset, "CONDUCTORES")
    ' Establecimiento de origen de vinculación para controles
    vinculo = New BindingSource()
    vinculo.DataSource = dset
    vinculo.DataMember = "CONDUCTORES"
    ' Vinculación de control ListBox a objeto origen de vinculación
    Me.ListBox1.DataSource = vinculo
    Me.ListBox1.DisplayMember = "NOMBRE" ' Nombre de la columna con los valores a mostrar
    Me.ListBox1.ValueMember = "DNI" ' Nombre de la columna con los valores a almacenar
    ' Vinculo de la propiedad Text del control, a la columna DNI del registro activo
    Me.txtDni.DataBindings.Add(New Binding("Text", vinculo, "DNI"))
    Me.txtNombre.DataBindings.Add(New Binding("Text", vinculo, "NOMBRE"))
    Me.txtApellido.DataBindings.Add(New Binding("Text", vinculo, "APELLIDOS"))
    Me.txtDireccion.DataBindings.Add(New Binding("Text", vinculo, "DIRECCION"))
    Me.txtMatricula.DataBindings.Add(New Binding("Text", vinculo, "MATRICULA"))
    Me.txtEdad.DataBindings.Add(New Binding("Text", vinculo, "EDAD"))
    Me.txtLicencia.DataBindings.Add(New Binding("Text", vinculo, "LICENCIA"))
End Sub
```

El control ListBox hace de selector de registro activo para el origen de vinculación. Cuando se selecciona un elemento, el registro correspondiente pasa a ser el registro activo, y las cajas de texto muestran automáticamente el valor del resto de sus campos.

Para obtener el registro seleccionado de un objeto *BindingSource* puede emplearse su propiedad **Current**.

La propiedad **Current** devuelve un valor de tipo *Object*. El valor será un objeto representante del registro activo en el conjunto de resultados. El valor de la propiedad puede consultarse cada vez que cambia el registro activo del objeto BindingSource atendiendo el evento *PositionChanged*.

Si empleamos como conjunto de resultados una colección de objetos entidad de Entity Framework, el valor devuelto por la propiedad Current será un objeto de la clase entidad correspondiente:

```
Private Sub Form2_Load(sender As System.Object, e As System.EventArgs) Handles MyBase.Load
    ' Inicialización del contexto de objetos
    contexto = New MULTASEntities()
    ' Obtención del conjunto de resultados con todos los conductores
    ' registrados en la base de datos.
    Dim todosConductores = contexto.CONDUCTORES

    ' Establecimiento de origen de vinculación para controles
    vinculo = New BindingSource()
    vinculo.DataSource = todosConductores
    ' Asignación de función manejadora al evento PositionChanged del objeto vinculo
    AddHandler vinculo.PositionChanged, AddressOf VinculoPositionChanged

    ' Vinculación de control ListBox a objeto origen de vinculación
    Me.ListBox.DataSource = vinculo
    Me.ListBox.DisplayMember = "NOMBRE" ' Nombre de la columna con los valores a mostrar
    Me.ListBox.ValueMember = "DNI" ' Nombre de la columna con los valores a almacenar

    ' Vinculo de la propiedad Text del control, a la columna DNI del registro activo
    Me.txtDni.DataBindings.Add(New Binding("Text", vinculo, "DNI"))
    Me.txtNombre.DataBindings.Add(New Binding("Text", vinculo, "NOMBRE"))
    Me.txtApellido.DataBindings.Add(New Binding("Text", vinculo, "APELLIDOS"))
    Me.txtDireccion.DataBindings.Add(New Binding("Text", vinculo, "DIRECCION"))
    Me.txtEdad.DataBindings.Add(New Binding("Text", vinculo, "EDAD"))
    Me.txtLicencia.DataBindings.Add(New Binding("Text", vinculo, "LICENCIA"))
End Sub

Private Sub VinculoPositionChanged(sender As Object, e As EventArgs)
    ' Obtención del objeto entidad correspondiente al conductor seleccionado
    Dim conductorSeleccionado As CONDUCTOR = CType(vinculo.Current, CONDUCTOR)
    ' Visualización del DNI
    MessageBox.Show("El conductor seleccionado tiene DNI: " & conductorSeleccionado.DNI)
End Sub
```

Si empleamos por otro lado un objeto DataSet, o DataTable como conjunto de resultados; el valor devuelto por la propiedad Current será un objeto de la clase **DataRowView**.

```
Private Sub Form2_Load(sender As System.Object, e As System.EventArgs) Handles MyBase.Load
    ' Recuperación de datos de la base de datos
    conexion = New SqlConnection(My.Settings.MULTASConexion)
    adaptador = New SqlDataAdapter("SELECT DNI, NOMBRE, APELLIDOS, DIRECCION, MATRICULA,
EDAD, LICENCIA FROM CONDUCTOR", conexion)
    adaptador.Fill(dset, "CONDUCTORES")

    ' Establecimiento de origen de vinculación para controles
    vinculo = New BindingSource()
    vinculo.DataSource = dset
    vinculo.DataMember = "CONDUCTORES"
    ' Asignación de función manejadora al evento PositionChanged del objeto vinculo
    AddHandler vinculo.PositionChanged, AddressOf VinculoPositionChanged
    ...
End Sub

Private Sub VinculoPositionChanged(sender As Object, e As EventArgs)
    ' Obtención del objeto entidad correspondiente al conductor seleccionado
    Dim registro As DataRow = CType(vinculo.Current, DataRowView).Row
    ' Visualización del DNI
    MessageBox.Show("El conductor seleccionado tiene DNI: " & registro.Item("DNI"))
End Sub
```

En C# para asociar una función manejadora al evento PositionChanged del objeto origen de vinculación se emplea la siguiente sintaxis:

```
// Establecimiento del vínculo a datos con conjunto de datos 'dset'
vínculo = new BindingSource(dset, "CONDUCTORES");
// Asignación de función manejadora a evento 'PositionChanged' del
// objeto origen de vinculación 'vínculo'.
vínculo.PositionChanged += new EventHandler(vínculo_PositionChanged);
```

La función manejadora permite obtener los datos vinculados al registro activo en ese momento. En el caso de emplear un DataSet como conjunto de resultados; el código necesario para obtener el objeto DataRow asociado al registro activo sería:

```
void vínculo_PositionChanged(object sender, EventArgs e)
{
    DataRow drow = (vínculo.Current as DataRowView).Row;
    MessageBox.Show("El DNI del registro activo es: " + drow["dni"].ToString());
}
```

En el caso de emplear como conjunto de resultados una colección de objetos entidad, el código sería:

```
void vínculo_PositionChanged(object sender, EventArgs e)
{
    CONDUCTOR conductor = vínculo.Current as CONDUCTOR;
    MessageBox.Show("El DNI del registro activo es: " + conductor.DNI.ToString());
}
```

7.5.- Modificación desde objeto origen de vinculación

El vínculo entre los controles vinculados y el conjunto de resultados obtenido desde la base de datos a través de un objeto origen de vinculación es bidireccional. Esto significa que los valores recuperados de la base de datos pueden ser modificados directamente desde controles vinculados con capacidad de edición tales como cajas de texto (*TextBox*) o controles tabla (*DataGridView*).

La clase *BindingSource* dispone de cuatro métodos para controlar la actualización de los datos contenidos en un conjunto de resultado con los valores modificados de controles vinculados:

Método	Acción
EndEdit()	Confirma las modificaciones realizadas en los controles vinculados, aplicando los cambios en el conjunto de resultados vinculado.
CancelEdit()	Cancela las modificaciones realizadas en los controles vinculados, y recupera los valores originales.
DeleteCurrent()	Elimina el registro activo actual del conjunto de resultados.
AddNew()	Inserta un nuevo registro en el conjunto de resultados con todos los campos vacíos.

Ejemplo: Sea el siguiente formulario con una lista y un conjunto de cajas de texto vinculadas a un mismo objeto origen de vinculación asociado al conjunto de registros de la tabla CONDUCTORES en la base de datos MULTAS.

El usuario puede seleccionar el nombre de cualquier conductor seleccionándolo en la lista de modo que se muestra automáticamente el valor del resto de sus campos en las cajas de texto.

El código de inicialización del formulario es el mismo que el de ejemplos anteriores:

```
Imports System.Data.SqlClient

Public Class Form2

    Private vinculo As BindingSource
    Private conexion As SqlConnection
    Private adaptador As SqlDataAdapter
    Private dset As New DataSet()

    Private Sub Form2_Load(sender As System.Object, e As System.EventArgs) Handles MyBase.Load
        ' Recuperación de datos de la base de datos
        conexion = New SqlConnection(My.Settings.MULTASCConexion)
        adaptador = New SqlDataAdapter("SELECT DNI, NOMBRE, APELLIDOS, DIRECCION, MATRICULA,
EDAD, LICENCIA FROM CONDUCTOR", conexion)
        adaptador.Fill(dset, "CONDUCTORES")
        ' Establecimiento de origen de vinculación para controles
        vinculo = New BindingSource()
        vinculo.DataSource = dset
        vinculo.DataMember = "CONDUCTORES"

        ' Vinculación de control ListBox a objeto origen de vinculación
        Me.ListBox.DataSource = vinculo
        Me.ListBox.DisplayMember = "NOMBRE" ' Nombre de la columna con los valores a mostrar
        Me.ListBox.ValueMember = "DNI" ' Nombre de la columna con los valores a almacenar
        ' Vinculo de la propiedad Text del control, a la columna DNI del registro activo
        Me.txtDni.DataBindings.Add(New Binding("Text", vinculo, "DNI"))
        Me.txtNombre.DataBindings.Add(New Binding("Text", vinculo, "NOMBRE"))
        Me.txtApellido.DataBindings.Add(New Binding("Text", vinculo, "APELLIDOS"))
        Me.txtDireccion.DataBindings.Add(New Binding("Text", vinculo, "DIRECCION"))
        Me.txtMatricula.DataBindings.Add(New Binding("Text", vinculo, "MATRICULA"))
        Me.txtEdad.DataBindings.Add(New Binding("Text", vinculo, "EDAD"))
        Me.txtLicencia.DataBindings.Add(New Binding("Text", vinculo, "LICENCIA"))
    End Sub
```

Cuando el usuario modifica los valores mostrados en las cajas de texto, las modificaciones no se filtran hasta el conjunto de resultados hasta que:

- Se invoca al método **EndEdit()** del objeto origen de vinculación.
- Se modifica el registro activo mediante la selección de otro registro en el control lista vinculado (o por programación modificando el valor de la propiedad *Position* o llamando a los métodos *MoveXXXX* del objeto *BindingSource*).

El botón OK del formulario realiza llamada al método **EndEdit()** que confirma los valores editados en los controles vinculados modificando los campos del registro activo actual:

```
' Guardado de cambios
Private Sub btnOK_Click(sender As System.Object, e As System.EventArgs) Handles
    btnOK.Click
        Me.vinculo.EndEdit()
    End Sub
```

El botón CANCELAR rechaza los valores modificados llamando al método **CancelEdit()**. Esto fuerza a los controles vinculados a recuperar los valores originales presentes en el registro activo actual.

```
' Rechazo de cambios
Private Sub btnCancel_Click(sender As System.Object, e As System.EventArgs) Handles
    btnCancel.Click
        Me.vinculo.CancelEdit()
    End Sub
```

El botón NUEVO añade un nuevo registro con todos los campos vacíos al final del conjunto de resultados. Para ello se llama al método **AddNew()**.

```
' Añadido de nuevo registro al conjunto de resultados.  
Private Sub btnNuevo_Click(sender As System.Object, e As System.EventArgs) Handles  
    btnNuevo.Click  
        Me.vinculo.AddNew()  
    End Sub
```

Finalmente, el botón ELIMINAR elimina el registro activo actual del conjunto de resultados llamando el método **RemoveCurrent()**.

```
' Eliminación del registro activo del conjunto de resultados  
Private Sub btnEliminar_Click(sender As System.Object, e As System.EventArgs)  
    Handles btnEliminar.Click  
        Me.vinculo.RemoveCurrent()  
    End Sub
```

Todas estas modificaciones a través al objeto *BindingSource* tienen lugar única y exclusivamente contra el conjunto de resultados, y no provocan la actualización de la base de datos por si mismos. Para ello sería necesario emplear el mecanismo de actualización necesario:

- **Si se emplean objetos DataSet / DataTable** → llamando al método *Update* del objeto *adaptador de datos* utilizado.
- **Si se emplean objetos colecciones de objetos entidad** → llamando el método *SaveChanged* del *contexto de objetos*.

Apendice: Configuración de SQL Server para conexión vía TCP/IP

Configuración del protocolo de conexión TCP / IP

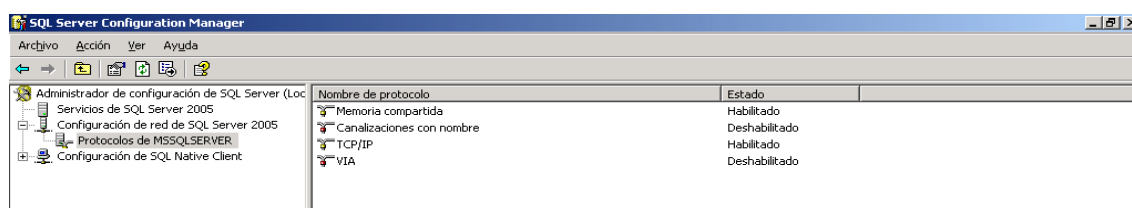
Para poder conectarnos al servidor de SQL Server es necesario que el servidor ‘escuche’ las solicitudes de conexión por parte de equipos remotos. Para ello debemos activar el protocolo TCP/IP en la configuración de red del SQL Server:

1.- Abrir el SQL Server Configuration Manager

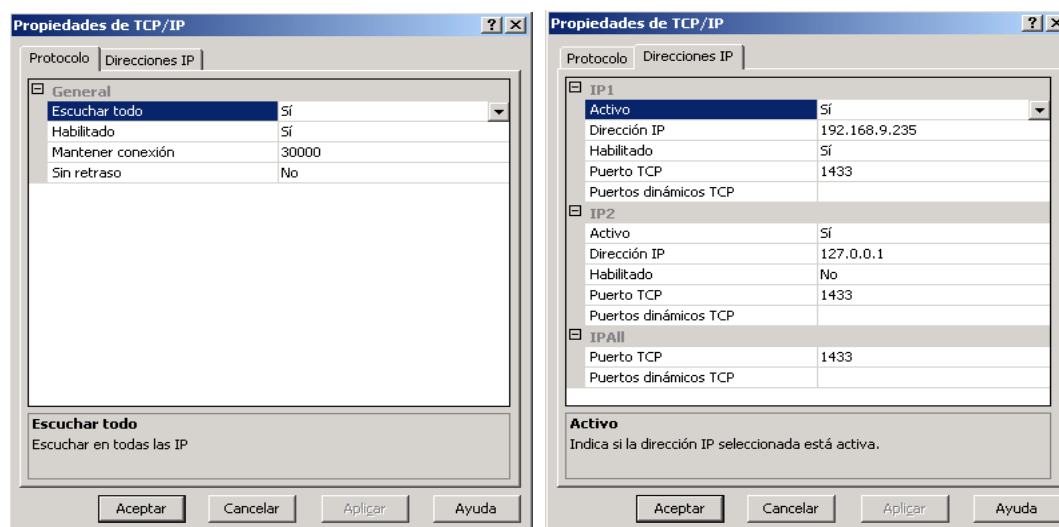
Por defecto suele encontrarse en:

Inicio → Todos los programas → Microsoft SQL Server → Herramientas de configuración →

2.- Una vez abierto el Administrador de configuración, seleccionar “*Protocolos de MSSQLSERVER*” dentro del nodo “*Configuración de red de SQL Server*”



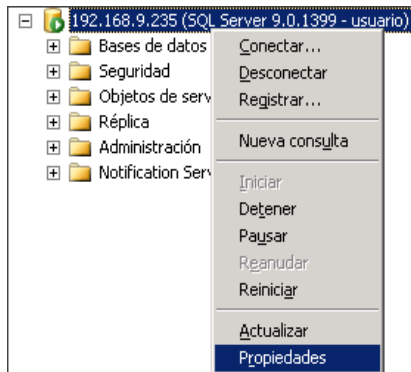
3.- Seleccionaremos el protocolo *TCP/IP*. Pulsando botón derecho sobre la opción aparecerá su menú contextual, donde seleccionaremos la opción “Propiedades”.



4.- En la pestaña **Protocolo** seleccionaremos “Sí” a las opciones “Escuchar todo” y “Habilitado”.

5.- En la pestaña **Direcciones IP** activaremos las conexiones TCP/IP que aparezcan disponibles. De esta manera SQL Server ‘escuchará’ las peticiones de conexión entrantes a través de las direcciones IP del equipo o máquina virtual en el que se ejecuta.

Configuración del servidor



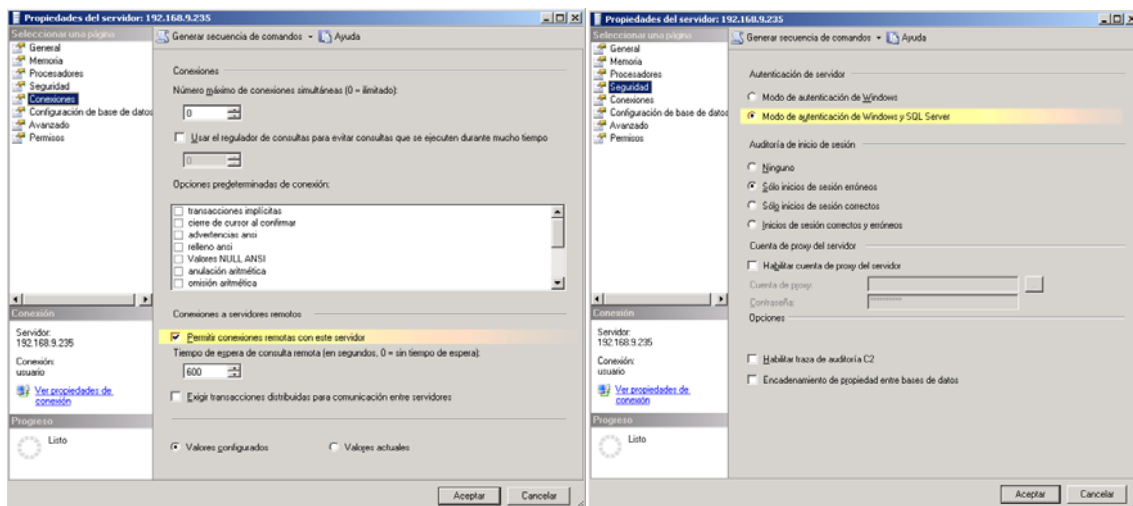
Abrir el **SQL Server Management Studio**

Por defecto suele encontrarse en:

Inicio → Todos los programas → Microsoft SQL Server →

Mediante el **SQL Server Management Studio** deberemos acceder al servidor como administradores y configurar una serie de ajustes destinados a permitir la conexión al servidor de base de datos desde otros equipos conectados en red.

Para acceder a las configuraciones generales del servidor, seleccionaremos la opción “Propiedades” del menú contextual que aparece al seleccionar con el botón derecho el icono del servidor en el árbol de elementos del administrador.



En la ventana de **Conexiones** → Comprobar que esté marcada la opción: “**Permitir conexiones remotas con el servidor.**”

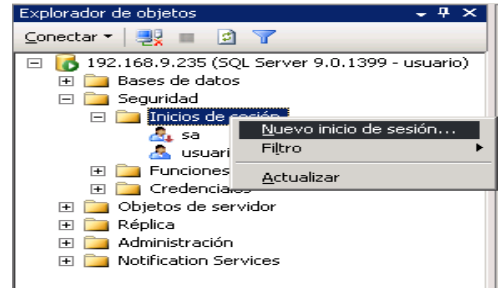
En la ventana de **Seguridad** → Comprobar que esté seleccionada la opción: “**Modo de autenticación de Windows y SQL Server.**”

Configuración cuenta de usuario para conexión

Además será necesaria la creación de una cuenta de usuario con autenticación SQL Server para su uso por parte de la aplicación. Evidentemente, esa cuenta deberá contar con los permisos necesarios para iniciar sesión en el servidor y manipular la base de datos de la aplicación:

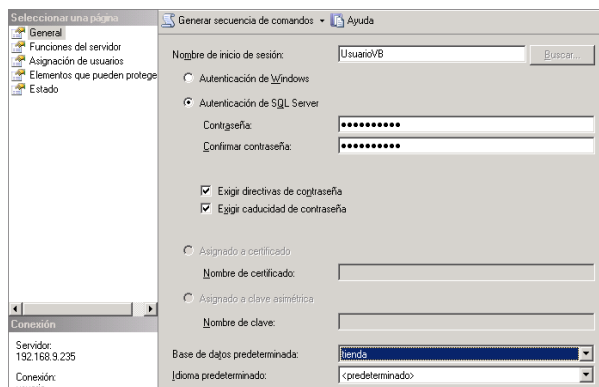
1.- Abrimos el **SQL Server Management Studio**

2.- Seleccionamos la opción “Nuevo Inicio de sesión” del menú contextual de la carpeta **Inicios de sesión**.



A continuación se abrirá el cuadro de dialogo para la configuración de los parámetros de la nueva cuenta de usuario. Esta cuenta la empleará la aplicación para conectarse al servidor de bases de datos como un usuario más.

➤ En la pestaña General:

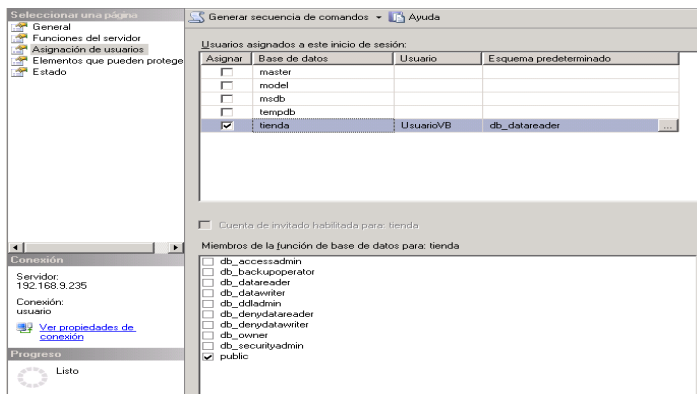


Nombre de inicio de sesión → Será el nombre de la cuenta que empleará el programa para identificarse al abrir la conexión con el servidor de bases de datos.

Autenticación de SQL Server y Contraseñas → Indican que la contraseña asociada a la cuenta.

➤ Base de datos predeterminada:

Indicamos la base de datos en el servidor a la que manipulará la aplicación a través de la cuenta de usuario que se está definiendo.



Asignación de Usuarios → Seleccionaremos la base de datos que manipulará la aplicación y configuraremos los esquemas de acceso que determinan las operaciones que podrá realizar la aplicación sobre la base de datos seleccionada.

Esquemas recomendados →

db_owner
db_datareader
db_datawriter

Apéndice: Modelo de aplicaciones MVC

La estrategia tradicional de utilizar aplicaciones compactas causa gran cantidad de problemas de integración en sistemas software complejos como pueden ser los sistemas de gestión de una empresa o los sistemas de información integrados consistentes en más de una aplicación. Estas aplicaciones suelen encontrarse con importantes problemas de escalabilidad, disponibilidad, seguridad, integración...

Para solventar estos problemas se ha generalizado la división de las aplicaciones en capas que normalmente serán tres: una capa implementa el manejo y estructura de los datos manejados por la aplicación (capa de datos o **modelo**), una capa para centralizar la lógica de negocio (capa de negocio o **controlador**) y por último una interfaz gráfica que facilite al usuario el uso del sistema (capa de presentación o **vista**)



Capa de presentación (Vista) : es la que ve el usuario (también se la denomina "capa de usuario"), presenta el sistema al usuario, le comunica la información y captura la información del usuario en un mínimo de proceso (realiza un filtrado previo para comprobar que no hay errores de formato). Esta capa se comunica únicamente con la capa de negocio. También es conocida como interfaz gráfica y debe tener la característica de ser "amigable" (entendible y fácil de usar) para el usuario.

Capa de negocio (Controlador) : es donde residen los programas que se ejecutan, se reciben las peticiones del usuario y se envían las respuestas tras el proceso. Se denomina capa de negocio (e incluso de lógica del negocio) porque es aquí donde se establecen todas las reglas que deben cumplirse. Esta capa se comunica con la capa de presentación, para recibir las solicitudes y presentar los resultados, y con la capa de datos, para solicitar al gestor de base de datos para almacenar o recuperar datos de él. También se consideran aquí los programas de aplicación.

Capa de datos (Modelo) : es donde residen los datos y es la encargada de acceder a los mismos. Está formada por uno o más gestores de bases de datos que realizan todo el almacenamiento de datos, reciben solicitudes de almacenamiento o recuperación de información desde la capa de negocio.

Ejercicios

La Agenda


Conéctate a la base de datos AGENDA utilizando los siguientes datos:

IP Servidor: 192.168.9.158

Base de Datos: AGENDA

Usuario / Contraseña: usuario – cipsa.

La base de datos consta de una única tabla EMPLEADO con las siguientes columnas y tipos de datos:

	Nombre de columna	Tipo de datos	Permitir valores NULL
	DNI	nchar(9)	<input type="checkbox"/>
	NOMBRE	nvarchar(15)	<input type="checkbox"/>
	APELLIDOS	nvarchar(30)	<input type="checkbox"/>
	TELEFONO	nchar(12)	<input checked="" type="checkbox"/>
	NACIMIENTO	smalldatetime	<input checked="" type="checkbox"/>
			<input type="checkbox"/>

Crear una aplicación que cumpla las siguientes especificaciones:

1. Crear una pequeña aplicación provista de un formulario que posea un menú de opciones con las siguientes opciones:

→ *Conectar...*

→ *Salir*

La opción *Conectar...* debe mostrar un formulario modal en el que se solicite la IP del servidor y el nombre y clave de usuario para establecer la conexión con la base de datos.

2. Modificar la opción *Conectar...* para que establezca la conexión con el servidor empleando una cadeba de conexión almacenada en el archivo de configuración (*app.config*).
3. Modificar la aplicación añadiendo una opción “Listas Personas...”. La opción debe abrir un formulario provisto de un DataGridView en el que deben mostrarse todos los datos de todas las personas registradas.
4. Modificar la aplicación añadiendo una opción “Buscar Persona...”. La opción debe mostrar un formulario que permita introducir el DNI de una persona y muestre sus datos en un conjunto de controles de tipo Label.

5. Modificar la aplicación añadiendo una opción “Gestión”. Esta opción debe mostrar un formulario provisto con los siguientes controles:

The screenshot shows a Windows application window titled "Listado". It contains a DataGrid with columns labeled "DNI", "NOMBRE", and "APELLIDOS". The first row of the grid has an asterisk (*) in the first column. Below the grid is a section titled "Detalles" containing text boxes for "DNI", "NOMBRE", "APELLIDO", "TELEFONO", and "FECHA". There are "Cancelar" and "Guardar" buttons at the bottom right of the "Detalles" section.

El control DataGridView debe mostrar el DNI, NOMBRE y APELLIDOS de todas las personas registradas en la base de datos. Cuando se seleccione una persona deben mostrarse automáticamente todos datos en las cajas de texto inferiores para poder modificarlos.

Si se pulsa el botón CANCELAR se rechazan las modificaciones volviendo a mostrar los valores originales. Si se pulsa el botón GUARDAR deben validarse las siguientes condiciones:

- **DNI** Este campo debe permanece no editable.
 - **NOMBRE** debe tener un máximo de 15 caracteres. No puede dejarse vacío.
 - **APELLIDOS** debe tener un máximo de 30 caracteres. No puede dejarse vacío.
 - **TELEFONO** debe tener un máximo de 12 caracteres, y sólo contener dígitos y guiones como caracteres permitidos.
 - **NACIMIENTO** debe ser una fecha válida con formato dd/mm/aaaa
6. Modificar el formulario añadiendo un botón ALTA que al ser pulsado genere un nuevo registro permitiendo introducir sus datos en las cajas de texto. En este caso el campo DNI sí es editable. Debe tener un máximo de 9 caracteres, no puede dejarse vacío, y no se admiten valores repetidos.
7. Modificar el formulario añadiendo un botón BAJA que al ser pulsado elimine el registro seleccionado en el control DataGridView.

Como inicio para desarrollar una aplicación de base de datos lo recomendable es crear en primer lugar una clase que permite el acceso a la base de datos al resto de las clases de la aplicación. Esta clase constituye la base del *modelo* o capa de datos de la aplicación.

El modelo debe ser independiente (al menos en su declaración) del proveedor de datos empleado. Por ello, se definen los métodos con los tipos de las clases padres definidas en *System.Data.Common* (***DbConnection***, ***DbCommand***, ***DbDataAdapter***, ***DbDataReader***), en vez de emplear las clases hijas específicas de cada proveedor de datos. Gracias a la herencia, los métodos admitirán al ser llamados objetos de cualquiera de sus clases hijas y la clase podrá emplearse con cualquier proveedor de datos.

A modo de ejemplo, se recomienda implementar la siguiente clase de acceso a base de datos incluyendo los siguientes métodos:

```
VB
Imports System.Data.Common

Public Class DataBase
    ''' <summary>
    ''' Cadena de conexion a la BBDD
    ''' </summary>
    ''' <remarks></remarks>
    Private cadenaConexion As String
    ''' <summary>
    ''' Objeto conexion
    ''' </summary>
    ''' <remarks></remarks>
    Private conexion As DbCommand = Nothing
    ''' <summary>
    ''' Objeto adaptador de datos
    ''' </summary>
    ''' <remarks></remarks>
    Private adaptador As DbDataAdapter = Nothing
    ''' <summary>
    ''' Objeto generador de comandos
    ''' </summary>
    ''' <remarks></remarks>
    Private combuilder As DbCommandBuilder = Nothing
    ''' <summary>
    ''' Constructor
    ''' </summary>
    ''' <param name="_cadenaConexion">Cadena de conexion</param>
    ''' <remarks></remarks>
    Public Sub New(ByVal _cadenaConexion As String)

    End Sub

    ''' <summary>
    ''' Abre la conexion a la base de datos
    ''' </summary>
    ''' <remarks>Producira una exception SQLException si la conexion no fuera posible.
    </remarks>
    Public Sub AbrirConexion()

    ''' <summary>
    ''' Cierra la conexion a la base de datos.
    ''' </summary>
    ''' <remarks></remarks>
    Public Sub CerrarConexion()

    ''' <summary>
    ''' Retorna un objeto Comando para ejecutar una operacion de consulta o modificacion
    ''' </summary>
    ''' <returns></returns>
    ''' <remarks></remarks>
    Public Function GenerarComando() As DbCommand
```

```

''' <summary>
''' Obtiene los resultados de una operacion de consulta. SELECT
''' </summary>
''' <param name="comando">Objeto comando configurado con la consulta</param>
''' <param name="tabla">Nombre de la tabla devuelta con los resultados</param>
''' <returns>Objeto DataSet con los datos</returns>
''' <remarks></remarks>
Public Function ObtenerConsulta(ByVal comando As DbCommand, ByVal tabla As String)
    As DataSet

''' <summary>
''' Obtiene un cursor con los resultados de una operacion de consulta. SELECT
''' </summary>
''' <param name="comando">Objeto comando configurado con la consulta</param>
''' <returns>Objeto DataReader</returns>
''' <remarks>La conexion debe estar abierta y se cerrada al final</remarks>
Public Function ObtenerReader(ByVal comando As DbCommand) As DbDataReader

''' <summary>
''' Ejecuta una operaci3n de actualizaci3n, modi ficaci3n o eliminaci3n de registros en la
BBDD
''' </summary>
''' <param name="command">Objeto comando configurado con la operaci3n</param>
''' <returns>Valor entero con n3mero de registros afectados por la operaci3n</returns>
''' <remarks></remarks>
Public Function EjecutarActualizaci3n(ByVal command As DbCommand) As Integer

''' <summary>
''' Actualiza un objeto DataSet obtenido previamente.
''' </summary>
''' <param name="dataset">Objeto DataSet</param>
''' <param name="tabla">Nombre de la tabla en el DataSet a actualizar en la BBDD</param>
''' <remarks></remarks>
Public Sub ActualizarDataSet(ByVal dataset As DataSet, ByVal tabla As String)

End Class

```

C#

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using System.Data.Common;
using System.Data;

namespace WindowsFormsAppli cation2
{
    public class DataBase
    {
        /// <summary>
        /// Cadena de conexion a la BBDD
        /// </summary>
        /// <remarks></remarks>
        private String cadenaConexion;
        /// <summary>
        /// Objeto conexion
        /// </summary>
        /// <remarks></remarks>
        private DbCommand conexion = null;
        /// <summary>
        /// Objeto adaptador de datos
        /// </summary>
        /// <remarks></remarks>
        private DbDataAdapter adaptador = null;
        /// <summary>
        /// Objeto generador de comandos
        /// </summary>
        /// <remarks></remarks>
        private DbCommandBuilder comBuilder = null;
        /// <summary>
        /// Constructor
        /// </summary>
        /// <param name="_cadenaConexion">Cadena de conexion</param>
        /// <remarks></remarks>
        public DataBase(String _cadenaConexion){

```

```

    /// <summary>
    /// Abre la conexión a la base de datos
    /// </summary>
    /// <remarks>Producirá una excepción SQLException si la conexión no fuera posible.
</remarks>
    public void AbrirConexion() {

        /// <summary>
        /// Cierra la conexión a la base de datos.
        /// </summary>
        /// <remarks></remarks>
        public void CerrarConexion() {

            /// <summary>
            /// Retorna un objeto Comando para ejecutar una operación de consulta o modificación
            /// </summary>
            /// <returns></returns>
            /// <remarks></remarks>
            public DbCommand GenerarComando() {

                /// <summary>
                /// Obtiene los resultados de una operación de consulta. SELECT
                /// </summary>
                /// <param name="comando">Objeto comando configurado con la consulta</param>
                /// <param name="tabla">Nombre de la tabla devuelta con los resultados</param>
                /// <returns>Objeto DataSet con los datos</returns>
                /// <remarks></remarks>
                public DataSet ObtenerConsulta(DbCommand comando, String tabla) {

                    /// <summary>
                    /// Obtiene un cursor con los resultados de una operación de consulta. SELECT
                    /// </summary>
                    /// <param name="comando">Objeto comando configurado con la consulta</param>
                    /// <returns>Objeto DataReader</returns>
                    /// <remarks>La conexión debe estar abierta y se cerrada al final</remarks>
                    public DbDataReader ObtenerReader(DbCommand comando) {

                        /// <summary>
                        /// Ejecuta una operación de actualización, modificación o eliminación de registros en la
BBDD
                        /// </summary>
                        /// <param name="command">Objeto comando configurado con la operación</param>
                        /// <returns>Valor entero con número de registros afectados por la operación</returns>
                        /// <remarks></remarks>
                        public int EjecutarActualizacion(DbCommand command) {

                            /// <summary>
                            /// Actualiza un objeto DataSet obtenido previamente.
                            /// </summary>
                            /// <param name="dataset">Objeto DataSet</param>
                            /// <param name="tabla">Nombre de la tabla en el DataSet a actualizar en la BBDD</param>
                            /// <remarks></remarks>
                            public void ActualizarDataSet(DataSet dset, String tabla)
                        }
                    }
                }
            }
        }
    }

```


Ejercicio

La Tienda

Se desea crear una aplicación para una tienda de alquiler de juegos y películas que permita almacenar la información de todos los artículos así como los alquileres y los clientes asociados.

Descripción modelo de datos.

La información que se desea almacenar por cada cliente es su NIF, nombre, apellidos, y fecha de nacimiento. El sistema buscará los clientes registrados por el NIF. No se permitirá el registro de dos clientes con un NIF.

La información que se desea almacenar para cada artículo es un número de artículo, el nombre del juego o película, un indicativo de si es para mayores de 18 de años o no, y otro que indica si el artículo está alquilado ya. El identificador de cada artículo se genera en el momento de darlo de alta por el propio sistema sumando 1 al identificador del último artículo dado de alta. Los artículos pueden a su vez ser Juegos o Películas. De los juegos se indica la consola a la que pertenece: XBOX360, PS3, WII. En el caso de las películas se indica el tipo de medio: DVD o BluRay. Ambos valores se declaran empleando tipos enumerados.

Finalmente de registrarse cada alquiler. La información a almacenar contiene la fecha y hora del alquiler, el identificador del artículo alquilado y el NIF del cliente que lo alquila.

Resumen de las clases entidad de la aplicación:

- Clase **Articulo**: Identificador (Int), Nombre (String), Mas18 (boolean), Alquilado (boolean)
- Clase **Juego**: (hereda de Artículo): Plataforma Tipo (Enumerado: XBOX360,PS3,WII).
- Clase **Película** (hereda de Artículo): Dispositivo Tipo (Enumerado: DVD, BluRay).
- Clase **Cliente**: NIF (String), Nombre (String), Apellidos (String), Nacimiento (DateTime)
- Clase **Alquiler**: Cliente, Articulo, Fecha(DateTime).

La información debe almacenarse en la base de datos Tienda instalada en el servidor de SQL Server con los siguientes datos:

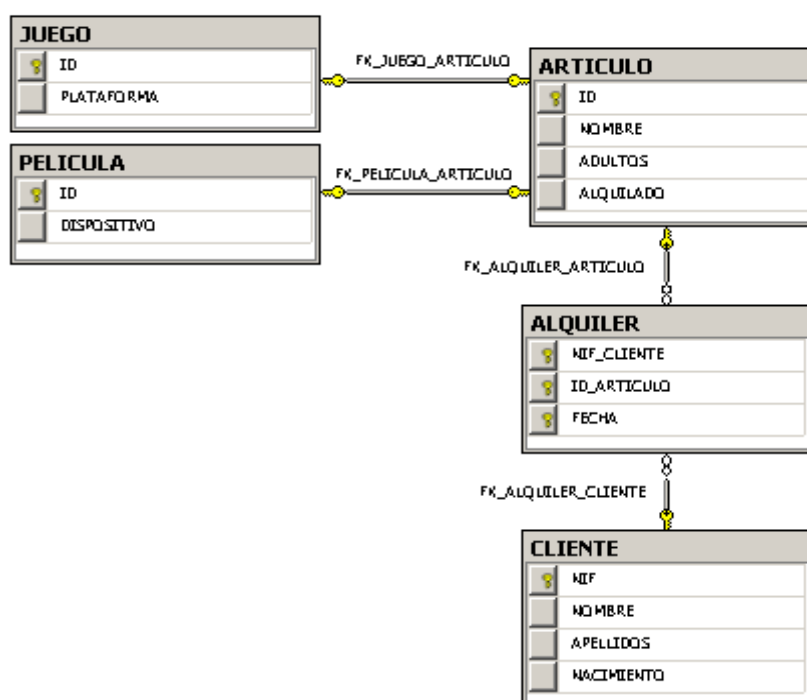
IP: 192.168.9.158

BASE DATOS: tienda

CUENTA: usuario

CLAVE: cipsa

La base de datos Tienda consta de las siguientes tablas y relaciones:



- **Tabla ARTICULO** → Contiene todos los artículos registrados indicando si están alquilados

	Nombre de columna	Tipo de datos	Permitir v...
🔑	ID	int	<input type="checkbox"/>
	NOMBRE	nvarchar(20)	<input type="checkbox"/>
	ADULTOS	bit	<input type="checkbox"/>
	ALQUILADO	bit	<input type="checkbox"/>

- **Tabla PELICULA** → Contiene el dispositivo asociado a los artículos que son películas.

	Nombre de columna	Tipo de datos	Permitir v...
🔑	ID	int	<input type="checkbox"/>
	DISPOSITIVO	nvarchar(10)	<input type="checkbox"/>

- **Tabla JUEGO** → Contiene la plataforma asociada a los artículos que son videojuegos.

	Nombre de columna	Tipo de datos	Permitir v...
🔑	ID	int	<input type="checkbox"/>
	PLATAFORMA	nvarchar(10)	<input type="checkbox"/>

- **Tabla CLIENTE** → Contiene los datos de todos los clientes registrados

	Nombre de columna	Tipo de datos	Permitir v...
🔑	NIF	nvarchar(9)	<input type="checkbox"/>
	NOMBRE	nvarchar(15)	<input type="checkbox"/>
	APELLIDOS	nvarchar(30)	<input type="checkbox"/>
	NACIMIENTO	smalldatetime	<input type="checkbox"/>

- **Tabla ALQUILER** → Contiene los datos de todos los artículos alquilados actualmente.

	Nombre de columna	Tipo de datos	Permitir v...
🔑	NIF_CLIENTE	nvarchar(9)	<input type="checkbox"/>
🔑	ID_ARTICULO	int	<input type="checkbox"/>
🔑	FECHA	smalldatetime	<input type="checkbox"/>

Descripción lógica de negocio (Controlador)

La lógica de la aplicación se reparte en las siguientes clases.

Clase ColeccionArticulos → Esta clase gestiona todas las operaciones de carga y guardado de datos de los artículos en la base de datos:

- **Alta(Artículo)** → Almacena el artículo pasado como parámetro en la base de datos. El atributo ID del artículo se establece en el método a partir del ID más grande de los artículos en la base de datos.
- **Obtener(ID)** → Devuelve el artículo con el identificador indicado, o NULL si no existe en la BD.
- **TodosPeliculas()** → Devuelve una matriz con todas las películas en la BD ordenadas por Nombre.
- **TodosJuegos()** → Devuelve una matriz con todos los juegos en la BD ordenados por Nombre.

Clase ColecciónClientes → Esta clase encapsula todas las operaciones de carga y guardado de datos referentes a los clientes en la base de datos.

- **Alta(Cliente)** → Inserta el cliente dado. Retorna un valor lógico FALSO si el cliente ya existe.
- **Obtener(NIF)** → Devuelve el cliente con el NIF indicado, o NULL si no existe en la base de datos.
- **Todos()** → Retorna una matriz con todos los Clientes registrados.

Clase ColecciónAlquileres → Esta clase encapsula todas las operaciones de gestión de alquileres en la base de datos.

- **Todos()** → Retorna una matriz con todos los alquileres pendientes indicando el nombre del artículo alquilado y el nombre del socio que lo alquila.
- **AltaAlquiler(Cliente, Artículo)** → Registra el nuevo alquiler. Si el artículo ya está alquilado se lanza una excepción de tipo '*ArticuloYaAlquiladoException*'. Si el artículo es para mayores de 18 años y el Cliente no tiene la edad requerida se lanza una excepción de tipo '*ClienteMenorEdadException*'. En caso contrario se crea el objeto Alquiler correspondiente y se almacena en la colección. El campo Alquilado debe marcarse como 'VERDADERO'.
- **BajaAlquiler(identificador)** → Elimina el alquiler del producto indicado. Si no existe ningún alquiler con el identificador existente se lanza una excepción de tipo: '*AlquilerInexistenteException*'. Si el alquiler existe se elimina de la colección. El campo Alquilado del artículo debe marcarse como 'FALSO'. El método debe devolver un valor entero indicando el número de días transcurridos desde la fecha de alquiler.
- **CantidadAlquileresCliente(NIF)** → Devuelve el número de alquileres pendientes de un cliente.
- **AlquileresCliente(NIF)** → Devuelve una matriz de objetos Artículo con todos los artículos alquilados por el cliente indicado.
- **AlquilerProducto(Identificador)** → Devuelve el Cliente que ha alquilado el producto o NULL si no está alquilado.
- **AlquileresFueraPlazo()** → Devuelve una matriz de objetos Alquiler, con todos los alquileres realizados hace más de tres días.

Descripción del interfaz (Vista)

La aplicación debe realizarse empleando una interfaz basada en ventanas de Windows. Debe emplearse un formulario principal de tipo MDI provisto de una barra de menú superior con las siguientes opciones:

1. Alta Película: Muestra un formulario que permite introducir los datos de una película: título, dispositivo (*DVD, BluRay*) y si es para mayores de edad, y muestra el identificador generado al darlo de alta.
2. Alta Juego: Muestra un formulario que permite introducir los datos de un juego: título y plataforma (*Wii, PS3, Xbox360*), y muestra el identificador generado al darlo de alta.
3. Alta Cliente: Muestra un formulario que permite introducir los datos de un nuevo cliente: DNI, Nombre, Apellidos, y fecha de nacimiento. Debe mostrarse un mensaje de advertencia si ya existe un cliente dado de alta con el mismo NIF.
4. Consultar Cliente: Muestra un formulario provisto de una lista desplegable para seleccionar el NIF de un cliente. Al seleccionar el cliente se debe pulsar el botón BUSCAR para que se muestren sus datos junto con el ID, Titulo y fecha de alquiler de todos los alquileres pendientes en su nombre.

5. Listar Artículos. Muestra un formulario provisto de un control tabla en el que se muestran los datos de todos los artículos registrados. Deben aparecer dos botones de opción (RadioButton) que permitan seleccionar si desea ver los artículos o las películas.

Los artículos pueden seleccionarse en la tabla. Si el artículo seleccionado está alquilado (sin importar que sea un videojuego o una película) debe mostrarse el NIF y el nombre del cliente que lo ha alquilado y la fecha.

6. Listar Películas: Muestra un formulario provisto de un control tabla en el que se muestran todos los datos de los artículos películas registradas.
7. Registrar alquiler: Muestra un formulario provisto de dos tablas.

En una de las tablas debe mostrarse todos los artículos no alquilados indicando su nombre e identificador. Deben aparecer dos botones de opción (*OptionButton*) que permite seleccionar “Películas” o “Videojuegos”. En la otra tabla deben mostrarse todos los clientes registrados indicando su NIF y nombre.

The screenshot shows a Windows application window with two panes. The left pane, titled 'Artículos', contains two radio buttons labeled 'Películas' and 'Videojuegos'. Below them is a table with two columns: 'ID_ARTICULO' and 'TITULO'. The right pane, titled 'Clientes', contains a table with three columns: 'NIF', 'NOMBRE', and 'APELLIDOS'. At the bottom center of the window is a button labeled 'Registrar Alquiler'.

Debe aparecer un botón “Registrar alquiler” que almacene el alquiler del artículo seleccionado por parte del cliente indicado. Si el artículo está es para mayores de edad y el cliente no es mayor de edad debe indicar el mensaje: "CLIENTE INAPROPIADO".

8. Devolver Producto: Solicita el identificador del producto y elimina el alquiler de la tabla ALQUILER de la base de datos, mostrando a continuación el número de días que el producto ha estado alquilado por el cliente.
9. Listado de alquileres fuera de plazo: Muestra una lista con la información de todos aquellos alquileres que excedan los tres días de tiempo desde que fueron alquilados.