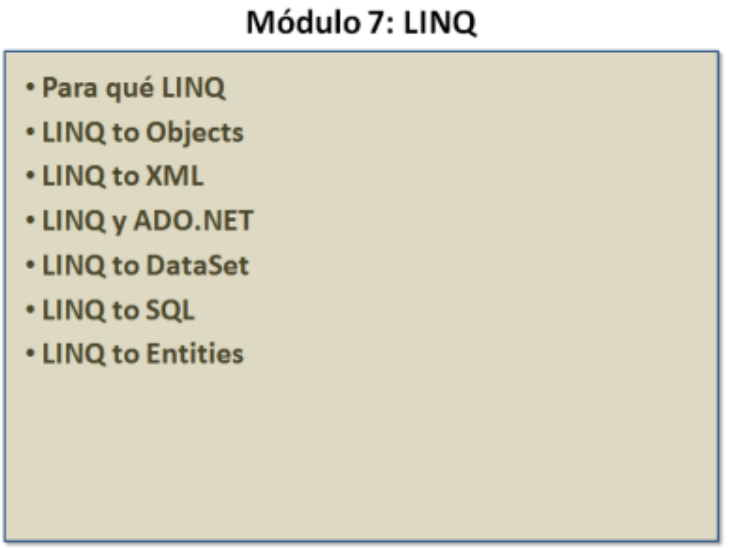
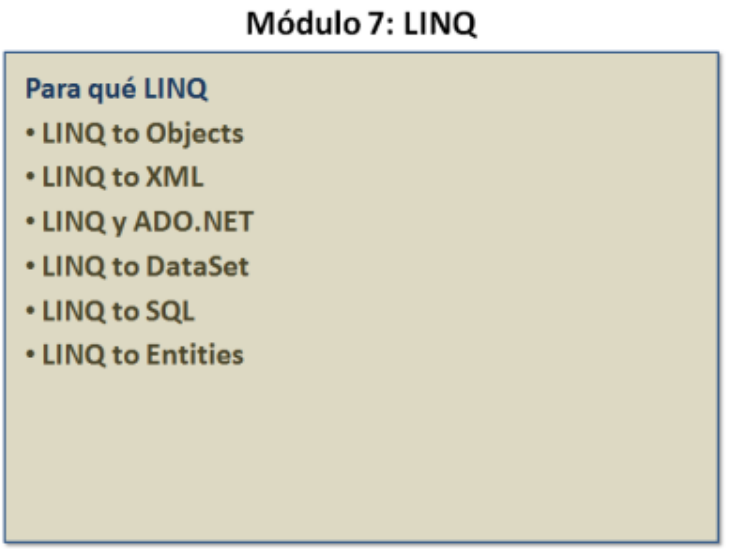


Contenido

En este módulo veremos los aspectos generales y particulares de LINQ

A continuación se exponen los apartados que se verán en este módulo.

- **Para qué LINQ**
- **LINQ to Objects**
- **LINQ to XML**
- **LINQ y ADO.NET**
- **LINQ to DataSet**
- **LINQ to SQL**
- **LINQ to Entities**



Para qué LINQ

LINQ o **L**anguage **IN**tegrated **Q**uery es una de las novedades más importantes de Microsoft .NET Framework 3.5 y que afecta por lo tanto a Visual Studio 2008 y con ello a Visual Basic 2008.

> **Nota:** LINQ no está disponible en Visual Studio 2005 ni versiones anteriores.

LINQ constituye ante todo una revolución a la hora de desarrollar aplicaciones Software. Mucha gente no lo ve así, pero es demasiado pronto como para que el fenómenos LINQ termine de calar entre los desarrolladores, si bien, quienes lo utilizan, quedan prendados de él.

La misión fundamental de LINQ es la de enlazar los datos con la programación y los lenguajes de programación tratando de eliminar las barreras que separaban a los datos de la programación, y dotando por otro lado a la aplicación, de más seguridad a los programadores a la hora de trabajar con datos.

LINQ entre otras cosas, nos permite comprobar los tipos de datos con los que estamos operando en tiempo de compilación, por lo que reducimos en gran medida los riesgos que solemos arrastrar los programadores en muchos proyectos Software.

Pero todo esto no tendría lugar sino fuera por los iteradores y las colecciones, aspectos todos ellos, que toman una especial importancia a la hora de trabajar con LINQ.

Para trabajar con LINQ, utilizaremos en muchas situaciones la interfaz *IQueryable* y *IEnumerable* que se encuentra en el nombre de espacio **System.Linq** del ensamblado **System.Core.dll**. Por defecto, al iniciar un proyecto con Visual Studio 2008, el entorno generará la estructura base de la aplicación y agregará las referencias necesarias para trabajar con LINQ.

¿Y qué es lo que nos permite LINQ?. LINQ nos permite por lo tanto, trabajar con datos, colecciones y elementos de una forma muy precisa, utilizando para ello sentencias integradas muy similares al lenguaje SQL.

Orígenes de datos LINQ

Para trabajar con colecciones, datos y elementos con LINQ, Microsoft nos ha proporcionado una serie de objetos o extensiones que nos permitirán interactuar con los datos adecuados y LINQ.

De acuerdo a la Arquitectura de LINQ, ésta se puede asemejar a la que se indica en la siguiente imagen:

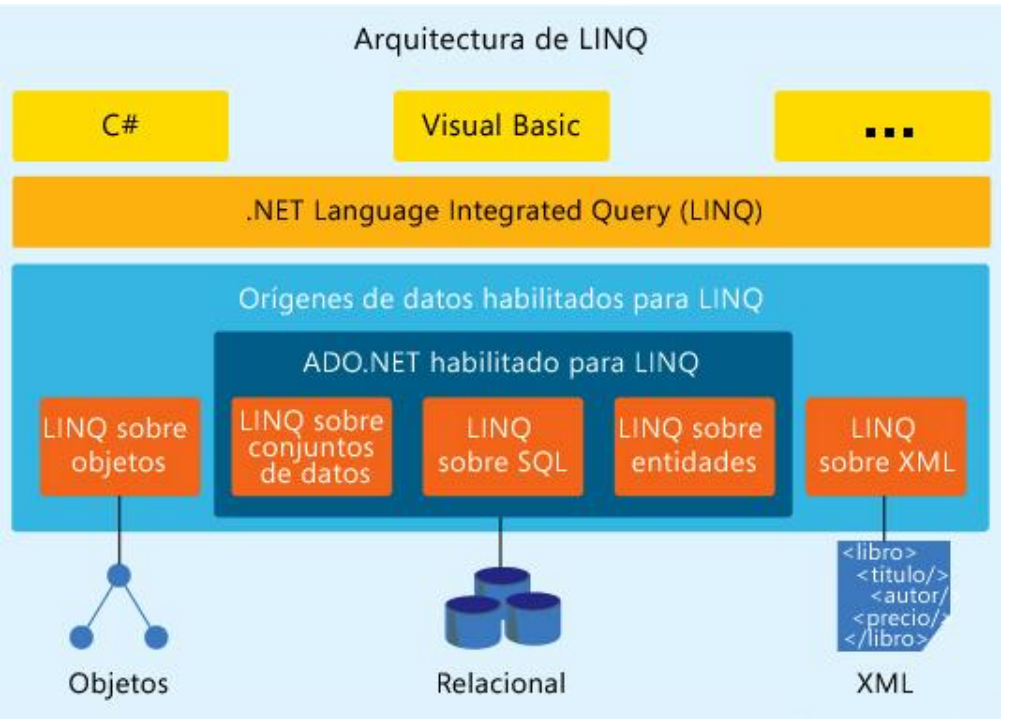


Figura 7.1.- Arquitectura de LINQ

En esta arquitectura, podemos ver a los lenguajes de programación en la capa más alta, y a continuación LINQ como lenguaje integrado de consultas, y debajo de éste, los orígenes de datos o extensiones que podremos utilizar con LINQ.

Respecto a las extensiones, debemos destacar las que pertenecen a los objetos de datos relaciones, las que tienen relación con los objetos propiamente dichos, y las que permiten trabajar con XML. Todos estos extensores, nos permiten trabajar con LINQ de una forma muy parecida.

Podría haber más extensores, incluso podríamos crear el nuestro propio. De hecho, en Internet hay diferentes proyectos, muchos de ellos de carácter código abierto, que tienen por misión crear extensores de LINQ para utilizar con .NET Framework 3.5. Muchos de esos proyectos, pueden encontrarse en CodePlex

Así por lo tanto, tenemos dentro de .NET Framework 3.5 los siguientes extensores:

- Si queremos trabajar con objetos, lo razonable es que trabajemos con LINQ to Objects.
- Si lo que queremos es trabajar con documentos XML, utilizaremos LINQ to XML.
- Pero si queremos acceder y manipular datos, entonces podremos trabajar con tres diferentes extensores dependiendo de nuestras necesidades. Estos extensores son LINQ to DataSet, también conocido como LINQ sobre conjunto de datos (recordemos el capítulo de datos donde se hablaba de DataSets), LINQ to SQL para trabajar con bases de datos SQL Server, y LINQ to Entities para trabajar con entidades, o lo que es lo mismo, con cualquier fuente de datos, no solo con SQL Server.

Cómo funciona LINQ

Las consultas de LINQ operan sobre entidades que implementan la interfaz **IEnumerable<T>**, pudiendo extraer su resultado a través de **IEnumerable<T>**.

Cuando trabaje con LINQ, oirá siempre comentar que estamos trabajando con consultas LINQ, porque en realidad, lo que hay por debajo es la ejecución de consultas muy parecidas a SQL, pero desde el código de nuestras aplicaciones.

A través de las consultas de LINQ, podemos trabajar con diferentes objetos como hemos indicado anteriormente, documentos XML, objetos en memoria, datos en bases de datos, etc.

LINQ constituye una forma más y novedosa de ejecutar sentencias parecidas a las sentencias SQL, directamente en nuestro código.

Operadores de LINQ

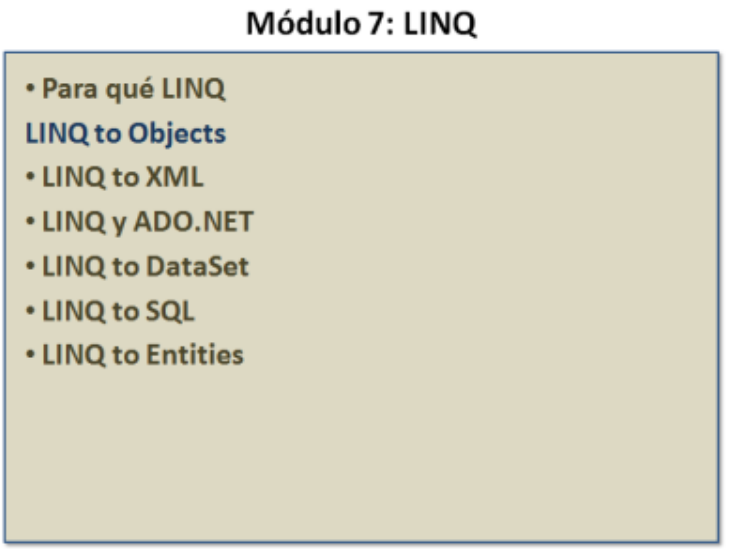
En LINQ tenemos la posibilidad de trabajar con un amplio conjunto de operadores. A continuación, se exponen los operadores de LINQ más destacables, con el fin de que los tenga presente.

Listado de operadores LINQ.

Operador	Descripción
Select SelectMany	Operadores de selección para filtrar y obtener datos.
Where OfType	Operadores de restricción utilizados para filtrar y obtener datos.
OrderBy OrderByDescending Reverse ThenBy ThenByDescending	Operadores de ordenación utilizados para ordenar el resultado de los datos obtenidos.
GroupBy	Operadores de agrupación, para agrupar la salida de los datos obtenidos.
Distinct Except Intersect	Operadores de trabajo con listas de datos.

Union	
All Any Contains	Operadores de cuantificación utilizados para obtener los datos de acuerdo a una determinada condición.
GroupJoin Join	Operadores de intersección utilizados para relacionar los elementos de listas de datos.
Skip SkipWhile Take TakeWhile	Operadores de extracción de una cantidad determinada de elementos. Útil por ejemplo para paginaciones de datos.
Aggregate Average Count LongCount Max Min Sum	Operadores de agregación utilizados para obtener resultados de acuerdo a la agregación de información o datos.
AsEnumerable AsQueryable Cast ToArray ToDictionary ToList ToLookup	Operadores utilizados para transformar la salida de los datos.
ElementAt ElementAtOrDefault First FirstOrDefault Last LastOrDefault Single SingleOrDefault	Operadores para acceder al elemento situado en una determinada posición.
Concat	Operadores de concatenación utilizados para unir datos de un par de listas, en una sola lista de datos.
SequenceEqual	Operadores de restricción utilizados para filtrar y obtener datos.
DefaultIfEmpty Empty Range Repeat	Operadores de generación para generar un conjunto de datos.

A continuación veremos como usar cada uno de los extensores de LINQ en Visual Basic 2008, sin entrar a detallar cada uno de los operadores que hemos visto en la lista anterior.



LINQ to Objects

Con LINQ to Objects trabajaremos con objetos, matrices y colecciones de datos que están en memoria, desde nuestras aplicaciones escritas en Visual Basic 2008.

Con LINQ to Objects, podemos trabajar con LINQ siempre y cuando los objetos con los que queremos trabajar, implementen las interfaces **IEnumerable** e **IEnumerable<T>**.

> **Nota aclaratoria:** recuerde que si una clase empieza por **I** seguida del nombre de la clase, suele indicar según la nomenclatura recomendada por Microsoft, que se trata de una interfaz. También es posible que encuentre en diferentes documentos, IEnumerable<T> como IEnumerable(Of T). Ambas significan lo mismo, y tienen relación directa con los datos tipados de la interfaz, o dicho de otra forma, tiene relación con los datos genéricos. En la documentación general de MSDN, lo encontrará siempre como IEnumerable<T>.

Para recorrer los elementos de una colección, utilizaremos normalmente el bucle **For Each**.

El ensamblado **System.Linq** que se incluye por defecto en cualquier aplicación de Visual Studio 2008 y que reside en **System.Core.dll**, nos permite utilizar directamente LINQ dentro de nuestras aplicaciones. Si eliminamos la referencia a este ensamblado, no podremos utilizar LINQ en nuestras aplicaciones.

Gracias a LINQ, podemos utilizar otra nomenclatura diferente para acceder a los elementos de un objeto o colección que están en memoria, algo que veremos en el siguiente apartado.

Practicando con LINQ to Objects

En este primer ejemplo, mostraremos como trabajar con una matriz de datos. Es un sencillo ejemplo que nos permitirá comprender mejor como funciona LINQ to Objects.

Lo primero que haremos es iniciar con Visual Studio 2008 un nuevo proyecto de Visual Basic 2008 y de tipo aplicación Windows.

Dentro del formulario Windows insertaremos un control **Button**, y a continuación, escribiremos el siguiente código fuente para nuestra aplicación.

Código

```
Public Class Form1



    Private Sub Button1_Click(ByVal sender As System.Object, ByVa
l e As System.EventArgs) Handles Button1.Click

        ' Declaramos la matriz y la inicializamos con valores

        Dim material() As String = {"mesa", "reloj", "libro", "pl
uma", "borrador", "pelota", "botella"}

        ' Ejecutamos LINQ para obtener los datos buscados

        ' Concretamente, buscamos los materiales que empiezan por
 "b"

        Dim materialBuscado = From busqueda In material _

                              Where busqueda.StartsWith("b") _

                              Select busqueda

        ' En materialBuscado tendremos toda la seleccirealizada,
por lo

        ' que deberemos recorrerla para extraer los elementos enc
ontrados

        Dim resultado As String = ""

        For Each elementos In materialBuscado

            resultado &= elementos & vbCrLf

        Next

        ' Mostramos la información

        MessageBox.Show(resultado)

    End Sub
```

```
End Class
```

En el ejemplo que acabamos de preparar, podemos ver como con LINQ, como lenguaje integrado de consultas, hemos accedido a los elementos de la matriz que están en la memoria.

Como habrá podido comprobar, LINQ es en su estructura, muy similar a SQL. Conviene destacar, que la forma de aprender a utilizar y manejar LINQ adecuadamente, es a base de práctica. También es interesante recalcar, que LINQ representa una revolución para el programador, y que Microsoft está haciendo grandes esfuerzos por dar a conocer y extender LINQ como un nuevo paradigma en el desarrollo de aplicaciones Software.

Si analizamos un poco el ejemplo que hemos realizado, veremos que dentro del código de Visual Basic podemos utilizar claúsulas como From, Where o Select, más propias de SQL que de un lenguaje de programación como Visual Basic. LINQ por lo tanto, nos proporciona o nos acerca el lenguaje natural de SQL para trabajar directamente con él en nuestras aplicaciones, seleccionando los elementos que queremos, proporcionando claúsulas de condición, u ordenando los elementos para obtenerlos en su salida directamente tal y como los queremos.

Dentro del ejemplo de LINQ, podemos comprobar que para seleccionar un subconjunto de elementos de un conjunto principal, utilizamos una serie de operandos diferentes. En primer lugar la claúsula u operando **From** en la que indicamos un alias dentro del conjunto de elementos *From <alias> In <conjunto_de_elementos>*. Posteriormente y dentro de la misma claúsula, utilizamos la claúsula u operando **Where** que nos permite indicar la condición o condiciones que queremos utilizar. Finalmente y en el ejemplo realizado, utilizamos la claúsula u operando **Select** que tiene por misión seleccionar el elemento o conjunto de elementos que queremos extraer.

A continuación, expondremos un ejemplo un poco más completo del uso de LINQ to Object. Aprovechando la base del ejemplo anterior, escribiremos una clase *Empleado* y luego, consumiremos esa clase para trabajar con ella desde LINQ to Objects. Veremos lo sumamente fácil, rápido e intuitivo que resulta trabajar con LINQ to Objects en Visual Basic 2008.

Código

```
Class Empleado

    ' Nombre del empleado

    Private m_Nombre As String

    Public Property Nombre() As String

        Get

            Return m_Nombre

        End Get

        Set(ByVal value As String)
```

```
            m_Nombre = value
        End Set
    End Property

    ' NIF del empleado
    Private m_NIF As String
    Public Property NIF() As String
        Get
            Return m_NIF
        End Get
        Set(ByVal value As String)
            m_NIF = value
        End Set
    End Property

    ' Edad del empleado
    Private m_Edad As Byte
    Public Property Edad() As Byte
        Get
            Return m_Edad
        End Get
        Set(ByVal value As Byte)
            m_Edad = value
        End Set
    End Property
End Class



Public Class Form1

    Private Sub Button1_Click(ByVal sender As System.Object, ByVa
l e As System.EventArgs) Handles Button1.Click
```

```
        ' Declaramos la matriz y la inicializamos con valores

        Dim personal() As Empleado = {New Empleado() With {.Nombr
e = "Juan", .NIF = "111A", .Edad = 32}, _

                                      New Empleado() With {.Nombr
e = "María", .NIF = "222C", .Edad = 43}, _

                                      New Empleado() With {.Nombr
e = "José", .NIF = "333R", .Edad = 45}, _

                                      New Empleado() With {.Nombr
e = "Rosa", .NIF = "333F", .Edad = 33}, _

                                      New Empleado() With {.Nombr
e = "Javier", .NIF = "444G", .Edad = 41}}

        ' Ejecutamos LINQ para obtener los datos buscados

        ' Concretamente, buscamos los empleados cuya edad es mayo
r de 40 años

        Dim empleadosBuscados = From busqueda In personal _

                                Where busqueda.Edad > 40 _

                                Select busqueda.Nombre, busqueda.Ed
ad _

                                Order By Edad Descending

        ' En empleadosBuscados tendremos toda la seleccirealizada
, por lo

        ' que deberemos recorrerla para extraer los elementos enc
ontrados

        Dim resultado As String = ""

        For Each elementos In empleadosBuscados

            resultado &= String.Format("{0} tiene {1} años", elem
entos.Nombre, elementos.Edad) & vbCrLf

        Next

        ' Mostramos la información

        MessageBox.Show(resultado)

    End Sub


End Class
```

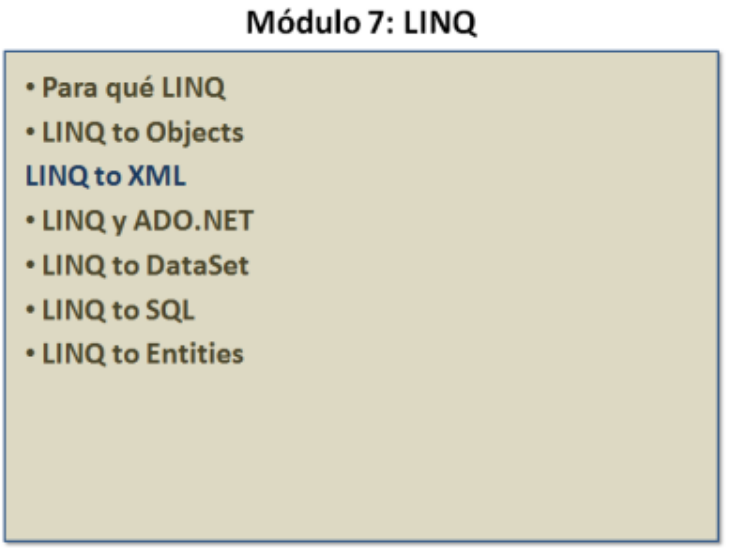
Nuevo en Visual Basic 2008: Una nueva característica de Visual Basic 2008, es la que permite inicializar la clase y sus propiedades directamente. Para ello, utilizaremos la claúsula **With** tal y como podemos ver en el ejemplo anterior.

Analizando el ejemplo que acabamos de ver, vemos que en este caso estamos trabajando con objetos *Empleado* que están en memoria, y con sus datos inicializados en la aplicación.

Dentro de la sentencia LINQ que estamos utilizando, hemos seleccionado dos elementos de los datos con los que estamos trabajando, y hemos realizado la ordenación de sus datos de acuerdo al operando o cláusula **Order By**. Incluso le hemos indicado el orden de ordenación mediante el uso de la palabra reservada **Descending**, indicando así que queremos ordenar los elementos por el campo **Edad** y en orden descendiente. Por defecto, la ordenación es en sentido ascendente y no hace falta indicarlo.

A la hora de trabajar con el entorno de desarrollo, *Intellisense* constituye para nosotros una herramienta de productividad enorme, ya que es capaz de indicarnos directamente en LINQ los campos disponibles, permitiéndonos así, seleccionar el campo o campos que deseamos utilizar.

- Ver video de esta lección - LINQ to Objects (Parte 1)
- Ver video de esta lección - LINQ to Objects (Parte 2)



LINQ to XML

Con LINQ to XML trabajaremos con documentos XML desde nuestras aplicaciones escritas en Visual Basic 2008.

LINQ to XML en Visual Studio 2008, posee diferencias notables y ventajosas con respecto a C# 3.0. Muchos son los programadores de C# que desean incluir algunas de las características que están incluidas en LINQ to XML para Visual Basic 2008.

Visual Basic 2008 proporciona por lo tanto, una integración total con XML, que permite manipular los documentos XML directamente.

> **Pruebe esto:** copie al portapapeles de Windows el contenido de un documento XML y escriba en Visual Studio 2008, y dentro de su código de Visual Basic 2008 la siguiente instrucción: **Dim variable =** y a continuación peque ahí el contenido del documento XML que tendrá en el portapapeles de Windows. Observará que Visual Basic 2008 es capaz de tratar el documento XML de forma directa.

Para trabajar con LINQ to XML, Microsoft nos ha proporcionado un nuevo ensamblado de nombre **System.Linq.Xml.dll**. A través de Visual Basic 2008, podemos utilizar los literales XML de forma directa, permitiéndonos explotar los documentos XML de una forma mucho más ágil y transparente, aumentando enormemente así la productividad de nuestros desarrollos.

Practicando con LINQ to XML

A continuación, realizaremos un ejemplo de como trabajar con LINQ to XML.

Para realizar el siguiente ejemplo, nos apoyaremos en el siguiente documento XML:

Código

```xml
<?xml version='1.0'?>
<!-- Este documento XML representa el inventario de coches -->
<coches>
    <coche marca="BMW">
        <modelo>520</modelo>
        <potencia>125CV</potencia>
    </coche>
    <coche marca="BMW">
        <modelo>525</modelo>
        <potencia>135CV</potencia>
    </coche>
    <coche marca="Citroen">
        <modelo>C3</modelo>
        <potencia>75CV</potencia>
    </coche>
    <coche marca="Citroen">
        <modelo>C4</modelo>
        <potencia>115CV</potencia>
    </coche>
    <coche marca="Citroen">
        <modelo>C5</modelo>
        <potencia>135CV</potencia>
    </coche>
</coches>
```

Insertaremos el documento XML anterior dentro del código de nuestra aplicación para utilizarlo de forma directa.

Basándonos en la base del ejemplo anterior, escribiremos un ejemplo que utilice el documento XML anterior en bruto. El código de nuestro ejemplo, quedará por lo tanto de la siguiente forma:

Código

```
Public Class Form1



    Private Sub Button1_Click(ByVal sender As System.Object, ByVa
l e As System.EventArgs) Handles Button1.Click

        Dim datos = SelectXML(120)

        Dim resultado As String = ""

        For Each dato In datos

            resultado &= String.Format("{0} tiene un coche {1}",
dato.marcaCoche, dato.modeloCoche) & vbCrLf

        Next

        MessageBox.Show(resultado)

    End Sub



    Private Function SelectXML(ByVal potencia As Byte) As Object

        Dim documentoXML = <?xml version='1.0'?>

                           <!--
 Este documento XML representa el inventario de coches -->

                           <coches>

                               <coche marca="BMW">

                                   <modelo>520</modelo>

                                   <potencia>125</potencia>

                               </coche>

                               <coche marca="BMW">

                                   <modelo>525</modelo>

                                   <potencia>135</potencia>

                               </coche>

                               <coche marca="Citroen">

                                   <modelo>C3</modelo>

                                   <potencia>75</potencia>

                               </coche>

                               <coche marca="Citroen">
```

```
                                    <modelo>C4</modelo>
                                    <potencia>115</potencia>
                                </coche>
                                <coche marca="Citroen">
                                    <modelo>C5</modelo>
                                    <potencia>135</potencia>
                                </coche>
                            </coches>
        Return From datos In documentoXML...<coche> _
               Where datos.<potencia>.Value > potencia _
               Select marcaCoche = datos.@marca, modeloCoche = da
tos.<modelo>.Value
    End Function


End Class
```

En este ejemplo, podemos ver cosas muy curiosas. Por un lado, la forma en la que accedemos a los bloques del documento XML lo hacemos mediante la instrucción **documentoXML...**. El uso de **...** nos permite situarnos en cada bloque de datos del documento XML, o lo que es lo mismo, en cada bloque de coche.

Por otro lado, para extraer el valor de un campo hijo en el bloque determinado, lo haremos utilizando su etiqueta entre los caracteres **<** y **>**, y usando el término **Value**, como por ejemplo datos..Value. Finalmente, para utilizar el valor de una etiqueta padre, utilizaremos el carácter **@**, como en el caso de **datos.@marca**.

Ahora bien, ¿que pasaría si el documento XML en lugar de estar incrustado dentro del código estuviera en una carpeta del disco duro?.

En ese caso, podremos cargar el documento XML en memoria para utilizarlo adecuadamente.

A continuación, veremos un ejemplo de como usar esta característica:

Código

```
Public Class Form1
```

```
    Private Sub Button1_Click(ByVal sender As System.Object, ByVa
l e As System.EventArgs) Handles Button1.Click

        Dim documentoXML As XDocument = XDocument.Load("C:\Desarr
ollaConMSDN.xml")

        Dim datosSeleccionados = From datos In documentoXML...<co
che> _

                Where datos.@marca.ToUpper() = "BMW" _

                Select modeloCoche = datos.<modelo>.Value, potenci
aCoche = datos.<potencia>.Value

        Dim resultado As String = ""

        For Each dato In datosSeleccionados

            resultado &= String.Format("{0} tiene un coche {1} de
 {2}CV", "BMW", dato.modeloCoche, dato.potenciaCoche) & vbCrLf

        Next

        MessageBox.Show(resultado)

    End Sub


End Class
```

Al ejecutar este ejemplo, veremos que el comportamiento no cambia demasiado, de hecho, el funcionamiento es exactamente el mismo, con la salvedad de que en un caso el documento XML se ha cargado en bruto, y en este último caso, se ha cargado desde una unidad de disco duro.

Trabajando con literales y LINQ to XML

Sin embargo, dentro del uso de documento XML y Visual Basic 2008, hay algo que resulta realmente atractivo para el programador.

Estoy hablando del uso de literales.

A continuación, veremos un ejemplo del uso de literales, que es a mi juicio, la mejor forma de entender en qué consiste esta característica agregada a Visual Basic 2008.

Basándonos una vez más en la base de los ejemplos anteriores, es decir, un formulario Windows con un control **Button**, vamos a desarrollar un ejemplo que nos muestre el uso de literales.

Para llevar a cabo nuestra tarea, vamos a agregar nuestro formulario un control **WebBrowser**. En mi caso, y para darle una estética más visual, he decidido agregar también dos controles **Panel** y he jugado con la propiedad **Dock** de los controles insertados en el formulario.

Una vez hecho esto, vamos a escribir el código de nuestro ejemplo que quedará de la siguiente forma:

Código

```
Public Structure Vehiculo


    Public Enum TipoVehiculo
        camion
        coche
        moto
    End Enum


    Public Tipo As TipoVehiculo
    Public Matricula As String
    Public Velocidad As Integer
End Structure


Public Class Form1


    Private Sub Button1_Click(ByVal sender As System.Object, ByVa
l e As System.EventArgs) Handles Button1.Click
        Dim listaVehiculos As New List(Of Vehiculo)
        Dim vehiculoUno As New Vehiculo With {.Matricula = "0000A
AA", .Tipo = Vehiculo.TipoVehiculo.coche, .Velocidad = 235}
        listaVehiculos.Add(vehiculoUno)
        Dim vehiculoDos As New Vehiculo With {.Matricula = "1111A
AA", .Tipo = Vehiculo.TipoVehiculo.moto, .Velocidad = 265}
        listaVehiculos.Add(vehiculoDos)
        Dim vehiculoTres As New Vehiculo With {.Matricula = "2222
AAA", .Tipo = Vehiculo.TipoVehiculo.coche, .Velocidad = 190}
        listaVehiculos.Add(vehiculoTres)
        Dim vehiculoCuatro As New Vehiculo With {.Matricula = "33
33AAA", .Tipo = Vehiculo.TipoVehiculo.camion, .Velocidad = 160}
        listaVehiculos.Add(vehiculoCuatro)
        Dim vehiculos = <?xml version='1.0'?>
```

```
                        <vehiculos>
                            <%= From datos In listaVehiculos _
                                Select <vehiculo>
                                           <tipo><%= datos.Tipo.T
oString() %></tipo>
                                           <matricula><%= datos.M
atricula %></matricula>
                                           <velocidad><%= datos.V
elocidad %></velocidad>
                                       </vehiculo> %>
                        </vehiculos>
        vehiculos.Save("C:\vehiculos.xml")
        Me.WebBrowser1.Navigate("C:\vehiculos.xml")
        MessageBox.Show("Datos guardados")
    End Sub


End Class
```

Observando con detenimiento este ejemplo, podemos ver que por un lado hemos creado una estructura de datos, que utilizaremos más adelante para crear los objetos que utilizaremos para crear al aire nuestro documento XML.

Dentro del código del evento **Click** del control **Button**, crearemos objetos de tipo **Vehiculo**, que corresponde con la estructura anteriormente creada, y construiremos un documento XML con los datos de los objetos creados.

Finalmente, guardaremos el documento XML creado en memoria al disco duro, y mostraremos su contenido en el control **WebBrowser**.

El resultado final obtenido, es el que se puede ver en la siguiente imagen:

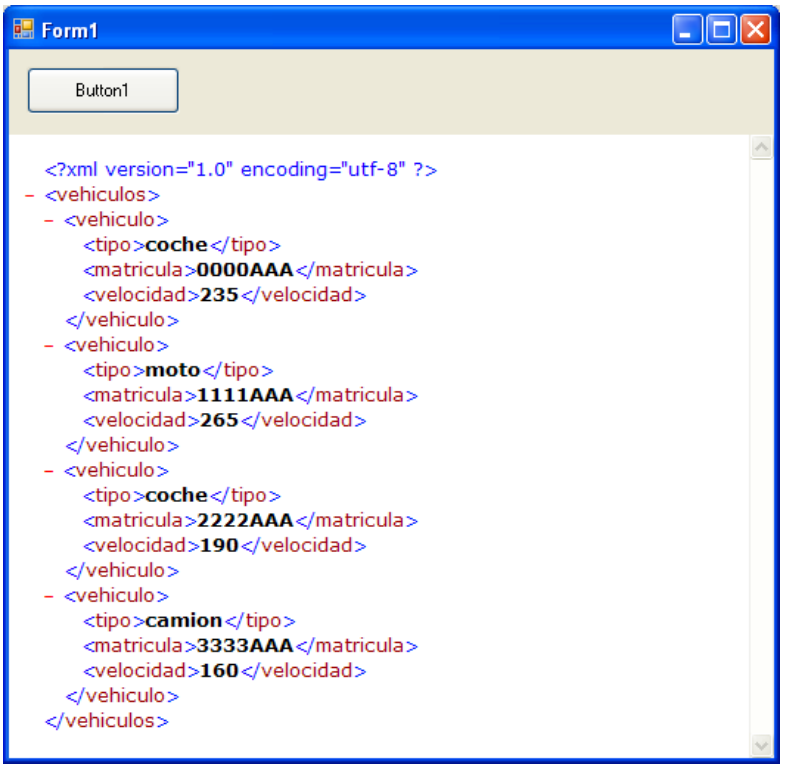
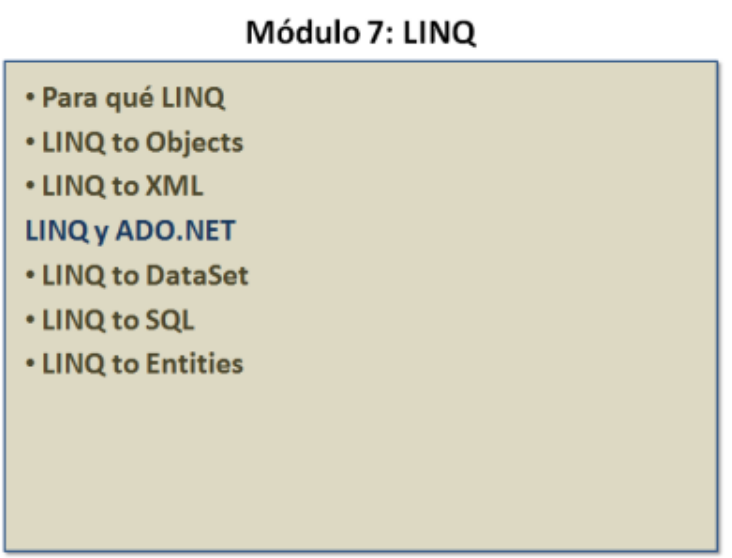


Figura 7.2.- Ejemplo en ejecución del uso de literales en LINQ to XML

Como podemos apreciar, el trabajo con documentos XML en Visual Basic 2008 es realmente sencillo y productivo gracias a LINQ.

- Ver video de esta lección - LINQ to XML (Parte 1)
- Ver video de esta lección - LINQ to XML (Parte 2)



LINQ y ADO.NET

Hasta el momento, hemos visto como trabajar con LINQ to Objects y LINQ to XML, sin embargo, aún no hemos visto como trabajar con fuentes de datos y LINQ.

Como recordará, al principio del capítulo dedicado a LINQ, vimos un resumen de los proveedores existentes de LINQ.

Vimos en una imagen, ese resumen. La imagen de recordatorio, es la siguiente:

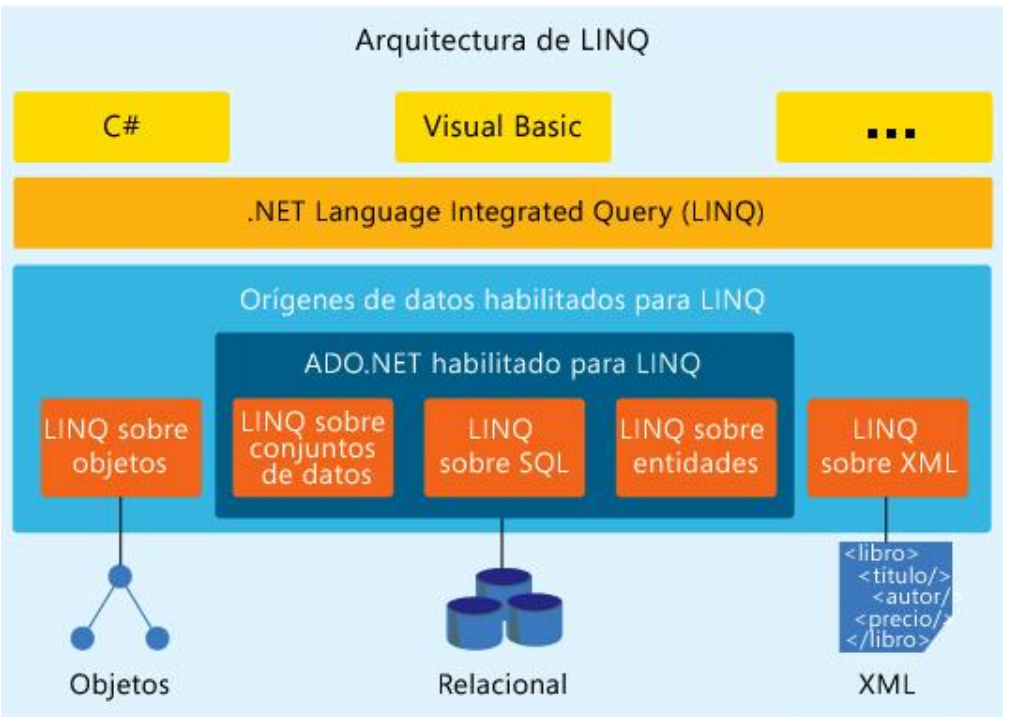
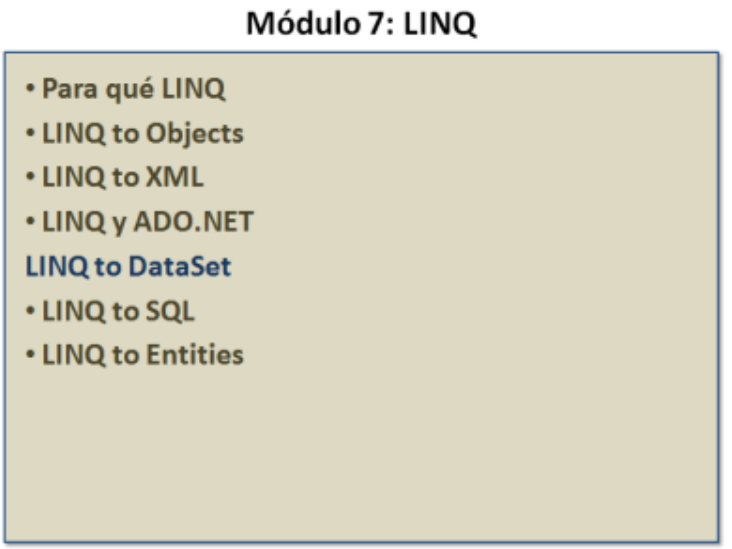


Figura 7.3.- Arquitectura de LINQ

Como puede observar en la imagen, la parte que corresponde al trabajo con LINQ y ADO.NET, está constituida por tres proveedores que nos permiten trabajar con datos. Esto son LINQ to DataSet, LINQ to SQL y LINQ to Entities.

A continuación, veremos cada uno de ellos y como se utiliza en Visual Basic 2008.



LINQ to DataSet

Como todos sabemos a estas alturas, los DataSet constituyen la base para trabajar con datos en memoria, desconectados de la fuente de datos.

LINQ to DataSet nos ofrece la posiblidad de trabajar con LINQ y fuentes de datos desconectadas, o lo que es lo mismo, usar LINQ para manipular y trabajar con los datos de un DataSet.

Para trabajar con LINQ y DataSet, no debemos pensar en nuevo paradigma de trabajo y manipulación de los datos, sino de pensar en LINQ como un servicio más que se ofrece para trabajar con DataSet.

Es decir, podremos trabajar con los DataSet tal y como lo hacemos hasta ahora, y aplicar LINQ para tratar de sacar todo el provecho posible.

La única regla de oro no obstante, es que los DataSet con los que trabajaremos, serán DataSet tipados, es decir, DataSet que poseen unos tipos de datos establecidos.

Practicando con LINQ to DataSet

En el siguiente ejemplo, veremos como utilizar LINQ en DataSets. Para ello, crearemos una estructura a la que iremos dando una serie de valores y crearemos así, una lista de estructuras para crear posteriormente un DataSet y su correspondiente DataTable, e ir agregando los elementos de la estructura al DataTable de ese DataSet. Posteriormente, utilizaremos LINQ para acceder a los datos del DataSet.

El ejemplo, quedará de la siguiente forma:

Código

```
Public Structure Persona

    Public Enum SexoPersona

        hombre

        mujer

    End Enum


    Public Nombre As String

    Public Sexo As SexoPersona

    Public Edad As Byte

End Structure


Public Class Form1



    Private Sub Button1_Click(ByVal sender As System.Object, ByVa
l e As System.EventArgs) Handles Button1.Click

        ' Creamos una lista de tipo Persona

        Dim listaPersonas As New List(Of Persona)

        Dim PersonaUno As New Persona With {.Nombre = "Daniel", .
Edad = 27, .Sexo = Persona.SexoPersona.hombre}

        listaPersonas.Add(PersonaUno)

        Dim PersonaDos As New Persona With {.Nombre = "Jimena", .
Edad = 29, .Sexo = Persona.SexoPersona.mujer}

        listaPersonas.Add(PersonaDos)

        Dim PersonaTres As New Persona With {.Nombre = "Alberto",
 .Edad = 24, .Sexo = Persona.SexoPersona.hombre}

        listaPersonas.Add(PersonaTres)

        ' Creamos el DataSet

        Dim personasDataSet As New DataSet()

        ' Agregamos una tabla al DataSet

        personasDataSet.Tables.Add("TablaPersonas")

        ' Agregamos tres columnas a la tabla
```

```
        personasDataSet.Tables("TablaPersonas").Columns.Add("Nombre")

        personasDataSet.Tables("TablaPersonas").Columns.Add("Edad")

        personasDataSet.Tables("TablaPersonas").Columns.Add("Sexo")

        ' Recorremos los elementos de la lista Persona

        ' y vamos agregando esos elementos al DataSet

        For Each empleado In listaPersonas

            ' Creamos una nueva fila

            Dim fila As DataRow = personasDataSet.Tables("TablaPersonas").NewRow()

            fila("Nombre") = empleado.Nombre

            fila("Edad") = empleado.Edad

            fila("Sexo") = empleado.Sexo

            ' Agregamos la fila

            personasDataSet.Tables("TablaPersonas").Rows.Add(fila)

        Next

        ' Buscamos datos dentro del DataSet

        Dim datosBuscados = From datos In personasDataSet.Tables("TablaPersonas") _

                            Where datos.Item("Edad") > 25 _

                            Select nombre = datos.Item("Nombre"), flagSexo = If(datos.Item("Sexo") = "hombre", True, False), sexo = datos.Item("Sexo")

        ' Mostramos los datos

        Dim resultado As String = ""

        For Each dato In datosBuscados

            resultado &= String.Format("{0} es {1} {2}", dato.nombre, If(dato.flagSexo, "un", "una"), dato.sexo) & vbCrLf

        Next

        MessageBox.Show(resultado)

    End Sub



End Class
```

Novedad en Visual Basic 2008: una de las novedades de Visual Basic 2008, es que ahora, **If** puede funcionar de la misma forma a como lo hacía **Iif**. En nuestros desarrollos podremos seguir utilizando **Iif**, pero **If**, también funciona desde Visual Basic 2008 de la misma forma.

En el ejemplo anterior, podemos ver que buena parte del código la hemos dedicado a crear el objeto DataSet y su objeto DataTable, alimentar ese DataTable con elementos, y posteriormente, crear la instrucción LINQ correspondiente para extraer los elementos resultantes.

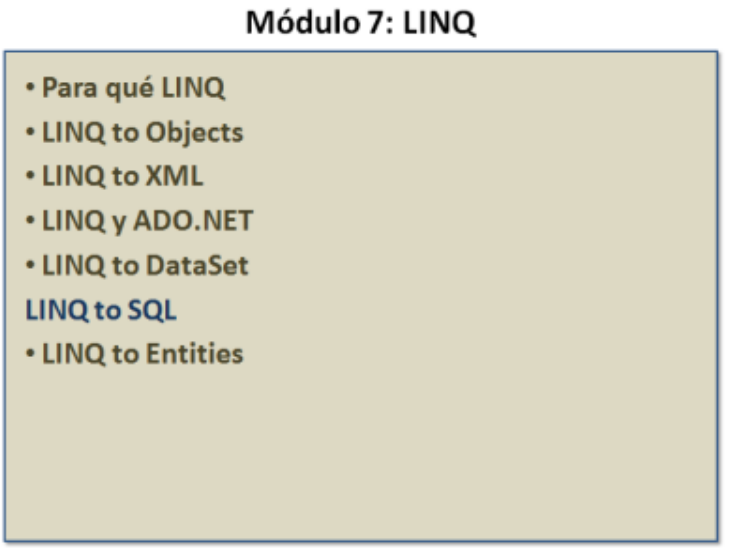
Como podemos apreciar, la forma de trabajar es siempre la misma.

Algo que aún no hemos mencionado cuando trabajamos con LINQ es todo lo relativo a la eficiencia, productividad e inmediatez a la hora de trabajar con LINQ y los datos.

El uso de LINQ es realmente rápido, mucho más de lo que mucha gente puede creer en un primer momento.

A continuación, veremos como trabajar con LINQ y otros modelos o fuentes de datos.

Ver video de esta lección - LINQ to DataSet



LINQ to SQL

Anteriormente, hemos visto como trabajar con LINQ to DataSet para trabajar con fuentes de datos desconectadas, pero, ¿y si queremos trabajar con fuentes de datos conectadas?.
En ese caso, podremos utilizar LINQ to SQL o LINQ to Entities.

¿Cuál de ellos debemos utilizar?.

LINQ to Entities está preparado para trabajar con cualquier fuente de datos conectada, mientras que LINQ to SQL está preparada para trabajar únicamente con fuentes de datos Microsoft SQL Server, por lo que parece claro, que si tenemos un motor de base de datos como Microsoft SQL Server, podemos trabajar tanto con LINQ to SQL como con LINQ to Entities, aunque si el motor de base de datos con el que estamos trabajando es Oracle, LINQ to SQL no nos servirá.

Aún y así, LINQ to Entities es ligeramente diferente a LINQ to SQL y ofrece unas características más avanzadas.

¿Sentencias LINQ o sentencias SQL?

LINQ to SQL destaca por permitirnos trabajar con objetos relacionales (O/R) a través del diseñador que ha sido incluido en Visual Studio 2008.

Adicionalmente, LINQ to SQL tiene la habilidad de construir la instrucción SQL correspondiente.
El programador debe olvidarse de como construir esa sentencia SQL, ya que es el propio compilador y el motor de ejecución de .NET Framework, el que se encarga de crear y lanzar por nosotros la sentencia SQL correspondiente.

Como comentábamos antes, LINQ to SQL trabaja con Microsoft SQL Server, por lo que las sentencias SQL creadas por LINQ to SQL serán para ese motor de base de datos.

De forma diferente a la que mucha gente piensa, esas sentencias generadas por LINQ son sentencias optimizadas en rendimiento, así que debemos estar tranquilos a la hora de utilizar LINQ.

Practicando con LINQ to SQL

Para practicar con LINQ to SQL deberemos iniciar Visual Studio 2008. En concreto, iniciaremos un proyecto de formulario Windows e insertaremos un control **Button** y un control **DataGridView** dentro del formulario.

Una vez hecho esto, haremos clic sobre el proyecto con el botón derecho del ratón, y aparecerá una ventana dentro de la cual, deberemos seleccionar la plantilla **Clases de LINQ to SQL** perteneciente al grupo **Datos**. Esto es lo que se puede ver en la siguiente imagen:

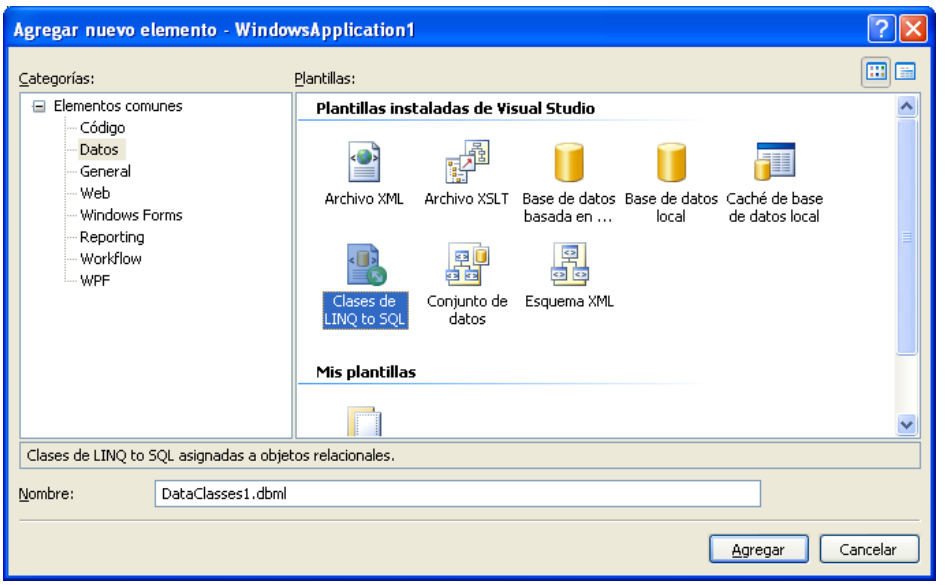


Figura 7.4.- Selección de la plantilla Clases de LINQ to SQL

Una vez seleccionada esta plantilla, el proyecto mostrará una nueva ventana que corresponde con el diseñador del modelo relacional, donde deberemos indicar que partes del modelo de datos de SQL Server queremos incluir en la plantilla.

A continuación, deberemos realizar la siguiente tarea, que consiste en crear una conexión con la base de datos de SQL Server. Para ello, podemos utilizar el **Explorador de servidores** y agregar en él la conexión que queremos utilizar en LINQ to SQL.

Cuando hayamos agregado la conexión con la base de datos, y dentro de la ventana del **Explorador de servidores**, podremos arrastrar y soltar sobre el diseñador, los objetos del modelo de datos que queramos utilizar en LINQ to SQL.

En el ejemplo con el que vamos a trabajar, los objetos que hemos arrastrado y soltado sobre el diseñador, son los que se muestran en la siguiente figura:

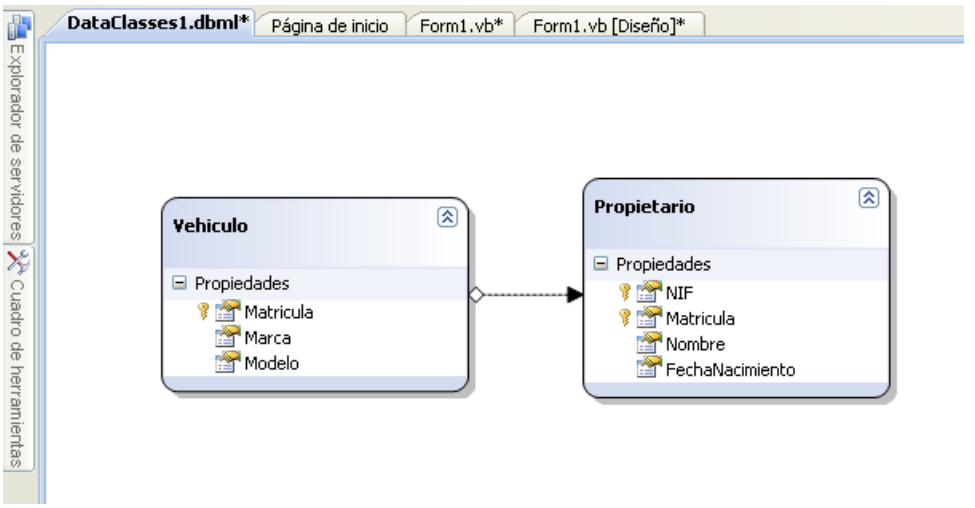


Figura 7.5.- Objetos de la base de datos en el diseñador de LINQ to SQL

> **A observar:** la extensión del diseñador del modelo relacional de LINQ to SQL es **dbml**, que corresponde con la plantilla de LINQ to SQL que hemos seleccionado en la figura 7.4.

Como podemos ver en la figura anterior, el diseñador del modelo relacional de LINQ to SQL, muestra en este caso, dos tablas y su posible relación. Las tablas contienen además, datos con los que podremos operar o trabajar. Aquí es donde entra LINQ y donde usándolo, obtendremos una gran productividad en nuestros desarrollos.

Para finalizar nuestro ejemplo, escribiremos el siguiente código fuente:

Código

```
Public Class Form1



    Private Sub Button1_Click(ByVal sender As System.Object, ByVa
l e As System.EventArgs) Handles Button1.Click

        ' Declaramos el contexto

        Dim contexto As New DataClasses1DataContext()

        ' Ejecutamos la sentencia LINQ correspondiente

        Dim datos = From tablaPropietario In contexto.Propietario
s, _

                                tablaVehiculos In contexto.Vehicu
los _

                                Where tablaPropietario.Matricula
= tablaVehiculos.Matricula _
```

```
                                Group tablaVehiculos By tablaProp
ietario.NIF Into Group _

                                Select NIF, Cantidad = NIF.Count(
)

        ' Mostramos el resultado final

        Me.DataGridView1.DataSource = datos

    End Sub


End Class
```

Nuestro ejemplo en ejecución, es el que se muestra en la figura siguiente:

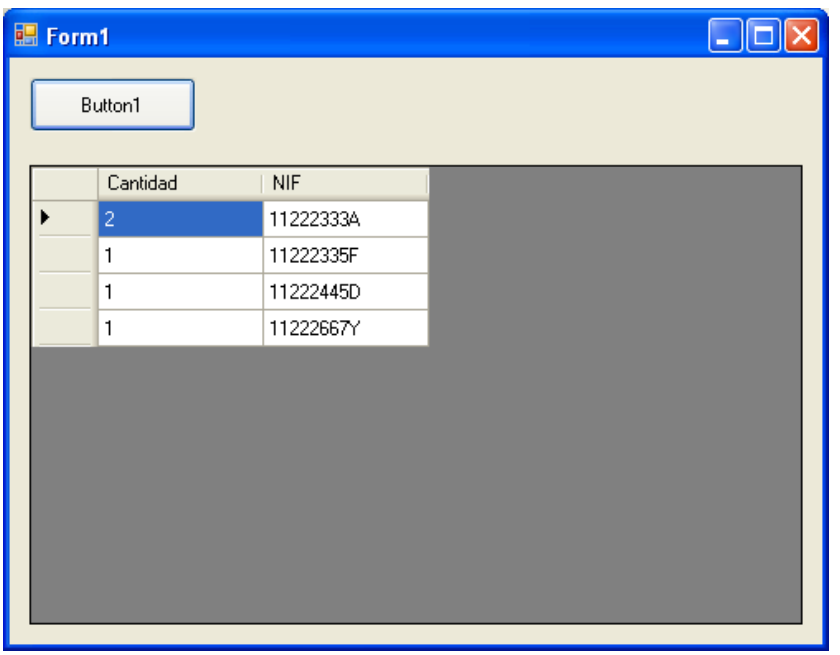


Figura 7.6.- Ejemplo del uso de LINQ to SQL en ejecución

Como podemos apreciar en el código fuente de la aplicación que hemos escrito, la cantidad de código fuente que hemos agregado es realmente pequeña. Analizando nuestro código fuente, vemos que por un lado, hemos declarado el contexto, o lo que es lo mismo, la clase del modelo relacional que hemos construido (el fichero de extensión **dbml**).

Si abrimos este fichero, observaremos que tiene un código fuente asociado que corresponde con el modelo relacional pasado a una clase, conservando en él cada uno de los campos de la tabla o de las tablas, con sus tipos de datos (fuertemente tipada) y sus relaciones si las hay.

De esta manera, estamos trabajando sobre una clase y sus propiedades, que corresponden con los campos de las tablas, por lo que resulta muy difícil equivocarse.

Otra particularidad del modelo relacional creado en Visual Studio 2008, es que el modelo relacional se conecta con SQL Server cuando es utilizado, y esto es prácticamente trasparente para el programador, ya que todo este proceso de conectarse, ejecutar la instrucción correspondiente, obtener los datos y desconectase nuevamente de la fuente de datos, la realiza el sistema dentro de la clase del modelo relacional (fichero **dbml**), pero la pregunta que podríamos hacernos ahora es... ¿dónde se encuentra por lo tanto la necesaria cadena de conexión para conectarse a la fuente de datos?.

La cadena de conexión la encontraremos en la configuración de la aplicación. Para acceder a ella, deberemos hacer clic con el botón derecho del ratón sobre el proyecto, y seleccionar la opción **Propiedades** del menú emergente. Finalmente, accederemos a la solapa **Configuración** para acceder al valor de la cadena de conexión que utiliza LINQ to SQL para conectarse.

Esto es lo que se puede ver en la siguiente imagen:

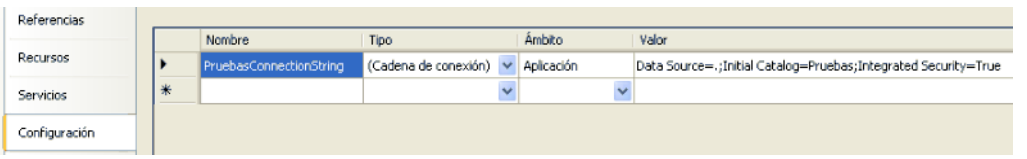
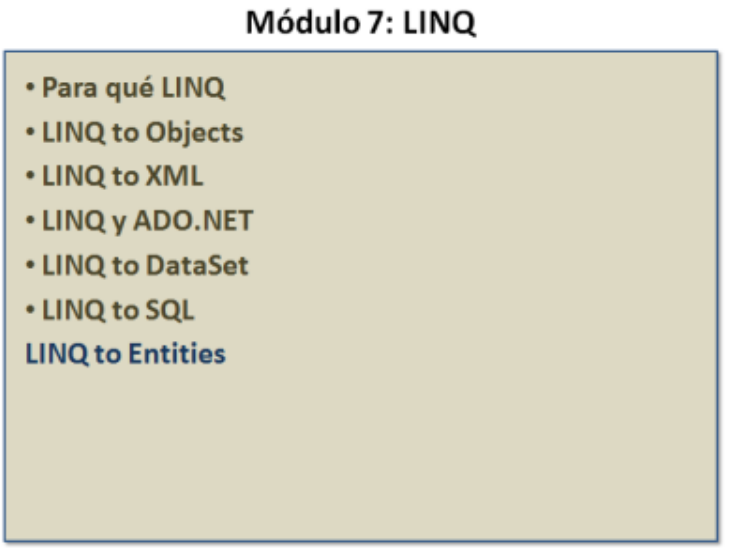


Figura 7.7.- Solapa de Configuración del proyecto, con la cadena de conexión del ejemplo de LINQ to SQL

Ver video de esta lección - LINQ to SQL



LINQ to Entities

A estas alturas, tendrá claro algunas de las ventajas de trabajar con LINQ, sin embargo, LINQ dentro de Microsoft .NET Framework 3.5 no proporciona un acceso directo a cualquier tipo de objeto, solamente a aquellos que estén preparados para trabajar con LINQ.

Dentro de los proveedores de LINQ, Microsoft proporciona uno muy especial, muy parecido a LINQ to SQL, pero con notables diferencias.

Este proveedor se llama LINQ to Entities, y es un proveedor preparado para trabajar con cualquier gestor de base de datos, proporcionando algunas ventajas que no proporciona LINQ to SQL.

Aún y así, también LINQ to Entities posee algunas diferencias con respecto a LINQ to SQL que le perjudican, como por ejemplo que LINQ to SQL sí está preparado para trabajar con procedimientos almacenados, mientras que la primera versión de LINQ to Entities no lo está. De hecho, ésta es una de las características más demandadas por la comunidad de desarrolladores y que el equipo de Microsoft de LINQ to Entities, tiene encima de la mesa para ponerse con ella cuanto antes.

Recordemos además, que mientras LINQ to SQL solo está preparado para trabajar con Microsoft SQL Server, LINQ to Entities está preparado para trabajar con cualquier motor de base de datos. Solo será necesario tener instalados los componentes adecuados para llevar a buen término nuestras necesidades. De hecho, en Internet puede encontrar ya algunos de estos proveedores listos para ser usados en motores de bases de datos como Oracle, DB2 o MySql.

Con LINQ to Entities, podemos trabajar por lo tanto, con motores de bases de datos y con EDM (Entity Data Model), o lo que es lo mismo, una representación lógica de un modelo de datos.

LINQ to SQL nos ofrecía un mapeo de datos muy sencillo, y para muchos desarrollos Software, será el proveedor adecuado, pero en otras circunstancias, necesitamos algo más. Ese "algo más" nos lo proporciona LINQ to Entities, que se

encarga de mapear los datos con mecanismos más complejos y potentes a lo que lo hace LINQ to SQL.

Finalmente, y para que no se lleve a engaños, debemos indicar que tanto LINQ to SQL como LINQ to Entities, no están preparados para soportar relaciones de muchos a muchos con carga útil (*Payload relationship*). El objetivo del equipo de trabajo de LINQ to Entities, es tener esta característica lista para la próxima versión de LINQ to Entities.

Pero para utilizar LINQ to Entities, necesitaremos no obstante, instalar un par de agregados a nuestra instalación de Visual Studio 2008.

Preparando Visual Studio 2008 para trabajar con LINQ to Entities

Para trabajar con LINQ to Entities, deberemos preparar adecuadamente nuestro entorno de trabajo, Visual Studio 2008. Deberemos descargar e instalar Microsoft .NET Framework 3.5 SP1 (Service Pack 1) en primer lugar, reiniciar el sistema (recomendado), e instalar Visual Studio 2008 SP 1 posteriormente.

Es posible que entre la instalación de Microsoft .NET Framework 3.5 SP1 y de Visual Studio 2008 SP1, el sistema le pida instalar algún paquete de Software necesario para llevar a buen término la instalación. Esos posibles avisos le aparecerán en la ventana de instalación, y lo único que tendrá que hacer es seguir las instrucciones.

Tenga a mano también, los CD o DVD originales de la instalación, porque es muy posible que el proceso de actualización le pida esa información.

Practicando con LINQ to Entities

En este punto, deberíamos estar listos para trabajar con LINQ to Entities, y es eso justamente, lo que vamos a hacer ahora.

Iniciaremos un nuevo proyecto de tipo aplicación Windows con Visual Studio 2008, e insertaremos en el formulario un control **Button** y un control **DataGridView**.

La siguiente acción que realizaremos, será la de agregar un elemento de tipo LINQ to Entities, que nos permitirá trabajar con la fuente de datos que elijamos.

Para agregar un nuevo elemento al proyecto, haremos clic dentro de la ventana del *Explorador de soluciones* sobre el proyecto con el botón derecho del ratón, y seleccionaremos la opción **Agregar > Nuevo elemento** del menú emergente.

Aparecerá una ventana dentro de la cual, deberemos encontrar la plantilla **ADO.NET Entity Data Model**, la cual localizaremos rápidamente en el grupo de plantillas de tipo **Datos**.

Esta es la ventana que se muestra a continuación:

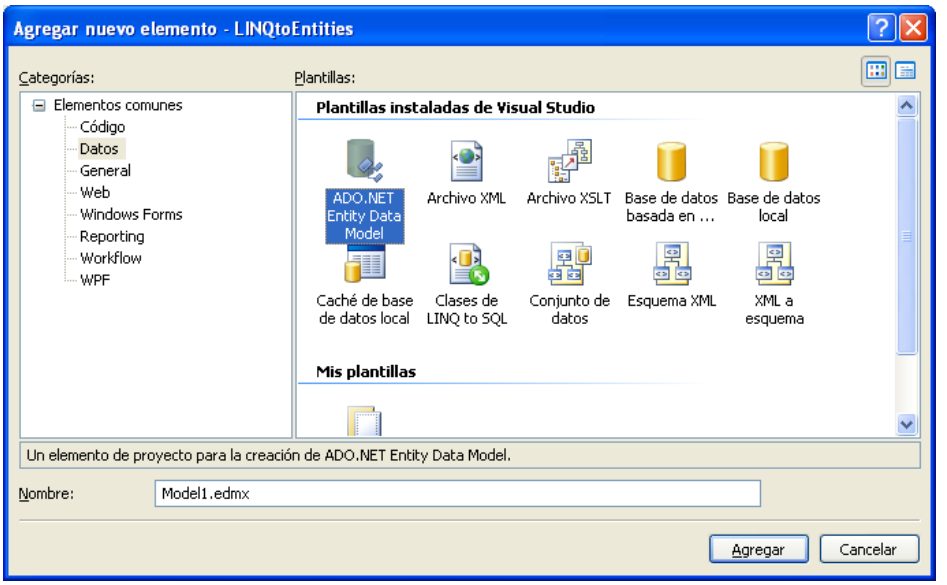


Figura 7.8.- Selección de la plantilla ADO.NET Entity Data Model

Haremos doble clic sobre la plantilla y ésta se nos incorporará al proyecto.

Al incorporarse esta plantilla al proyecto, Visual Studio 2008 lanzará un asistente, cuya primera ventana, consiste en seleccionar el contenido del modelo. Dentro de la selección, podemos elegir entre generar el modelo desde la base de datos, o generar el modelo partiendo de un modelo vacío. Nuestra intención es seleccionar todas las tablas en el modelo, por lo que seleccionaremos la primera opción.

Esto es lo que se puede ver en la siguiente imagen:

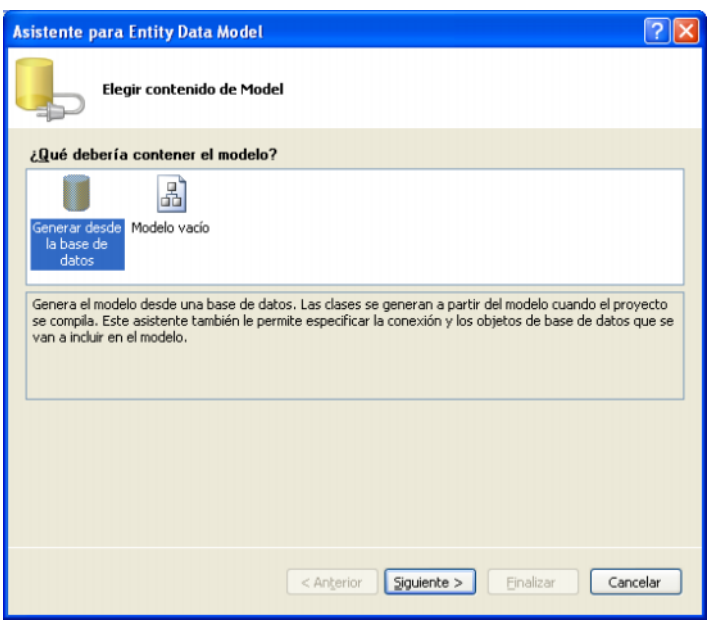


Figura 7.9.- Ventana principal del asistente para el EDM - Entity Data Model

A continuación, haremos clic sobre el botón **Siguiente**.
De esta forma, aparecerá una ventana dentro de la cual, podremos elegir la conexión de datos que queremos utilizar.

Nos aseguraremos de que tenemos la conexión de datos adecuada, y en caso contrario, crearemos una.

En esta misma ventana, podremos crear un nombre para el nombre de la conexión que será utilizado en el archivo de configuración de la aplicación **App.config**. Podríamos de todas las maneras, no seleccionar esta opción que aparece seleccionada por defecto. En nuestro, hemos dejado las opciones tal y como aparecen en la imagen.

Para este ejemplo, utilizaré una conexión de datos de Microsoft SQL Server. En mi caso, la ventana con la configuración creada es la que se indica en la siguiente imagen:

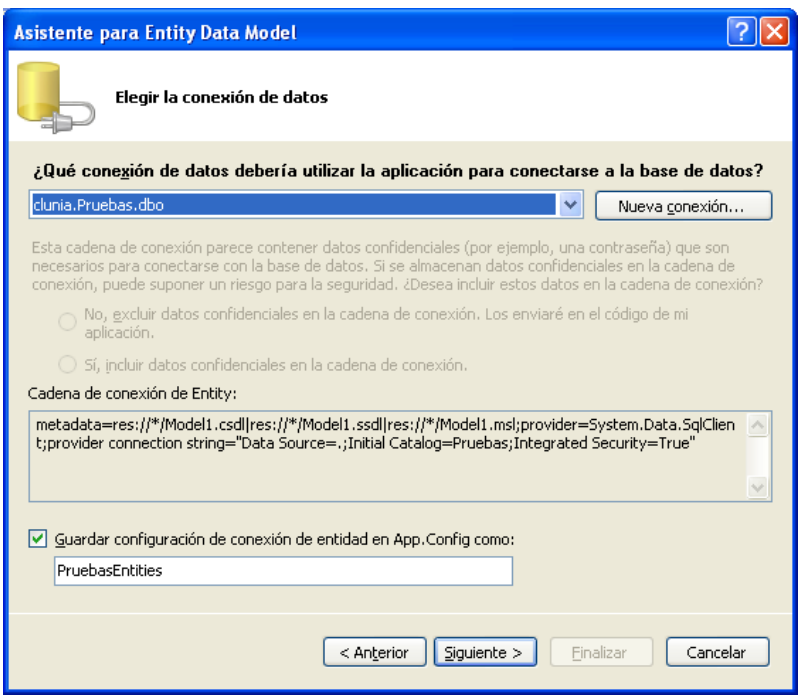


Figura 7.10.- Ventana de selección de la conexión de datos

Una vez realizado estos pasos, haremos clic en el asistente sobre el botón **Siguiente**.
Aparecerá entonces una ventana en el asistente, dentro de la cual podremos seleccionar los objetos de la base de datos que queremos utilizar.

Como podremos ver en esa ventana, podremos indicar no solo las tablas que queremos agregar al modelo, sino también las vistas y los procedimientos almacenados.

Para este ejemplo, solo seleccionaremos las tablas de nuestro modelo, que es exactamente el mismo que utilizamos en el capítulo de LINQ to SQL.

Esta selección es la que se muestra en la siguiente imagen:

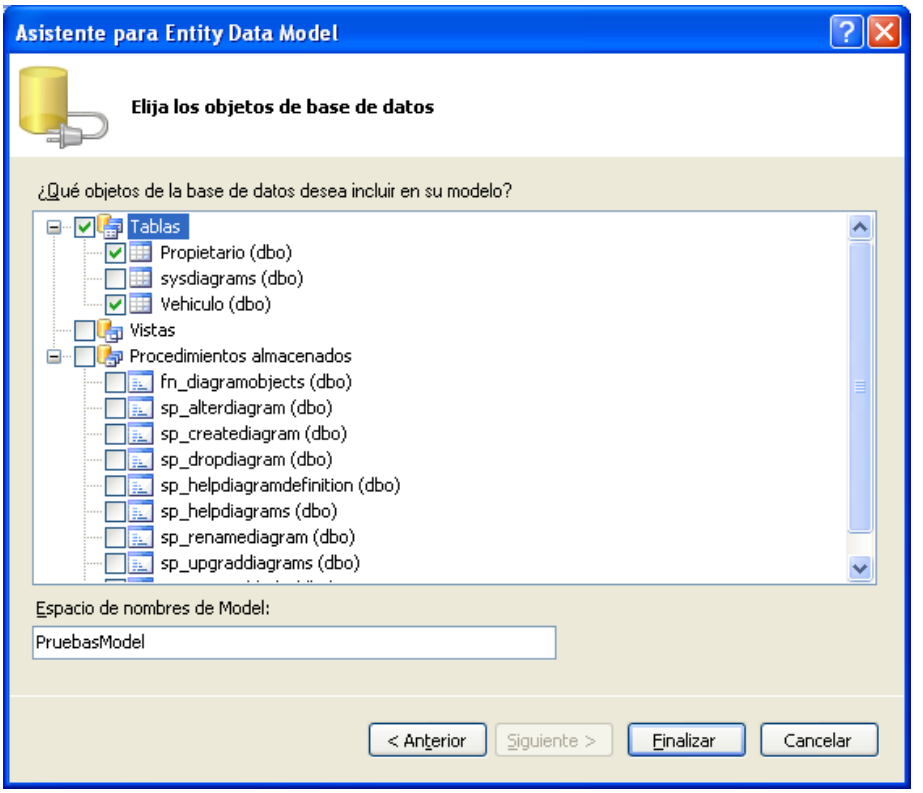


Figura 7.11.- Objetos de la base de datos seleccionados en LINQ to Entities

Para concluir con el asistente, haremos clic sobre el botón **Finalizar**. De esta manera, se generará la configuración del modelo tal y como lo hemos indicado en el asistente, y se generarán las siguientes partes:

La primera cosa que nos llamará la atención es el modelo de datos que se habrá dibujado en Visual Studio 2008, y que en nuestro ejemplo corresponderá con el siguiente:

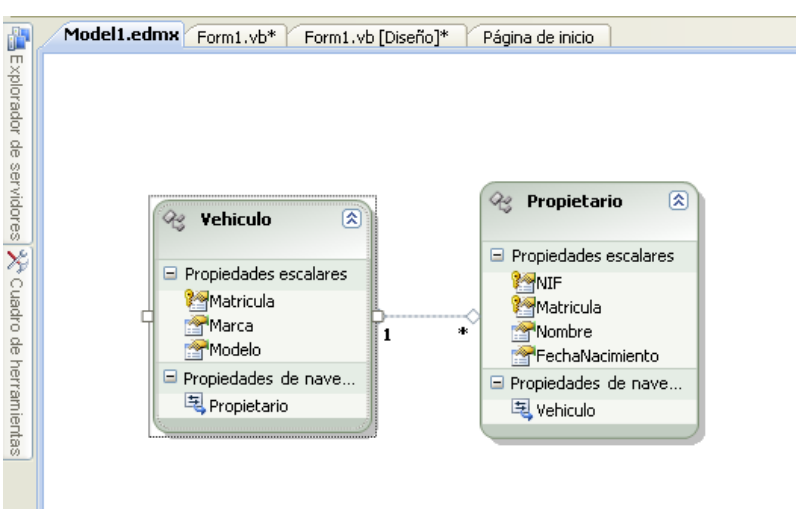


Figura 7.12.- EDM representado en Visual Studio 2008 después de ejecutar el asistente de LINQ to Entities

En nuestro diagrama, podemos localizar la entidad *Vehiculo* y la entidad *Propietario*.
Como podemos observar en el diagrama del EDM, el sistema ha determinado por nosotros una relación de tipo 1 a muchos entre la entidad *Vehiculo* y la entidad *Propietario*, por lo que podríamos decir rápidamente viendo el diagrama, que una persona puede ser propietario de más de un vehículo.

En la parte inferior de Visual Studio 2008, podremos encontrar también, una ventana que contiene los detalles de asignación de cada una de las entidades.

Un ejemplo de los detalles de asignación de la tabla *Vehiculo* es la que se indica en la siguiente imagen:

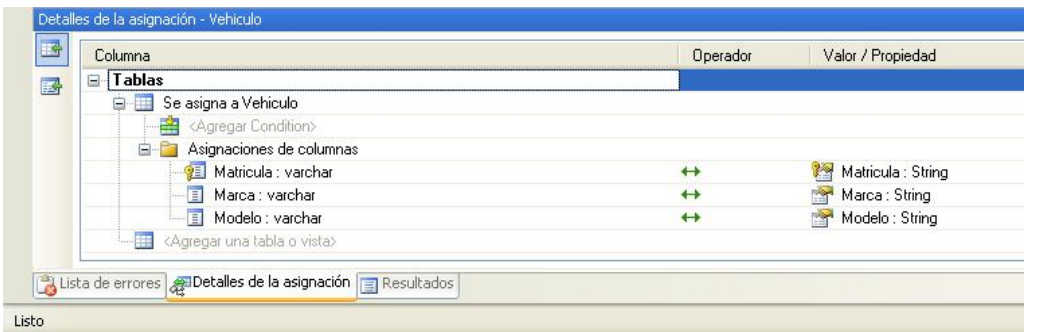


Figura 7.13.- Detalles de asignación de la entidad Vehiculo de nuestro EDM en LINQ to Entities

Si observamos detenidamente la ventana con los detalles de asignación, veremos que los campos de la tabla se han convertido en propiedades, y que los campos clave de la tabla, están así indicados en el EDM.

En sí, el EDM representa fielmente el modelo de datos relacional, permitiéndonos trabajar con él de una forma rápida y directa, evitando errores de asignación y asegurándonos una fiabilidad adecuada.

Pero hasta aquí, la preparación del EDM y algunos comentarios que hemos hecho para que se familiarice con él.

¿Cuál es nuestra siguiente tarea?. Desarrollar un ejemplo práctico que nos muestre como utilizar LINQ to Entities en nuestros desarrollos. Eso es justamente lo que haremos a continuación.

Ya tenemos preparado nuestro formulario Windows con los controles que queremos utilizar. Eso ya lo indicamos al principio de este capítulo. A continuación, deberemos escribir el código de nuestra aplicación, que quedará de la siguiente manera:

Código

```
Public Class Form1
```

```
    Private Sub Button1_Click(ByVal sender As System.Object, ByVa
l e As System.EventArgs) Handles Button1.Click

        ' Declaramos el contexto

        Dim contexto As New PruebasEntities()

        ' La conexila abre y cierra el contexto por nosotros

        ' Indicamos la instrucciLINQ que vamos a ejecutar

        Dim datos = From tablaPropietario In contexto.Propietario
, _

                    tablaVehiculo In contexto.Vehiculo _

                    Where tablaPropietario.Matricula = tablaVehic
ulo.Matricula _

                    Order By tablaPropietario.Nombre _

                    Select tablaPropietario.Nombre, tablaPropieta
rio.NIF, tablaVehiculo.Marca, tablaVehiculo.Modelo

        ' Mostramos los datos en el control DataGridView

        Me.DataGridView1.DataSource = datos

        ' Ajustamos el tamadel control DataGridView

        ' al tamade los datos de las filas y columnas

        Me.DataGridView1.AutoResizeColumns(DataGridViewAutoSizeCo
lumnsMode.AllCells)

        Me.DataGridView1.AutoResizeRows(DataGridViewAutoSizeRowsM
ode.AllCells)

    End Sub



End Class
```

Nuestro ejemplo en ejecución, es el que se muestra en la figura siguiente:

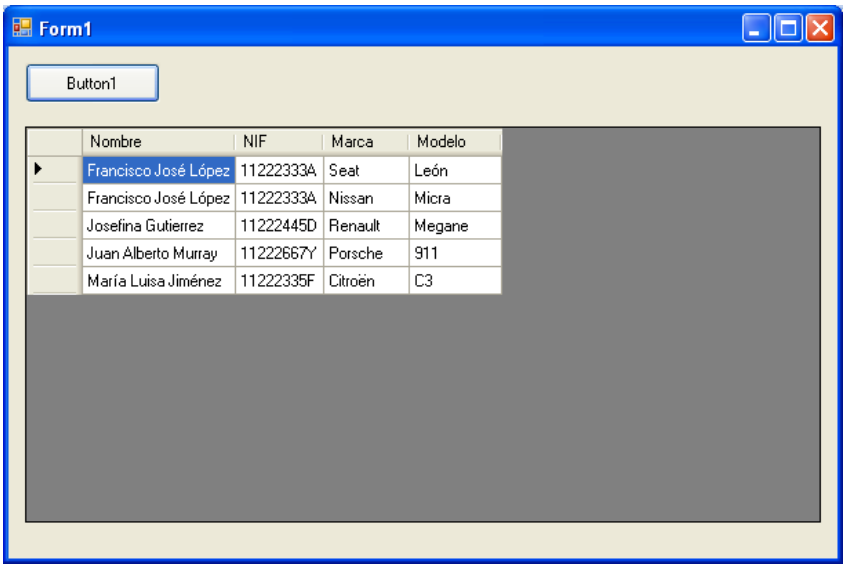


Figura 7.14.- Ejemplo del uso de LINQ to Entities en ejecución

Como podemos apreciar en el código fuente de la aplicación que hemos escrito, hemos declarado el contexto para utilizarlo en el proceso de obtención de los datos.

Posteriormente hemos declarado la instrucción LINQ a ejecutar. Como podemos ver, no hemos abierto ni cerrado ninguna conexión con la base de datos. Esto lo realiza LINQ to Entities por nosotros de forma transparente. Finalmente, hemos mostrado los datos obtenidos en la ejecución de la instrucción LINQ, dentro del control **DataGridView**.

Ahora bien, ¿dónde se encuentra la cadena de conexión?.

Si accedemos al fichero **App.Config**, veremos un apartado que en nuestro caso es de la siguiente forma:

Código

```
<connectionStrings>

  <add name="PruebasEntities" connectionString="metadata=res://*/
Model1.csdl|res://*/Model1.ssdl|res://*/Model1.msl;

provider=System.Data.SqlClient;provider connection string=&quot;D
ata Source=.;Initial Catalog=Pruebas;

Integrated Security=True;MultipleActiveResultSets=True&quot;" pro
viderName="System.Data.EntityClient" />

</connectionStrings>
```

Si nos fijamos en lo que aquí se indica, podemos observar que LINQ to Entities está compuesto por el modelo de entidades (EDM), que consiste en un fichero de extensión (**edmx**), y tres ficheros que por defecto estarán incrustados en el

ensamblado como archivo de recursos, y que contienen la extensión (**csdl**), (**ssdl**) y (**msl**).

En esta parte del documento XML, podemos ver también el proveedor de acceso a datos que vamos a utilizar, la cadena de conexión con la fuente de datos, y el nombre del proveedor de LINQ **(System.Data.EntityClient)**.

Esta parte de la aplicación es la que contiene información relativa a la propia conectividad con la fuente de datos y a los parámetros de configuración necesarios para acceder y manipular eficientemente los datos.

Ver video de esta lección - LINQ to Entities

Aplicación de ejemplo MSDN Video

A continuación vamos a desarrollar una aplicación que nos permita explorar las principales características de Visual Studio 2005. Puede descargar el código fuente en su máquina y explorarlo o modificarlo para conocer de forma práctica cómo usar Visual Basic para desarrollar aplicaciones reales. También puede explorar los videos donde construimos esta aplicación paso a paso.

La aplicación

MSDN Video es una aplicación de ejemplo empresarial desarrollada en España por la comunidad de desarrollo y Microsoft. Puede utilizarse para comprobar las mejores prácticas en la construcción aplicaciones distribuidas, escalables y con distintos tipos de clientes (Windows, Web y móviles).</P<

La aplicación que desarrollaremos en este tutorial es una versión reducida de MSDN Video donde podrá ver Visual Studio 2005 en acción en un escenario cliente / servidor sencillo.

Nota:
MSDN Video utiliza una base de datos para almacenar su estado. Si desea ejecutar la aplicación en su máquina necesitará SQL Server 2005 Express instalado.

Videos explicativos

Si desea ver cómo se hizo esta aplicación puede visionar estos videos donde desarrollamos la aplicación paso a paso:

- Video 1. Creación de la base de datos y el acceso a datos
- Video 2. Esqueleto de formularios
- Video 3. Formulario Nuevo Socio
- Video 4. Formulario Alquilar
- Video 5. Formulario Devolver
- Video 6. Conclusiones