**Introducción a LINQ en Visual Basic**

Language-Integrated Query (LINQ) agrega funcionalidades de consulta a Visual Basic y proporciona funcionalidades sencillas y eficaces para trabajar con todo tipo de datos.

La ventaja es que LINQ presenta las consultas como parte del lenguaje de Visual Basic. Utiliza una sintaxis unificada independientemente del tipo de datos.

LINQ permite consultar datos desde una base de datos de SQL Server, XML, matrices y colecciones en memoria, conjuntos de datos ADO.NET o cualquier otro origen de datos remoto o local que admita LINQ.

Los resultados se devuelven como objetos con establecimiento inflexible de tipos. Estos objetos admiten IntelliSense lo que facilita la edición y depuración.

Las consultas LINQ se pueden usar como el origen de consultas adicionales para refinar los resultados. También se pueden enlazar a los controles para que los usuarios puedan ver y modificar con facilidad los resultados de la consulta.

Por ejemplo, el código siguiente muestra una consulta LINQ que devuelve una lista de clientes de una colección y los agrupa basándose en su ubicación.

Dim customers As List(Of Customer) = GetCustomerList()

Dim customersByCountry = From cust In customers \_

Order By cust.Country, cust.City \_

Group By CountryName = cust.Country \_

Into RegionalCustomers = Group, Count() \_

Order By CountryName

For Each country In customersByCountry

Console.WriteLine(country.CountryName & \_

" (" & country.Count & ")" & vbCrLf)

For Each customer In country.RegionalCustomers

Console.WriteLine(vbTab & customer.CompanyName & \_

" (" & customer.City & ")")

Next

Next

**Proveedores linq**

**Linq to objects:** Permite consultar colecciones y objetos de memoria. Para ello, el objeto tiene que implementar la interfaz IEnumerable o IEnumerable(T).

**Linq to sql:** Permite consultar y modificar los datos de una base de datos SQLServer. El diseño de los objetos LINQTOSQL se realiza mediante el diseñador relacional de objetos que permite crear las clases LINQ a partir de las tablas de la base de datos, así como los métodos a partir de los procedimientos almacenados de las mismas.

**Linq to XML:** Permite consultar y modificar XML, bien desde memoria (por ejemplo sobre una “isla” XML) bien desde un archivo.

**Linq to DataSet:** Permite consultar y modificar un DataSet mediante una consulta de Linq.

**Estructura de una consulta linq y operadores de consulta.**

Está formada por una combinación de cláusulas de consulta que identifican los orígenes de datos y las variables de iteración de la consulta. Una expresión de consulta también puede incluir instrucciones para ordenar, filtrar, agrupar y combinar los cálculos que se van a aplicar a los datos de origen.

**Claúsula FROM:** Se necesita una cláusula From o Aggregate para iniciar una consulta. Una cláusula From especifica una colección de origen y una variable de iteración de una consulta.

Ejemplo:

VB

' Returns the company name for all customers for whom

' State is equal to "WA".

Dim names = From cust In customers \_

Where cust.State = "WA" \_

Select cust.CompanyName

C#

from cust In customers where cust.state=”WA”

select new {cust.CompanyName};

**Claúsula SELECT:** Se necesita una cláusula From o Aggregate para iniciar una consulta. Una cláusula From especifica una colección de origen y una variable de iteración de una consulta y la claúsula SELECT selecciona las columnas a devolver.

Ejemplo:

Dim customerList = From cust In customers \_

Select cust.CompanyName, cust.CustomerID

C#

from cust In customers \_

Select new {cust.CompanyName,cust.CustomerID}

**Claúsula WHERE:** Especifica una condición de filtrado de una consulta.

Ejemplo:

Dim names = From product In products \_

Where product.Category = "Beverages" \_

Select product.Name

**Cláusula Order by:** Especifica las columnas por las que se ordenarán los resultados de la consulta.

Ejemplo:

Dim titlesAscendingPrice = From b In books \_

Order By b.price

**Cláusula join:** Combina 2 colecciones en una sola relacionando registros con campos coincidentes en ambas colecciones. Dichos campos se especifican en la cláusula On.

Ejemplo:

Dim customerList = From cust In customers \_

Join ord In orders On \_

cust.CustomerID Equals ord.CustomerID \_

Select cust.CompanyName, cust.CustomerID, \_

Ord.OrdId

**Cláusula Group by:** Se utiliza para agrupar los resultados de una consulta que tenga un mismo valor para una columna. Puede emplearse para la aplicación de funciones de agregado a cada uno de los grupos.

Ejemplo:

Dim orderList = From order In orders \_

Order By order.OrderDate \_

Group By OrderDate = order.OrderDate \_

Into OrdersByDate = Group

**Cláusula Group join:** Combina 2 colecciones en una única colección jerárquica, en la práctica funciona igual que el LEFT OUTER JOIN de SQL.

Ejemplo:

Dim customerList = From cust In customers \_

Group Join ord In orders On \_

cust.CustomerID Equals ord.CustomerID \_

Into CustomerOrders = Group, \_

TotalOfOrders = Sum(ord.Total) \_

Select cust.CompanyName, cust.CustomerID, \_

CustomerOrders, TotalOfOrders

**Claúsula Aggregate:** Aplica una o más funciones de agregado a una colección.

Ejemplo:

Dim orderTotal = Aggregate order In orders \_

Into Sum(order.Total)

También puede usarse para crear un campo calculado en la consulta.

Dim customerMax = From cust In customers \_

Aggregate order In cust.Orders \_

Into MaxOrder = Max(order.Total) \_

Select cust.CompanyName, MaxOrder

**Cláusula** [**Let**](http://msdn.microsoft.com/es-es/library/bb531379.aspx) **:** Calcula un valor y lo asigna a una nueva variable de la consulta.

Ejemplo:

Dim discountedProducts = From prod In products \_

Let Discount = prod.UnitPrice \* 0.1 \_

Where Discount >= 50 \_

Select prod.Name, prod.UnitPrice, Discount

[**Cláusula Distinct:**](http://msdn.microsoft.com/es-es/library/bb384803.aspx) Opcional. Restringe los valores de la variable de iteración actual para eliminar los valores duplicados de los resultados de la consulta.

Ejemplo:

Dim cities = From item In customers \_

Select item.City \_

Distinct

**Cláusula Skip:** Omite un número especificado de elementos de una colección y, a continuación, devuelve los elementos restantes.

Ejemplo:

Dim customerList = From cust In customers \_

Skip 10

[**Skip While**](http://msdn.microsoft.com/es-es/library/bb531331.aspx) : Opcional. Omite los elementos de una colección en tanto que una condición especificada sea true y, a continuación, devuelva los elementos restantes.

Ejemplo

Dim customerList = From cust In customers \_

Skip While IsSubscriber(cust)

[**Cláusula**](http://msdn.microsoft.com/es-es/library/bb531351.aspx) **Take:** Devuelve un número especificado de elementos contiguos desde el principio de una colección.

Ejemplo:

Dim customerList = From cust In customers \_

Take 10

**Cláusula** [**Take While:**](http://msdn.microsoft.com/es-es/library/bb513984.aspx) Opcional. Incluye los elementos de una colección en tanto que una condición especificada sea true y omita los elementos restantes.

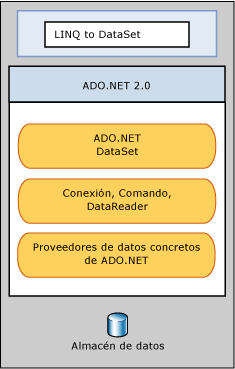
Ejemplo:

Dim customersWithOrders = From cust In customers \_

Order By cust.Orders.Count Descending \_

Take While HasOrders(cust)

**Linq to DataSet**



Linq to dataset permite escribir consultas de interrogación de datos permitiendo la escritura de las mismas en el mismo lenguaje de programación utilizado, en lugar de tener que hacerlo en otro diferente. Como vemos en la figura se basa en ADO.NET 2.0.

Vamos a ver algunos ejemplos de consultas en DataSets tipados.

**Consultas de tabla única:** La siguiente consulta recupera el id y el nombre del usuario cuyo código es “aaaaa”.

Dim **query =** From **u** In **DsForo1.Usuarios** Where **u.idusuario =** "aaaaa" **\_** Select **u.idusuario, u.Nombre**

ForEach **x** In **query**

**Console.WriteLine(“{0}-{1}”,x.idusuario,x.Nombre)**

Next

**Consultas entre tablas:** La siguiente consulta recupera los nombres de todos los usuarios junto con los mensajes que han escrito en un foro.

Dim q = From us In DsForo1.Usuarios Join men In DsForo1.Mensajes \_

On men.Idusuario Equals us.idusuario Select us.Nombre, us.Apellidos, men.Mensaje

**Creación de Dataview:** En el siguiente código vemos como crear un DataView a partir de una consulta LINQ to DataSet. Es necesario convertir la colección de resultados devueltos por la query para poder, por ejemplo, enlazarlos a un DatagridView.

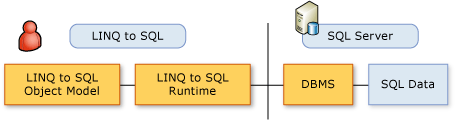
Dim q = From us In DsForo1.Usuarios Where us.idusuario = "aaaaa" Select us

Me.DataGridView1.DataSource = q.AsDataView

También hay que tener en cuenta que para que la query disponga del método AsDataView, la consulta ha de devolver un resultado del tipo Colección de filas, ya que de otro modo no existiría tal método. Por ejemplo, hay problemas si la consulta representa una join, o especificamos uno o varios campos de la tabla en la sentencia Select.

**Linq to SQL**

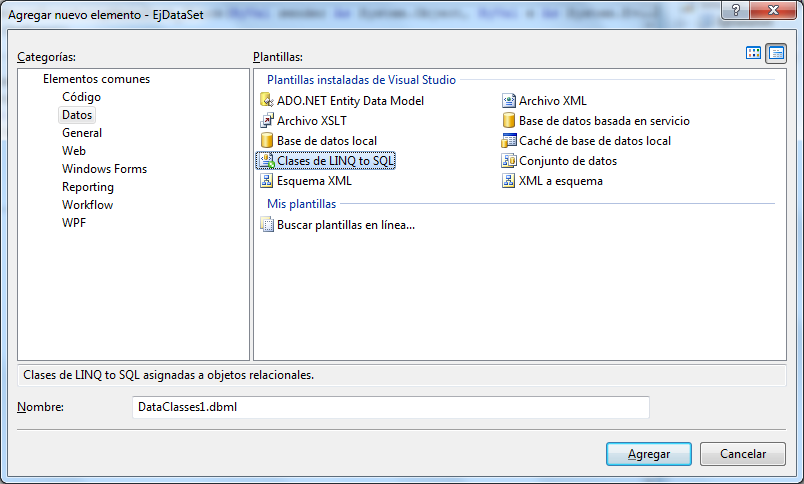
En LINQ to SQL, el modelo de datos de una base de datos relacional se asigna a un modelo de objetos de negocio expresado en el lenguaje de programación del programador. Cuando la aplicación se ejecuta, LINQ to SQL convierte a SQL las consultas integradas en el lenguaje en el modelo de objetos y las envía a la base de datos para su ejecución. Cuando la base de datos devuelve los resultados, LINQ to SQL los vuelve a convertir en objetos con los que pueda trabajar en su propio lenguaje de programación.



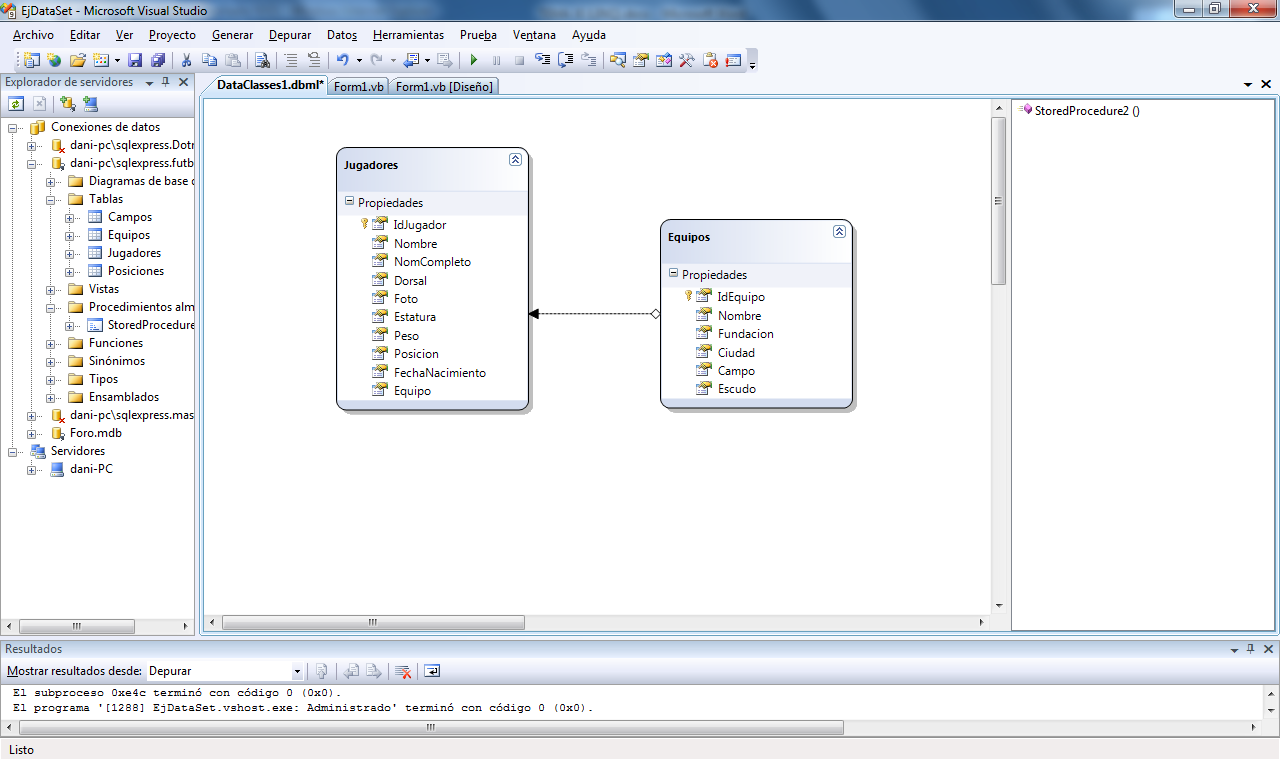
|  |  |
| --- | --- |
| Modelo de objetos de LINQ to SQL | Modelo de datos relacionales |
| Clase de entidad | Tabla |
| Miembro de clase | Column |
| Asociación | Relación de clave externa |
| Method | Procedimiento almacenado o función |

La correspondencia entre los elementos del modelo relacional y los objetos de negocio es como sigue.

**La creación de las clases de negocio** se realiza mediante el diseñador relacional de objetos. Para ello agregamos un nuevo elemento al proyecto de tipo *clases de LINQ to SQL.*



El Diseñador relacional de objetos (Diseñador relacional de objetos) proporciona una superficie de diseño visual para crear clases de entidad y asociaciones (relaciones) de [LINQ to SQL](http://msdn.microsoft.com/es-es/library/bb386976.aspx) basadas en los objetos de una base de datos. Es decir, el Diseñador relacional de objetos se usa para crear un modelo de objetos en una aplicación que se asigna a los objetos de una base de datos. También genera una clase [DataContext](http://msdn.microsoft.com/es-es/library/system.data.linq.datacontext.aspx) con establecimiento inflexible de tipos que se usa para enviar y recibir datos entre las clases de entidad y la base de datos. El Diseñador relacional de objetos también proporciona la funcionalidad para asignar los procedimientos almacenados y funciones a los métodos de [DataContext](http://msdn.microsoft.com/es-es/library/system.data.linq.datacontext.aspx) con el fin de devolver datos y rellenar las clases de entidad. Por último, el Diseñador relacional de objetos permite diseñar relaciones de herencia entre las clases de entidad.



**Agregar las clases de entidad**: arrastraremos las tablas desde la conexión elegida del el explorador de servidores al panel izquierdo.

**Agregar métodos:**  arrastraremos los procedimientos almacenados en lugar de las tablas.

Los campos de la base de datos se convierten en propiedades de las clases de entidad, las relaciones en asociaciones y los procedimientos almacenados en métodos.

Una vez generada la clase DataContext correspondiente veamos algunos ejemplos de consultas con objetos LINQTOSQL.

**Consultas de tabla única:** Con la siguiente consulta recuperamos el id y el nombre de todos los jugadores del equipo cuyo código es 1.

Dim d As New DataClasses1DataContext

Dim query = From j In d.Jugadores Where j.Equipo = 1 Select j.Nombre

Nótese que previamente a la consulta creamos una instancia al DataContext generado por el diseñador. Las colecciones de dicha instancia, que se corresponden con las tablas de la base de datos, serán las que se utilizan como colecciones a consultar en la query.

**Consultas entre tablas:**  Con la siguiente consulta recuperamos el nombre completo de un jugador, junto con el equipo en el que milita.

Dim query = From j In d.Jugadores Join eq In d.Equipos On j.Equipo Equals eq.IdEquipo Select j.Nombre, EquipoActual = eq.Nombre

Nótese que ha habido que utilizar un “alias” para la columna nombre del equipo, debido a que ya recuperábamos una columna con ese identificador para el nombre del jugador.

**Agrupación y resumen:** Con la siguiente consulta recuperamos los nombres de los equipos existentes, junto con el número de jugadores registrados de cada equipo, ordenados por nombre de jugador.

Dim d As New DataClasses1DataContext

Dim query = From j In d.Jugadores Join k In d.Equipos On j.Equipo Equals k.IdEquipo \_

Order By j.Nombre Group By Equipos = k.Nombre Into Count(j.IdJugador)

Observamos como al agrupar por nombre de equipo ponemos un alias a esa columna, en el ejemplo “Equipos”.

**Actualizar la base de datos mediante una instancia de la clase de entidad.**

**Actualizar filas:** En el siguiente fragmento de código actualizamos las columnas ciudad y fundación de una fila de la tabla equipos.

Dim f As New futbolDataContext()

Dim eq = (From x In f.Equipos Where x.IdEquipo = 2 Select x).First

eq.Nombre = "F.C.Barcelona"

eq.Ciudad = "Ciudad Condal"

f.SubmitChanges()

En primer lugar seleccionamos la fila a modificar. Nótese que aplicamos el método First para indicar que la consulta devuelve un resultado de fila única (de lo contrario devolvería una colección de un solo elemento).

Modificamos la fila seleccionada y mediante la orden *SubmitChanges*  el datacontext actualiza la base de datos asociada.

**Insertar filas:** En el siguiente fragmento de código añadimos una nueva fila a la tabla de equipos.

Dim x As New Equipos

x.Nombre = "Real Sociedad"

x.Fundacion = "09/09/1910"

x.Campo = "3"

x.Ciudad = "Donostia"

Dim EQ As New futbolDataContext

EQ.Equipos.InsertOnSubmit(x)

EQ.SubmitChanges()

Primero creamos una instancia de la clase de entidad Equipo. A continuación actualizamos sus propiedades, que se corresponden con las columnas de la tabla en la Base de Datos.

A continuación, creamos un DataContext y a partir de la propiedad Equipos del mismo, insertamos la nueva fila mediante *InsertOnSubmit*.

Por último enviamos los cambios a la base de datos mediante el método *SubmitChanges.*

**Eliminar filas:** La siguiente consulta elimina un equipo de la base de datos.

Dim f As New futbolDataContext

Dim eq = (From x In f.Equipos Where x.IdEquipo = 3).First

f.Equipos.DeleteOnSubmit(eq)

f.SubmitChanges()

Primero elegimos la fila a eliminar mediante una consulta de linq. Nótese que para que la sentencia devuelva una única fila usamos el método first (de lo contrario devolvería una colección de un elemento).

A continuación mediante el método *submitChanges* de la colección de equipos del Datacontext la marcamos para eliminar mediante *DeleteOnSubmit.*

Por último actualizamos los cambios en la Base de Datos mediante el método *submitchanges.*

**Nota:** tanto en las actualizaciones como en los borrados, si lo que queremos es hacerlos de forma masiva, recuperaremos una colección de filas y la recorreremos mediante un *for each,* aplicando los cambios o marcando para eliminar (según proceda) cada fila individualmente.

**Para saber más.**

En este documento nos hemos centrado en las clases linq to Dataset y linq to SQL. Se propone ampliar información sobre los demás proveedores.

Referencia: <http://msdn.microsoft.com/es-es/library/bb397926.aspx>

Tampoco hemos explorado en profundidad las clases de entidad generadas por el diseñador de objetos relacionales. Para ampliar información al respecto.

Referencia: <http://msdn.microsoft.com/es-es/library/bb386989.aspx>

<http://thinkingindotnet.wordpress.com/2007/05/30/linq-to-sql-2%c2%aa-parte-definiendo-nuestras-clases-del-modelo-de-datos/>

Por último, un aspecto interesante a tener en cuenta para la programación con LINQ es el uso de expresiones lambda.

Referencias: <http://msdn.microsoft.com/es-es/library/system.linq.expressions.lambdaexpression.aspx>

<http://msdn.microsoft.com/es-es/library/bb397687.aspx> (para C#)