**Introducción a LINQ en C#**

Language-Integrated Query (LINQ) proporciona funcionalidades de consulta a Visual C# 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.

var customers As List< Customer> = GetCustomerList();

var customersByCountry = from cust In customers \_

Order By cust.Country, cust.City \_

Group cust By cust.Country \_

Into RegionalCustomers select RegionalCustomer.Count();

foreach (var country In customersByCountry

Console.WriteLine(country.CountryName + " (" + country.Count + ")\n")

Foreach(varcustomer In country.RegionalCustomers

Console.WriteLine(vbTab + customer.CompanyName +

" (" + customer.City + ")\n");

**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:

// Returns the company name for all customers for whom

// State is equal to "WA".

var names = from cust in customers

where cust.State == "WA"

select cust.CompanyName;

**Claúsula SELECT:** Se necesita una cláusula from para iniciar una consulta. Una cláusula from especifica una colección de origen y una variable de iteración de una consulta.

Ejemplo:

var customerList = from cust In customers \_

select new{compañia=cust.CompanyName, IdCliente=cust.CustomerID};

Cuando se quieren devolver más de una columna, pero no todas las de la tabla en cuestión, en c# es necesario crear una nueva clase, que será anónima, para determinar de qué tipos son los elementos de la colección devuelta.

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

Ejemplo:

Var names = from product in products \_

where product.Category == "Beverages" \_

select product.Name

//Recordar que el operador de comparación es == en c#

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

Ejemplo:

var 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:

var customerList = from cust In customers

join ord In orders on

cust.CustomerID equals ord.CustomerID

select new{compañia=cust.CompanyName,

Idcliente= cust.CustomerID};

**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:

var orderList = from order in orders \_

Order By order.OrderDate \_

Group order By order.OrderDate into gr

Select new {Fecha=gr.Key,Cuantos=gr.Count()};

//devuelve todas las fechas de pedido, junto con el número de //pedidos en esa fecha

**Combinar el group by con el join, y devolver claves de más de un campo**

Ejemplo:

Var orderList=from order in orders join detail in OrderDetails

On orders.orderId equals details.orderId

Group order by new {Orden=order.OrderId ,

cliente=order.CustomerId} into gr

select new {Cliente=gr.key.cliente, Orden=gr.key.orden,gr.count()};

Para poder utilizar 2 columnas de agrupación, hay que crear una nueva clase anónima de agrupación, con dos campos que formarán parte de la clave de agrupación.

No existen funciones de agregado(cláusula Aggregate de VB) pero podemos utilizar la función Count de cada grupo, ya que cada grupo es una Colección IEnumerable.

[**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. Se aplica al conjunto de resultados devueltos.

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:

var 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

var customerList = (From cust In customers select cust). 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 select cust). 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.

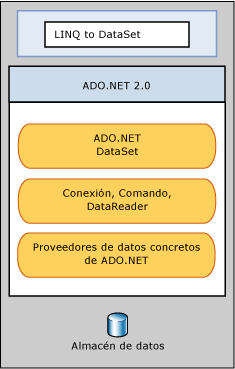
Ejemplo:

Var customersWithOrders = (from cust In customers

Order By cust.Orders.Count Descending

Select cust).Take While asOrders(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”.

var **query =** from **u** In **DsForo1.Usuarios** Where **u.idusuario =** "aaaaa" **\_** Select new{Id= **u.idusuario,Nombre= u.Nombre}**

foreach(var **x** in **query)**

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

}

**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 new{Nom= us.Nombre,Ape= us.Apellidos,Mens= 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.

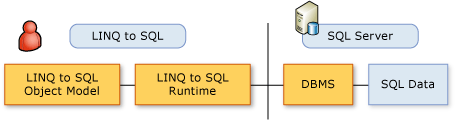
Dim q = From us In DsForo1.Usuarios Where us.idusuario = "aaaaa" Select us

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**

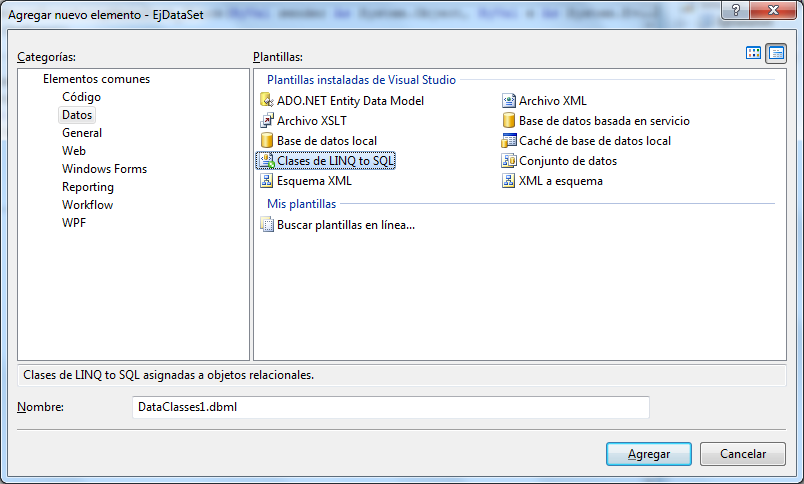
En LINQ to SQL, el modelo de datos de una base de datos relacional se asigna a un modelo de objetos de negocio expresado en el lenguaje de programación del programador. Cuando la aplicación se ejecuta, LINQ to SQL convierte a SQL las consultas integradas en el lenguaje en el modelo de objetos y las envía 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.



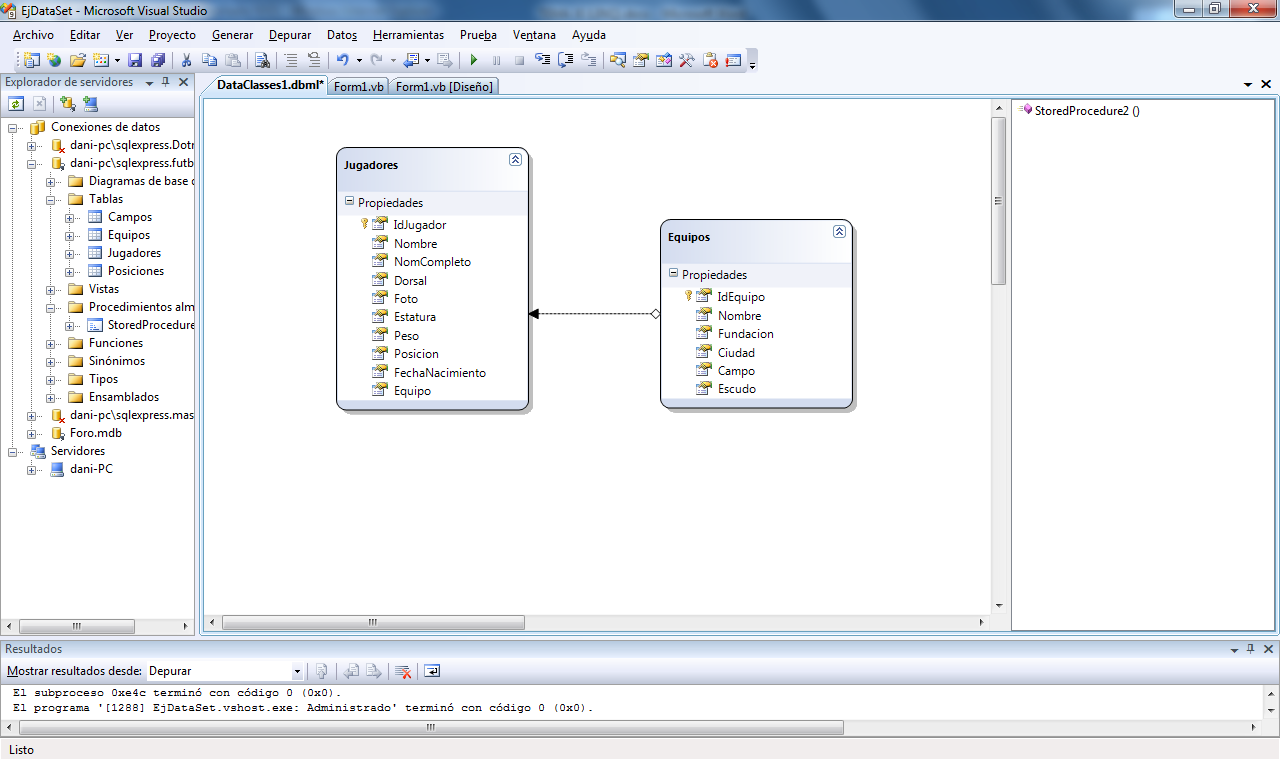
|  |  |
| --- | --- |
| Modelo de objetos de LINQ to SQL | Modelo de datos relacionales |
| Clase de entidad | Tabla |
| Miembro de clase | Column |
| Asociación | Relación de clave externa |
| Method | Procedimiento almacenado o función |

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.

DataClasses1DataContext d = New DataClasse1DataContext

var 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 new {Nom= 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.

DataClasses1DataContext d= New DataClasses1DataContext()

var 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 gr select new{cuantos= gr.Count};

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.

futbolDataContext f = New futbolDataContext()

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

eq.Nombre = "F.C.Barcelons";

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.

Equipos x = New Equipos();

x.Nombre = "Real Sociedad";

x.Fundacion = "09/02/1910";

x.Campo = "3";

x.Ciudad = "Donostia";

futbolDataContext EQ = 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.

futbolDataContext f = New futbolDataContext()

var 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#)