AT

Funcionalidad del lenguaje integrado de consultas (LINQ), con ejemplos en Visual Basic.NET

Functionality of integrated queries language (LINQ), with examples in Visual Basic.NET

Carlos Alberto Vanegas*

Fecha de recepción. Septiembre 19 de 2011

Fecha de aceptación. Octubre 14 de 2011

Resumen

Uno de los grandes retos de la programación orientada a objetos es facilitar la combinación y el acceso a cualquier tipo de información, como una característica integrada a un lenguaje de programación. Con el lenguaje integrado de consultas LINQ de .NET Framework se pueden crear consultas no solo de bases de datos relacionales y XML, sino también de matrices, colecciones en memoria, conjunto de datos ADO.NET o cualquier otro tipo de datos que admita LINQ. Las consultas integradas en los lenguajes .NET definen un conjunto de operadores de consulta estándar que hacen posible las operaciones de consulta, filtrado, enumeración y proyección. LINQ fue integrado en Visual Studio 2008 en el .NET Framework 3.5 e incluidas en los lenguajes de programación C# y Visual Basic. Este artículo hace una descripción de las funcionalidades y características del lenguaje estructurado de consultas con ejemplos en el lenguaje de programación Visual Basic.NET.

Palabras clave:

Marcos de trabajo, LINQ (lenguaje estructurado de consultas), origen de datos, consultas, colección.

* Ingeniero de Sistemas, Universidad Incca de Colombia, Especialista en Ingeniería de Software, Universidad Distrital Francisco José de Caldas, Magíster en Ingeniería de Sistemas, Universidad Nacional de Colombia, docente investigador del grupo CompuParalela adscrito a la Facultad Tecnológica de la Universidad Distrital Francisco José de Caldas. Correo electrónico: cavanegas@udistrital.edu.co

Abstract

One of the great challenges of object-oriented programming is to facilitate the combination and access to any information, such as an integrated feature of a programming language. With the integrated language of queries LINQ of .NET Framework can not only create queries relational databases and XML, but also arrays, collections in memory, ADO.NET Dataset or any other data that supports LINQ. The integrated query .NET language defines a set of standard query operators that allow query operations, filtering, enumeration and projection list. LINQ was integrating into the Visual Studio 2008 in .NET Framework 3.5 and included in the programming languages C # and Visual Basic. This article gives a description of the features and characteristics of structured query language with examples in the programming language Visual Basic.NET.

Key words:

Framework, LINQ (language integrated query), data source, query, collection.

Introducción

El lenguaje integrado de consultas LINQ (Language Integrated Query) es un proyecto de Microsoft que permite realizar consultas similares a las de SQL[1], cuya funcionalidad fue incluida en Visual Studio 2008, inicialmente en los lenguajes de programación Visual Basic .NET y C# de la plataforma .NET Framework 3.5[2]. LINQ es capaz de agregar esquemas estándares, eficaces y sencillos a la sintaxis de los lenguajes de programación C# y Visual Basic.NET para consultar y renovar datos.

Generalmente las consultas de datos se formulan como cadenas sencillas que no comprueban los tipos de datos en tiempo de compilación ni contienen las características intellisense[3]. LINQ permite que las consultas sean datos estructurados que representan información, además verifican la sintaxis del lenguaje en tiempo de compilación, los errores en compilación y se obtienen todas las características intellisense del lenguaje. Es por ello que LINQ define un conjunto de operadores de consulta estándares que permiten las operaciones de consulta, filtrado, enumeración y proyección de diferentes de datos, matrices, XML, bases de datos relacionales y orígenes de datos de terceros. Esto permite que los datos que estén en memoria se puedan manipular de una forma más rápida, lo que beneficia el rendimiento de las aplicaciones.

Proveedores LINQ

Un proveedor LINQ realiza una consulta y la traduce en comandos que podrá ejecutar el origen de datos, además convierte los datos del origen en los objetos que obtienen los resultados de la consulta. Microsoft suministra los siguientes proveedores LINQ:

LINQ to Objects: Permite consultar colecciones y matrices en memoria, implementando IEnumerable[4][1].

1 Lenguaje de consulta estructurado.

2 Componente integral de Windows que proporciona un entorno para varios lenguajes basado en estándares.

3 Proporciona elementos de código lógicos que se pueden seleccionar en un menú desplegable cuando se escribe código.

4 Interface que posee un conjunto de elementos de un mismo tipo (array, vector, lista) que permiten el acceso de manera estándar.

- LINQ to XML: Permite consultar y modificar archivos XML. Puede modificar XML en memoria o cargarlos desde un archivo. Admite crear literales XML5, lo que permite mediante programación la creación de elementos, documentos y fragmentos XML [2].

- LINQ to ADO.NET6: [3] Permite la utilización de consultas en objetos de bases de datos. Esta a su vez se divide en:

  - LINQ to SQL: permite consultar, modificar, actualizar y eliminar datos de una base de datos SQL Server. .NET incluye un diseñador relacional de objetos, el cual permite crear un modelo de objetos en una aplicación que se asigna a objetos de una base de datos. Es decir, convierte a SQL las consultas en un modelo de objetos y las envía a la base de datos para su ejecución.

  - LINQ to DataSet7:[4] permite consultar, actualizar y agregar los datos de un conjunto de datos ADO.NET ya sea un DataSet normal(solo código) o tipeado (se incluyen métodos y se puede ver su organización gráficamente).

  - LINQ to Entities: permite la consulta de los modelos de datos de entidad creados por el ORM8 Entity Framework9 [5].

- DBLinq: Proveedor similar a LINQ a SQL que permite usar bases de datos diferentes a SQL Server.

5 Permite incorporar directamente XML en el código de Visual Basic.NET.

6 Conjunto de clases que permiten el acceso a datos de .NET.

7 Representa un conjunto de datos en memoria.

8 Objeto relacional de mapeo de una base de datos.

9 Componente de ADO.NET que habilita el acceso a bases de datos mediante LINQ.

Operadores de consulta LINQ

Los operadores de consultas LINQ son métodos que constituyen el modelo LINQ y se pueden operar con cualquier proveedor y funcionan en secuencias, donde una secuencia es un objeto que implementa las interfaces IEnumerable y IQueryable[10]. Con LINQ se pueden adoptar uno o varios operadores de consulta estándar, que permiten obtener un resultado eficaz sobre un conjunto de datos. Estos operadores de consulta se encuentran en el ensamblado System.Core.dll dentro del espacio de nombres **System.Linq**. Pueden usarse en objetos que implementan IEnumerable<T> o IQueryable<T>[6]. Esto les permite operar con varios tipos, desde colecciones y matrices (secuencias) en memoria hasta bases de datos remotas que usan proveedores como LINQ to Entities y LINQ to SQL [7].

Los operadores de consulta tienen diferentes modos de ejecución: inmediato, aplazado, con transmisión por secuencias, sin transmisión por secuencias. La ejecución inmediata lee el origen de datos y origina el resultado en donde se declara la consulta; en la ejecución aplazada el resultado depende del origen de los datos, es decir que los resultados pueden ser diferentes al ejecutarse la consulta; los operadores con transmisión por secuencia no leen todos los datos del origen para proporcionar los elementos del resultado; los operadores sin transmisión por secuencias tiene que leer todos los datos de origen para proporcionar los elementos de resultado.[8]

Existen diferentes tipos de operadores, los principales se muestran a continuación:

1. **Operadores de agregación**: son operadores que calculan un valor único a partir de una colección de valores. Los métodos de realizan las operaciones de agregación son:

   - **Aggregate**: realiza una operación de agregación de una colección de valores.

10 Interface que proporciona funciones para evaluar consultas con respecto a un origen de datos.

- **Average**: calcula el promedio de una serie de valores numéricos.
- **Count**: devuelve un número entero con la cantidad de elementos de una colección de valores.
- **LongCount**: devuelve un número decimal con la cantidad de elementos de una colección de valores.
- **Min**: encuentra el número menor de una colección de números.
- **Max**: encuentra el número mayor de una colección de números.
- **Sum**: obtiene la sumatoria de una colección de números.

2. **Operadores de concatenación**: realiza la operación de unir una colección con otra. El método de concatenación es:
   - **Concat**: une dos colecciones de datos para formas una nueva colección.
3. **Operadores de conversión**: son operaciones que permiten cambiar el tipo de los objetos de entrada de una colección. Sus métodos son:
   - **Cast:** convierte los elementos de una colección en un tipo de datos específico.
   - **OfType**: permite filtrar los elementos de una colección por un tipo de datos específico.
   - **ToArray**: convierte en un vector una colección de datos.
   - **ToList**: convierte a una lista una colección de datos.
4. **Operadores de elementos**: son operaciones que devuelven un único elemento específico de una colección. Los métodos de los operadores de elementos son:
   - **ElementAt**: devuelve el elemento de un índice específico de una colección de datos.
   - **ElementatOrDefault**: devuelve el elemento de un índice específico de una colección de datos o un valor determinado si el índice se encuentra fuera de un intervalo.
   - **First**: devuelve el primer elemento de una colección.
   - **FistOrDefault**: devuelve el primer elemento de una colección o el primer valor especifico si no se encuentra ningún elemento.
   - **Last**: devuelve el último elemento de una colección.
   - **LastOrDefault**: devuelve el último elemento de una colección o el último valor especifico si no se encuentra ningún elemento.
5. **Igualdad**: se considera una operación de igualdad si los elementos correspondientes de dos colecciones son iguales y las dos colecciones tiene el mismo número de elementos. El método de esta operación es:
   - **SecuenceEqual**: compara dos colecciones para determinar si son iguales.
6. **Operadores de generación**: operadores que generan una nueva colección a partir de otra. Sus métodos son:
   - **Empty:** genera una colección vacía.
   - **Range**: genera una colección en relación con un intervalo.
   - **Repeat**: genera una colección de un único elemento, un número determinado de veces.
7. **Operadores de agrupación**: colocan los elementos de una colección en grupos que compartan una condición común. Sus métodos son:
   - **GroupBy**: agrupa los elementos de una colección por una agrupación definida.

- **Join**: une dos colecciones.
- **GroupJoin**: realiza una unión agrupada de elementos de dos colecciones.

8. **Operadores de ordenación**: operaciones que ordenan los elementos de una colección según uno o más condiciones. Los métodos son:
   - **OrderBy**: ordena los elementos de una colección en forma ascendente.
   - **OrderByDescending**: ordena los elementos de una colección en orden descendente.
   - **ThenBy**: operación secundaria que ordena los elementos de una colección en forma descendente.
   - **ThenByAscending**: operación secundaria que ordena los elementos de una colección en forma ascendente.
   - **Reverse**: ordena los elementos de una colección en forma inversa.
9. **Operadores de proyección**: son operaciones que transforman o construyen un nuevo conjunto de elementos generado a partir de una colección.
   - **Select**: permite crear una proyección de los elementos de una colección.
   - **SelectMany**: permite crear una proyección de uno o varios elementos de una colección.
10. **Operadores de cuantificación**: estos operadores devuelven un valor booleano si alguno o todos los elementos de un colección cumplen una condición.
    - **All**: devuelve verdadero si todos los elementos de una colección cumplen una condición.
    - **Any**: devuelve verdadero si alguno de los elementos de una colección cumplen una condición.
    - **Contains**: devuelve verdadero si en una colección un elemento cumple una condición específica.
    - **Where**: devuelve verdadero si se cumple una condición de filtrado.
11. **Operadores de conjunto**: son operaciones que generan un conjunto de resultados a partir de la existencia o no de elementos de una colección o varias colecciones.
    - **Distinct**: elimina los elementos repetidos de una colección.
    - **Except**: devuelve los elementos de una colección que no existen en una segunda colección.
    - **Intersect**: devuelve los elementos comunes entre dos colecciones de datos.
    - **Union**: devuelve todos los elementos de dos colecciones.

Estructuración de una consulta LINQ

Una consulta es una expresión que recupera datos u obtiene resultados de un origen de datos, la cual está conformada por una combinación de cláusulas que identifican los orígenes de datos y las variables de iteración de la consulta. . En una consulta LINQ siempre se trabajan objetos, utilizando los mismos modelos de instrucciones para consultar y convertir los datos de diferentes orígenes de datos. Para definir y utilizar consultas LINQ se deben realizar tres acciones:

1. **Obtener el origen de datos u orígenes de datos**: Se debe especificar cuáles serán los datos que se van a utilizar y donde están.
2. **Crear la consulta**: Se especifica la información que se desea obtener del origen de datos, estos, se pueden ordenar, filtrar, agrupar, etc.
3. **Ejecutar la consulta**: La ejecución de la consulta es un proceso independiente de la creación. Se puede ejecutar cuando este definida (ejecución inmediata) o se puede

guardar y ejecutarse posteriormente (ejecución diferida).

La ejecución y la creación de una consulta LINQ son operaciones distintas, el hecho de crear una consulta, no significa la recuperación de datos.

Una expresión de consulta contiene tres cláusulas: **From** o **Aggregate**, **Select** y **Where**. **From** es una cláusula inicial de una consulta y especifica el origen de datos y las variables de iteración que se usan sirven para referenciar cada elemento del origen de datos por separado. **Aggregate**, también es una cláusula inicial y aplica una o más funciones de agregado a un origen de datos, en este caso la cláusula **From** es opcional. **Select** especifica la forma y el contenido de los datos que serán devueltos y **Where** se utiliza para filtrar datos de un origen de datos.

En el siguiente ejemplo se muestran las tres partes de una operación de consulta. Se utiliza un vector de números como origen de datos. La consulta debe dar como resultado todos los números enteros mayores de 10.

- **Origen de datos**
  variable vector() es entero = {12,2,4,54,89,75,1,21,18,6}

- **Creación de la consulta**
  variable obtenernumeros =
  From numeros **In** vector
  Where números > 10
  Select numeros

- **Ejecutar la consulta**
  Para cada número
  En obtenernumeros
  Imprimir (número)

Consultas LINQ en Visual Basic.NET

El lenguaje integrado de consultas LINQ es una de las funcionalidades de consulta que tiene el lenguaje de programación Visual Basic .NET [9], y permite la creación de consultas sencillas, fáciles y eficaces con todo tipo de datos en otras palabras con LINQ se pueden crear consultas como parte de la sintaxis del lenguaje de programación Visual Basic. NET independiente del tipo de dato. [10]

En este aparte se crean ejemplos de consultas LINQ utilizando el entorno de programación del lenguaje Visual Basic .NET utilizando el tipo de proyecto **aplicación de consola**, donde se manipularán algunos de los operadores de consultas LINQ mencionados anteriormente.

Ejemplo 1: Si se tiene la siguiente matriz:

numeros={10,20,5,89,41,56,18,23,52}

a. Obtener el promedio de los valores de la matriz números.

   Dim promedio As **Double** = **Aggregate** valores_promedio **In** numeros **Into Average**()

   Console.**Write**("El promedio de los números es:" & promedio)

Figura 1. Promedio de los números.

b. Obtenerla suma de los valores de la matriz **numeros**.

   Dim suma As **Integer** = **Aggregate** sumanumeros **In** números **Into sum**()

   Console.**Write**("La suma de los números es:" & suma)

Figura 2. Suma de los números.

c. Imprimir los valores de la matriz **numeros** en forma ascendente.

   Dim ordenar = **From** valores_ordenados **In** numeros **Order By** valores_ordenados

   For Each valor **In** ordenar

Console.**Write**(valor & “,”)

Next

Figura 3. Ordenamiento de los números en forma ascendente.

d. Determinar si existe el número 89 en la matriz **numeros**.

Dim valor_existe = **Aggregate** numero **In** numeros **into Any**(numero=89)

IF (valor_existe=true) then

Console.**Write**(“Existe el número 89”)

Else

Console.**Write**(“No Existe el número 89”)

End If

Figura 4. Existencia de un número especifico en el vector.

Ejemplo 2: Si se tiene la siguiente matriz:

Nombres= {“carlos”, ”rosa”, “cristian” ,”angelita”, “andres”, “julia”}

a. Ordenar cada elemento de la matriz **nombres** en orden descendente.

Dim ordenar_nombres = **From** ordena **In** nombres **Order By** ordena **Descending**

For Each orden In ordenar_nombres

Console.Write(orden & “,”)

Next

Figura 5. Ordenamiento de la matriz nombres.

b. Obtener los nombres que contengan la letra i en la matriz **nombres**.

Dim obtener_nombres = **From** obtener_i **In** nombres **Where** obtener_i**. Contains(“i”)**

For Each nombres_con_i **In** obtener_ nombres

Console.Write(nombres_con_i & “,”)

Next

Figura 6. Nombres que contiene la letra i.

c. Imprimir la primera letra de cada elemento de la matriz **nombres.**

Dim primera = **From** letra **In** nombres **Select** letra.**Substring**(0, 1)

For Each primera_letra **In** primera

Console.Write(primera_letra & “,”)

Next

Figura 7. Primera letra de los elementos de la matriz nombres.

d. Imprimir todos los elementos de la matriz **nombres** cuya longitud sea mayor que 6.

Dim palabras = **From** palabras_6 **In** nombres **Where** palabras_6**.Length >** 6

For Each palabra_mayor_6 **In** palabras

Console.Write(palabra_mayor_6 & “,”)

Next

Figura 8. Elementos de la matriz nombres cuya longitud es mayor que 6.

Ejemplo 3: Se supondrá que se tiene en una base de datos (se utiliza la plantilla **Clases de LINQ To SQL** de Visual Basic .NET para crear la base de datos y las tablas) que contiene dos tablas: **productos** y **ventas** con la siguiente información:

Tabla 1: Tablas productos y ventas.

Productos					Ventas	
Código	**Articulo**	**Cantidad**	**Valor**		**Código**	**Cantidad**
10	reglas	12	1200		10	15
20	bolígrafos	24	2000		10	20
30	lápices	35	700		20	48
40	cuadernos	8	1500		30	12
50	colores	14	2000		20	87

a. Imprimir el código y el nombre de los artículos de la tabla productos cuyo código sea mayor que 30.

```
Dim consulta = From misproductos In vista.productos Where misproductos.codigo > 30

    Select misproductos.codigo, misproductos.articulo

For Each codigos In consulta

    Console.WriteLine(codigos)

Next
```

Figura 9. Códigos y nombre de los articulos cuyo codigo es mayor que 30.

b. Imprimir los registros de la tabla **productos** cuyo nombre de artículo sea igual a **cuadernos**.

```
Dim consulta = From misproductos In vista.productos
    Where misproductos.articulo="cuadernos"

  Select misproductos.codigo, misproductos.articulo, misproductos.cantidad,
      misproductos.valor

For Each articulos In consulta

    Console.WriteLine(articulos)

Next
```

Figura 10. Articulos cuyo nombre es igual a cuadernos.

c. Imprimir de la tabla **ventas** todos los códigos existentes sin repetir códigos.

```
Dim unicocodigo = From obtenercodigo In vista.ventas

  Select obtenercodigo.codigo Distinct

Console.WriteLine("Imprimir un único codigo de la tabla ventas")

For Each vercodigos In unicocodigo

  Console.Write(vercodigos & ",")

Next
```

Figura 11. Imprimir un único código de la tabla ventas.

d. Imprimir por cada registro de la tabla **ventas**, el código, el nombre y el valor de la tabla **productos** y la cantidad de la tabla **ventas**.

```
Dim consulta = From misproductos In vista.productos Join misventas In vista.ventas On misventas.codigo Equals misproductos.codigo

Select misproductos.codigo, misproductos.articulo, misventas.cantidad, misproductos.valor
```

```
Console.WriteLine("Ventas de Productos")

For Each codigos In consulta

    Console.WriteLine(codigos)
Next
```

Figura 12. Impresión de los productos vendidos.

Conclusiones

El lenguaje integrado de consultas LINQ de .NET Framework permite crear consultas con bases de datos relacionales, XML, matrices, colecciones en memoria, conjunto de datos ADO.NET o cualquier otro tipo de datos.

- Las consultas LINQ se pueden crear fácilmente desde el entorno de programación del lenguaje Visual Basic.NET.
- Los operadores de consultas LINQ son similares a las consultas SQL.
- Los operadores de consultas LINQ se pueden utilizar con cualquier proveedor.
- Las consultas LINQ se pueden comprobar en tiempo de compilación y además contienen la característica Intellisense.

Referencias

Infografías primarias

[1] IEnumerable Interface. Disponible en: http://msdn.microsoft.com/en-us/library/system.collections.ienumerable.aspx

[2] Información general sobre literales. Disponible en: XML, http://msdn.microsoft.com/es-es/library/bb384629.aspx

[3] ADO.NET. Disponible en: http://msdn.microsoft.com/es-es/library/e80y5yhx(v=vs.80).aspx

[4] DataSet. Disponible en: http://msdn.microsoft.com/es-es/library/ss7fbaez.aspx

[5] ADO.Net Entity Framework. Disponible en: http://geeks.ms/blogs/ciin/archive/2008/01/25/ado-net-entity-framework-linq-to-entities-entity-sql-y-entity-services-i.aspx

[6] IQueyable (interfaz). Disponible en: http://msdn.microsoft.com/es-es/library/system.linq.iqueryable.aspx

[7] Operadores de consulta estándar con LINQ. Disponible en: http://msdn.microsoft.com/es-es/magazine/cc337893.aspx

[8] Clasificación de operadores. Disponible en: http://msdn.microsoft.com/es-es/library/bb882641.aspx

[9] Inicio de Visual Basic. Disponible en: http://msdn.microsoft.com/es-es/vbasic/ms789056

[10] Introducción a LINQ en Visual Basic. Disponible en: http://msdn.microsoft.com/es-es/library/bb763068.aspx

Infografías secundarias

[11] http://msdn.microsoft.com/es-es/library/bb546138.aspx

[12] http://msdn.microsoft.com/es-es/library/bb546162.aspx

[13] http://msdn.microsoft.com/es-es/library/bb546139.aspx

[14] http://msdn.microsoft.com/es-es/library/bb546160(v=vs.90).aspx

[15] http://msdn.microsoft.com/es-es/library/bb546129(v=vs.90).aspx

[16] http://msdn.microsoft.com/es-es/library/bb546145.aspx

[17] http://msdn.microsoft.com/es-es/library/bb546140(v=vs.90).aspx

[18] http://msdn.microsoft.com/es-es/library/bb546168.aspx

[19] http://msdn.microsoft.com/es-es/library/bb546128.aspx

[20] http://msdn.microsoft.com/es-es/library/bb546153.aspx

[21] ^ a b "What is Object/Relational Mapping?" . *Hibernate Overview*. JBOSS Hibernate.

[22] http://www.hibernate.org/about/orm. Retrieved 19 April 2011.