

Entendiendo LINQ (Language Integrated Query) con C# y Mono

L.I.S. Martín Márquez <xomalli@gmail.com>

Introducción

Hoy día es común encontrar aplicaciones en las cuales hay una necesidad de reemplazar o migrar de una o de varias fuente de datos hacia otra u otras fuentes de datos similares o totalmente diferentes, si bien este cambio se da por razones de desempeño, económicas o de capacidad, este cambio implica que el desarrollador ajuste o reescriba el lenguaje de acceso a la fuente de datos (SQL, Macros, DOM, Xquery) para poner en marcha la aplicación utilizando la fuente de datos actualizada.

Un escenario así presenta las siguientes dificultades:

- Los errores de sintaxis del lenguaje de acceso a la fuente de datos no son localizables por el compilador y solo se encuentran en tiempo de ejecución.
- Obliga a que el desarrollador entienda a determinado nivel el lenguaje de manipulación de la fuente de datos, por ejemplo si es una base de datos se debe entender SQL, si es un XML entender DOM o XQuery.
- Es inevitable la mezcla de lenguajes en la solución, para el desarrollo un lenguaje de programación imperativo orientado a objetos (como C#) y un lenguaje imperativo (como SQL) para la manipulación de datos.

En este escenario LINQ proporciona una solución estupenda, ya que las consultas estarían integradas dentro del lenguaje de programación sin importar cual fuese la fuente de datos y con la consistencia de utilizar el mismo patrón para todas las consultas.

Acerca de LINQ

Language Integrated Query o LINQ es una tecnología integrada en .NET que proporciona la capacidad para consultar o manipular diversas fuentes de datos, independientes del proveedor utilizando de forma nativa la sintaxis de cualquier lenguaje de programación soportado por .NET, lo cual nos proporciona el soporte del compilador y nos permite concentrarnos únicamente en las búsquedas en lugar de cómo hacer la rutina para cada búsqueda, además la sintaxis de LINQ es similar a SQL lo que nos proporciona un estándar ya que es la misma sintaxis para todas las fuentes de datos diferentes o similares.

Dependiendo de la fuente de datos a trabajar, es el componente LINQ a utilizar, los componentes se agrupan en:

LINQ to SQL: Es el conjunto de clases, estructuras, interfaces y enumeraciones utilizadas para escribir consultas a bases de datos relacionales como PostgreSQL, SQL Server o MySQL.

LINQ to Objects: Es la API predeterminada de LINQ y permite escribir consultas para arreglos, estructuras y colecciones de objetos en memoria.

LINQ to XML: Proporciona la habilidad de escribir consultas para procesar fuentes de datos XML.

LINQ to DataSet: Es la API dedicada a trabajar con clases DataSets y DataTables, ya que aun existen aplicaciones y desarrolladores que utilizan esta solución.

En este tutorial mostraremos una serie de ejemplos con cada una de las expresiones y operadores de consulta de LINQ, utilizando el API predeterminada de LINQ o sea *LINQ to Objects*.

Trabajando LINQ con Mono

Antes de empezar a trabajar debemos tener instalado y configurado la última versión del proyecto Mono, esto se consigue desde el sitio de descarga: http://www.mono-project.com/Main_Page

Empezaremos creando una aplicación de consola, para ejemplificar sobre todo las expresiones de consulta que se utilizan con LINQ.

Esta aplicación tendrá tres colecciones de objetos (entidades) Hospital, Almacén y Artículos la relación entre un hospital, un almacén y los artículos está dada por una relación de uno a muchos, donde un hospital tiene desde uno a varios almacenes y en cada almacén hay uno o varios artículos diferentes.

Para empezar con el ejemplo, escribiremos el código de las objetos que utilizaremos en el ejemplo, este código es el siguiente:

```
class Hospital {
public int ID{set;get;}
public string NombreH {set;get;}
}
class Almacen {
public Hospital Hospital {set;get;}
public int IDAlmacen{ set;get;}
public string NombreA { set; get;}
}
class Articulo{
public Almacen Almacen {set;get;}
public int IDArticulo {set;get;}
public string Nombre {set;get;}
public double Precio{set;get;}
}
```

Ahora dentro del método **Main** escribimos el siguiente código, para iniciar las colecciones con datos donde realizaremos las búsquedas con LINQ.

```
List<Hospital> lHospitales = new List<Hospital>();
var hospitales = new Hospital[] {
new Hospital{ID = 1,NombreH="Angeles Pedregal"},
new Hospital{ID = 2, NombreH="Iero Octubre"},
new Hospital{ID = 3, NombreH="Ignacio Zaragoza"},
new Hospital{ID = 4, NombreH="Angeles Torreon"}
};
List<Almacen> lAlmacenes = new List<Almacen>();
var almacenes = new Almacen[]{
new Almacen{Hospital = hospitales[0],IDAlmacen = 1,NombreA = "Patriotismo"},
new Almacen{ Hospital = hospitales[0],IDAlmacen = 2,NombreA = "Zaragoza"},
new Almacen{ Hospital = hospitales[1],IDAlmacen = 3,NombreA = "101"},
new Almacen{ Hospital = hospitales[2],IDAlmacen = 4,NombreA = "404"},
new Almacen{ Hospital = hospitales[1],IDAlmacen = 5,NombreA = "Monterrey"},
new Almacen{ Hospital = hospitales[2],IDAlmacen = 6,NombreA = "Guadalajara"}
};
```

```

List<Articulo> lArticulos = new List<Articulo>();
var articulos = new Articulo[]{
new Articulo{Almacen = almacen[0], IDArticulo = 1005, Nombre = "HOJA SIERRA
18MM", Precio = 1.5},
new Articulo{Almacen = almacen[0], IDArticulo = 1006, Nombre = "DRENAJE BLAKE
2229 15FR C/PUNZON°", Precio = 1.5},
new Articulo{Almacen = almacen[1], IDArticulo = 1007, Nombre = "VICRYL
ETHICON 3-0", Precio = 6.22},
new Articulo{Almacen = almacen[1], IDArticulo = 1008, Nombre = "PDS II*
VIOLETA / ETHICON", Precio = 9.2},
new Articulo{Almacen = almacen[1], IDArticulo = 1009, Nombre = "IOBAN 2
ANTIMICROBIAL / 3M", Precio = 3.11},
new Articulo{Almacen = almacen[2], IDArticulo = 1010, Nombre = "COMP FEM ANAT
R/C IZQ NO 1", Precio = 10.20},
new Articulo{Almacen = almacen[3], IDArticulo = 1011, Nombre = "BASE TIBIAL
ANAT DER NO 7", Precio = 10.45},
new Articulo{Almacen = almacen[4], IDArticulo = 1012, Nombre = "INSERT ART
UNIV UHMPE 3-4 X 9MM", Precio = 12.80},
};
//Agregamos la información
foreach (Almacen a in almacen)
lAlmacen.Add(a);
foreach (Articulo r in articulos)
lArticulos.Add(r);

```

Uso de la palabra reservada var

La palabra reservada `var` se utiliza para lograr una escritura más compacta en el código ya que le indica al compilador que infiera el tipo de la variable que se le asigna al declararle el valor inicial, por ejemplo en la forma clásica donde se declara el tipo de variable de forma explícita se escribe:

```

string s = "Revista Atix";
int i = 14;
DataSet ds = new DataSet();

```

utilizando la escritura de forma implícita con `var` se escribe:

```

var s = "Revista Atix";
var i = 14;
var ds = new DataSet();

```

El uso de la palabra reservada `var` tiene las siguientes limitaciones:

- siempre se debe asignar un valor inicial a la variable al declararse
- no se puede asignar a `var` un valor `null`
- no se puede asignar más de una variable `var` en una sentencia.
- Las variables `var` solo tienen alcance local no pueden utilizarse como variables globales de clase.

Sintaxis y operadores de las consultas en LINQ

Básicamente la sintaxis de las consultas en LINQ es similar a las consultas SQL, utilizando las palabras reservadas `select`, `from`, `where`, `orderby` y `join`.

Aquí a diferencia de las consultas SQL la palabra reservada `from` va al comienzo de la consulta y al final se emplea la palabra reservada `select`, como en el código siguiente:

```
var fromQuery = from a in lAlmacenes select a;
```

Similar a SQL tenemos la palabra reservada `where` , como en el código siguiente:

```
var whereQuery = from a in lAlmacenes where a.Hospital.ID == 2 select a;
```

El uso de `orderby` y `descending` se ejemplifica en los códigos siguientes:

```
var orderbyQuery = from a in lAlmacenes orderby a.NombreA select a;
```

```
var orderbyQueryDesc = from a in lAlmacenes orderby a.NombreA descending select a;
```

Continuando con la similitud con SQL no debe faltar la palabra reservada `join`, como en el siguiente código:

```
var joinQuery = from r in lArticulos join a in lAlmacenes on r.Almacen.IDAlmacen equals a.IDAlmacen select new {r.Nombre,a.NombreA };
```

Cada una de las consultas que se realizan con LINQ, regresan una interfaz `IEnumerable` , por eso en el método `Display` El parámetro que recibe es un `Ienumerable`, el código completo del ejemplo se lista a continuación:

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace LinqAtix
{
class Program{
public static void Main(string[] args){
    var hospitales = new Hospital[] {
        new Hospital{ID = 1,NombreH="Angeles Pedregal"},
        new Hospital{ID = 2, NombreH="Ilero Octubre"},
        new Hospital{ID = 3, NombreH="Ignacio Zaragoza"},
        new Hospital{ID = 4, NombreH="Angeles Torreón"}
    };
    List<Almacen> lAlmacenes = new List<Almacen>();
    var almacenes = new Almacen[]{
        new Almacen{Hospital = hospitales[0],IDAlmacen = 1,NombreA = "Patriotismo"},
        new Almacen{ Hospital = hospitales[0],IDAlmacen = 2,NombreA = "Zaragoza"},
        new Almacen{ Hospital = hospitales[1],IDAlmacen = 3,NombreA = "101"},
        new Almacen{ Hospital = hospitales[2],IDAlmacen = 4,NombreA = "404"},
        new Almacen{ Hospital = hospitales[1],IDAlmacen = 5,NombreA = "Monterrey"},
        new Almacen{ Hospital = hospitales[2],IDAlmacen = 6,NombreA = "Guadalajara"}
    };
    List<Articulo> lArticulos = new List<Articulo>();
    var articulos = new Articulo[]{
        new Articulo{Almacen = almacenes[0],IDArticulo = 1005,Nombre = "HOJA SIERRA
18MM",Precio = 1.5},
        new Articulo{Almacen = almacenes[0],IDArticulo = 1006,Nombre = "DRENAJE BLAKE
2229 15FR C/PUNZON°",Precio = 1.5},
        new Articulo{Almacen = almacenes[1],IDArticulo = 1007,Nombre = "VICRYL
ETHICON 3-0",Precio = 6.22},
        new Articulo{Almacen = almacenes[1],IDArticulo = 1008,Nombre = "PDS II*
VIOLETA / ETHICON",Precio = 9.2},
        new Articulo{Almacen = almacenes[1],IDArticulo = 1009,Nombre = "IOBAN 2
ANTIMICROBIAL / 3M",Precio = 3.11},
        new Articulo{Almacen = almacenes[2],IDArticulo = 1010,Nombre = "COMP FEM ANAT
R/C IZQ NO 1",Precio = 10.20},
```

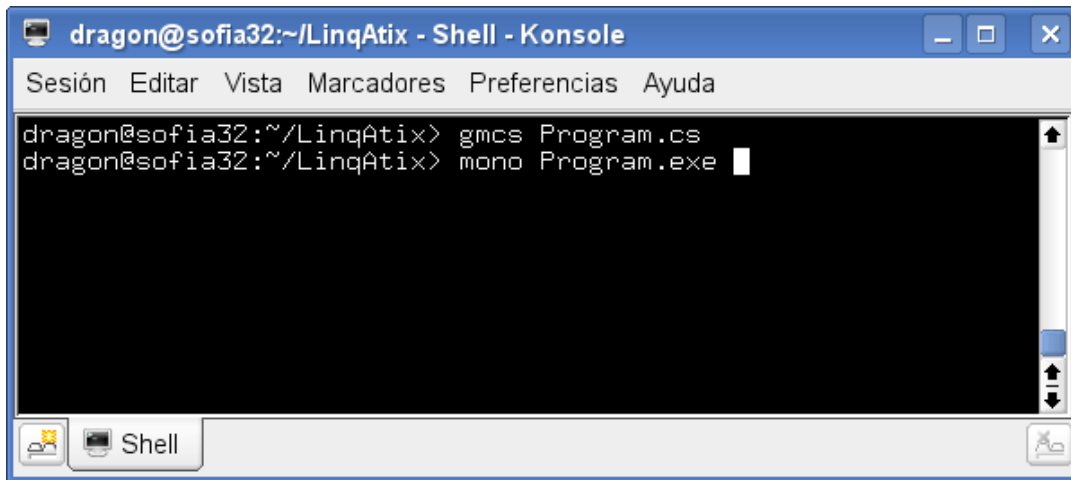
```

    new Articulo{Almacen = almacenes[3], IDArticulo = 1011, Nombre = "BASE TIBIAL
ANAT DER NO 7", Precio = 10.45},
    new Articulo{Almacen = almacenes[4], IDArticulo = 1012, Nombre = "INSERT ART
UNIV UHMPE 3-4 X 9MM", Precio = 12.80},
};
//Agregamos la información
foreach(Almacen a in almacenes)
lAlmacenes.Add(a);
foreach(Articulo r in articulos)
lArticulos.Add(r);
//Consultas Linq:Trabajando con FROM
var fromQuery = from a in lAlmacenes select a;
//listamos todos los almacenes
Display(fromQuery);
//listamos todos los almacenes donde el id Hospital sea el 2
var whereQuery = from a in lAlmacenes where a.Hospital.ID == 2 select a;
Display(whereQuery);
//listamos todos los almacenes en orden alfabetico
var orderByQuery = from a in lAlmacenes orderby a.NombreA select a;
Display(orderbyQuery);
//listamos todos los almacenes en orden alfabetico en forma descendente
var orderByQueryDesc = from a in lAlmacenes orderby a.NombreA descending
select a;
Display(orderbyQueryDesc);
//listamos el nombre de los articulos y el nombre del Hospital
var joinQuery = from r in lArticulos join a in lAlmacenes on
r.Almacen.IDAlmacen equals a.IDAlmacen select new {r.Nombre, a.NombreA };
Console.WriteLine("=====Resultados del Query=====");
foreach(var j in joinQuery)
Console.WriteLine("Articulo: {0}\tAlmacén: {1}", j.Nombre, j.NombreA);
Console.ReadKey(true);
}
static void Display(IEnumerable<Almacen> ie){
Console.WriteLine("=====Resultados del Query=====");
foreach(Almacen a in ie)
Console.WriteLine("{0}\t{1}", a.IDAlmacen, a.NombreA);
}
}

class Hospital {
public int ID{set;get;}
public string NombreH {set;get;}
}
class Almacen {
public Hospital Hospital {set;get;}
public int IDAlmacen{ set;get;}
public string NombreA { set; get;}
}
class Articulo{
public Almacen Almacen {set;get;}
public int IDArticulo {set;get;}
public string Nombre {set;get;}
public double Precio{set;get;}
}
}

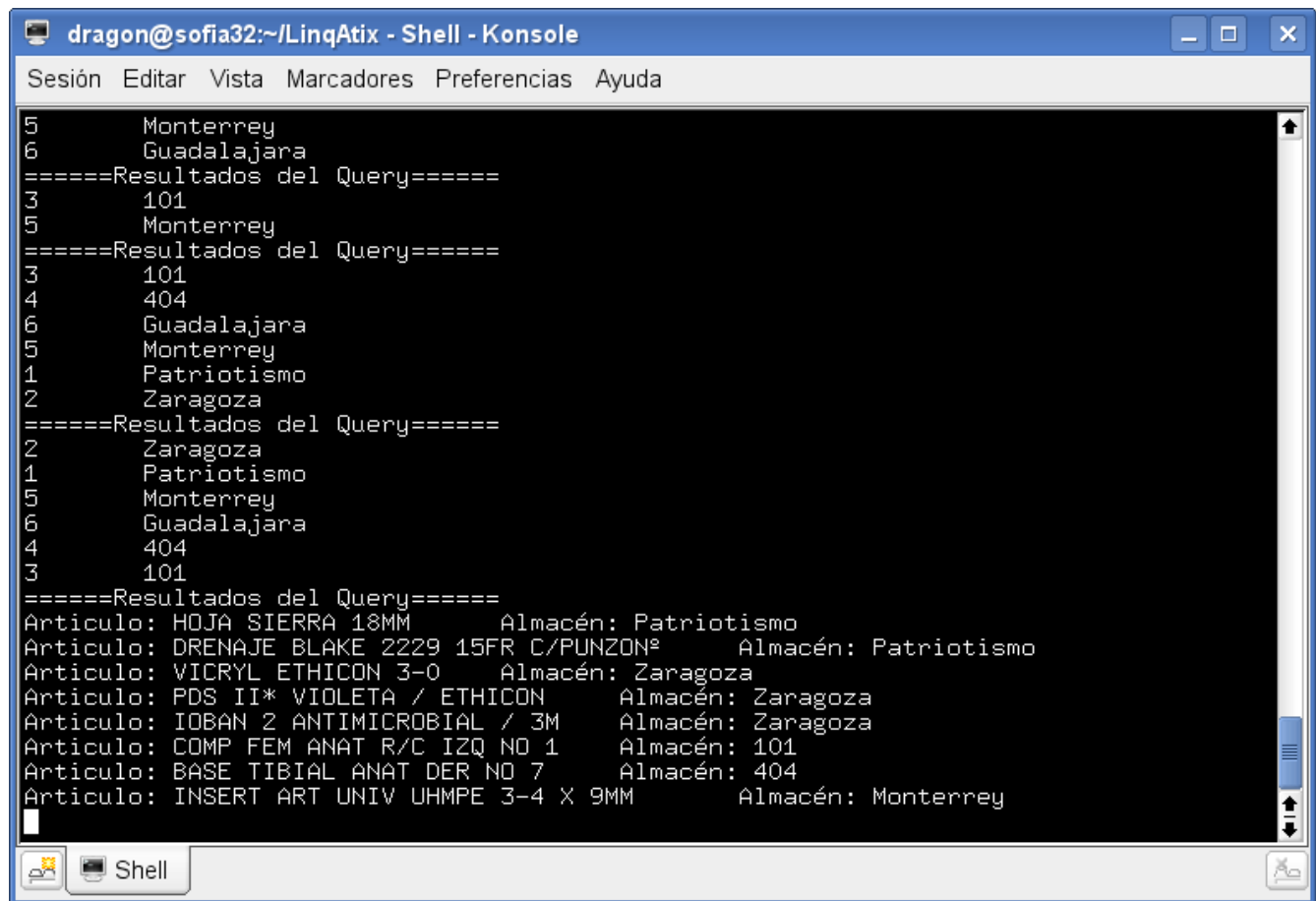
```

Para compilar el ejemplo, debemos de utilizar el comando gmcs, como se muestra en la imagen siguiente:



```
dragon@sofia32:~/LinqAtix - Shell - Konsole
Sesión Editar Vista Marcadores Preferencias Ayuda
dragon@sofia32:~/LinqAtix> gmcs Program.cs
dragon@sofia32:~/LinqAtix> mono Program.exe
```

Para ejecutar el programa utilizamos el comando mono Program.exe, y la salida se muestra en la imagen siguiente:



```
dragon@sofia32:~/LinqAtix - Shell - Konsole
Sesión Editar Vista Marcadores Preferencias Ayuda
5     Monterrey
6     Guadalajara
=====Resultados del Query=====
3     101
5     Monterrey
=====Resultados del Query=====
3     101
4     404
6     Guadalajara
5     Monterrey
1     Patriotismo
2     Zaragoza
=====Resultados del Query=====
2     Zaragoza
1     Patriotismo
5     Monterrey
6     Guadalajara
4     404
3     101
=====Resultados del Query=====
Artículo: HOJA SIERRA 18MM           Almacén: Patriotismo
Artículo: DRENAJE BLAKE 2229 15FR C/PUNZON® Almacén: Patriotismo
Artículo: VICRYL ETHICON 3-0       Almacén: Zaragoza
Artículo: PDS II* VIOLETA / ETHICON Almacén: Zaragoza
Artículo: IOBAN 2 ANTIMICROBIAL / 3M Almacén: Zaragoza
Artículo: COMP FEM ANAT R/C IZQ NO 1 Almacén: 101
Artículo: BASE TIBIAL ANAT DER NO 7 Almacén: 404
Artículo: INSERT ART UNIV UHMPE 3-4 X 9MM Almacén: Monterrey
```

CONCLUSIÓN

LINQ es una más de las muchas tecnologías que ofrece .NET y Mono para facilitar el trabajo a los desarrolladores en cuanto al trabajo con fuentes de datos, LINQ es un enfoque total de llevar la manipulación de datos en los lenguajes orientados a objetos, librando los inconvenientes de trabajar con los lenguajes de manipulación de datos que cada proveedor posee.

Los ejemplos pueden ser descargados de <http://www.humansharp.com/>

Este documento está protegido bajo la licencia de documentación libre Free Documentacion License del Proyecto GNU, para consulta ver el sitio <http://www.gnu.org/licenses/fdl.txt> , toda persona que lo desee está autorizada a usar, copiar y modificar este documento según los puntos establecidos en la «Licencia FDL»