1.- Introducción

A lo largo del tiempo, los programadores han tenido que aprender varios lenguajes de consulta según el origen de datos (SQL, XQuery, …) LINQ simplifica este problema unificando los distintos formatos y orígenes de datos. Todas las consultas hechas en LINQ constan de tres partes:

- Obtener el origen de datos.
- Crear la consulta.
- Ejecutar la consulta.

1.1.- SQL vs LINQ

SQL	LINQ

`select <campos>`
`from <tabla>`
`where <condiciones>`
	`from <alias> in <tabla>`
	`where <condiciones>`
	`select <campos>`

1.2.- Ejemplos

En el ejemplo se muestra cómo se expresan las tres partes de una operación de consulta en código fuente. Se usa una matriz de enteros, distinta por cada ejemplo, como origen de datos:

1.2.1.- Obtener todos los números (consulta básica)

```
//No olvidar añadir la librería correspondiente:
using System.Linq;

//Las 3 partes de la consulta con LINQ
//Origen de datos
int[] _numeros = { 9, 5, 7, 6, 4, 1, 8, 3, 2, 8, 6, 7, 5, 2, 4, 7, 6, 3, 2, 4, 2, 0, 5, 7 };

//Crear la consulta
int [] _ej1 = (from num in _numeros select num).ToArray();

//Ejecutar la consulta
Console.WriteLine("Obteniendo todos los números: ");
foreach(int item in _ej1) {
	Console.Write("{0} ", item);
}
```

1.2.2.- Obtener los números mayores que 4 (where)

```
//Crear la consulta (usamos el mismo origen de datos que antes)
int[] _ej2 = (from num in _numeros where num > 4 select num).ToArray();

//Ejecutar la consulta
Console.WriteLine("Obteniendo los números mayores que 4: ");
foreach (int item in _ej2) {
	Console.Write("{0} ", item);
}
```

1.2.3.- Obtener los 5 primeros números mayores que 4 (.Take(n))

El método .Take(n) (al igual que otros) no es exclusivo de las consultas en LINQ, si no que se puede utilizar con todos aquellos elementos que admitan implícitamente la interfaz genérica IEnumerable <T>.

Todas las consultas en LINQ se recorren en una instrucción foreach, que requiere IEnumerable o IEnumerable<T>. Los tipos compatibles con IEnumerable<T> o una interfaz derivada, como la interfaz genérica IQueryable<T> se denominan **tipos consultables**.

```
//Crear la consulta (usamos el mismo origen de datos que antes)
int[] _ej3 = (from num in _numeros where num > 4 select num).Take(5).ToArray();

//Ejecutar la consulta
Console.WriteLine("Obteniendo los 5 primeros números mayores que 4: ");
foreach (int item in _ej3) {
	Console.Write("{0} ", item);
}
```

1.2.4.- Obtener los números sólo una vez (.Distinct())

```
//Crear la consulta (usamos el mismo origen de datos que antes)
int[] _ej4 = (from num in _numeros select num).Distinct().ToArray();

//Ejecutar la consulta
Console.WriteLine("Obteniendo los distintos números mayores que 4: ");
foreach (int item in _ej4) {
	Console.Write("{0} ", item);
}
```

1.2.5.- Ordenar una consulta (order by <campo> <cláusula>)

La cláusula ascending es el valor de ordenación predeterminado y puede omitirse. Usaremos el ejemplo 1.2.4.

```
//Crear la consulta (usamos el mismo origen de datos que antes)
int[] _ej4 = (from num in _numeros orderby num descending where num > 4 select
num).Distinct().ToArray();

//Ejecutar la consulta
Console.WriteLine("Obteniendo los distintos números mayores que 4 ordenados de mayor a
menor: ");
foreach (int item in _ej4) {
	Console.Write("{0} ", item);
}
```

1.2.6.- Obtener sólo los números pares (usando el alias en la cláusula where)

```
//Crear la consulta (usamos el mismo origen de datos que antes)
int[] _ej5 = (from num in _numeros orderby num descending where (num % 2) == 0 select
num).Distinct().ToArray();

//Ejecutar la consulta
Console.WriteLine("Obteniendo los distintos números pares ordenados de mayor a menor: ");
foreach (int item in _ej5) {
	Console.Write("{0} ", item);
}
```

1.2.7.- Agrupar libros por categorías (group by)

1.2.7.1.- Con un diccionario clave-valor(Dictionary<TKey, TValue>)

```
 Dictionary<string, string> _lista = new Dictionary<string, string>();

_lista.Add("Cien años de soledad", "Novelas");
_lista.Add("Las hijas del Capitán", "Novelas");
_lista.Add("Cálculo diferencial I", "Matemáticas");
_lista.Add("Álgebra I", "Matemáticas");
_lista.Add("El imperio Bizantino", "Historia");
_lista.Add("Historia de Puertollano", "Historia");
_lista.Add("Integración y derivación", "Matemáticas");

var consulta = (from libro in _lista
                orderby libro.Key
                group libro by libro.Value into grupo
                //¡CUIDADO! grupo.Key hace referencia a la clave de agrupación, no a la
                clave del diccionario.
                orderby grupo.Key
                select grupo);

foreach (var e in consulta) {
       Console.WriteLine("Categoría: {0} ----------------------------------------", e.Key);
       foreach(KeyValuePair<string, string> ex in e) {
              Console.WriteLine("Título: {0}", ex.Key);
       }
}
```

1.2.7.2.- Con una lista (List<T>)

```
/// <summary>
/// Clase Libro.cs
/// </summary>
public class Libro {
	public string Categoria { get; set; }
	public string Titulo { get; set; }
}

/// <summary>
/// Método Main
/// </summary>
/// <param name="args"></param>
static void Main(string[] args) {
	List<Libro> _lista = new List<Libro>();
	_lista.Add(new Libro {
		Categoria = "Novelas",
		Titulo = "Cien años de soledad"
	});
	_lista.Add(new Libro {
		Categoria = "Novelas",
		Titulo = "Las hijas del Capitán"
	});
	_lista.Add(new Libro {
		Categoria = "Matemáticas",
		Titulo = "Cálculo diferencial I"
	});
	_lista.Add(new Libro {
		Categoria = "Matemáticas",
		Titulo = "Algebra I"
	});
	_lista.Add(new Libro {
		Categoria = "Matemáticas",
		Titulo = "Integración y derivación"
		    });
	_lista.Add(new Libro {
		Categoria = "Historia",
		Titulo = "El imperio Bizantino"
	});
	_lista.Add(new Libro {
		Categoria = "Historia",
		Titulo = "Historia de Puertollano"
	});

	var consulta = (from c in _lista
	                orderby c.Titulo
	                group c by c.Categoria into grupo
	                orderby grupo.Key
	                select grupo);

	foreach (var e in consulta) {
		Console.WriteLine("Categoría: {0} ----------------------------------------",
e.Key);
		foreach (Libro ex in e) {
			Console.WriteLine("Título: {0}", ex.Titulo);
		}
	}
	Console.ReadLine();
}
```

2.- Operadores de proyección

Permiten modificar el resultado de la consulta sin alterar el contenido del origen de datos.

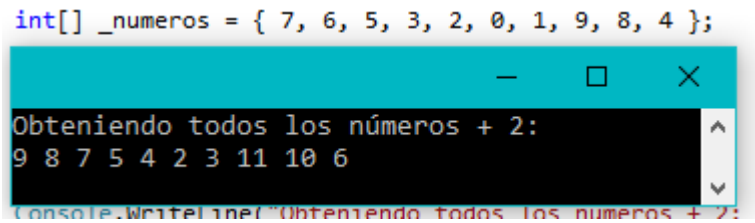
2.1.- Ejemplos

2.1.1.- Sumar 2 a los números de un array

```
//Las 3 partes de la consulta con LINQ
//Origen de datos
int[] _numeros = { 7, 6, 5, 3, 2, 0, 1, 9, 8, 4 };

//Crear la consulta
var _ej6 = from num in _numeros select num + 2;

//Ejecutar la consulta
Console.WriteLine("Obteniendo todos los números + 2: ");
foreach(int item in _ej6) {
       Console.Write("{0} ", item);
}
```



2.1.2.- Obtener un listado de animales de tres formas distintas (Operador new)

Podemos modificar la salida cuantas veces queramos, pero no es necesario hacer la consulta varias veces. El operador `new` ayuda en esta tarea.

```
using System.Globalization; //Necesario para trabajar con la clase CultureInfo

//Las 3 partes de la consulta con LINQ
//Origen de datos
string[] _ej7 = { "VaCa", "tOrO", "PeRrO", "Gato", "RaNa", "ToRTuGA" };

//Crear la consulta
var _resultado = from anim in _ej7
     orderby anim ascending
     select new { mayusculas = anim.ToUpper(),
      minusculas = anim.ToLower(),
      capital = CultureInfo.CurrentCulture.TextInfo.ToTitleCase(anim)
     };

//Ejecutar la consulta
Console.WriteLine("Obteniendo todos los animales: ");
foreach (var item in _resultado) {
       Console.WriteLine("Mayúscula: {0}, minúscula: {1}, letra capital: {2}",
item.mayusculas, item.minusculas, item.capital);
}
```

2.1.3.- Hacer consultas a dos array al mismo tiempo (Operador new)

Este código obtiene todos los números en _numerosA que son menores que todos los números en _numerosB

Para el 0 obtendría: 0 < 1, 0 < 5, 0 < 6, 0 < 7, y 0 < 9.

En la segunda consulta, a parte de cumplirse la primera condición, debe cumplirse también que los números en _numerosA sean mayores que 5.

```
//Las 3 partes de la consulta con LINQ
//Origen de datos
int[] _numerosA = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
int[] _numerosB = { 1, 5, 6, 7, 9 };

//Crear la consulta
var _resultado = from numA in _numerosA
                 from numB in _numerosB
                 where numA < numB
                 select new { numA, numB };

//Ejecutar la consulta
Console.WriteLine("Obteniendo todos los números que cumplan la condición: ");
foreach (var item in _resultado) {
       Console.WriteLine("{0} < {1}", item.numA, item.numB);
}

//Creación de otra consulta (en la cláusula where, se usan los
//operadores lógicos && (and) y || (or) e igual que (==)
var _resultado2 = from numA in _numerosA
                  from numB in _numerosB
                  where numA < numB && numA > 5
                  select new { numA, numB };

//Ejecutamos la segunda consulta
Console.WriteLine("Obteniendo todos los números que cumplan la condición: ");
foreach (var item in _resultado2) {
    Console.WriteLine("{0} < {1}", item.numA, item.numB);
}
```

2.2.- Ejercicios prácticos

2.2.1.- Tabla de multiplicar

Sacar las tablas de multiplicar del 0 al 10 usando LINQ y el siguiente vector de enteros como origen de datos:

```
int[] _numeros = { 9, 5, 1, 6, 2, 8, 4, 7, 3, 10 };
```

Solución 1:

```
int[] _numeros = { 9, 5, 1, 6, 2, 8, 4, 7, 3, 10 };

foreach (int i in _numeros.OrderBy(x => x)) {
	Console.WriteLine("Tabla de multiplicar del {0} ", i);

	int[] _tabla = (from num in _numeros orderby num select (num * i)).ToArray();

	//Con operadores lambda:
	//int[] _tabla = _numeros.OrderBy(x => x).Select(x => x * i).ToArray();

	for (int j = 0; j < _tabla.Length; j++) {
		Console.WriteLine("{0} * {1} = {2}", i, j, _tabla[j]);
	}
	Console.WriteLine("-----------------------------------------------");
}

Console.ReadLine();
```

Solución 2

```
int[] _numerosOrdenados = _numeros.OrderBy(x => x).ToArray();
for (int i = 0; i < _numerosOrdenados.Length; i++) {
	Console.WriteLine("Tabla de multiplicar del {0} ", i + 1);
	for (int j = 0; j < _numerosOrdenados.Length; j++) {
	Console.WriteLine("{0} * {1} = {2}", i + 1 , j + 1, (_numerosOrdenados[j] *
		_numerosOrdenados[i]));
	}

	Console.WriteLine("-----------------------------------------------");

}
Console.ReadLine();
```

2.2.2.- De números a letras

Imprimir por pantalla una sucesión de números en forma de texto y ordenados de menor a mayor.

```
//Las 3 partes de la consulta con LINQ
//Origen de datos
int[] _numeros = { 10, 7, 3, 5, 2, 6, 0, 1, 9, 8, 4 };
string[] _aLetras = { "cero", "uno", "dos", "tres", "cuatro", "cinco", "seis", "siete",
"ocho", "nueve", "diez"};

//Crear la consulta
string []_letras = from num in _numeros orderby num ascending select _aLetras[num];

//Ejecutar la consulta
Console.WriteLine("Obteniendo todos los números en forma de letras: ");
foreach (string item in _letras) {
       Console.Write("{0} ", item);
}

Console.ReadLine();
```

3.- Expresiones lambda

Las consultas en LINQ se pueden escribir como una expresión lambda. Una expresión lambda es una función anónima que se puede usar para crear tipos delegados o de árbol de expresión.

Realizaremos los ejemplos anteriores con expresiones lambda.

3.1.- Consultas sobre el array de números

LINQ normal	LINQ con expresión lambda
`from num in _numeros select num`	`_numeros.Select(x => x)`
`from num in _numeros where num > 4 select num`	`_numeros.Where(x => x > 4)`
`(from num in _numeros where num > 4 select num).Take(5)`	`_numeros.Where(x => x > 4).Take(5)`
`(from num in _numeros where num > 4 select num).Distinct()`	`_numeros.Where(x => x>4).Distinct()`
`(from num in _numeros orderby num descending where num > 4 select num).Distinct()`	`_numeros.OrderByDescending(x => x).Where(x => x > 4).Distinct()`
`(from num in _numeros orderby num descending where (num % 2) == 0 select num).Distinct()`	`_numeros.OrderByDescending(x => x).Where(x => (x % 2) == 0).Distinct()`

3.2.- Consultas group by

LINQ normal	LINQ con expresión lambda

`from libro in _lista`
`orderby libro.Value`
`group libro by libro.Value into grupo`
`orderby grupo.Key`
`select grupo`
	`_lista.OrderByDescending(x=> x.Value).GroupBy(x=>x.Value).OrderBy(x => x.Key)`

`from c in _lista`
`orderby c.Titulo`
`group c by c.Categoria into grupo`
`orderby grupo.Key`
`select grupo`
	`_lista.OrderBy(x => x.Titulo).GroupBy(x => x.Categoria).OrderBy(x => x.Key)`

3.3.- Operadores de proyección

LINQ normal	LINQ con expresión lambda
`from num in _numeros select num + 2`	`_numeros.Select(x => x + 2)`

`from anim in _ej7`
`orderby anim ascending`
`select new { mayusculas = anim.ToUpper(),`
`minusculas = anim.ToLower(),`
`capital = CultureInfo.CurrentCulture.TextInfo.ToTitleCase(anim)`
`}`
	`_ej7.OrderBy(x => x).Select(x => new { mayusculas = x.ToUpper(), minusculas = x.ToLower(), capital = CultureInfo.CurrentCulture.TextInfo.ToTitleCase(x)})`

`from numA in _numerosA`
`from numB in _numerosB`
`where numA < numB`
`select new { numA, numB }`
	`_numerosA.SelectMany(numB => _numerosB, (numA, numB) => new { numA, numB }).Where(x => x.numA < x.numB)`

`from numA in _numerosA`
`from numB in _numerosB`
`where numA < numB && numA > 5`
`select new { numA, numB }`
	`_numerosA.SelectMany(numB => _numerosB, (numA, numB) => new { numA, numB }).Where(x => x.numA < x.numB && x.numA > 5)`

4.- LINQ to XML

https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/concepts/linq/basic-queries-linq-to-xml

Para este apartado, se utilizarán los siguientes documentos XML:

- OrdenesDeCompra.xml
- OrdenesDeCompraConEspacioDeNombres.xml
- Agenda.xml

4.1.- Buscar un elemento con un atributo específico

Este primer ejemplo muestra como buscar el elemento **Direccion**, que tiene un atributo **tipo**, con un valor **"Factura"**

```
// Las tres partes de una consulta en LINQ:
//  1. El origen de datos.
XElement xml = XElement.Load(@"D:\OrdenesDeCompra.xml");

// 2. La creación de la consulta.
IEnumerable<XElement> direccion = from el in xml.Elements("Direccion")
where (string)el.Attribute("tipo") == "Factura"
select el;

// 3. Ejecución de la consulta.
foreach(XElement el in direccion) {
    Console.WriteLine("Nombre: {0}\nCalle: {1},\nCiudad: {2},\nEstado: {3},\nCódigo postal: {4},\nPaís: {5}",
    el.Element("Nombre").Value,
    el.Element("Calle").Value,
    el.Element("Ciudad").Value,
    el.Element("Estado").Value,
    el.Element("CodPost").Value,
    el.Element("Pais").Value);
}
```

Salida:

```
Nombre: Tai Yee
Calle: 8 Avenida Oak
Ciudad: Old Town
Estado: PA
Código postal: 95819
País: USA
```

El siguiente ejemplo muestra la misma consulta sobre un XML que se encuentra en un espacio de nombres.

```
// Las tres partes de una consulta en LINQ:
//  1. El origen de datos.
XElement xml = XElement.Load(@"D:\OrdenesDeCompraConEspacioDeNombres.xml");
XNamespace aw = "http://www.adventure-works.com";

// 2. La creación de la consulta.
IEnumerable<XElement> direccion = from el in xml.Elements(aw + "Direccion")
where (string)el.Attribute(aw + "tipo") == "Factura"
select el;

// 3. Ejecución de la consulta.
foreach (XElement el in direccion) {
    Console.WriteLine(el);
    Console.WriteLine("----------------------------------------");
}

foreach (XElement el in direccion) {
    Console.WriteLine("Nombre: {0}\nCalle: {1}\nCiudad: {2}\nEstado: {3}\nCódigo postal:
{4}\nPaís: {5}",
    el.Element(aw + "Nombre").Value,
    el.Element(aw + "Calle").Value,
    el.Element(aw + "Ciudad").Value,
    el.Element(aw + "Estado").Value,
    el.Element(aw + "CodPost").Value,
    el.Element(aw + "Pais").Value);
    Console.WriteLine("----------------------------------------");
}
```

Salida:

```
<aw:Direccion aw:tipo="Factura" xmlns:aw="http://www.adventure-works.com">
  <aw:Nombre>Tai Yee</aw:Nombre>
  <aw:Calle>8 Avenida Oak</aw:Calle>
  <aw:Ciudad>Old Town</aw:Ciudad>
  <aw:Estado>PA</aw:Estado>
  <aw:CodPost>95819</aw:CodPost>
  <aw:Pais>USA</aw:Pais>
</aw:Direccion>
----------------------------------------
Nombre: Tai Yee
Calle: 8 Avenida Oak
Ciudad: Old Town
Estado: PA
Código postal: 95819
País: USA
----------------------------------------
```

4.2.- Buscar un elemento con un elemento secundario específico

```
// Las tres partes de una consulta en LINQ:
//  1. El origen de datos.
XElement xml = XElement.Load(@"D:\Agenda.xml");

// 2. La creación de la consulta.
// Buscamos el/los contacto/s con el apellido López
IEnumerable<XElement> contacto = from con in xml.Elements("contacto")
         where (string)con.Element("apellido") == "López"
         select con;

// 3. Ejecución de la consulta.
foreach (XElement el in contacto) {
    Console.WriteLine(el);
    Console.WriteLine("----------------------------------------");
}

foreach (XElement el in contacto) {
    Console.WriteLine("Nombre: {0}\nApellido: {1}\nDirección: {2}\nTeléfono de casa:
{3}\nTeléfono de móvil: {4}\nTeléfono de trabajo: {5}",
    el.Element("nombre").Value,
    el.Element("apellido").Value,
    el.Element("direccion").Value,
    el.Element("telefonos").Element("telcasa").Value,
    el.Element("telefonos").Element("telmovil").Value,
    el.Element("telefonos").Element("teltrabajo").Value);
    Console.WriteLine("----------------------------------------");
}
```

```
<contacto>
  <nombre>Pedro</nombre>
  <apellido>López</apellido>
  <direccion>Calle del Carmen, 13</direccion>
  <telefonos>
    <telcasa>44332211</telcasa>
    <telmovil>55443322</telmovil>
    <teltrabajo>66554433</teltrabajo>
  </telefonos>
</contacto>
----------------------------------------
<contacto>
  <nombre>Álvaro</nombre>
  <apellido>López</apellido>
  <direccion>Calle del Silencio</direccion>
  <telefonos>
    <telcasa>59437981</telcasa>
    <telmovil>44678618</telmovil>
    <teltrabajo>35487971</teltrabajo>
  </telefonos>
</contacto>
----------------------------------------
Nombre: Pedro
Apellido: López
Dirección: Calle del Carmen, 13
Teléfono de casa: 44332211
Teléfono de móvil: 55443322
Teléfono de trabajo: 66554433
----------------------------------------
Nombre: Álvaro
Apellido: López
Dirección: Calle del Silencio
Teléfono de casa: 59437981
Teléfono de móvil: 44678618
Teléfono de trabajo: 35487971
----------------------------------------
```