Unidad 2. LINQ

Language Integrated Query

1.1. Fundamentos de LINQ

¿Qué es LINQ?

- Language Integrated Query (Lenguaje integrado de consulta)
- Hacer Consulta una parte del lenguaje
- Framework 3.5

Consultas sin LINQ

- Objetos usando bucles y condiciones

```
foreach(Customer c in customers)
   if (c.Region == "UK") …
```

- Basede Datos usando SQL

```
SELECT * FROM Customers WHERE Region='UK'
```

- XML usando XPath/XQuery

```
//Customers/Customer[@Region='UK']
```

Consultas con LINQ

- C#

```
var myCustomers = from c in customers
    where c.Region == "UK"
    select c;
```

Otra consulta con LINQ

- C#

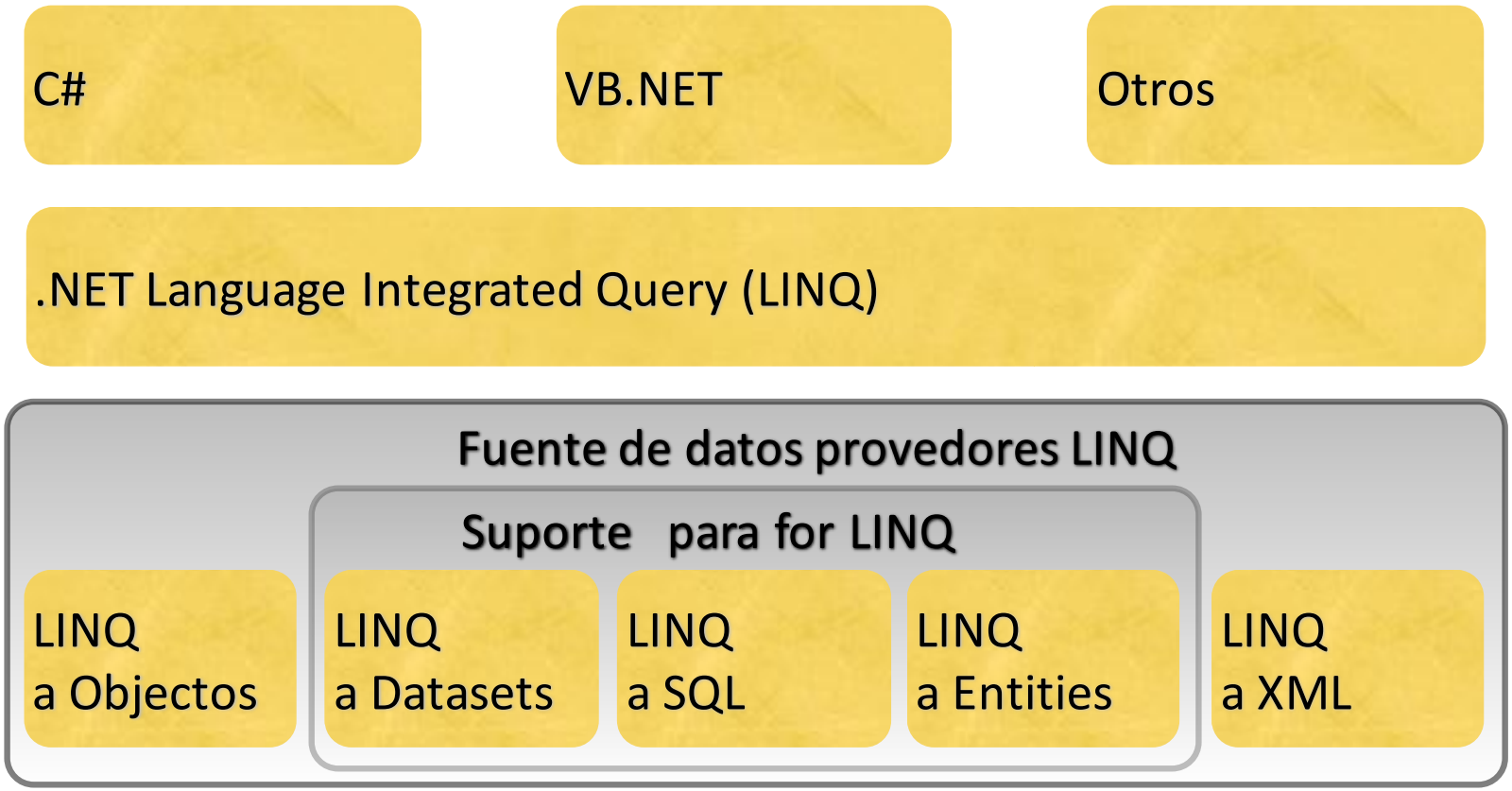
```
var goodCusts = (from c in db.Customers
    where c.PostCode.StartsWith("GY")
    orderby c.Sales descending
    select c).Skip(10).Take(10);
```


Objects
MongoDB
CSV Files
File System
LINQ is Everywhere
SQL Database
HL7 XML
JSON

Ventajas

- Acceso unificado a datos
  Sintaxis simple para aprender y recordar
- Fuertemente tipado
  Capturar errores durante la compilacion
- IntelliSense
  Solicitud de sintaxis y atributos
- Conjuntos de resultados enlazables

Arquitectura



LINQ a Objectos

C#

```
int[] nums = new int[] {0,4,2,6,3,8,3,1};
double average = nums.Take(6).Average();
var above = from n in nums
            where n > average
            select n;
```

Dos Sintaxis de Expresar Linq

- **Syntaxis consulta**

```
var vendedorconsulta=from v in vendedores
                     where v.empresa.Contains("El")
                     orderby v.empresa
                     select b;
```

- **Syntaxis Método**

```
var vendedorconsulta= vendedores
                     .where(v=>v.empresa.Contains("El"))
                     .OrderBy()
```

Operaciones de consulta

Consulta (Query)

- En una consulta, se especifica la información que se desea recuperar del origen de datos.
- Puede especificar cómo se debe ordenar, agrupar y conformar esa información antes de que se devuelva.
- LINQ proporciona un conjunto de métodos de consulta estándar que se puede utilizar en una consulta
- Una consulta se ejecuta en una instrucción foreach y foreach requiere IEnumerable o IEnumerable<T>

Método Filtrado

- El filtro hace que la consulta devuelva solo los elementos para los que la expresión es verdadera.
- El resultado se genera mediante la cláusula **where**.
- El filtro aplicado especifica qué elementos se deben excluir de la secuencia de origen.

Métodos de ordenación

- OrderBy,
  - Ordera Acendentemente
- OrderByDescending,
  - Ordena Descendentemente
- ThenBy,
  - Cuando se duplica el primer criterio ordenar por
- ThenByDescending
  - Cuando se duplica el primer criterio ordenar por
- Reverse.

Persona.sln

```
//3.- listadealumnos ordenada por apellidos
var consulta3 = from alm in Alumnos
                orderby alm.Apellidos
                select new { alm.Nombre, alm.Apellidos};

//4.- listadealumnos ordenada por nombre luego por apellido descendiente
var consulta4 = from alm in Alumnos
                orderby  alm.Nombre, alm.Apellidos descending

                select alm;
```

Métodos de agrupamiento

- Es la operación de colocar los datos en grupos de manera que los elementos de cada grupo compartan un atributo común
- GroupBy

Persona.sln

```
//5.-Edades de los alumnos
var consulta5 = from almu in Alumnos
                group almu by almu.Edad into edad
                select edad;
// Grupo Edades
foreach (var GrupoEdades in consulta5)
{
    Console.WriteLine(GrupoEdades.Key);
    foreach (Persona alumno in GrupoEdades)
    {
        Console.WriteLine("{0} {1}", alumno.Nombre, alumno.Apellidos);

    }
}
Console.WriteLine("********************************");
```

Métodos de agregación

- Una operación de agregación calcula un valor único a partir de una colección de valores.
- Por ejemplo, el cálculo de la temperatura media diaria a partir de los valores de la temperatura diaria de un mes es una operación de agregación.

Persona.sln

```
    var consulta5 = from a in Alumnos
                    group a by a.Sexo into Sexo
                    select new {Sexo=Sexo.Key,Detalle=Sexo,Promedio=Sexo.Average(Persona=>Persona.Edad) };
    foreach (var grupoSexo in consulta5)
    {
        Console.WriteLine("Sexo {0}",grupoSexo.Sexo);
        Console.WriteLine("*****************************************");
        Console.WriteLine("Sexo:{0} promedio{1} ", grupoSexo.Sexo, grupoSexo.Promedio);
        foreach (Persona p in grupoSexo.Detalle)
        {
            Console.WriteLine("Nombre:{0} Edad:{1}",p.Nombre,p.Edad);
        }
    }
    Console.ReadKey();
}
```

- Aggregate,
- Average,
- Count,
- LongCount,
- Max,
- Min
- Sum.

Skip y Take

- Para obtener de la lista los primeros valores se utiliza take.
- Para obtener de la lista desde un determinado elemento se utiliza skip.

Persona.sln

```
Console.WriteLine("9.- Los tres almumnos con menor Peso  ");

var consulta9 = (from almu in Alumnos
                 orderby almu.Peso
                 select  almu).Take(3);
foreach (var al in consulta9)
{
    Console.WriteLine("{0} {1} {2}", al.Nombre,al.Apellidos,al.Peso);

}
Console.WriteLine("***");
```

Operadores LINQ

Agregacion	Conversion	Ordenar	Particion	Conjuntos
Aggregate	Cast	OrderBy	Skip	Concat
Average	OfType	ThenBy	SkipWhile	Distinct
Count	ToArray	Descending	Take	Except
Max	ToDictionary	Reverse	TakeWhile	Intersect
Min	ToList			Union
Sum	ToLookup			
	ToSequence			

Transformaciones de Datos

Transformaciones

- Combinar varias secuencias de entrada en una sola secuencia de salida que tiene un tipo nuevo. Se utiliza el método **Concat**

```
// Consulta
var personasEnSeattle = (from e in Estudiantes
                         where e.Ciudad == "Seattle"
                         select e.Apellidos)
            .Concat(from d in Docentes
                    where d.Ciudad == "Seattle"
                    select d.Apellidos);

Console.WriteLine("Los siguientes estudiantes y docentes viven en Seattle:");
// ejecuta la consulta
foreach (var persona in personasEnSeattle)
{
    Console.WriteLine(persona);
}

Console.WriteLine("Presione Cualquier tecla para salir.");
Console.ReadKey();
```

Seleccionar un subconjunto de cada elemento de origen

- Crear secuencias de salida cuyos elementos estén formados por una o varias propiedades de cada elemento de la secuencia de origen.

```
// Crea un subconjuto con un solo valor

var consulta = from e in Estudiantes
               select e.Ciudad;

// Crea un subconjunto con dos valores

var consulta1 = from e in Estudiantes
               select new { Nombre = e.Nombre , Ciudad = e.Ciudad};
```

Seleccionar un subconjunto de cada elemento de origen

- Crear secuencias de salida cuyos elementos estén formados por los resultados de las operaciones realizadas en el origen de datos.

```
// Crea un subconjunto con dos valorescalculados del conjunto

var consulta2= from e in Estudiantes
               select new { NombreCompleto = e.Nombre+" "+e.Apellidos};

Console.WriteLine("Presione Cualquier tecla para salir.");
Console.ReadKey();
```

Transformar objetos en memoria en XML

- Crear secuencias de salida en un formato diferente. Por ejemplo, se pueden transformar datos de filas de SQL o archivos de texto en XML.

```
    // transforma en XML
    var EstudiantesToXML = new XElement("Root",
    from e in Estudiantes
    let scores = string.Join(",", e.Calificaciones)
    select new XElement("estudiante",
              new XElement("Nombre", e.Nombre),
              new XElement("Apellidos", e.Apellidos),
              new XElement("Calificaciones", scores)
          ) // end "estudiante"
        ); // end "Root"

    // Execute the query.
    Console.WriteLine(EstudiantesToXML);


    Console.WriteLine("Presione Cualquier tecla para salir.");
    Console.ReadKey();
}
```

Referencias

- https://linqsamples.com
- https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/concepts/linq/basic-linq-query-operations
- https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/concepts/linq/data-transformations-with-linq