UNIVERSIDAD PRIVADA DE TACNA

LINQ

Curso: Programación II

Ing. Enrique Lanchipa Valencia

Resumen

- Arrays
- Vectores
- Matrices
- Sintaxis
- GITHUB
- .
- .

Introducción

LINQ es un conjunto de extensiones de lenguaje añadidas a C # y VB.NET. Proporciona un modelo de programación unificado a los dominios de datos diferentes para la gestión de datos

LINQ es un componente del Framework Microsoft .NET que permite realizar consultas en forma nativa haciendo uso de los lenguajes de programación del .NET, esto utilizando una sintaxis similar a SQL.

Una de las características mas importantes de LINQ es que simplifica y unifica la implementación de acceso a cualquier tipo de dato. Además de que no impone el uso de una arquitectura, sino es que ayuda a varias arquitecturas ya existentes en el acceso a datos.

A partir de la versión 2008 de Visual Studio y el NET Framework, LINQ se incluyó como un estándar en la instalación de Visual Studio.


```
List<string> names =
    new List<string>{"John", "Rick", "Maggie", "Mary"};

IEnumerable<string> nameQuery = from name in names
                                where name[0] == 'M'
                                select name;

foreach (string str in nameQuery)
{
    Console.WriteLine(str);
}
```

1
2
3

¿Qué es LINQ?

LINQ (Language Integrated Query)

Es un lenguaje de consulta que nació en :NET Framework 3,5. Las consultas que se realizan con LINQ son similares al lenguaje SQL, pero su ventaja se encuentra que es posible extraer información de arrays, clases enumerables, archivos XML y bases de datos relacionales.

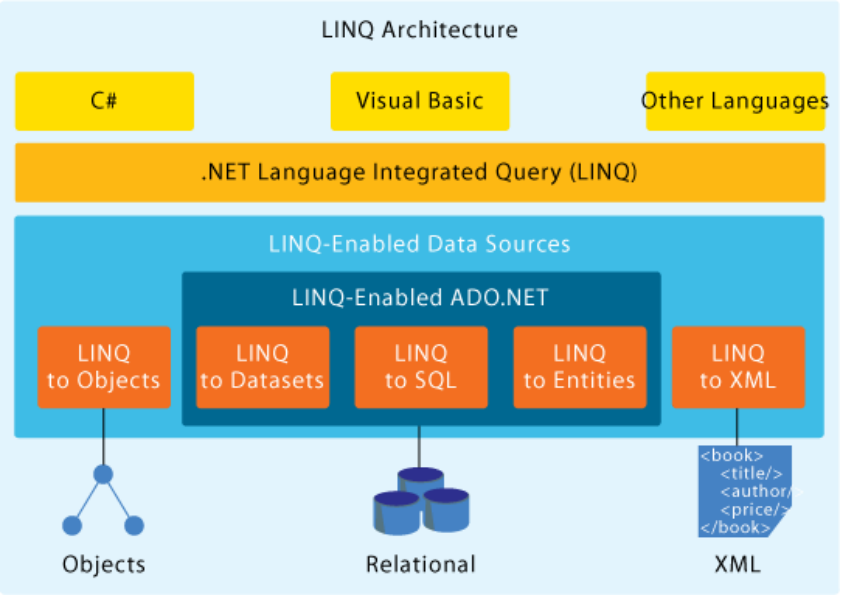
LINQ es un conjunto de operadores estándar que ofrece poderosas facilidades de consulta a lenguajes como C#.

El framework de LINQ facilita la capacidad de traer datos con el poder de manipularlos.

Los operadores proveen la capacidad de expresar las operaciones de consulta directa y declarativamente con cualquier lenguaje basado en .NET.

LINQ (Language Integrated Query) es una característica agregada a los lenguajes .NET que extiende las capacidades de consulta de estos lenguajes.

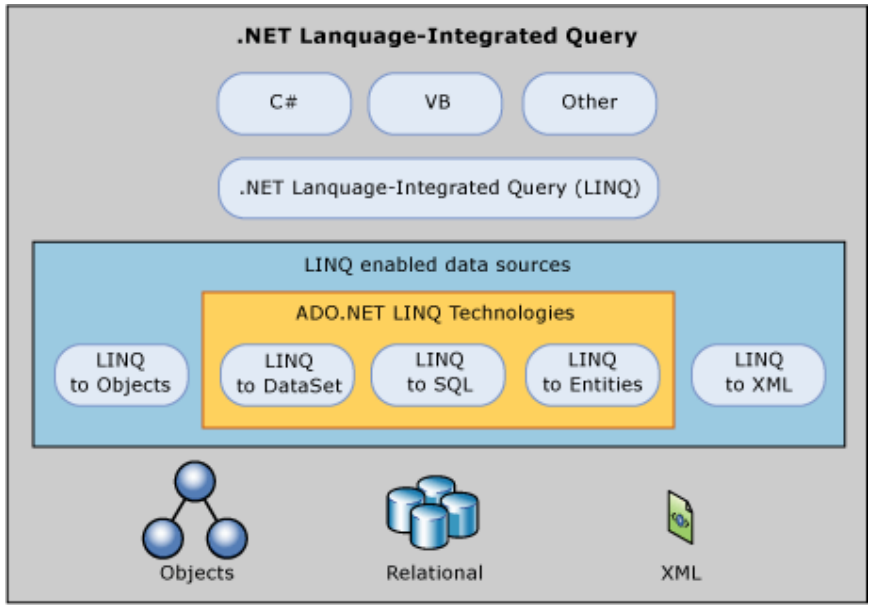
Visual Studio viene con assemblies de LINQ que habilitan el uso de LINQ con diferentes fuentes de datos, tales como colecciones en memoria, bases de datos relacionales SQL, Datasets ADO.NET, documentos XML, etc. En Visual Studio, Visual C# y Visual Basic son los lenguajes que implementan las extensiones de LINQ. Las extensiones de LINQ usan los nuevos operadores estandard de consulta, para cualquier colección que implemente IEnumerable<T> en C# o IEnumerable(Of T) (en VB). Esto significa que todas las colecciones y los arrays pueden ser consultados usando LINQ. Las colecciones, simplemente necesitan implementar IEnumerable<T>, para habilitarlas para que LINQ consulte las colecciones.



Fuentes de Datos

Existen cinco (5) fuentes de datos a los que se puede acceder con LINQ. De estas tres (3) de ellas se realizan con ADO.NET.

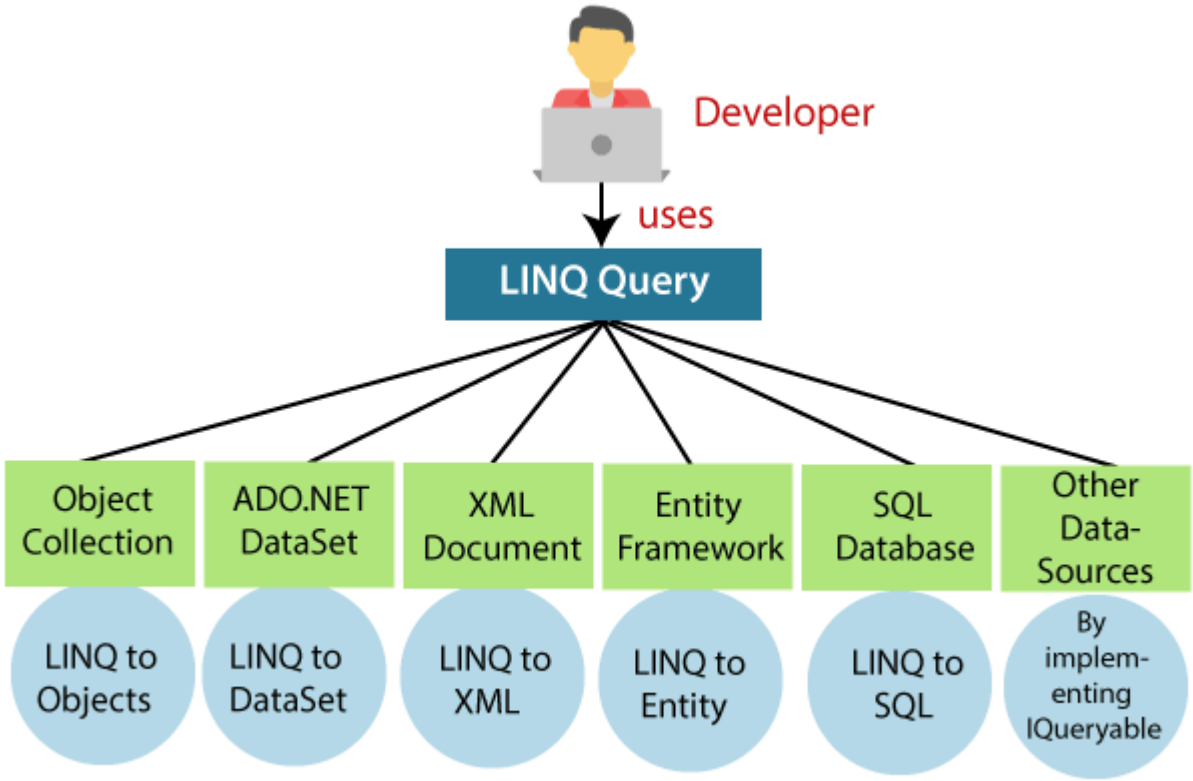
1. **LINQ to Objects:** Consultas LINQ para accesar estructuras en memoria. Podemos consultar cualquier tipo que suporte IEnumerable(Of T) (Visual Basic) or IEnumerable<T> (C#).
2. **LINQ to Dataset:** Hace más fácil consultar los datos guardados en Datasets. Un Dataset está formado por datos desconectados y consolidados desde diferentes fuentes de datos.
3. **LINQ to SQL:** Consultas LINQ para accesar estructuras en memoria. Podemos consultar cualquier tipo que suporte IEnumerable(Of T) (Visual Basic) or IEnumerable<T> (C#).
4. **LINQ to Entities:** El modelo de entidad de datos (Entity Data Model) es un modelo de datos conceptual que puede ser usado para modelar los datos, para que las aplicaciones puedan interactuar con los datos como entities (entidades) u objetos. A través del modelo de entidad de datos, ADO.NET expone las entidades como objetos.
5. **LINQ to XML:** Provee capacidades de modificación de documentos en memoria del Document Object Model (DOM) y suporta consultas LINQ. Usando LINQ to XML, podemos consultar, modificar, navegar, y grabar los cambios de un documento XML. Nos capacita para escribir consultas para navegary recuperar una colección de elementos y atributos. Es similar a XPath y XQuery.



Fuente:

https://msdn.microsoft.com/es-es/library/bb399365(v=vs.110).aspx

LINQ


Developer
uses
LINQ Query
Object Collection
ADO.NET DataSet
XML Document
Entity Framework
SQL Database
Other Data-Sources
LINQ to Objects
LINQ to DataSet
LINQ to XML
LINQ to Entity
LINQ to SQL
By implem-enting IQueryable

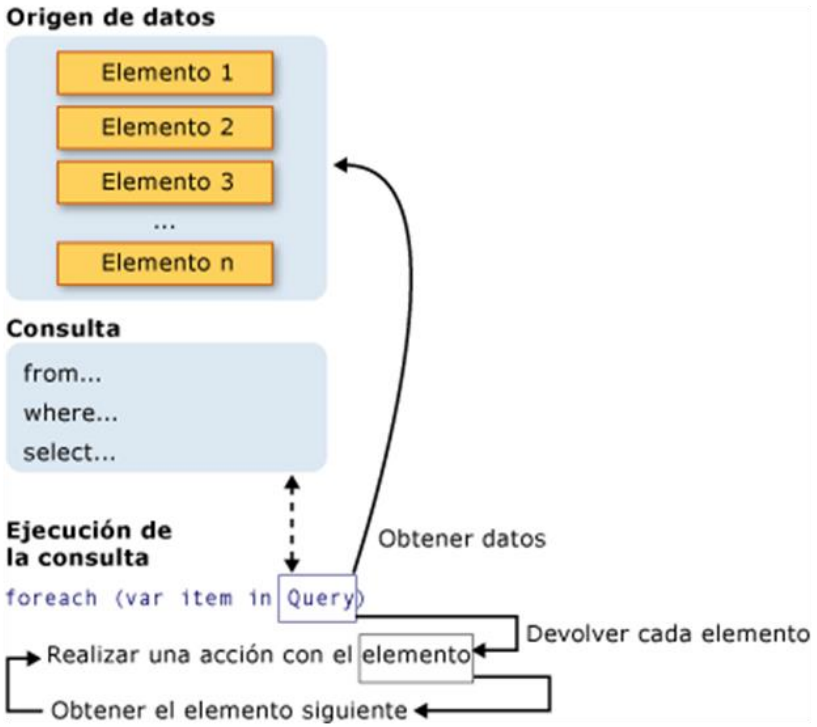
LINQ

LINQ es una innovación para disminuir la brecha entre el mundo de los objetos y el mundo de los datos.

```
namespace Ejemplo_LINQ
{
    class LINQQuerySyntax
    {
        static void Main(string[] args)
        {
            // string collection
            IList<string> stringList = new List<string>() {
            "C# ",
            "SQL",
            "LINQ",
            "MV" ,
            };
            // LINQ Query Syntax
            var result = from s in stringList
                         where s.Contains("LINQ")
                         select s;
        }
    }
}
```

Colecciones de negocio

Operación de consulta LINQ



Las 3 partes de la operación de consulta

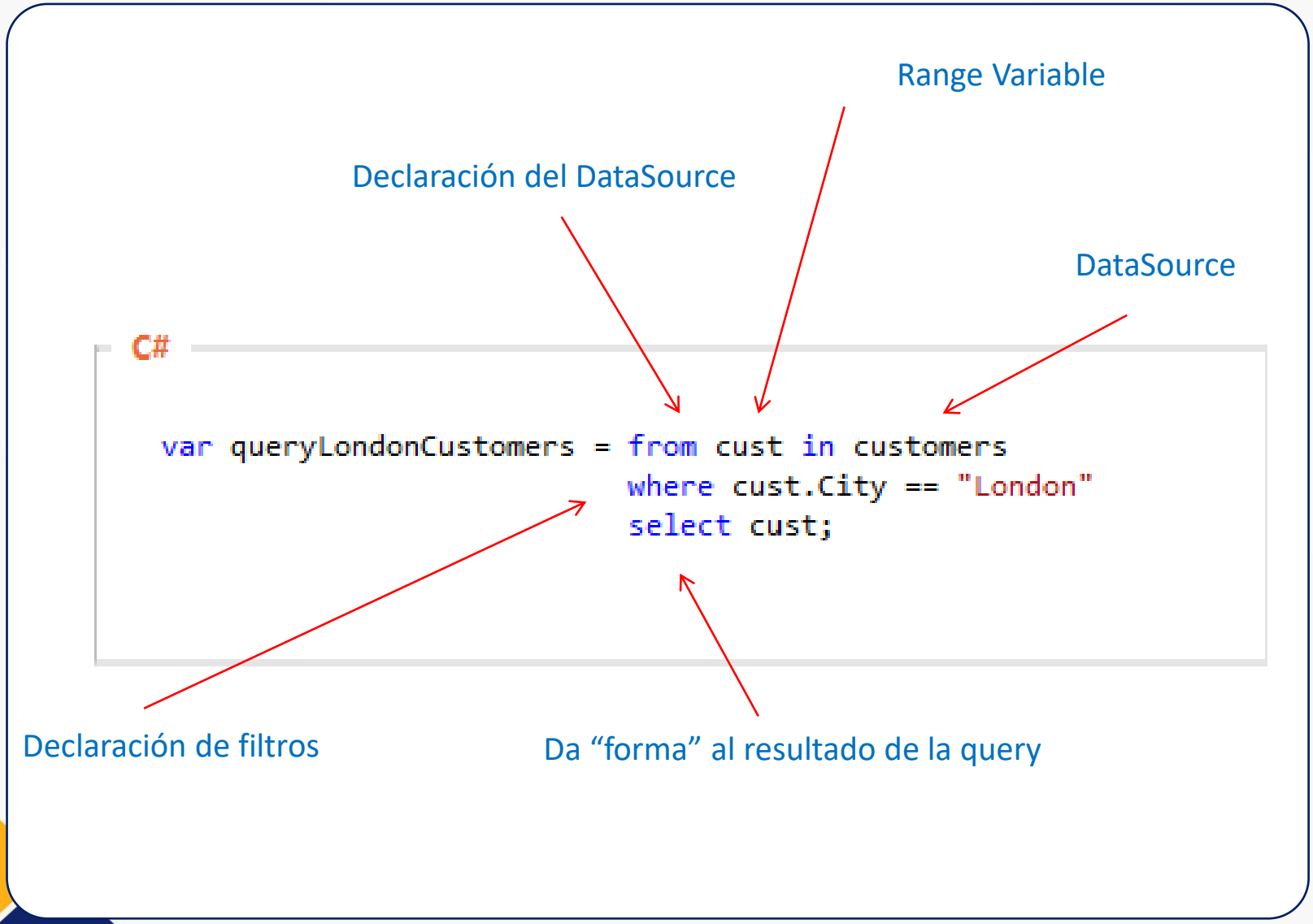
Todas las operaciones de consulta LINQ se componen de tres acciones distintas:

1. **Obtención del origen de datos.**
2. **Creación de la consulta.**
3. **Ejecución de la consulta**

Una query tiene 3 clausulas:

- **from:** Especifica el data source
- **where:** aplica el filtro
- **select:** especifica el tipo de los elementos retornados

Partes básicas de una consulta



Operadores LINQ – de Proyección

Proyección se refiere al acto de transformar los elementos de una secuencia en una forma definida para el desarrollador.
Los operadores de proyección son: **Select** y **SelectMany**

Select

Este operador muestra una proyección de la colección, es decir Select proyecta los valores de una sola secuencia o colección.

Ejemplo: Select

```
static void Proyeccion01()
{
    using (tiendaEntities db = new tiendaEntities())
    {
        var sql = from p in db.PRODUCTOS
                  where p.NOMBRE.StartsWith("M")
                  select new { p.NOMBRE,p.PRECIO };

        foreach (var item in sql)
        {
            Console.WriteLine("Producto: {0} - {1}", item.NOMBRE, item.PRECIO);
        }
    }
}
```

Utilizando la sintaxis del método

```
static void Proyeccion_SelectV2()
{
    using (tiendaEntities db = new tiendaEntities())
    {
        var sql = db.PRODUCTOS
            .Select(c => new { c.NOMBRE, c.PRECIO } )
            .Where(c => c.NOMBRE.StartsWith("M"));


        foreach (var item in sql)
        {
            Console.WriteLine("Producto: {0} - {1}", item.NOMBRE, item.PRECIO);
        }
    }
}
```

Operadores LINQ – de proyección

SelectMany

Provee la multiple capacidad de combinación de clausulas from con los resultados de cada objeto, en una sola secuencia.

Ejemplo: SelectMany

```
static void Proyeccion_SelectMany()
{
    string[] dias =
        {
            "lunes",
            "martes",
            "miercoles"
        };

    // Convierte cada string en un array de string
    // y combina en un solo array.
    var result = dias.SelectMany(element => element.ToCharArray());

    // Muestra letras
    foreach (char letter in result)
    {
        Console.WriteLine(letter);
    }
}
```

Utilizando la sintaxis del método con base de datos. Retorna las columnas nombre y precio

```
static void Proyeccion_SelectManyV2()
{
    using (tiendaEntities db = new tiendaEntities())
    {

        var cat = from c in db.CATEGORIAS
                  select c;
        foreach (var p in cat.SelectMany(x => x.PRODUCTOS))
        {
            Console.WriteLine(p.NOMBRE);
        }
    }

}
```

Operadores LINQ – de Restricción

Where

Este operador permite agregar una condición a la consulta, con el propósito de poder filtrar por criterio.

Ejemplo: Where

Muestra los productos que cumplan con la condición que el precio sea mayor a 1000.

```
static void Proyeccion_Where()
{
    using (tiendaEntities db = new tiendaEntities())
    {
        var sql = from p in db.PRODUCTOS
                  where p.PRECIO > 1000
                  select new { p.NOMBRE, p.DESCRIPCION };

        foreach (var item in sql)
        {
            Console.WriteLine("Producto: {0} - {1}", item.NOMBRE, item.DESCRIPCION);
        }
    }
}
```

Operadores LINQ – de Ordenamiento

Los operadores de ordenamiento proveen la capacidad de ordenar los resultados de manera ascendente o descendente.
Entre los cuales se encuentran: **OrderBy, OrderByDescending, ThenBy, ThenByDescending, y Reverse.**

OrderBy

Este operador ordena los resultados de manera ascendente.

Ejemplo: OrderBy

Muestra la lista de productos ordenados ascendentemente.

```
static void Proyeccion_OrderBy()
{
    using (tiendaEntities db = new tiendaEntities())
    {
        var sql = from p in db.PRODUCTOS
                  orderby p.NOMBRE
                  select new { p.NOMBRE, p.DESCRIPCION };

        foreach (var item in sql)
        {
            Console.WriteLine("Producto: {0} - {1}", item.NOMBRE, item.DESCRIPCION);
        }
    }
}
```

Utilizando la sintaxis del método

```
static void Proyeccion_OrderBy_v2()
{
    using (tiendaEntities db = new tiendaEntities())
    {
        var sql = db.PRODUCTOS
            .Select(c => new { c.NOMBRE, c.DESCRIPCION }).OrderBy(c=>c.NOMBRE);

        foreach (var item in sql)
        {
            Console.WriteLine("Producto: {0} - {1}", item.NOMBRE, item.DESCRIPCION);
        }
    }
}
```

Operadores LINQ – de Ordenamiento

OrderByDescending

Este operador ordena los resultados de manera descendente.

Ejemplo: OrderByDescending

Muestra la lista de productos ordenados descendentemente.

```
static void Proyeccion_OrderByDescending()
{
    using (tiendaEntities db = new tiendaEntities())
    {
        var sql = from p in db.PRODUCTOS
                  orderby p.NOMBRE descending
                  select new { p.NOMBRE, p.DESCRIPCION };

        foreach (var item in sql)
        {
            Console.WriteLine("Producto: {0} - {1}", item.NOMBRE, item.DESCRIPCION);
        }
    }
}
```

Utilizando la sintaxis del método

```
static void Proyeccion_OrderByDescending_v2()
{
    using (tiendaEntities db = new tiendaEntities())
    {
        var sql = db.PRODUCTOS
            .Select(c => new { c.NOMBRE, c.DESCRIPCION })
            .OrderByDescending(c => c.NOMBRE);

        foreach (var item in sql)
        {
            Console.WriteLine("Producto: {0} - {1}", item.NOMBRE, item.DESCRIPCION);
        }
    }
}
```

Operadores LINQ – de Ordenamiento

ThenBy

Este operador se aplica a la segunda columna que se desea ordenar.

Ejemplo: ThenBy

Se ordena primero la columna idcategoria y luego a la segunda columna nombre.

```
static void Proyeccion_ThenBy()
{
    using (tiendaEntities db = new tiendaEntities())
    {
        var sql = from p in db.PRODUCTOS
                  orderby p.IDCATEGORIA descending,p.NOMBRE
                  select new { p.IDCATEGORIA, p.NOMBRE};

        foreach (var item in sql)
        {
            Console.WriteLine("Producto: {0} - {1}", item.IDCATEGORIA, item.NOMBRE);
        }
    }
}
```

Utilizando la sintaxis del método

```
static void Proyeccion_ThenBy_v2()
{
    using (tiendaEntities db = new tiendaEntities())
    {
        var sql = db.PRODUCTOS
            .Select(c => new { c.IDCATEGORIA, c.NOMBRE})
            .OrderByDescending(c => c.IDCATEGORIA)
            .ThenBy (c=>c.NOMBRE);

        foreach (var item in sql)
        {
            Console.WriteLine("Producto: {0} - {1}", item.IDCATEGORIA, item.NOMBRE);
        }
    }
}
```

Operadores LINQ – de Ordenamiento

ThenByDescending

Este operador ordena los valores resultantes en forma descendente.

Ejemplo: ThenByDescending

Se ordena primero la columna idcategoria y luego a la segunda columna nombre de manera descendente.

```
static void Proyeccion_ThenByDescending()
{
    using (tiendaEntities db = new tiendaEntities())
    {
        var sql = db.PRODUCTOS
            .Select(c => new { c.IDCATEGORIA, c.NOMBRE })
            .OrderByDescending(c => c.IDCATEGORIA)
            .ThenByDescending(c => c.NOMBRE);

        foreach (var item in sql)
        {
            Console.WriteLine("Producto: {0} - {1}", item.IDCATEGORIA, item.NOMBRE);
        }
    }
}
```

Operadores LINQ – de Ordenamiento

Reverse

Este operador ordena los valores resultantes en forma descendente.

Ejemplo: Reverse

```
static void Reverse()
{
    string[] meses = { "Enero","Febrero","Marzo","Abril","Mayo","Junio",
        "Julio", "Agosto","Septiembre","Octubre","Noviembre","Diciembre"};

    string[] mesesReverse = meses.Reverse().ToArray();
    foreach (var item in mesesReverse)
    {
        Console.WriteLine("{0}", item);
    }

}
```

Operadores LINQ – de Unión

Unión es la operación de relacionar un objeto de una fuente de datos con un segundo objeto. Se relacionan a través de un atributo en común.

Join

El operador Join es similar a Inner Join (SQL), que une una fuente de datos con otra.

Ejemplo: Join

Se unen las tablas producto y categoría, mostrando los campos nombre y precio de la tabla producto y el nombre de la categoría de la tabla categoría.

```
static void join()
{
    using (tiendaEntities db = new tiendaEntities())
    {
        var sql = from pro in db.PRODUCTOS
                  join cat in db.CATEGORIAS
                on pro.IDCATEGORIA equals cat.IDCATEGORIA
                      select new { pro.DESCRIPCION, pro.PRECIO, cat.NOMBRE };

        foreach (var item in sql)
        {
            Console.WriteLine("Producto: {0} - {1}", item.DESCRIPCION, item.NOMBRE);
        }
    }
}
```

Utilizando la sintaxis del método

```
static void join_v2()
{
    using (tiendaEntities db = new tiendaEntities())
    {
        var sql = db.PRODUCTOS.Join(db.CATEGORIAS,
            cat => cat.IDCATEGORIA,
            pro => pro.IDCATEGORIA, (cat, pro) => new { pro.NOMBRE, nomcat = cat.NOMBRE });

        foreach (var item in sql)
        {
            Console.WriteLine("Producto: {0} - {1}", item.NOMBRE, item.nomcat);
        }
    }
}
```

Operadores LINQ – de Unión

GroupJoin

El operador GroupJoin une cada valor o elemento de la llave primaria (primera o izquierda), con los valores correspondientes de la derecha. Este tipo de unión es muy útil cuando se desea crear una estructura de datos jerárquica.

Ejemplo: GroupJoin

Se unen las tablas producto y categoría, Muestra los productos de cada categoría.

```
static void GroupJoin()
{
    using (tiendaEntities db = new tiendaEntities())
    {
        var sql = db.CATEGORIAS.GroupJoin(db.PRODUCTOS,
            p => p.IDCATEGORIA,
            c => c.IDCATEGORIA,
            (prod,cat) => new {
                cate=prod.NOMBRE,
                listaProd=cat.Select(c=>c.NOMBRE)} );

        foreach (var item in sql)
        {
            Console.WriteLine("{0} ", item.cate);
            foreach (var item2 in item.listaProd)
            {
                Console.WriteLine("    {0}",item2);
            }
        }
    }
}
```

Operadores LINQ – de Unión

GroupBy

El operador GroupBy agrupa valores por una columna específica.

Ejemplo: GroupBy

Muestra los productos agrupados por el código de la categoria

```
static void GroupBy()
{
    using (tiendaEntities db = new tiendaEntities())
    {

        var sql = db.PRODUCTOS.Where(p=>p.IDPRODUCTO > 0).GroupBy
            (cat=>cat.IDCATEGORIA,
            cat=>cat.IDPRODUCTO);

        foreach (var item in sql)
        {
            Console.WriteLine(item.Key);
            foreach (var prod in item)
            {
                Console.WriteLine("  {0}",prod);
            }
        }
    }
}
```

Gracias