EXPRESIONES DE CONSULTA USANDO LINQ

LINQ

LINQ (*Language INtegrated Query*) fue inicialmente soportado por .NET Framework 3.0 con la finalidad de ofrecer la posibilidad de expresar las operaciones de consulta en el propio lenguaje (C#, por ejemplo) y no como literales de cadena pertenecientes a otro lenguaje incrustados en el código de aplicación (sirva como ejemplo las sentencias SQL que en el capítulo anterior incrustábamos en el código para acceder a una base de datos). Finalmente, es con la versión 3.5 de .NET Framework, espacios de nombres *System.Linq* y *System.Data.Linq*, cuando LINQ queda totalmente integrado en este marco de trabajo, junto con otras bibliotecas como WPF, WCF, WF o ASP.NET AJAX, haciendo realidad la implementación de aplicaciones que contengan única y exclusivamente código .NET.

Para explicar con más claridad las expresiones de consulta, supongamos que hemos definido las clases *CPais* y *CPersona* así:

```
public sealed class CPais
{
  public string Codigo { get; set; }
  public string Nombre { get; set; }
}

public sealed class CPersona
{
  public string Nombre { get; set; }
  public DateTime FechaNac { get; set; }
  public string PaisNac { get; set; }
}
```

Desde el punto de vista de LINQ una consulta no es más que una expresión que recupera datos de un origen de datos. Todas las operaciones de consulta LINQ se componen de tres acciones distintas:

1. Obtención del origen de datos. Por ejemplo:

```
List<CPersona> listPersona = new List<CPersona> {
  new CPersona {
    Nombre = "Elena", FechaNac = new DateTime(1990, 10, 25), PaisNac = "ES" },
  new CPersona {
    Nombre = "Manuel", FechaNac = new DateTime(1991, 9, 21), PaisNac = "US"},
  new CPersona {
    Nombre = "Javier", FechaNac = new DateTime(1990, 7, 2), PaisNac = "ES" },
  new CPersona {
    Nombre = "María", FechaNac = new DateTime(1991, 10, 1), PaisNac = "EN"}
};
```

2. Creación de la consulta. Por ejemplo:

```
var personas1990 =
        from p in listPersona
        where p.FechaNac.Year == 1990
        orderby p.Nombre
        select new { Nombre = p.Nombre };
```

3. Ejecución de la consulta. En LINQ, la ejecución de la consulta es distinta de la propia consulta; dicho de otra forma, no se recuperan los datos con la simple creación de la variable de consulta (*personas1990* en nuestro caso) sino que hay que ejecutarla, por ejemplo, en una instrucción **foreach**:

```
foreach (var persona in personas1990)
  Console.WriteLine(persona.Nombre);
```

Sintaxis de las expresiones de consulta

Básicamente, una expresión de consulta siempre comienza con la cláusula **from ... in**, en la que se especifica una variable local que representa a cada elemento en la secuencia de origen así como el origen de datos, y a continuación se pueden escribir una o más cláusulas **from ... in**, **let**, **where**, **join ... in ... on ... equals**, **join ... in ... on ... equals ... into**, **orderby ... [ascending | descending]**, **select**, **group ... by**, o **group ... by ... into**. Opcionalmente, al final de la expresión puede escribirse una cláusula de continuación que comienza con **into** y continúa con el cuerpo de otra consulta.

Cláusula group

La clausula **group** permite agrupar los resultados según la clave que se especifique. Por ejemplo, la siguiente expresión de consulta genera una secuencia *personasPorAño* con las personas de la lista *listPersona* agrupadas por años:

```
var personasPorAño =
    from p in listPersona
    group p by p.FechaNac.Year; // Key: año

foreach (var grupoPersonas in personasPorAño)
{
  Console.WriteLine(grupoPersonas.Key); // año
  foreach (var persona in grupoPersonas)
    Console.WriteLine("    {0}", persona.Nombre);
}
```

El resultado es una secuencia de elementos de tipo **IGrouping**<*TKey*, *T*> (hereda de **IEnumerable**<*T*>) que define una propiedad **Key** del tipo de la clave de agrupación, en nuestro caso de tipo **int**.

Este otro ejemplo, haciendo uso de la lista de países y de la de personas, agrupa las personas según su país de nacimiento:

```
var personasPais =
    from pais in listPais
    join pers in listPersona
              on pais.Codigo equals pers.PaisNac
    group new { Nombre = pers.Nombre } by pais.Nombre;

foreach (var grupoPerPais in personasPais)
{
  Console.WriteLine(grupoPerPais.Key); // nombre país
  foreach (var persona in grupoPerPais)
    Console.WriteLine("    {0}", persona.Nombre);
}
```

Productos cartesianos

En bases de datos, el producto cartesiano de dos tablas no es más que otra tabla resultante de combinar cada fila de la primera con cada fila de la segunda. En LINQ podemos aplicar esta definición utilizando dos cláusulas **from** (un producto cartesiano se implementa mediante el método **SelectMany**). Por ejemplo:

```
var productoCartesiano =
    from pais in listPais
    from pers in listPersona
```

```
    select new { NomPais = pais.Nombre, NomPers = pers.Nombre };

foreach (var elem in productoCartesiano)
  Console.WriteLine("{0}  {1}", elem.NomPers, elem.NomPais );
```

Es recomendable evitar los productos cartesianos por la explosión combinatoria que generan, o crearlos con restricciones. Por ejemplo, la siguiente expresión de consulta muestra el nombre las personas junto con el país donde nacieron:

```
var productoCartesiano =
    from pais in listPais
    from pers in listPersona
    where pais.Codigo == pers.PaisNac
    select new { NomPers = pers.Nombre, NomPais = pais.Nombre };
```

Cláusula join

La clausula **join** se utiliza para realizar una operación de combinación (**join** funciona siempre con colecciones de objetos, en lugar de con tablas de base de datos, aunque en LINQ no es necesario utilizar esta cláusula tan a menudo como en SQL, porque las claves externas en LINQ se representan en el modelo de objetos como propiedades que contienen una colección de elementos). Con **join** se trata de limitar las combinaciones que produciría un producto cartesiano, manteniendo únicamente los elementos de las secuencias que casan de acuerdo con el criterio establecido. Por ejemplo, la combinación que realizamos en el apartado anterior (producto cartesiano) podríamos escribirla mejor así, ya que el rendimiento es muy superior:

```
var combinacion =
    from pais in listPais
    join pers in listPersona
              on pais.Codigo equals pers.PaisNac
    select new { NomPers = pers.Nombre, NomPais = pais.Nombre };

foreach (var elem in combinacion)
  Console.WriteLine("{0}  {1}", elem.NomPers, elem.NomPais);
```

Cláusula into

La clausula **into** puede utilizarse para crear un identificador temporal que almacene los resultados de una cláusula **group**, **join** o **select** en un nuevo identificador. Por ejemplo, la siguiente expresión de consulta genera una secuencia *persNacidasPorAño* con los años de nacimiento de las personas de la lista *listPersona* agrupadas por año de nacimiento más el número de personas de cada grupo, pero sólo los grupos con un número mínimo de personas:

```
var persNacidasPorAño =
    from p in listPersona
    group p by p.FechaNac.Year into grupoPersAño
    where grupoPersAño.Count() >= 2
    select new { Año = grupoPersAño.Key,
                 PorAño = grupoPersAño.Count() };

foreach (var persona in persNacidasPorAño)
{
  Console.WriteLine("En {0} nacieron {1} o más personas.",
    persona.Año, persona.PorAño);
}
```

¿Recuerda el resultado de la expresión de consulta siguiente?

```
var personasPais =
    from pais in listPais
    join pers in listPersona
              on pais.Codigo equals pers.PaisNac
    group new { Nombre = pers.Nombre } by pais.Nombre;
```

Era éste:

```
España
    Elena
    Javier
Inglaterra
    María
Estados Unidos
    Manuel
```

¿Cómo modificamos la expresión de consulta para que los países aparezcan en orden ascendente?. Pues introduciendo ese resultado en una secuencia y ordenando ésta. Para ello utilizaremos **into** con **group**. Esto es:

```
var persPais =
    from pais in listPais
    join pers in listPersona
              on pais.Codigo equals pers.PaisNac
    group new { Nombre = pers.Nombre }
          by pais.Nombre into persPaisOrd orderby persPaisOrd.Key
    select persPaisOrd;
```

Otro ejemplo. ¿Cómo obtenemos una lista de países ordenada ascendentemente con el número de personas por país?. Combinamos cada país con las personas de ese país y las contamos. Para ello utilizaremos **into** con **join**. Esto es:

```
var PaisNumPers =
```

```
  from pais in listPais orderby pais.Nombre
  join pers in listPersona
            on pais.Codigo equals pers.PaisNac
            into grupoPais
  select new { NomPais = pais.Nombre, NumPers = grupoPais.Count()
};

foreach (var pais in PaisNumPers)
{
  Console.WriteLine("En {0} hay {1} personas.",
                    pais.NomPais, pais.NumPers);
}
```

Cláusula let

La clausula **let** se utiliza para almacenar el resultado de una subexpresión con el fin de utilizarlo en cláusulas posteriores. Por ejemplo, la siguiente expresión de consulta, haciendo uso de la cláusula **let**, genera una secuencia *persPorAñoMes* con las personas que han nacido en un mes y un año determinados:

```
int unMes = 10, unAño = 1991;

var persPaisAñoMes =
    from pais in listPais orderby pais.Nombre
    join pers in listPersona
              on pais.Codigo equals pers.PaisNac
    let mesNac = pers.FechaNac.Month
    let añoNac = pers.FechaNac.Year
    where añoNac == unAño && mesNac == unMes
    group new { Nombre = pers.Nombre } by pais.Nombre into perPais
    select perPais;

Console.WriteLine("Nacidos en el mes {0} del año {1}:", unMes, unAño);
foreach (var grupoPerPais in persPaisAñoMes)
{
  Console.WriteLine(grupoPerPais.Key);
  foreach (var persona in grupoPerPais)
    Console.WriteLine("    {0}", persona.Nombre);
}
```