Para ver el artículo en inglés, active la casilla Inglés. También puede ver el texto en inglés en una ventana emergente si pasa el puntero del mouse por el texto.

Basic LINQ Query Operations (C#)

En este tema se ofrece una breve introducción a las expresiones de consulta LINQ y algunas de las clases de operaciones típicas que se realizan en una consulta. En los temas siguientes se ofrece información más detallada:

Expresiones de consultas LINQ (Guía de programación de C#)

Standard Query Operators Overview

Walkthrough: Writing Queries in C# (LINQ)

Nota

Si ya está familiarizado con un lenguaje de consultas como SQL o XQuery, puede omitir la mayoría de este tema. Lea la parte dedicada a la cláusula **from** en la sección siguiente para obtener información sobre el orden de las cláusulas en las expresiones de consulta LINQ.

Obtener un origen de datos

En una consulta LINQ, el primer paso es especificar el origen de datos. En C#, como en la mayoría de los lenguajes de programación, se debe declarar una variable antes de poder utilizarla. En una consulta LINQ, la cláusula **from** aparece en primer lugar para introducir el origen de datos (`customers`) y la *variable de rango* (`cust`).

C#

```
//queryAllCustomers is an IEnumerable<Customer>
var queryAllCustomers = from cust in customers
                        select cust;
```

La variable de rango es como la variable de iteración en un bucle **foreach**, con la diferencia de que en una expresión de consulta realmente no se produce ninguna iteración. Cuando se ejecuta la consulta, la variable de rango actúa como referencia para cada elemento sucesivo de `customers`. Dado que el compilador puede deducir el tipo de `cust`, no tiene que especificarlo explícitamente. Una cláusula **let** puede introducir variables de rango adicionales. Para obtener más información, consulte let (Cláusula, Referencia de C#).

Nota

Para los orígenes de datos no genéricos, como ArrayList, el tipo de la variable de rango debe establecerse explícitamente. Para obtener más información, consulte How to: Query an ArrayList with LINQ y from (Cláusula, Referencia de C#).

Filtrar

Probablemente la operación de consulta más común es aplicar un filtro en forma de expresión booleana. El filtro hace que la consulta devuelva sólo 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. En el ejemplo siguiente, sólo se devuelven los `customers` cuya dirección se encuentra en Londres (London).

C#

```
var queryLondonCustomers = from cust in customers
                           where cust.City == "London"
                           select cust;
```

Puede utilizar los operadores lógicos **AND** y **OR** de C#, con los que ya estará familiarizado, para aplicar las expresiones de filtro que sean necesarias en la cláusula **where**. Por ejemplo, para devolver sólo los clientes con dirección en "London" **AND** cuyo nombre sea "Devon", escribiría el código siguiente:

C#

```
where cust.City=="London" && cust.Name == "Devon"
```

Para devolver los clientes con dirección en Londres o París, escribiría el código siguiente:

C#

```
where cust.City == "London" || cust.City == "Paris"
```

Para obtener más información, consulte where (Cláusula, Referencia de C#).

Ordering

A menudo es necesario ordenar los datos devueltos. La cláusula **orderby** hará que los elementos de la secuencia devuelta se ordenen según el comparador predeterminado del tipo que se va a ordenar. Por ejemplo, la consulta siguiente se puede extender para ordenar los resultados según la propiedad `Name`. Dado que `Name` es una cadena, el comparador predeterminado realiza una ordenación alfabética de la A a la Z.

C#

```
var queryLondonCustomers3 =
    from cust in customers
    where cust.City == "London"
    orderby cust.Name ascending
    select cust;
```

Para ordenar los resultados en orden inverso, de la Z a la A, utilice la cláusula `orderby…descending`.

Para obtener más información, consulte orderby (Cláusula, Referencia de C#).

Grupo

La cláusula **group** permite agrupar los resultados según la clave que se especifique. Por ejemplo, podría especificar que los resultados se agrupen por `City` para que todos los clientes de London o París estén en grupos individuales. En este caso, la clave es `cust.City`.

C#

```
// queryCustomersByCity is an IEnumerable<IGrouping<string, Customer>>
  var queryCustomersByCity =
      from cust in customers
      group cust by cust.City;

  // customerGroup is an IGrouping<string, Customer>
  foreach (var customerGroup in queryCustomersByCity)
  {
      Console.WriteLine(customerGroup.Key);
      foreach (Customer customer in customerGroup)
      {
          Console.WriteLine("    {0}", customer.Name);
      }
  }
```

Al finalizar una consulta con una cláusula **group**, los resultados adoptan la forma de una lista de listas. Cada elemento de la lista es un objeto que tiene un miembro `Key` y una lista de elementos agrupados bajo esa clave. Al procesar una iteración en una consulta que genera una secuencia de grupos, debe utilizar un bucle **foreach** anidado. El bucle exterior recorre en iteración cada grupo y el bucle interior recorre en iteración los miembros de cada grupo.

Si debe hacer referencia a los resultados de una operación de grupo, puede utilizar la palabra clave **into** para crear un identificador con el que se puedan realizar más consultas. La consulta siguiente devuelve sólo los grupos que contienen más de dos clientes:

C#

```
// custQuery is an IEnumerable<IGrouping<string, Customer>>
var custQuery =
    from cust in customers
    group cust by cust.City into custGroup
    where custGroup.Count() > 2
    orderby custGroup.Key
    select custGroup;
```

Para obtener más información, consulte group (Cláusula, Referencia de C#).

Combinación

Las operaciones de combinación crean asociaciones entre las secuencias que no se modelan explícitamente en los orígenes de datos. Por ejemplo, puede realizar una combinación para buscar todos los clientes y distribuidores que tengan la misma ubicación. En LINQ, la cláusula **join** funciona siempre con colecciones de objetos, en lugar de con tablas de base de datos directamente.

C#

```
var innerJoinQuery =
    from cust in customers
    join dist in distributors on cust.City equals dist.City
    select new { CustomerName = cust.Name, DistributorName = dist.Name };
```

En LINQ no es necesario utilizar **join** 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. Por ejemplo, un objeto `Customer` contiene una colección de objetos `Order`. En lugar de realizar una combinación, se tiene acceso a los pedidos utilizando la notación de punto:

```
from order in Customer.Orders...
```

Para obtener más información, consulte join (Cláusula, Referencia de C#).

Selección (proyecciones)

La cláusula **select** genera resultados de consulta y especifica la "forma" o el tipo de cada elemento devuelto. Por ejemplo, puede especificar si sus resultados estarán compuestos de objetos `Customer` completos, un solo miembro, un subconjunto de miembros o algún tipo de resultado completamente diferente basado en un cálculo o en un objeto nuevo. Cuando la cláusula **select** genera algo distinto de una copia del elemento de origen, la operación se denomina *proyección*. El uso de proyecciones para transformar los datos es una eficaz funcionalidad de las expresiones de consulta LINQ. Para obtener más información, consulte Transformaciones de datos con LINQ (C#) y select (Cláusula, Referencia de C#).

Vea también

Getting Started with LINQ in C#
Expresiones de consultas LINQ (Guía de programación de C#)
Walkthrough: Writing Queries in C# (LINQ)
Palabras clave de consultas (Referencia de C#)
Tipos anónimos (Guía de programación de C#)
Operaciones básicas de consulta (Visual Basic)