LINQ: ЯЗЫК ИНТЕГРИРОВАННЫХ ЗАПРОСОВ

LINQ (Language Integrated Query – язык интегрированных запросов) представляет собой новую технологию обработки данных на уровне языка программирования.

Запрос представляет собой выражение, позволяющее получить данные из источника данных. Запросы обычно выражаются на специальном языке запросов. С развитием языков программирования были разработаны различные языки запросов для различных типов источников данных, например SQL для реляционных баз данных и XQuery для XML. Таким образом, программисты вынуждены изучать новый язык запросов для каждого типа источника и каждого типа формата данных. LINQ упрощает эту ситуацию, предлагая согласованную модель для работы с данными в различных видах источников данных и в различных форматах.

В языке C# LINQ-запрос можно адресовать к базам данных SQL Server, XML-документам, наборам данных ADO.NET и любой коллекции объектов, реализующей интерфейс IEnumerable.

Выполнение типового запроса LINQ состоит из трех последовательных действий:

1. Получение источника данных.
2. Создание запроса.
3. Собственно выполнение запроса.

Рассмотрим различные варианты шаблонов запросов.

Шаблон from-where-select

Шаблон from-where-select используется для выборки из источника данных, удовлетворяющих некоторому условию. В следующем примере рассмотрим применение данного шаблона; при этом в качестве источника данных будем использовать массив целых чисел.

```
using System;
using System.Linq;
namespace Example
{
   class Program
   {
      static void Main()
      {
         //получение источника данных
         int[] number = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 };
         //создание запроса
         var lowNums =
                     from n in number
                     where n <= 5
                     select n;
         //выполнение запроса
         foreach (var x in lowNums)
         {
            Console.Write("{0} ", x);
         }
      }
   }
}
```

Результат работы программы:

```
1 2 3 4 5 0
```

Рассмотрим данный пример более подробно.

При создании запроса используется ключевое слово *var*. Это слово относится к нововведениям C# 3.0 и позволяет объявлять переменную без указания ее типа. Компилятор самостоятельно определит тип переменной, исходя из представления данных. В данном случае для переменной lowNums тип будет определяться исходя из типа выражения, стоящего после оператора присваивания. В нашем случае после оператора присваивания стоит выражение запроса.

В общем случае, выражение запроса должно начинаться предложением *from* и оканчиваться предложением *select*, или *group*. Между предложением *from* и предложением *select*, или *group* может содержаться одно, или несколько необязательных предложений: *where, orderby, join, let*. Допускается вкладывать одно выражение запроса в другое.

Как уже было сказано, выражение запроса должно начинаться с предложения *from*, которое определяет:

1) источник данных, применительно к которому будет выполняться запрос;
2) локальную *переменную диапазона*, представляющую каждый элемент исходной последовательности.

В нашем случае источником данных является массив number, а локальной переменной диапазона – переменная n. Тип этой переменной определяется типом элементов источника данных. В данном случае n будет иметь тип int.

Предложение *where* используется для указания элементов, возвращаемых из источника данных. Это предложение применяет логическое условие к каждому исходному элементу из источника данных и возвращает элементы, для которых заданное условие является истинным. В нашем случае логическое условие отбирает из источника данных элементы, значение которых меньше, или равно 5. Таким образом, предложение where является фильтром, позволяющим отобрать необходимые данные.

Следует отметить, что в одном выражении запроса может присутствовать несколько предложений *where*.

Заканчивается запрос предложением *select*, которое задает тип значений, получаемых при выполнении запроса. Результат основывается на анализе всех предыдущих предложений и на любых выражениях внутри предложения *select*. В нашем случае предложение *select* определяет тип переменной запроса lowNums в соответствии с типом переменной диапазона n. В результате возвращенная последовательность содержит элементы с тем же типом, что и у элементов в источнике данных, т.е., с типом int.

Задание

Измените запрос так, чтобы на экран выводились только четные элементы массива.

Замечание

Назначение предложений group, orderby, join и let будет рассмотрено позже.

Предложение where может содержать более сложное логическое выражение, в котором используются стандартные логические операции и, возможно, внешние функции. Например:

```
using System;
using System.Linq;
```

```
namespace Example
{
   class Program
   {
      static void Main()
      {
         //получение источника данных
         int[] number = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 };
         //создание запроса
         var lowNums =
                     from n in number
                     where n <= 5 || n >= 8
                     select n;
         //выполнение запроса
         foreach (var x in lowNums)
         {
            Console.Write("{0} ", x);
         }
      }
   }
}
```

Результат работы программы:

```
1 2 3 4 5 8 9 0
```

Задание

Измените запрос так, чтобы на экран выводились только четные элементы массива, за исключением нулевых.

Как было сказано ранее, один и тот же запрос можно применять к различным источникам данных. Изменим первый пример так, чтобы в качестве источника данных использовался список.

```
using System;
using System.Linq;
using System.Collections.Generic;
namespace Example
{
   class Program
   {
      static void Main()
      {
         //источник данных - список целых чисел
         List<int> number =
                 new List<int> { 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 };
         var lowNums =
                     from n in number
                     where n <= 5
                     select n;
         foreach (var x in lowNums)
         {
            Console.Write("{0} ", x);
         }
```

```
        }
    }
}
```

Результат работы программы:

```
1 2 3 4 5 0
```

Как видим, в данном примере изменился только источник данных, а сам запрос остался неизменным.

Модифицируем предыдущий пример так, чтобы его можно было использовать для запроса к более сложным структурам данных:

```
using System;
using System.Linq;
using System.Collections.Generic;

namespace Example
{
    class Student
    {
        public string Name;
        public int ID;
        public Student(string name, int id)
        {
            Name = name;
            ID = id;
        }
    }
    class Program
    {
        static void Main()
        {
            //источник данных - список студентов
            List<Student> list =
                  new List<Student> {new Student("Иванов", 1),
                  new Student("Петров", 8),
                  new Student("Сидоров", 6),
                  new Student("Ткачев", 3),
                  new Student("Смирнов", 9),
                  new Student("Цукерман", 2)};
            //создаем запрос по номеру студента
            var students =
                        from n in list
                        where n.ID <= 5
                        select n;
            //выполняем запрос
            foreach (var x in students)
            {
                Console.WriteLine("{0} {1}", x.Name, x.ID);
            }
        }
    }
}
```

Результат работы программы:

```
Иванов 1
Ткачев 3
Цукерман 2
```

Задание

Измените запрос так, чтобы:

1) *на экран выводились данные о студентах, фамилия которых начинается на букву С.*
2) *на экран выводились данные о студентах, фамилия которых начинается на букву С, и номер которых четный.*

Следует помнить, что предложение where является необязательным элементом в выражении запроса и, в общем случае, может отсутствовать. Например, использование следующего запроса позволит нам создать массив, элементы которого на 1 больше элементов исходного массива:

```
using System;
using System.Linq;
namespace Example
{
   class Program
   {
      static void Main()
      {
         //получение источника данных
         int[] number = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 };
         //создание запроса
         var lowNums =
                     from n in number
                     select n+1;
         //выполнение запроса
         foreach (var x in lowNums)
         {
            Console.Write("{0} ", x);
         }
      }
   }
}
```

Результат работы программы:

```
2 3 4 5 6 7 8 9 10 1
```

Задание

Измените запрос так, чтобы создавался новый массив, значение каждого элемента которого было бы в два раза меньше чем значение элементов исходного массиву.

Рассмотрим еще один пример, в котором источником данных служит файловая система компьютера:

```
using System;
using System.Linq;
using System.IO;
```

```
namespace Example
{
   class Program
   {
      static void Main()
      {
         //получение источника данных
         DirectoryInfo dir = new DirectoryInfo(@"d:\");
         //создание запроса
         var files =
                   from n in dir.GetFiles("*.doc")
                   select n;
         //выполнение запроса
         foreach (var x in files)
         {
            Console.WriteLine("{0} {1}", x.FullName, x.CreationTime);
         }
      }
   }
}
```

Результат работы программы:

```
d:\letter.doc   01.04.2008 11:24:03
d:\отчет.doc    23.07.2009 14:56:45
d:\реферат.doc  11.05.2006 20:01:50
```

Замечание

Данный пример используется только в демонстрационных целях. На практике для решения поставленной задачи достаточно выполнить команду: FileInfo [] files = dir.GetFiles(".doc").*

Задание

Измените запрос так, чтобы на экран выводилась информация о файлах, размер которых не превышает 10Кбайт.

Шаблон from-orderby-select

В некоторых случаях целесообразно возвращать отсортированную последовательность данных. Это можно сделать с помощью предложения *orderby*. Рассмотрим различные варианты использования данного предложения.

Пусть дан неупорядоченный массив. Извлечем из него данные в отсортированном виде.

```
using System;
using System.Linq;
namespace Example
{
   class Program
   {
      static void Main()
      {
         int[] number = { 14, 0, 23, -3, 10,19, 45};
```

```
            var sortedNums =
                            from n in number
                            orderby n
                            select n;
            foreach (var x in sortedNums)
            {
               Console.Write("{0} ", x);
            }
         }
      }
   }
```

Результат работы программы:

```
-3 0 10 14 19 23 45
```

Следует отметить, что сортировка данных происходит по возрастанию значения переменной диапазона, указанной после предложения orderby. Для того, чтобы сортировка производилась по убыванию сортируемой величины необходимо указать ключевое слово descending. Например, чтобы предыдущий запрос сортировал данные по убыванию необходимо внести следующие изменения:

```
var sortedNums =
                from n in number
                orderby n descending
                select n;
```

Как мы уже говорили, предложения where и orderby могут комбинироваться между собой, что дает возможность не просто отбирать нужные данные, но и сразу сортировать их:

```
using System;
using System.Linq;
namespace Example
{
   class Program
   {
      static void Main()
      {
         int[] number = { 14, 0, 23, -3, 10,19, 45};
         var sortedNums =
                         from n in number
                         where n%2 == 0
                         orderby n
                         select n;
         foreach (var x in sortedNums)
         {
            Console.Write("{0} ", x);
         }
      }
   }
}
```

Результат работы программы:

```
0 10 14
```

Таким образом, мы отобрали только четные числа и упорядочили их по возрастанию.

Задание

Измените запрос так, чтобы на экран выводились только положительные числа массива, отсортированные по убыванию.

Рассмотрим возможность сортировки более сложных структур:

```
using System;
using System.Linq;
using System.Collections.Generic;
namespace Example
{
   class Student
   {
      public string Name;
      public int ID;
      public Student(string name, int id)
      {
         Name = name;
         ID = id;
      }
   }
   class Program
   {
      static void Main()
      {
         List<Student> list = new List<Student>{
               new Student("Иванов", 1),
               new Student("Петров", 8),
               new Student("Сидоров", 6),
               new Student("Ткачев", 3),
               new Student("Смирнов", 9),
               new Student("Цукерман", 2)};
         var students = from n in list
                        orderby n.Name descending
                        select n;
         foreach (var x in students)
         {
            Console.WriteLine("{0} {1}", x.Name, x.ID);
         }
      }
   }
}
```

Результат работы программы:

```
Цукерман 2
Ткачев 3
Смирнов 9
Сидоров 6
Петров 8
Иванов 1
```

Задание

Измените запрос так, чтобы на экран выводились, данные о студентах, отсортированные по возрастанию поля ID.

Рассмотрим еще один пример, в котором источником данных служит файловая система компьютера:

```
using System;
using System.Linq;
using System.IO;
namespace Example
{
   class Program
   {
      static void Main()
      {
         DirectoryInfo dir = new DirectoryInfo(@"d:\");
         var files =
                  from n in dir.GetFiles()
                  orderby n.Length //сортировка по размеру файла
                  select n;
         foreach (var x in files)
         {
            Console.WriteLine("{0}", x.Name);
         }
      }
   }
}
```

Результат работы программы:

```
letter.doc
список.txt
отчет.doc
реферат.doc
picture.bmp
```

Задание

Измените запрос так, чтобы на экран выводились в алфавитном порядке имена файлов, размер которых превышает 10Кбайт.

Шаблон from-join-select

Предложение *join* используется для связывания элементов из различных последовательностей источников, которые не имеют прямых связей в объектной модели. Единственное требование – элементы в каждом источнике должны совместно использовать некоторое значение, которое можно сравнить на предмет равенства.

Например, некоторое учебное заведение имеет список групп и список студентов. Тогда предложение join можно использовать для создания списка студентов, распределенных по группам.

Предложение *join* в качестве ввода принимает две последовательности источников. Элементы в каждой последовательности должны быть свойством, или содержать свойство, которое можно сравнить с соответствующим свойством в другой последовательности. Предложение

join сравнивает указанные ключи на предмет равенства при помощи специального ключевого слова *equals*. Формат результата предложения *join* зависит от определенного типа выполняемого соединения. Наиболее распространенными типами соединения являются внутреннее и групповое.

Замечание

Другие типы соединений можно изучить в документации C#.

Внутреннее соединение создает простую последовательность пар, например, группа/студент. Групповое соединение создает иерархическую последовательность результатов, которая связывает элементы одного источника с одним, или несколькими совпадающими элементами другого источника данных. Чтобы сравнить эти типы соединений, рассмотрим следующий пример:

```
using System;
using System.Linq;
using System.Collections.Generic;
namespace Example
{
   class Demo
   {
      class Group
      {
         public string Name { get; set; }
         public int GroupID { get; set; }
      }
      class Student
      {
         public string Name { get; set; }
         public int GroupID { get; set; }
      }
      //создали первый источник данных
      List<Group> Groups = new List<Group>()
      {
         new Group(){ Name="Начальный уровень", GroupID=001 },
         new Group(){ Name="Базовый уровень", GroupID=002 },
         new Group(){ Name="Продвинутый уровень", GroupID=003 }
      };
      //создали второй источник данных
      List<Student> students = new List<Student>()
      {
         new Student(){ Name="Иванов",  GroupID=001 },
         new Student(){ Name="Петров",  GroupID=002 },
         new Student(){ Name="Сидоров", GroupID=002 },
         new Student(){ Name="Смирнов", GroupID=002 },
         new Student(){ Name="Ткачев", GroupID=003 },
         new Student(){ Name="Цукерман", GroupID=003 },
         new Student(){ Name="Токарев", GroupID=001 },
         new Student(){ Name="Оганесян", GroupID=002 },
      };
      static void Main()
      {
```

```
            Demo list = new Demo();
            list.InnerJoin();
            list.GroupJoin();
        }
        //организуем внутреннее соединение
        void InnerJoin()
        {
            //запрос: если значение  GroupID первого источника данных
            //совпадает со значением GroupID второго источника данных,
            //то создаем новую структуру данных, в которую добавляем
            //пары название группы/имя студента
            var query = from itemGroup in Groups
                        join itemStud in students
                        on itemGroup.GroupID
                        equals itemStud.GroupID
                        select new
                        {
                            Group = itemGroup.Name,
                            Student = itemStud.Name
                        };
            //выводим результаты запроса в виде пар название группы/имя
            //студента
            Console.WriteLine("Внутреннее соединение:");
            foreach (var item in query)
            {
                Console.WriteLine("{0} {1}", item.Group, item.Student);
            }
            Console.WriteLine();
        }
        //организуем групповое соединение
        void GroupJoin()
        {
            //запрос: если значение GroupID первого источника данных
            //совпадает со значением GroupID второго источника данных,
            //то создаем новую структуру данных, в которую каждой
            //группе ставится в соответствие отсортированная
            //последовательность студентов
            var query = from itemGroup in Groups
                        orderby itemGroup.GroupID
                        join itemStud in students
                        on itemGroup.GroupID
                        equals itemStud.GroupID into newGroup
                        select new
                        {
                            Group = itemGroup.Name,
                            Students = from x in newGroup
                                       orderby x.Name
                                       select x
                        };
            //выводим результаты запроса по группам
            Console.WriteLine("Групповое соединение:");
```

```
            foreach (var Grouping in query)
            {
               Console.Write("{0}: ", Grouping.Group);
               foreach (var item in Grouping.Students)
               {
                  Console.Write("{0}  ", item.Name);
               }
               Console.WriteLine();
            }
            Console.WriteLine();
         }
      }
   }
```

Результат работы программы:

```
Внутреннее соединение:
Начальный уровень Иванов
Начальный уровень Токарев
Базовый уровень Петров
Базовый уровень Сидоров
Базовый уровень Смирнов
Базовый уровень Оганесян
Продвинутый уровень Ткачев
Продвинутый уровень Цукерман

Групповое соединение:
Начальный уровень Иванов Токарев
Базовый уровень Оганесян Петров Сидоров Смирнов
Продвинутый уровень Ткачев Цукерман
```

Задание

Измените групповое соединение так, чтобы на экран выводились данные о студентах только начального и базового уровней.

Шаблон from-let-select

В выражении запроса иногда полезно сохранить результат выполнения какой-то составной части выражения, чтобы использовать его в последующих предложениях. Это можно выполнить с помощью ключевого слова *let*, создающего новую переменную диапазона и инициализирующего ее результатом предоставленного выражения. После инициализации значением переменная диапазона не может использоваться для хранения другого значения. Однако если в переменной диапазона хранится запрашиваемый тип, то его можно использовать для вложенного запроса. Например:

```
using System;
using System.Linq;
namespace Example
{
   class Program
   {
      static void Main()
      {
```

```
            string[] strings = {"один два три",
                                "четыре пять шесть",
                                "семь восемь девять десять"};
            //создали новую переменную диапазона и
            // организовали к ней запрос
            var query = from line in strings
                        let words = line.Split(' ')
                           from word in words
                           where word[0] == 'д'
                           select word;
            foreach (var x in query)
            {
               Console.Write("{0} ", x);
            }
         }
      }
   }
```

Результат работы программы:

```
два девять десять
```

Рассмотрим возможность применения данного шаблона к файловой системе:

```
using System;
using System.Linq;
using System.IO;
namespace Example
{
   class Program
   {
      static void Main()
      {
         DirectoryInfo directory = new DirectoryInfo(@"d:\");
         //создали переменную диапазона и
         //организовали к ней запрос
         var query = from item in directory.GetDirectories()
                     let files = item.GetFiles()
                        from file in files
                        orderby file.Length
                        select file;
         foreach (var x in query)
         {
            Console.WriteLine("{0} {1}", x.Name,x.Length);
         }
      }
   }
}
```

Задание

Объясните, информация о каких файлах будет выведена на экран.

Шаблон from-…-group

Предложение *group* позволяет группировать результаты на основе указанного ключа. Например, можно указать, что результаты должны быть сгруппированы по номеру группы так, чтобы все студенты оказались распределенными по соответствующим группам.

```
using System;
using System.Linq;
using System.Collections.Generic;
namespace Example
{
   class Demo
   {
      class Student
      {
         public string Name { get; set; }
         public int GroupID { get; set; }
      }
      static void Main()
      {
         //создали источник данных
         List<Student> students = new List<Student>()
         {
            new Student(){Name="Иванов",  GroupID=001},
            new Student(){Name="Петров",  GroupID=002},
            new Student(){Name="Сидоров", GroupID=002},
            new Student(){Name="Смирнов", GroupID=002},
            new Student(){Name="Ткачев", GroupID=003},
            new Student(){Name="Цукерман", GroupID=003},
            new Student(){Name="Токарев", GroupID=001},
            new Student(){Name="Оганесян", GroupID=002},
         };
         //создали запрос
         var query = from student in students
                     group student by student.GroupID;
         //выполнили запрос
         foreach (var items in query)
         {
            Console.Write("Группа {0}: ",items.Key);
            foreach (var item in items)
            {
               Console.Write("{0} ", item.Name);
            }
            Console.WriteLine();
         }
         Console.WriteLine();
      }
   }
}
```

Результат работы программы:

```
Группа 1: Иванов Токарев
Группа 2: Петров Смирнов Сидоров Оганесян
Группа 3: Ткачев Цукерман
```

На первый взгляд, результат очень похож на групповое соединение, выполненное с помощью предложения join. Различие заключается в том, что обрабатывается один источник информации. Более того, когда запрос завершается предложением *group*, результаты представляются в виде списка из списков. Каждый элемент в списке является объектом, имеющим член *Key* и список элементов, сгруппированных по этому ключу. При итерации запроса, создающего последовательность групп, необходимо использовать вложенный цикл *foreach*. Внешний цикл выполняет итерацию каждой группы, а внутренний цикл – итерацию членов каждой группы.

Если необходимо ссылаться на результаты операции группировки, то можно использовать ключевое слово *into* для создания идентификатора, который можно будет запрашивать. Следующий запрос возвращает только те группы, которые содержат более двух студентов:

```
using System;
using System.Linq;
using System.Collections.Generic;
namespace Example
{
   class Demo
   {
      class Student
      {
         public string Name { get; set; }
         public int GroupID { get; set; }
      }
      static void Main()
      {
         List<Student> students = new List<Student>()
         {
            new Student(){Name="Иванов",  GroupID=001},
            new Student(){Name="Петров",  GroupID=002},
            new Student(){Name="Сидоров", GroupID=002},
            new Student(){Name="Смирнов", GroupID=002},
            new Student(){Name="Ткачев", GroupID=003},
            new Student(){Name="Цукерман", GroupID=003},
            new Student(){Name="Токарев", GroupID=001},
            new Student(){Name="Оганесян", GroupID=002},
         };
         //записали результаты в новую группу
         var query = from student in students
                     group student by student.GroupID into Groups
               //выбрали те группы, в которых более 2 человек
                     where Groups.Count() > 2
                     select Groups;
         foreach (var items in query)
         {
            Console.Write("Группа {0}: ",items.Key);
```

```
            foreach (var item in items)
            {
               Console.Write("{0} ", item.Name);
            }
            Console.WriteLine();
         }
         Console.WriteLine();
      }
   }
}
```

Результат работы программы:

```
Группа 2: Петров Смирнов Сидоров Оганесян
```

Рассмотрим пример использования данного шаблона для организации запроса к файловой системе:

```
using System;
using System.Linq;
using System.Collections.Generic;
using System.IO;

namespace Example
{
   class Demo
   {
      static void Main()
      {
         DirectoryInfo directory = new DirectoryInfo(@"d:\");
         //группируем по расширению
         var query = from file in directory.GetFiles()
                     group file by file.Extension
                     into Groups
                     orderby Groups.Key
                     select Groups;
         foreach (var items in query)
         {
            Console.Write("Группа {0}: ", items.Key);
            foreach (var item in items)
            {
               Console.Write("{0} ", item.Name);
            }
            Console.WriteLine();
         }
         Console.WriteLine();
      }
   }
}
```

Результат работы программы:

```
Группа .bmp: pucture.bmp
Группа .doc: letter.doc отчет.doc реферат.doc
Группа .txt: список.txt
```

Задание

Измените запрос так, чтобы файлы в каждой группе сортировались по убыванию размера.

Практикум №20

1. В заданиях 8-10 практикума из раздела «Коллекции» реализуйте поиск информации с помощью LINQ запросов.
2. Реализуйте поддержку LINQ запросов для рассмотренных ранее АТД «список» и АТД«дерево», после чего продемонстрируйте использование запросов.
3. Объясните, можно ли реализовать поддержку LINQ запросов для АТД «стек» или АТД «очередь».

Замечание

Помните, что LINQ запросы можно применять только к экземплярам классов, реализующих интерфейс IEnumerable.