

## Практична робота 2. Основи LINQ

**Мета.** Ознайомитись з основними засобами мови LINQ для роботи з даними.

### План практичного заняття

1. Вступ
2. Запити до масивів
3. Впорядковування результатів запиту
4. Запит до колекції об'єктів
5. Проекція: створення нових об'єктів в запитах
6. З'єднання
7. Версії мови LINQ

### Зміст заняття

#### 1. Вступ

*Мова інтегрованих запитів LINQ (Language Integrated Query) — розширення мови C#, яке з'явилося у версії C# 3.0 і використовується для вибору даних з різних джерел - колекцій об'єктів, баз даних SQL, документів XML і зовнішніх застосувань (веб-служби Amazon і корпоративні каталоги). Синтаксис LINQ і методи однакові для усіх джерел даних.*

Використання мови LINQ спрощує код, усуває потребу у використанні операторів циклу для фільтрації і впорядкування, дозволяє зосередитися на об'єктах, якими оперує програма. Крім мови запитів, LINQ містить ще методи розширення, які спрощують впорядкування, групування і обчислення.

#### 2. Запити до масивів

Базові системні класи підтримки LINQ знаходяться в просторі імен `System.Linq`.

Оператор запиту LINQ має вигляд:

```
var queryResults = from n in names
                    where n.StartsWith("S")
                    select n;
```

Створити запит для пошуку даних в масиві об'єктів з використанням LINQ.

```
using System;
using System.Linq;

namespace ConsoleApplication1
{
    class Program
    {
        public static void Main(string[] args)
        {
            string[] names =
                {"Alonso", "Zheng", "Smith", "Jones", "Smythe", "Small", "Pertro",
                "Ruiz", "Hsieh", "Jorgenson", "Ilyich", "Singh", "Samba", "Fatimah" };
            // Імена, які розпочинаються з S
        }
    }
}
```

```

var queryResults = from n in names
                    where n.StartsWith("S")
                    select n;

Console.WriteLine("Names beginning with S :");

foreach (var item in queryResults)
{
    Console.WriteLine(item);
}

Console.Write("Program finished, press Enter/Return to continue:");
Console.ReadLine();
}
}
}

```

- Результат `queryResults` запиту описується як змінна невизначеного типу **var**. Її тип визначається на етапі компіляції і є типом результату запиту, тобто посиланням на колекцію об'єктів.
- Конструкція **from** вказує джерело з якого дані вибираються, в прикладі – це масив рядків `names`; з масиву буде сформована вибірка.
- Змінна **n** називається змінною запиту (аналог змінної циклу, але це не цикл) і використовується для доступу до елементів колекції. Вказувати її тип компілятору не потрібно. Він автоматично визначає його на основі тип елементів масиву `names`.
- Конструкція **where** називається *операцією обмеження*, може містити логічний вираз, який застосовується для вибору тих значень з колекції джерела даних, для яких умова істинна. Конструкція **where** необов'язкова.

В прикладі, елемент з масиву рядків `name` повинен розпочинатися з букви `S`.

Інші приклади застосування **where**:

- довжина має бути більша 10 - `where n.Length > 10;`
- ім'я повинне містити в букву `Q` - `where n.Contains("Q").`

- Конструкція **select** вказує, які елементи з'являться в результаті. Її використання буде продемонстровано в наступних прикладах.

### 3. Впорядковування результатів запиту

Конструкція **orderby** використовується для впорядкування колекції об'єктів.

Розглянемо приклад

```

var queryResults= from n in names where
                  n.StartsWith("S") orderby n
                  select n;

```

За замовчуванням використання **orderby** в запиті упорядковує елементи за збільшенням (від A до Z).

Конструкція **orderby n descending** задає порядок за спаданням.

Можна впорядковувати за значеннями довільного виразу. Щоб упорядкувати, наприклад, за останньою літерою імені можна використати вираз:

**orderby n.Substring(n.Length - 1)**

#### 4. Запит до колекції об'єктів

Розглянемо приклад, де колекція формується як сукупність об'єктів класу з декількома полями.

**Приклад.** Вибрати із списку користувачів із країн Азії

```
namespace ConsoleApplication1
{
    class Customer
    {
        public string ID { get; set; }
        public string City { get; set; }
        public string Country { get; set; }
        public string Region { get; set; }
        public decimal Sales { get; set; }
        public override string ToString()
        {
            return String.Format("ID:{0} City:{1} Country:{2} Region{3} Sales{4}",
                ID, City, Country, Region, Sales);
        }
    }
    class Program
    {
        public static void Main(string[] args)
        {
            List<Customer> customers = new List<Customer>{
                new Customer {ID="A",City="New York",Country="USA",Region ="North America",Sales=9999},
                new Customer { ID="B", City="Peking", Country="China", Region ="Asia", Sales=9999},
                new Customer { ID="C", City="Kiev", Country="Ukraine", Region ="Europe", Sales=9999},
                new Customer { ID="D", City="Moskva", Country="Russia", Region ="Asia", Sales=9999} };

            var queryResults = from c in customers
                               where c.Region == "Asia"
                               select c;

            Console.WriteLine("Customers in Asia :");
            foreach (Customer c in queryResults)
            {
                Console.WriteLine(c);
            }
            Console.WriteLine("Program finished, press any key to continue");
            Console.ReadLine();
        }
    }
}
```

#### 5. Проекція: створення нових об'єктів в запитах

*Проекція* використовується для створення нових типів даних з декількох інших типів.

Створимо колекцію об'єктів, яка буде містити лише поле `City` зі списку `customer` попереднього прикладу.

```
var queryResults = from c in customers
                    where c.Region == "Asia"
                    select c.City;
```

Проте, на відміну від SQL, LINQ не дозволяє використання декількох полів в конструкції `select`.

Використання конструкції `select c.City, c.Country, c.Sales` призведе до помилки під час компіляції.

Для формування колекції об'єктів з декількома полями можна використати наступну конструкцію

```
var queryResults = from c in customers
                    where c.Region == "North America"
                    select new { c.City, c.Country, c.Sales};
```

```
foreach (var item in queryResults)
{
    Console.WriteLine(item);
}
```

У конструкції `select` створюються об'єкти *анонімного типу* з властивостями `City`, `Country` і `Sales`.

*Примітка.* Порівняйте з попереднім прикладом виведені результати.

## 6. З'єднання

Зміст операції з'єднання в LINQ близький до її змісту в мові SQL.

**Приклад.** Розглянемо приклад формування нової колекції об'єктів з двох інших.

Додамо в попередню програму ще й список замовлень гіпотетичного клієнта і підрахуємо загальну суму замовлень кожного користувача.

### Постановка задачі

Утворити з тих об'єктів двох списків у яких співпадають ідентифікатори новий список об'єктів, який буде містити загальну суму продаж кожного.

```
List<Customer> customers = new List<Customer>
{
    new Customer { ID="A", City="New York", Country="USA", Region="North America", Sales=9999 },
    new Customer { ID="B", City="Peking", Country="China", Region="Asia", Sales=9999 },
    new Customer { ID="C", City="Kiev", Country="Ukraine", Region="Europe", Sales=9999 },
    new Customer { ID="D", City="Moskva", Country="Russia", Region="Asia", Sales=9999 }
};

List<Order> orders = new List<Order>
{
    new Order { ID=1, CustomerID="A", Amount=100 },
    new Order { ID=2, CustomerID="Q", Amount=200 }
};
```

c.Sales + o.Amount

Рис. 1. Операція з'єднання

## Програма

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

namespace ConsoleApplication1
{
    class Customer
    {
        public string ID { get; set; }
        public string City { get; set; }
        public string Country { get; set; }
        public string Region { get; set; }
        public decimal Sales { get; set; }
        public override string ToString()
        {
            return String.Format("ID:{0} City:{1} Country:{2} Region{3} Sales{4}",
                ID, City, Country, Region, Sales);
        }
    }
    class Order
    {
        public int ID { get; set; }
        public string CustomerID { get; set; }
        public decimal Amount { get; set; }
    }
    class Program
    {
        public static void Main(string[] args)
        {
            List<Customer> customers = new List<Customer>{
                new Customer{ID="A", City = "New York",Country="USA",Region="North America",Sales=9999},
                new Customer{ID ="B",City="Peking",Country="China", Region = "Asia", Sales = 9999},
                new Customer{ ID ="C",City="Kiev",Country="Ukraine", Region ="Europe", Sales = 9999},
                new Customer{ID ="D",City= "Moskva",Country="Russia", Region= "Asia", Sales = 9999}
            };

            List<Order> orders = new List<Order>
            {
                new Order { ID = 1, CustomerID = "A", Amount = 100 },
                new Order { ID = 2, CustomerID = "Q", Amount = 200 }
            };

            var queryResults = from c in customers
                               join o in orders on c.ID equals o.CustomerID
                               select new
                               {
                                   c.ID,
                                   c.City,
                                   SalesBefore = c.Sales,
                                   NewOrder = o.Amount,
                                   SalesAfter = c.Sales + o.Amount
                               };

            Console.WriteLine("\n Client's orders: \n");
            foreach (var item in queryResults)
            {
                Console.WriteLine(item);
            }
            Console.WriteLine("\n Program finished, press any key to continue \n");
            Console.ReadLine();
        }
    }
}
```

## Результат

```
Client's orders:
< ID = A, City = New York, SalesBefore = 9999, NewOrder = 100, SalesAfter = 10099 >
Program finished, press any key to continue
```

За допомогою конструкції `join` можна створити новий об'єкт з тих об'єктів обох списків, у яких значення ключів (ID і CustomerID) співпадають. Зміст цієї операції аналогічний змісту оператора `JOIN` в мові SQL.

Оператор `select` утворює новий об'єкт з вибраними із попередніх об'єктів та новими властивостями.

## 7. Версії мови LINQ

Існує декілька версій мови для різних предметних областей.

**LINQ to Objects** - використовується для реалізації запитів до колекцій динамічних колекцій об'єктів C# таких як масиви, списки та інші.

**LINQ to XML** використовується для реалізації запитів, створення та керування документами XML.

**LINQ to ADO.NET**. ADO.NET або Active Data Objects (активні об'єкти даних) для .NET — загальний термін, який охоплює різні класи і бібліотеки .NET, призначені для доступу до баз даних, у тому числі Microsoft SQL Server, Oracle і т. п.. LINQ to ADO.NET включає LINQ to Entities, LINQ to DataSet і LINQ to SQL.

**LINQ to Entities**. ADO.NET Entity Framework — набір класів інтерфейсу до даних в .NET 4, рекомендований Microsoft для сучасної розробки.

**LINQ to DataSet**. Об'єкт `DataSet` з'явився в першій версії .NET Framework.

Видозміна LINQ to DataSet дозволяє надсилати запити до джерел даних .NET за допомогою LINQ.

**LINQ to SQL**. Альтернативний інтерфейс LINQ для .NET 3.5, орієнтований головним чином на Microsoft SQL Server, який у версії .NET 4 був замінений LINQ to Entities.

**PLINQ**. PLINQ, або Parallel LINQ (Паралельний LINQ), розширює LINQ to Objects бібліотекою для паралельного програмування, яка дозволяє виконувати паралельні запити на багатоядерному процесорі.

## Література

1. Харви Дейтел. C# в подлиннике 2006.
2. Visual C# 2008 Базовый курс.
3. Нейгел К., Ивсен Б., Глинн Дж., Уотсон К. C# 4.0 и платформа .NET4 для профессионалов (2011).
4. Уотсон К., Нейгел К., Педерсен Я.Х., Рид Дж., Скиннер М. Visual C# 2010 полный курс (2011).djvu
5. Эндрю Троелсен. Язык программирования C# 2010 и платформа .NET 4 - 2010.djvu