LINQ to Object

Три частини LINQ -запиту

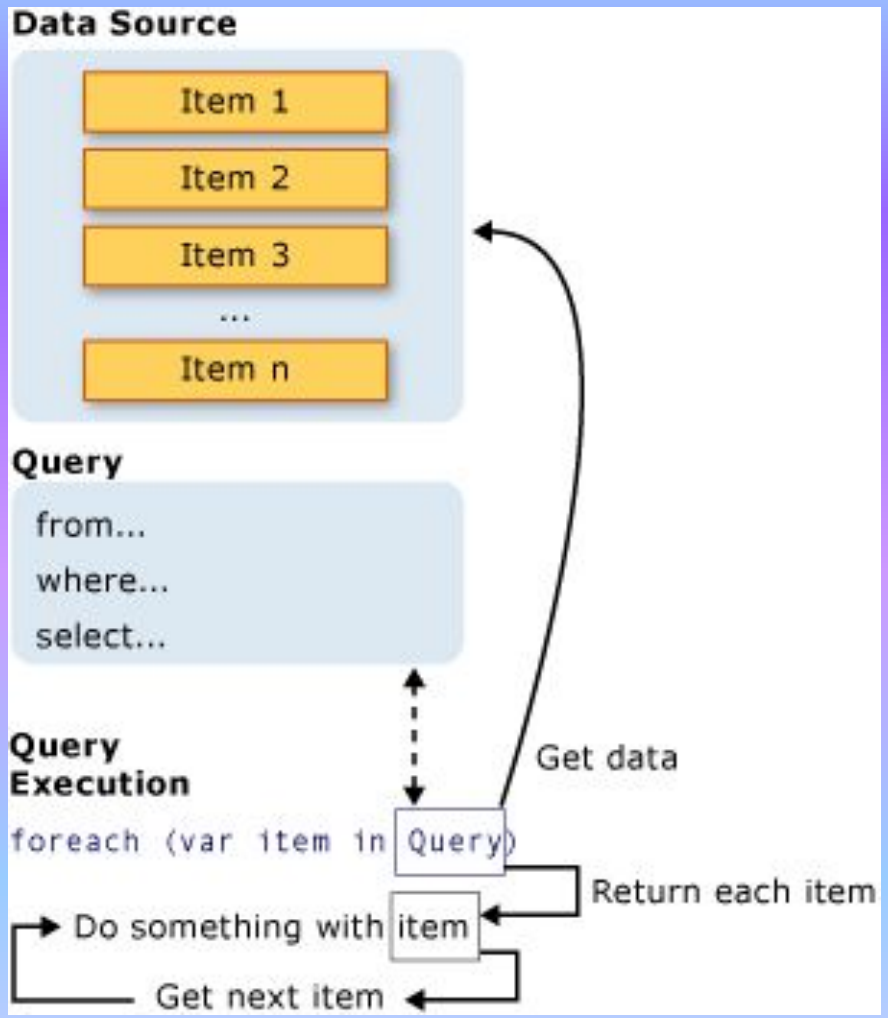
```
//  1. Джерело даних
int[] numbers = new int[7]{0,1,2,3,4,5,6};

// 2. Формування запиту (query)
IEnumerable<int> numQuery =
                 from num in numbers
                 where (num % 2) == 0
                 select num;

// 3. Виконання запиту
foreach (int num in numQuery)
{
    Console.Write("{0} ", num);
}
```

Три частини LINQ -запиту

1. **Джерело даних –** всі послідовності, що реалізують інтерфейс `IEnumerable<T>` - колекції, в тому числі і масиви. (System.Collections.Generic)
2. **Формування запиту (query)–** комбінація операторів стандартних запитів, які є розширеними методами класу `System.Linq.Enumerable` `(where, select, from,…)`.
3. **Виконання запиту:**

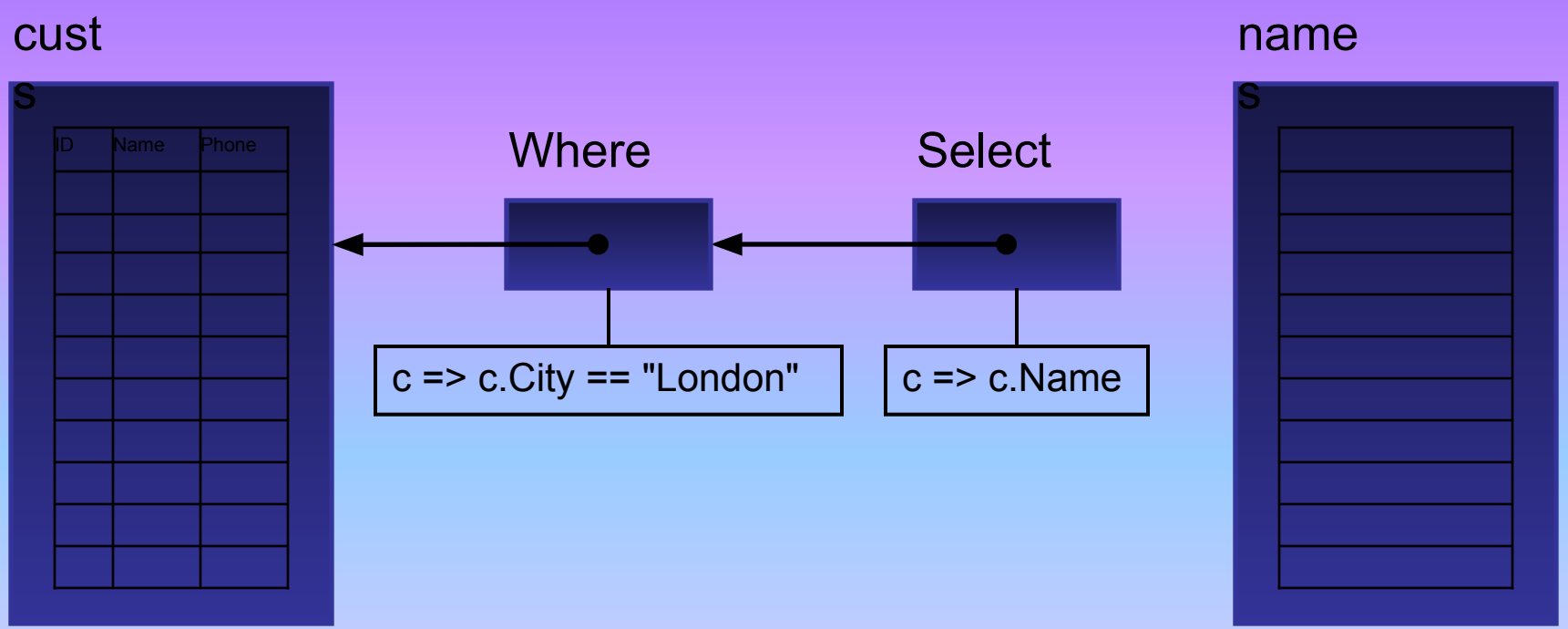
   - **Відкладені запити** . В результаті формування такого запиту утворюється об’єкт `IEnumerable<T>`. Виконання запиту (отримання послідовності) відбувається при переліченні об’єкта запиту в `foreach`.
   - **Невідкладені запити** - виконуються при формуванні запиту – надають послідовність елементів .(ToList)

Відкладене виконання запитів



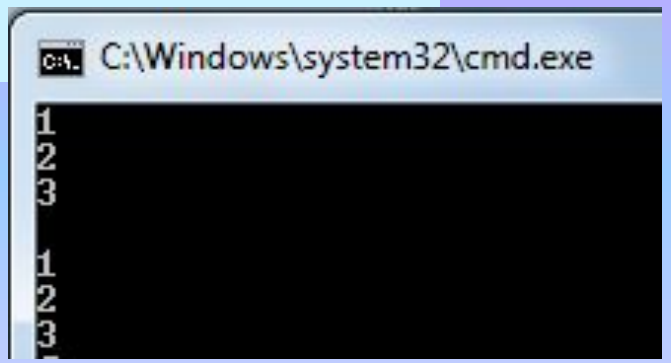
Відкладене виконання запиту

```
Customer[] custs = SampleData.GetCustomers();
var query = from c in custs where c.City == "London" select c.Name;
var query = custs.Where(c => c.City == "London").Select(c => c.Name);
string[] names = query.ToArray();
```



Приклади виконання запитів

```
int [] intArray = new int [] { 1, 2, 3 };

// Формування і виконання запиту
IEnumerable<int> ints = intArray.Select(i => i).ToList();

//  друк отриманої послідовності
foreach (int i in ints)
        Console.WriteLine(i);

// міняємо елемент в джерелі даних.
      intArray[0] = 5;

// друк отриманої послідовності.
foreach (int i in ints)
        Console.WriteLine(i);
```



Standard Query Operators

Restriction	Where
Projection	Select, SelectMany
Ordering	OrderBy, ThenBy
Grouping	GroupBy
Quantifiers	Any, All
Partitioning	Take, Skip, TakeWhile, SkipWhile
Sets	Distinct, Union, Intersect, Except
Elements	First, FirstOrDefault, ElementAt
Aggregation	Count, Sum, Min, Max, Average
Conversion	ToArray, ToList, ToDictionary
Casting	OfType<T>

Оператори стандартних запитів

- Оператори стандартних запитів – це статичні методи класу **`Enumerable`** , визначені як розширюючі методи того типу, яким вони оперують.
- Більшість методів оперують послідовностями (які реалізують **`IEnumerable(T)`** або `IQueryable(T)` інтерфейси).
- Забезпечують можливості запитів щодо фільтрування, проектування, агрегації, сортування і ін.
- методи можуть бути викликані, як статичні методи і як методи об’єкта.
- Методи `Where<T>` і `Select<T>` , що є `where` і `select` операторами – розширюючі методи для `IEnumerable(T)` інтерфейсу.

Where()

```
public static IEnumerable <S> Where<S>(this IEnumerable<S>
source, Func<S,bool> predicate );

public static IEnumerable <S> Where<S>(this IEnumerable<S>
source, Func<S,int,bool> predicate );
```

Делегати Func

- **Оператори стандартних запитів приймають як параметр - делегат `Func`** (останній в списку параметрів).

```
public delegate TR Func<TR>();
public delegate TR Func<T0, TR>(T0 a0);
public delegate TR Func<T0, T1, TR>(T0 a0, T1 a1);
public delegate TR Func<T0, T1, T2, TR>(T0 a0, T1 a1, T2 a2);
public delegate TR Func<T0, T1, T2, T3, TR>(T0 a0, T1 a1, T2 a2, T3 a3);
```

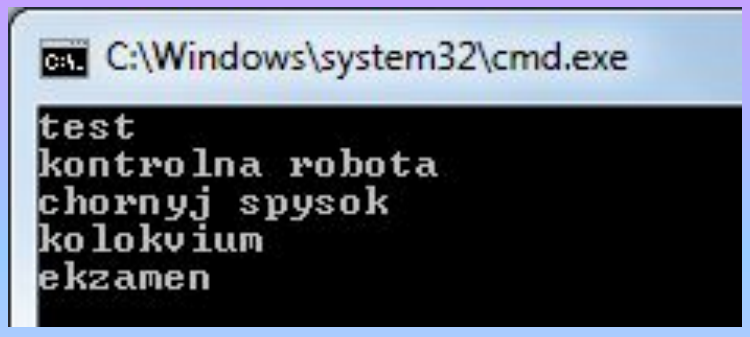
- ✔ `T0, T1, T2, T3` **- типи вхідних параметрів**
- ✔ `TR` **– тип, що повертається**
- ✔ **один з параметрів може бути** `int` **– індекс елемента в послідовності**

Приклад оголошення і використання делегата

```
int [ ] ints = new int [ ] { 1, 2, 3, 4, 5, 6 };

 //  оголошеня делегата через лямбда-вираз
 Func<int, bool> GreaterThanTwo = i => i > 2;

 IEnumerable<int> intsGreaterThanTwo =
                                  ints.Where(GreaterThanTwo);

 foreach (int i in intsGreaterThanTwo)
        Console.WriteLine(i);
```

Лямбда-вираз і виклик делегата

```
string [ ] controls = {"test","samostijna robota",
                       "kontrolna robota","opytuvannia",
            "chornyj spysok","vidviduvannia",
            "kolokvium", "zalik", "ekzamen"};

IEnumerable<string> sequence =
                    controls.Where((p, i) => (i % 2) == 0);

foreach (string s in sequence)
        Console.WriteLine("{0}", s);
```



Агрегація & Об’єднання

OPERATOR	DESCRIPTION
Aggregate	Застосовує функцію до кожного елемента послідовності
Average	Обчислює середнє арифметичне послід.
Count/LongCount	Підраховує кількість елем. в послідовності
Max	Повертає найбільше значення з послідовності числових значень
Min	Повертає найменше значення з послідовності числових значень
Sum	Повертає суму всіх числових значень послідовності
Concat	Об’єднує елементи з двох послідовностей

Aggregate()

```
public static TSource
Aggregate <TSource> ( this IEnumerable<TSource> source,
        Func <TSource, TSource, TSource> func )
```

- Здійснює обчислення над елементами послідовності.
- Застосовує func до кожного елемента послідовності.
- У **func** перший аргумент – результат, другий – елемент послідовності.
- Після кожного застосування **func** перший аргумент замінюється результатом
- перший елемент послідовності – початкове значення для результату.
-

```
int [ ] numbers =  new int [6] {  1, 2, 3, 4, 5, 6 };

int dob =numbers.Aggregate((d,e)=>d*e);

Console.Write("dob={0} ", dob); //720
```

Конвертування

Operator	Description
AsEnumerable	Конвертування послідовності до IEnumerable<T>
AsQueryable	Конвертування послідовності до IQueryable<T>
Cast	Приведення елемента послідовності у вказаний тип
OfType	Вибрати з послідовності елементи лише вказаного типу
ToArray	Конвертування послідовності в масив
ToList	Утворення List<T> з послідовності
ToLookup	Утворення Lookup<K,T> з послідовності (подібної до Dictionary<K, T>
ToSequence	Повертає свій аргумент як IEnumerable<T>

OfType()

```
//повертає послідовність елементів лише вказаного типу

public static IEnumerable<T>
  OfType <TSource> ( this IEnumerable<TSource> source)
```

```
object[] sequence = {1, "Hello", 2.0, 0, -7.9, 'f'};

var rez=sequence.OfType<double>();

foreach (var r in rez)
    Console.Write("{0} ", r);
```

Element

Operator	Description
DefaultIfEmpty	Забезпечує значеннями за замовчуванням для порожньої послідовності
ElementAt	Повертає елемент за вказаним індексом
ElementAtOrDefault	Так як і ElementAt , але повертає default-елемент, якщо індекс за межами
First	Повертає перший елемент послідовності (що задовільняє предикат)
FirstOrDefault	Подібно до First , але повертає значення за замовчуванням, якщо не знайдено
Last	Повертає останній елемент послідовності
LastOrDefault	Подібно до Last , але повертає значення за замовчуванням, якщо не знайдено
Single	Повертає елемент, що задовільняє умову, яка вказана як аргумент
SingleOrDefault	Подібно до Single, але повертає значення за замовчуванням, якщо елемент не знайдено

FirstOrDefault(), LastOrDefault()

```
//Повертає перший елемент (що задовільняє предикат) або default

public static T FirstOrDefault<T>(this IEnumerable<T> source);

public static T FirstOrDefault<T>(this IEnumerable<T> source,
                                  Func<T, bool> predicate);
```

```
int[] numbers = {1, 2, 3, 4, 5, 6, 7, 8, 9};

var query = numbers.FirstOrDefault(n => n > 5);
Console.Write("{0} ", query);

query = numbers.LastOrDefault(n => n > 5);
Console.Write("{0} ", query);
```

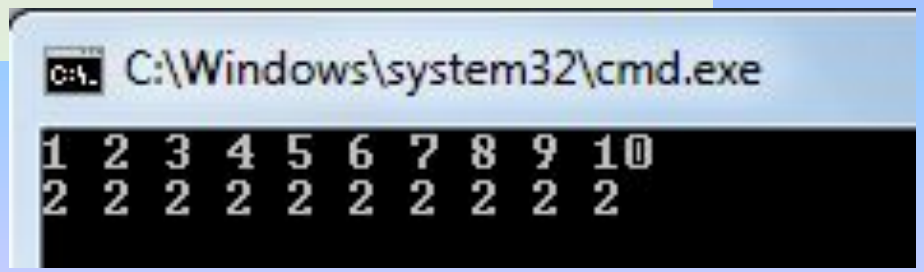
Рівність& Генерування & Групування

Operator	DESCRIPTION
SequenceEqual	Перевірка чи дві послідовності є рівні
Empty	Повертає порожню послідовність елементів вказаного типу
Range	Генерує числову послідовність з вказаної кількості елементів
Repeat	Генерує послідовність з вказаного елемента, повторюючи його вказану кількість разів
GroupBy	Групує елементи послідовності
GroupJoin	Здійснює групування з’єднанням двох послідовностей, спираючись на ключ
Join	Здійснює об’єднання включенням двох послідовностей на основі ключів

Range(), Repeat()

```
//генерує числову послідовність з count ел. починаючи з start
 public static IEnumerable<int> Range(int start,int count);

//генерує послідовність з count елементів element
 public static IEnumerable<T> Repeat<T>(T element, int count);
```

```
IEnumerable<int> ints = Enumerable.Range(1, 10);
foreach (int i in ints)
    Console.Write("{0} ", i);

IEnumerable<int> ints = Enumerable.Repeat(2, 10);
foreach (int i in ints)
     Console. Write("{0} ", i);
```



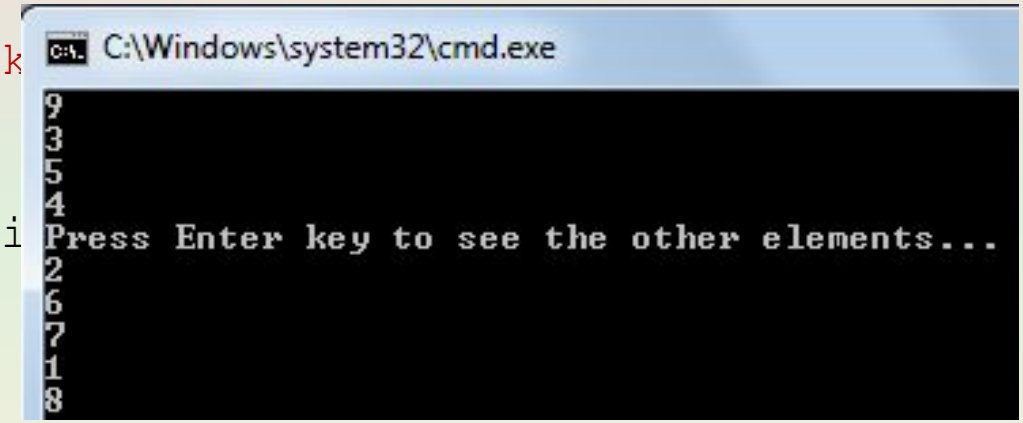
Сортування & Розбиття

Operator	Description
OrderBy	Сортування елементів, за одним або декількома ключами
OrderByDescending	Сортування в оберненому порядку
Reverse	Обертання елементів в послідовності
ThenBy	Використовується для сортування за додатковим ключем, після OrderBy або OrderByDescending
ThenByDescending	Подібно до ThenBy, але сортується послідовність обернено
Skip	Пропускає вказану кількість елементів і повертає всі решта
SkipWhile	Аналогічно до Skip, але будуть пропущені ті перші елементи, що задовільняють предикат
Take	Отримати вказану кількість елементів з вхідної послідовності
TakeWhile	Отримати ті перші елементи з вхідної послідовності , що задовільняють вказаний предикат

TakeWhile(), SkipWhile()

```
public static IEnumerable<T> TakeWhile<T>(
    this IEnumerable<T> source, Func<T, bool> predicate);

public static IEnumerable<T> TakeWhile<T>(
    this IEnumerable<T> source, Func<T, int, bool> predicate);
```

```
int[] numbers = { 9, 3, 5, 4, 2, 6, 7, 1, 8 };
var query = numbers.TakeWhile((n, index) => n >= index);
foreach (var i in query)
    Console.WriteLine(i);

Console.Write("Press Enter k
Console.ReadLine();

var query2 = numbers.SkipWhi
foreach (var i in query2)
    Console.WriteLine(i);
```



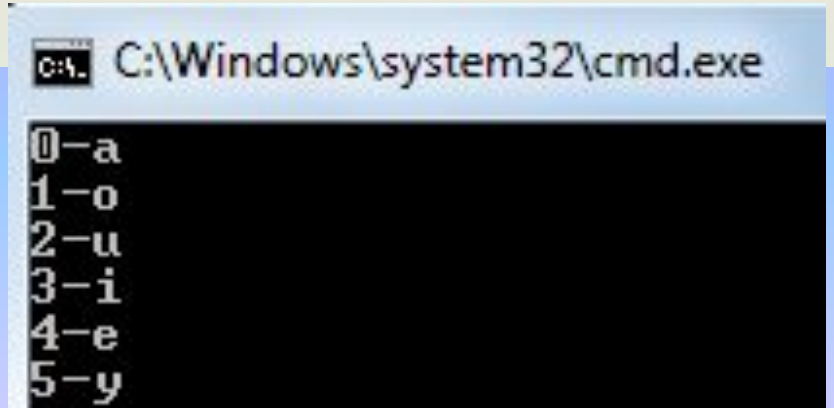
Проекція& Обмеження& Множини

Operator	Description
Select	Визначає елементи, що будуть вибиратись з послідовності
SelectMany	Утворює один-до-багатьох проекцію над послідовністю
All	Повертає true, якщо всі елементи послідовності задовільняють предикат
Any	Повертає true, якщо хоча б один елемент послідовності задовільняють предикат
Contains	Перевірка чи елемент є в послідовності
Where	Фільтрування послідовності згідно умови
Distinct	Повертає унікальні елементи послідовності
Except	Утворює послідовність – різницю двох послідовностей
Intersect	Утворює послідовність – перетин двох заданих
Union	Утворює послідовність, що є об’єднанням двох заданих

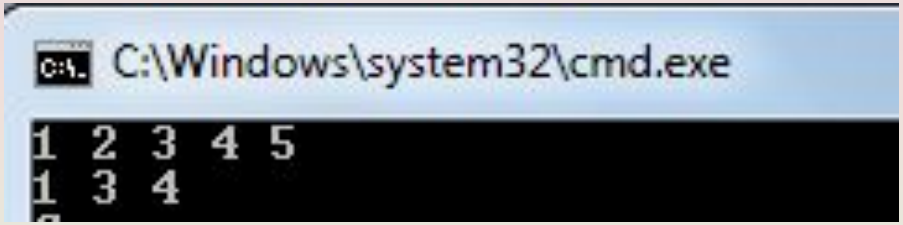
Select()

```
public static IEnumerable<S> Select<T, S>(
       this IEnumerable<T> source, Func<T, S> selector);
public static IEnumerable<S> Select<T, S>(
        this IEnumerable<T> source, Func<T, int, S> selector);
```

```
char[] letters = {'a','o','u','i','e','y' };

var query = letters.Select((l, i) =>
                                  new { index = i, letter = l });
foreach (var i in query)
            Console.WriteLine("{0}-{1}",i.index,i.letter);
```



Оператори над множинами

```
public static IEnumerable<T> Distinct<T>(
                                        this IEnumerable<T> source)
public static IEnumerable<T> Intersect<T>(
                this IEnumerable<T> first,IEnumerable<T> second)
```

```
int [ ] numbers = { 1,2,3,4,1,1,2,5};
var query = numbers.Distinct();
foreach (var i in query)
   Console.Write("{0} ", i);

Console.WriteLine();
int[ ] numbers2 = {1, 3, 4, 9 };
var query2 = numbers.Intersect(numbers2);
foreach (var i in query2)
   Console.Write("{0} ", i);
```