**UNIVERSITATEA DE STAT DIN MOLDOVA**

**FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ**

**DEPARTAMENTUL DE INFORMATICĂ**

**Operațiuni de generare, egalitate și unire în LINQ**

Lucrare individuală

Verificat: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **Vsevolod Arnăut, profesor universitar**

Executor: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **Țurcanu Cristian, student al grupei MIA2201**

CUPRINS

[1 Introducere 3](#_Toc135341075)

[2 Operațiuni de generare 4](#_Toc135341076)

[2.1 DefaultIfEmpty 4](#_Toc135341077)

[2.2 Empty 4](#_Toc135341078)

[2.3 Range 5](#_Toc135341079)

[2.4 Repeat 5](#_Toc135341080)

[3 Operațiune de egalitate 7](#_Toc135341081)

[3.1 SequenceEqual 7](#_Toc135341082)

[4 Operațiuni de unire 9](#_Toc135341083)

[4.1 Join 9](#_Toc135341084)

[4.2 GroupJoin 10](#_Toc135341085)

[5 Concluzii 13](#_Toc135341086)

[6 Bibliografie 14](#_Toc135341087)

# I Introducere

Operațiunile LINQ permit parcurgerea, filtrarea și convertirea colecțiilor în orice mod posibil. În cazul dat vor fi descrise operațiunile de generare, egalitate și unire. Primele au rolul de a crea ușor date noi printr-o sintaxă comodă, următoarele asigură egalitatea seriilor de obiecte, iar cele de unire combină elementele a două colecții diferite pentru a forma una nouă.

# II Operațiuni de generare

## 2.1 DefaultIfEmpty

Aceste metode returnează o nouă colecție cu un singur element în cazul în care colecția parametru nu are elemente, în caz contrar vor returna tot aceeași colecție. Supraîncărcările acestei metode permit specificarea obiectului ce va fi prezent în colecția nouă, precum valoarea default(TSource) sau o valoare specifică. Aceastea lucrează ca o asigurare adițională în cazul în care colecția în niciun caz nu trebuie să fie goală, precum în metodele Average și Max.

Aceste metode pot fi realizate în asemenea mod:

public static IEnumerable<TSource> DefaultIfEmpty<TSource>(this IEnumerable<TSource> source) {  
 return source.DefaultIfEmpty(default);  
}  
  
public static IEnumerable<TSource> DefaultIfEmpty<TSource>(this IEnumerable<TSource> source, TSource fallback) {  
 return source.Any() ? source : new List<TSource> { fallback };  
}

Exemplu:

public static void Main()  
{  
 Console.WriteLine(Array.Empty<int>().Average());  
 // Unhandled exception. System.InvalidOperationException: Sequence contains no elements  
}

public static void Main()  
 {  
 Console.WriteLine(Array.Empty<int>().DefaultIfEmpty().Average());  
 // 0  
 }

## 2.2 Empty

Această metodă returnează un IEnumerable gol de tipul generic dat de user, și este utilă în cazul în care trebuie imediat generată o colecție goală, care ulterior va fi folosită în cod.

Această metodă poate fi realizată în asemenea mod:

public static IEnumerable<TSource> Empty<TSource>() => new List<TSource>();

Această metodă nu are vreo valoare serioasă prin implementarea sa, și pur și simplu oferă o altă opțiune de a crea o colecție, fiind potențial folosită pentru curățirea codului. De asemenea, acesta poate fi folosit ca un parametru neutru pentru Union.

## 2.3 Range

Generează o secvență de numere întregi din diapazonul specificat. Primul parametru start este valoarea de start, iar al doilea parametru count este numărul elementelor, astfel metoda va eșua în cazul când count este mai mic ca zero sau start + count este mai mare sau egal ca Int32.MaxValue. Metoda este folositoare pentru inițializarea rapidă a seriilor, precum pentru testarea unor altor metode.

Exemplu de realizare:

public static IEnumerable<int> Range(int start, int count)  
{  
 var result = new List<int>();  
  
 for (int i = 0; i < count; i++)   
 result.Add(start + i);  
  
 return result;  
}

Exemplu de utilizare:

public static void Main()  
{  
 var numerePatrate = Enumerable.Range(1, 10).Select(x => x \* x);  
 // 1 4 9 16 25 36 49 64 81 100  
}

## 2.4 Repeat

Generează o colecție de count elemente duplicate ale element. Returnează eroare în cazul în care count este mai mic ca zero, în rest poate fi utilizat pentru a genera rapid colecții, spre exemplu pentru testarea unor altor metode.

Exemplu de realizare:

public static IEnumerable<TResult> Repeat<TResult>(TResult element, int count)  
{  
 var result = new List<TResult>();  
  
 for (int i = 0; i < count; i++)   
 result.Add(element);  
  
 return result;  
}

Exemplu de utilizare:

public static void Main()  
 {  
 var repeat = Enumerable.Repeat("C# Rocks!", 10);  
 // C# Rocks!  
 // C# Rocks!  
 // C# Rocks!  
 // C# Rocks!  
 // C# Rocks!  
 // C# Rocks!  
 // C# Rocks!  
 // C# Rocks!  
 // C# Rocks!  
 // C# Rocks!  
 }

# III Operațiune de egalitate

## 3.1 SequenceEqual

Această metodă determină dacă secvențele date sunt egale sau nu, folosind comparatorul inplicit sau dat ca parametru. Desigur, această metodă returnează o eroare în cazul în care vreun parametru este null.

Exemple de realizare:

public static bool SequenceEqual<TSource>(this IEnumerable<TSource> first, IEnumerable<TSource> second, IEqualityComparer<TSource> comparer)  
{  
 if (first.Count() != second.Count())  
 return false;  
  
 var firstEnumerator = first.GetEnumerator();  
 var secondEnumerator = second.GetEnumerator();  
  
 for (var i = 0; i < first.Count(); i++)  
 if (!comparer.Equals(firstEnumerator.Current, secondEnumerator.Current))  
 return false;  
  
 return true;  
}  
  
public static bool SequenceEqual<TSource>(this IEnumerable<TSource> first, IEnumerable<TSource> second)  
{  
 if (first.Count() != second.Count())  
 return false;  
  
 var firstEnumerator = first.GetEnumerator();  
 var secondEnumerator = second.GetEnumerator();  
  
 for (var i = 0; i < first.Count(); i++)  
 if (!firstEnumerator.Current.Equals(secondEnumerator.Current))  
 return false;  
  
 return true;  
}

Exemplu de utilizare:

public static void Main()  
{  
 var first = new List<int> { 1, 25, 49, 81, 4, 16, 9, 36, 64, 100 };  
 first.Sort();  
  
 var second = Enumerable.Range(1, 10).Select(x => x \* x);  
   
 Console.WriteLine(first.SequenceEqual(second));  
 // True  
}

# IV Operațiuni de unire

## 4.1 Join

Unirile deja sunt operațiuni tipice bazelor de date. Ele ne permit asocierea elementelor după un anumit parametru, numit cheie. Fundamental, un Join este pur și simplu un for în for care compară fiecare pereche de elemente. În cazul nostru, cheile sunt obținute în baza unor funcții de convertire transmise ca parametru, iar egalitatea lor este asigurată prin comparatorul, care poate fi obținut implicit sau poate fi transmis la chemare prin supraîncărcare. Elementele unite de asemenea pot fi convertite cu ajutorul unei funcții, spre exemplu restructurate într-un obiect nou.

Exemple de realizare:

public static IEnumerable<TResult> Join<TOuter, TInner, TKey, TResult>(  
 this IEnumerable<TOuter> outer,  
 IEnumerable<TInner> inner,  
 Func<TOuter, TKey> outerKeySelector,  
 Func<TInner, TKey> innerKeySelector,  
 Func<TOuter,TInner,TResult> resultSelector)  
 {  
 var result = new List<TResult>();  
   
 foreach (var outerValue in outer)  
 foreach (var innerValue in inner)  
 {  
 var outerKey = outerKeySelector.Invoke(outerValue);  
 var innerKey = innerKeySelector.Invoke(innerValue);  
  
 if (outerKey.Equals(innerKey))  
 result.Add(resultSelector.Invoke(outerValue, innerValue));  
 }  
  
 return result;  
 }  
   
 public static IEnumerable<TResult> Join<TOuter, TInner, TKey, TResult>(  
 this IEnumerable<TOuter> outer,  
 IEnumerable<TInner> inner,  
 Func<TOuter, TKey> outerKeySelector,  
 Func<TInner, TKey> innerKeySelector,  
 Func<TOuter,TInner,TResult> resultSelector,  
 IEqualityComparer<TKey> comparer)  
 {  
 var result = new List<TResult>();  
   
 foreach (var outerValue in outer)  
 foreach (var innerValue in inner)  
 {  
 var outerKey = outerKeySelector.Invoke(outerValue);  
 var innerKey = innerKeySelector.Invoke(innerValue);  
  
 if (comparer.Equals(outerKey, innerKey))  
 result.Add(resultSelector.Invoke(outerValue, innerValue));  
 }  
  
 return result;  
 }

Exemplu de utilizare:

public static void Main()  
{  
 var first = new List<int> { 1, 25, 49, 81, 4, 16, 9, 36, 64, 100, 1 };  
 first.Sort();  
  
 var second = Enumerable.Range(1, 10).Select(x => x \* x);  
  
 var result = first.Join(  
 second,   
 i => i,   
 j => j,   
 (i, j) => (i, j));  
  
 foreach (var num in result)  
 Console.Write($"{num}, ");  
 // (1, 1), (1, 1), (4, 4), (9, 9), (16, 16), (25, 25), (36, 36), (49, 49), (64, 64), (81, 81), (100, 100),  
}

## 4.2 GroupJoin

Această metodă se folosește analog Join-ului, cu diferența esențială în faptul că din a doua colecție primim nu un singur element, ci toate elementele ce au cheia egală cu cheia corespunzătoare din prima colecție, fiind folosită în cadrul relațiilor one-to-many. Această operație nu are un analog direct în SQL, însă poate fi convertită prin mai multe inner join-uri și left outer join-uri.

Exemplu de realizare:

public static IEnumerable<TResult> GroupJoin<TOuter, TInner, TKey, TResult>(  
 this IEnumerable<TOuter> outer,  
 IEnumerable<TInner> inner,  
 Func<TOuter, TKey> outerKeySelector,  
 Func<TInner, TKey> innerKeySelector,  
 Func<TOuter, IEnumerable<TInner>, TResult> resultSelector)  
{  
 var result = new List<TResult>();  
  
 foreach (var outerValue in outer)  
 {  
 var innerResult = new List<TInner>();  
 foreach (var innerValue in inner)  
 {  
 var outerKey = outerKeySelector.Invoke(outerValue);  
 var innerKey = innerKeySelector.Invoke(innerValue);  
  
 if (outerKey.Equals(innerKey))  
 innerResult.Add(innerValue);  
 }  
  
 result.Add(resultSelector.Invoke(outerValue, innerResult));  
 }  
  
 return result;  
}  
  
public static IEnumerable<TResult> GroupJoin<TOuter, TInner, TKey, TResult>(  
 this IEnumerable<TOuter> outer,  
 IEnumerable<TInner> inner,  
 Func<TOuter, TKey> outerKeySelector,  
 Func<TInner, TKey> innerKeySelector,  
 Func<TOuter, IEnumerable<TInner>, TResult> resultSelector,  
 IEqualityComparer<TKey> comparer)  
{  
 var result = new List<TResult>();  
  
 foreach (var outerValue in outer)  
 {  
 var innerResult = new List<TInner>();  
  
 foreach (var innerValue in inner)  
 {  
 var outerKey = outerKeySelector.Invoke(outerValue);  
 var innerKey = innerKeySelector.Invoke(innerValue);  
  
 if (comparer.Equals(outerKey, innerKey))  
 innerResult.Add(innerValue);  
 }  
   
 result.Add(resultSelector.Invoke(outerValue, innerResult));  
 }  
  
 return result;  
}

Exemplu de utilizare:

public record Obiect(int id, string name, int professorId);  
public record Professor(int id, string name, int age);  
  
public static void Main()  
{  
 Professor popescu = new(id: 1, name: "Popescu", age: 40);  
 Professor ionescu = new(id: 2, name: "Ionescu", age: 50);  
 Professor georgescu = new(id: 3, name: "Georgescu", age: 60);  
  
 var professors = new List<Professor> { popescu, ionescu, georgescu };  
  
 Obiect html = new(id: 1, name: "HTML", professorId: 1);  
 Obiect css = new(id: 2, name: "CSS", professorId: 1);  
 Obiect js = new(id: 3, name: "JS", professorId: 1);  
 Obiect csharp = new(id: 4, name: "C#", professorId: 2);  
 Obiect java = new(id: 5, name: "Java", professorId: 2);  
 Obiect python = new(id: 6, name: "Python", professorId: 3);  
 Obiect machineLearning = new(id: 7, name: "Machine Learning", professorId: 3);  
  
 var obiecte = new List<Obiect> { html, css, js, csharp, java, python, machineLearning };  
  
 var result = professors.GroupJoin(  
 obiecte,  
 professor => professor.id,  
 obiect => obiect.professorId,  
 (professor, obiecte) => (professor, obiecte));  
  
 foreach (var (professor, localObiecte) in result)  
 {  
 Console.WriteLine($"{professor.name}: ");  
 foreach (var obiect in localObiecte)  
 Console.WriteLine($" {obiect.name}");  
 }  
  
 // Popescu:  
 // HTML  
 // CSS  
 // JS  
 // Ionescu:  
 // C#  
 // Java  
 // Georgescu:  
 // Python  
 // Machine Learning  
  
}

# V Concluzii

Operațiunile de generare sunt extrem de comode pentru generarea rapidă a colecțiilor, fie pentru a testa o metodă, completa o nouă bază de date cu informații, sau pentru a începe o serie de operații LINQ, folosind o sintaxă comodă, elegantă, și respectând stilul funcțional. Operațiunile de egalitate permit verificarea egalității elementelor unor serii, fiind folositoare în cadrul unor situații foarte specifice. Operațiunile de unire imitează funcționalitatea unirilor din SQL, însă folosind sintaxa funcțională în loc de cea asemănătoare limbajului natural, micșorând astfel riscul de a face vreo eroare.

# VI Bibliografie

* Microsoft Learn - Generation Operations - https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/generation-operations
* Microsoft Learn - Equality Operations - https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/equality-operations
* Microsoft Learn - Join Operations - https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/join-operations