TDA-LABORATOR 3

Limbajul C#; Language Integrated Queries (LINQ)

Limbajul LINQ ca sistem ORM pe platforma .NET. ORM (Object-Relational Mapping) este o tehnică de programare pentru convertirea datelor stocate în doua medii diferite: sistemele de baze de date relationale(MySQL, Microsoft SQL Server, etc) şi limbajele de programare orientate obiect (Java, C++, C#, etc). Thenica constă în crearea unei baze de date virtuale formată din obiecte pentru a putea fi folosită în cadrul limbajului de programare.

LINQ (Language Integrated Query) este o componenta a platformei Microsoft .NET care adauga suport pentru interogari de date direct din sintaxa limbajelor .NET. Această componentă defineşte un set de operatori de interogare care pot fi folositi pentru chestionarea, proiectarea si filtrarea datelor din vectori, clase numarabile, fisiere XML, baze de date relaţionale şi alte surse de date, cu condiţia ca informaţia sa fie încapsulată în obiecte. Aşadar, daca sursa de date nu stochează datele sub formă de obiecte, este necesar ca să existe o corelare între aceasta şi modelul obiectual. Interogarile scrise folosind operatorii de interogare sunt fie executate de motorul de procesare LINQ fie, prin intermediul unor extensii, transmise LINQ providerilor care fie implementează un motor de interogare separat fie traduc interogarea într-un format diferit pentru a fi executat pe un suport de date diferit (cum ar fi un server de date SQL). Rezultatul unei interogari este o colectie de obiecte stocate în memoria programului care pot fi enumerate folosind o funcţie de parcurgere standard cum ar fi funcţia foreach din C#.

Interogari LINQ. Sunt trei stiluri de a scrie interogari **LINQ**, si trebuie sa le folosim pe toate pentru a putea utiliza la capacitate maxima operatorii de interogare pusi la indemana :

1. Formatul Expresiilor de Interogare (Query)

```
int[] nums = new int[] {0,4,2,6,3,8,3,1};

var result = from n in nums
        where n < 5
        orderby n
        select n;
```

2. Formatul Metodei Extinse/Extension Method (Dot Notation)

```
int[] nums = new int[] {0,4,2,6,3,8,3,1};

var result = nums
        .Where(n => n < 5)
        .OrderBy (n => n);
```

Sau cand este scrisa pe o singura linie pentru arata mai bine posibilitatile Metodei Extinse

```
var result = nums.Where(n => n < 5).OrderBy (n => n);
```

3. O combinatie a acestor doua formate (Query Dot syntax)
Sintaxa unei astfel de expresii de interogare care returneaza valori IEnumerabile poate fi urmata de o insiruire de Metode Extinse.

```
int[] nums = new int[] {0,4,2,6,3,8,3,1};

var result = (from n in nums
        where n < 5
        orderby n
        select n).Distinct();
```

Sintaxa Expresiei de Interogare a fost conceputa sa faca scrierea interogarilor mai atractiva si chiar mai asemanatoare celor din SQL. Compilatorul C# converteste toate interogarile pentru a folosi Metodele Extinse astfel incat Expresiile de Interogare sa poata face munca utilizatorului mai usoara acestea avand totodata o sintaxa simpla la folosirea unificarilor si gruparilor.

Operatori Standard de Interogare. Operatorii Standard de Interogare formeaza coloana vertebrala a functiilor de interogare ca parte a LINQ in Obiecte . In general sunt cunoscuti de la limbajul SQL.

Operator Type	Operator	Description
Agregare	Aggregate	Operatorul Aggregate aplica o functie peste o secventa.

	Average	Operatorul Average calculeaza media unei secvente de valori numerice.
	Count	Operatorul Count contorizeaza numarul de elemente dintr-o secventa, tipul returnat fiind Int.

	Count	Operatorul Count contorizeaza numarul de elemente dintr-o secventa, tipul returnat fiind Int.
	LongCount	Operatorul LongCount contorizeaza numarul de elemente dintr-o secventa, tipul returnat fiind Long.
	Max	Operatorul Max afla maximul unei secvente de valori numerice.
	Min	Operatorul Min afla minimul unei secvente de valori numerice.
	Sum	Operatorul Sum calculeaza suma unei secvente de valori numerice.
Concatenare	Concat	Operatorul Concat concateneaza doua secvente.
Conversie	Cast	Operatorul Cast converteste elementele unei secvente la un tip dat.
	OfType	Operatorul OfType filtreaza elementele unei secvente bazate pe un tip.
	ToArray	Operatorul ToArray creeaza un array dintr-o secventa.
	ToDictionary	Operatorul ToDictionary creeaza un Dictionar dintr-o secventa.
	ToList	Operatorul ToList creeaza o lista dintr-o secventa.
	ToLookup	Operatorul ToLookup creeaza un Lookup dintr-o secventa.

	ToSequence	Operatorul ToSequence returneaza argumentul transmis ca IEnumerabil.
Element	DefaultIfEmpty	Operatorul DefaultIfEmpty asigura un element standard pentru o secventa vida.
	ElementAt	Operatorul ElementAt returneaza elementul de la un anumit index din cadrul unei secvente.
	ElementAtOrDefault	Operatorul ElementAtOrDefault returneaza elementul de la un anumit index din cadrul unei secvente sau o valoare default daca indexul este in afara.
	First	Operatorul First returneaza primul element dintr-o secventa.
	FirstOrDefault	Operatorul FirstOrDefault returneaza primul element dintr-o secventa sau o valoare default daca nu este gasit nici un element.
	Last	Operatorul Last returneaza ultimul element dintr-o secventa.
	LastOrDefault	Operatorul LastOrDefault returneaza ultimul element dintr-o secventa sau o valoare default daca nu este gasit nici un element.
	Single	Operatorul Single returneaza singurul element dintr-o secventa
	SingleOrDefault	Operatorul SingleOrDefault returneaza singurul element dintr-o secventa sau o valoare default daca nu este gasit nici un element.
Egalitate	SequenceEqual	Operatorul SequenceEqual verifica daca doua secvente sunt egale.
Generare	Empty	Operatorul Empty returneaza o secventa vida a unui tip dat.
	Range	Operatorul Range genereaza o secventa de numere intregi.

	Repeat	Operatorul Repeat genereaza o secventa prin repetarea unei valori de un numar de ori dat .
Grupare	GroupBy	Operatorul GroupBy grupeaza elementele unei secvente.
Unificare	GroupJoin	Operatorul GroupJoin executa o unificare grupata a doua secvente bazate pe chei potrivite extrase din cadrul elementelor.
	Join	Operatorul Join executa o unificare interioara a doua secvente bazate pe chei potrivite extrase din cadrul elementelor.
Ordonare	OrderBy	Operatorul OrderBy ordoneaza o secventa conform unei sau mai multor chei in ordine crescatoare.
	ThenBy	Operatorul ThenBy ordoneaza o secventa ordonata conform unei sau mai multor chei in ordine crescatoare.
	OrderBy	Operatorul OrderBy ordoneaza o secventa conform unei sau mai multor chei in ordine crescatoare.
	ThenByDescending	Operatorul ThenBy ordoneaza o secventa conform unei sau mai multor chei in ordine descrescatoare.
	Reverse	OperatorulReverse inverseaza elementele unei secvente.
Partitionare	Skip	Operatorul Skip sare un numar dat de elemente dintr-o secventa si apoi cedeaza restul secventei.
	SkipWhile	Operatorul SkipWhile sare elementele dintr-o secventa cat timp testul este adevarat di poi cedeaza restul secventei.
	Take	Operatorul Take cedeaza un numar dat de elemente dintr-o secventa si apoi sare restul secventei.
	TakeWhile	Operatorul TakeWhile cedeaza elemntele dintr-o secventa cat timp testul este adevarat si apoi sare restul secventei.

Cuantificatori	All	Operatorul All verifica daca toate elementele unei secvente satisfac o conditie.
	Any	Operatorul Any verifica daca vreun element al secventei satisface o conditie.
	Contains	Operatorul Contains verifica daca o secventa contine un elemnt dat.
Restrictie	Where	Operatorul Where filtreaza o secventa bazandu-se pe un predicat.
Proiectie	Select	Operatorul Select executa o proiectie peste o secventa.
	SelectMany	Operatorul SelectMany executa o proiectie unul la mai multe elemente la nivelul unei secvente.
Multime	Distinct	Operatorul Distinct elimina elementele duplicate din cadrul unei secvente.
	Except	Operatorul Except produce diferenta dintre doua secvente.
	Intersect	Operatorul Intersect produce intersectia dintre doua secvente.the set intersection of two sequences.
	Union	Operatorul Union produce reuniunea a doua secvente.

Variantele limbajului LINQ. LINQ se prezinta în diferite variante, desi sintaxa de utilizare a lui este aceiasi in toate variantele.

1. **LINQ to Objects** – este numele dat API-ului IEnumerable<T> pentru operatorii de cereri standard. LINQ la obiecte vă permite să interogați obiecte C # care sunt rezidente în memorie. Putem astfel face interogari asupra tablourilor sau colectiilor din memorie. Operatorii standard sunt metode statice ale clasei System.Linq.Enumerable.
2. **LINQ to XML** – API dedicat sa lucreze cu XML. Trebuie sa adaugam o referinta la proiect System.Xml.Linq.dll si apoi using System.Xml.Linq. **LINQ to XML**, este un mod foarte convenabil și puternic de a crea, prelucra și interoga conținut XML.

3. **LINQ to DataSet** – API Linq pentru DataSet-uri.
4. **LINQ to SQL** – API IQuerable<T> ce permite Linq sa lucreze cu SQL Server. Trebuie referinta la System.Data.Linq.dll si apoi using System.data.Linq ;

5. **LINQ to Entities** – este varianta Linq folosit pentru interfata cu bazele de date. Se decupleaza entitatea model obiect de baza de date (fizic) prin crearea unei logici intre cele doua niveluri.

Asa cum reese din nume variant **LINQ to Entities**, permite efectuarea de întrerogari LINQ pe datele obținute din Entity Framework. Entity Framework este Microsoft's ORM framework, care face parte din platforma mai largă ADO.NET. Un ORM vă permite să lucrați cu date relaționale folosind obiecte C# și este mecanismul pe care îl vom folosi în laboratoarele urmatoare pentru a accesa datele stocate în bazele de date.

Interfața IQueryable <T> este derivată din IEnumerable <T> și este utilizată pentru a specifica rezultatul unei interogări executate pe o sursă de date specifică. Nu este necesară utilizarea IQueryable <T> direct. Una dintre caracteristicile remarcabile ale lui LINQ este că aceeași interogare poate fi efectuată pe mai multe tipuri de surse de date (obiecte, XML, baze de date etc.).

LINQ paralel, cunoscut sub numele de PLINQ, este o implementare paralelă a variantei **LINQ to Objects** care permite programatorului să efectueze interogări LINQ în paralel, astfel încât mai multe elemente de date să fie procesate simultan. Paralel LINQ este un superset de **LINQ to Objects** care acceptă executarea de interogări LINQ sincronizate pe mai multe procesoare sau nuclee.

Toate funcțiile descrise mai sus pot fi folosite în interogarile LINQ.

LINQ este facilitate importanta si convingătoare adaugata la .NET. LINQ are o sintaxă asemănătoare cu SQL pentru interogarea datelor din clase.

Sa presupunem că avem o colecție de obiecte de tip Product și dorim să găsim și să afișăm cele mai mari trei prețuri. Fără LINQ, am ajunge la ceva similar cu Exemplul 1.

Exemplul 1. Querying without LINQ

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Text;
namespace LanguageFeatures {
        public partial class Default : System.Web.UI.Page {
                protected void Page_Load(object sender, EventArgs e) {
                }
                protected string GetMessage() {
                        Product[] products = {
                                new Product {Name = "Kayak", Category = "Watersports", Price
                                = 275M},
```

```
                    new Product {Name = "Lifejacket", Category = "Watersports",
                    Price = 48.95M},
                    new Product {Name = "Soccer ball", Category = "Soccer", Price =
                    19.50M},
                    new Product {Name = "Corner flag", Category = "Soccer", Price =
                    34.95M}
                };

                Product[] foundProducts = new Product[3];
                Array.Sort(products, (item1, item2) => {
                return Comparer<decimal>.Default.Compare(item1.Price, item2.Price);
                });
                Array.Copy(products, foundProducts, 3);
                StringBuilder result = new StringBuilder();
                foreach (Product p in foundProducts) {
                result.AppendFormat("Price: {0} ", p.Price);
                }
                return result.ToString();
        }
    }
}
```

Cu LINQ, putem simplifica în mod semnificativ procesul de interogare, așa cum s-a demonstrat în Exemplul 2.

Exemplul 2. Using LINQ to query data

```
...
protected string GetMessage() {
    Product[] products = {
        new Product {Name = "Kayak", Category = "Watersports", Price = 275M},
        new Product {Name = "Lifejacket", Category = "Watersports", Price = 48.95M},
        new Product {Name = "Soccer ball", Category = "Soccer", Price = 19.50M},
        new Product {Name = "Corner flag", Category = "Soccer", Price = 34.95M}
    };
    var foundProducts = from match in products
                        orderby match.Price descending
                        select match.Price;
    int count = 0;
    StringBuilder result = new StringBuilder();
        foreach (var price in foundProducts) {
        result.AppendFormat("Price: {0} ", price);
        if (++count == 3) {
        break;
        }
    }
```

```
        return result.ToString();
}
...
```

Folosirea LINQ este mult mai bună. Puteți vedea interogarea asemănătoare cu SQL. Ordonăm obiectele Produs în ordine descrescătoare și folosim cuvântul cheie select pentru a returna doar valorile proprietății Preț. Acest stil de LINQ este cunoscut sub numele de sintaxa de interogare (query syntax) și este cel pe care dezvoltatorii îl găsesc cel mai confortabil atunci când încep să folosească LINQ.

Alternativa la aceasta sintaxa este sintaxa dot-notation syntax, sau dot notation, care se bazează pe metode de extensie. Exemplul 3 arată cum putem folosi această sintaxă alternativă pentru a procesa obiectele de tip Product.

Exemplul 3. Using LINQ dot notation

```
...
protected string GetMessage() {
        Product[] products = {
                new Product {Name = "Kayak", Category = "Watersports", Price = 275M},
                new Product {Name = "Lifejacket", Category = "Watersports", Price = 48.95M},
                new Product {Name = "Soccer ball", Category = "Soccer", Price = 19.50M},
                new Product {Name = "Corner flag", Category = "Soccer", Price = 34.95M}
        };
        var foundProducts = products.OrderByDescending(e => e.Price)
                                    .Take(3)
                                    .Select(e => e.Price);
        StringBuilder result = new StringBuilder();
        foreach (var price in foundProducts) {
                result.AppendFormat("Price: {0} ", price);
        }
        return result.ToString();
}
...
```

Desi această interogare LINQ, nu este la fel de plăcută ochiului ca cea exprimată prin sintaxa de interogare, nu toate funcțiile LINQ au cuvinte cheie in sintaxa de interogare si de aceia pentru programarea LINQ serioasă, trebuie să trecem la metode de extensie. Fiecare dintre metodele de extensie LINQ din Exemplul 3 este aplicată unui IEnumerable <T> și returnează tot un IEnumerable <T>, ceea ce ne permite să îmbricam metodele pentru a forma interogări complexe.

O mulțime de metode de extensie LINQ se află în spațiul de nume System.Linq, pe care trebuie introdus în aplicatie cu o declarație **using** înainte de a putea face interogări. Visual Studio adaugă automat spațiul de nume la clasele din spatele codurilor și le face disponibile pentru utilizare în aplicatie.

Metoda OrderByDescending rearanjează elementele din sursa de date. În acest caz, expresia lambda returnează valoarea pe care dorim să o folosim pentru comparații. Metoda Tak returnează un număr specificat de elemente din partea anterioară a rezultatelor (asta nu am putea face cu ajutorul sintaxei de interogare). Metoda Select ne permite să ne proiectăm rezultatele, specificând rezultatul dorit. În acest caz, proiectăm proprietățile Price.

Toate metodele LINQ prezentate mai sus funcționează pe IEnumerable <T>.

Există o variație interesantă în modul în care metodele de extensie sunt executate într-o interogare LINQ. O interogare care conține doar metode cu executare intarziata nu este executată până când sunt enumerate elementele din rezultat, așa cum se arată în Exemplul 5.

Exemplul 5. Using deferred LINQ extension methods in a auery

```
...
protected string GetMessage() {
        Product[] products = {
                new Product {Name = "Kayak", Category = "Watersports", Price = 275M},
                new Product {Name = "Lifejacket", Category = "Watersports", Price = 48.95M},
                new Product {Name = "Soccer ball", Category = "Soccer", Price = 19.50M},
                new Product {Name = "Corner flag", Category = "Soccer", Price = 34.95M}
        };
        var foundProducts = products.OrderByDescending(e => e.Price)
                                    .Take(3)
                                    .Select(e => e.Price);
        products[2] = new Product { Name = "Stadium", Price = 79600M };
        StringBuilder result = new StringBuilder();
        foreach (var price in foundProducts) {
                result.AppendFormat("Price: {0} ", price);
        }
        return result.ToString();
}
...
```

Între definirea interogării LINQ și enumerarea rezultatelor, am schimbat unul dintre elementele din gama de produse. Rezultatul din acest exemplu este următorul:

Price: 79600 Price: 275 Price: 48.95

Puteți vedea că interogarea nu este evaluată până la enumerarea rezultatelor și astfel modificările pe care le-am făcut - introducerea Stadium-ului în tabloul **products** - se reflectă în

rezultat. În schimb, folosirea oricăreia dintre metodele de extensie neamanate face ca o interogare LINQ să fie efectuată imediat.

Pentru a demonstra acest lucru, am folosit metoda de extindere Sum în interogarea noastră, așa cum se arată în Exemplul 6.

Exemplul 6. An immediately executed LINQ query

```
...
protected string GetMessage() {
        Product[] products = {
                new Product {Name = "Kayak", Category = "Watersports", Price = 275M},
                new Product {Name = "Lifejacket", Category = "Watersports", Price = 48.95M},
                new Product {Name = "Soccer ball", Category = "Soccer", Price = 19.50M},
                new Product {Name = "Corner flag", Category = "Soccer", Price = 34.95M}
        };
        var totalPrice = products.OrderByDescending(e => e.Price)
                                 .Take(3)
                                 .Select(e => e.Price)
                                 .Sum(e => e);
        products[2] = new Product { Name = "Stadium", Price = 79600M };
        return String.Format("Total: {0}", totalPrice.ToString());
}
...
```

Acest exemplu utilizează metoda Sum, care nu este amânată și produce următorul rezultat:

Total: 358.90

Puteți vedea că articolul Stadium, cu prețul său foarte mare, nu a fost inclus în total - aceasta se datorează faptului că rezultatele din metoda Sum sunt evaluate imediat după apelarea metodei, si nu sunt amânate până la utilizarea rezultatelor.

AICI

PLINQ este o implementare paralelă a limbajului LINQ to objects care permite programatorului să efectueze interogări LINQ în paralel, astfel încât mai multe elemente de date să fie procesate simultan. PLINQ funcționează cu aceleași surse de date pe care le utilizează LINQ pentru obiecte, și anume IEnumerable < > și IEnumerable.

Exemplul 7. Parallel Language Integrated Query

```
namespace Exemplul_7 {

   class Listing_01 {

      static void myMain() {

         int[] sourceData = new int[100];
```

```
        for (int i = 0; i < sourceData.Length; i++) {
            sourceData[i] = i;
        }
        IEnumerable<int> results =
            from item in sourceData.AsParallel()
            where item % 2 == 0
            select item;
    }
}
}
```

C# conține câteva cuvinte cheie specifice LINQ-ului, cum ar fi:

from, **where** si **select**.

Aceste cuvinte cheie sunt mapate la metodele de extensie din clasa Enumerable, astfel că o interogare cum ar fi următoarea:

```
IEnumerable<int> result =
        from item in sourceData
        where (item % 2 == 0)
        select item;
```

este mapat la metodele de extensie corespunzatoare:

```
IEnumerable<int> result =
sourceData
        .Where(item => item % 2 == 0)
    .Select(item => item)
```

Folosirea metodelor de extensie în locul cuvintelor cheie poate fi utilă, deoarece putem utiliza expresii lambda complexe.

Metodele de extensie sunt adesea denumite operatori de interogare standard și se folosesc aplicând notația cu punct.

PLINQ functioneaza pe două clase publice: **ParallelEnumerable-ParallelQuery.**

ParallelEnumerable - conține metode de extensie care funcționează pe tipul **ParallelQuery**.

ParallelEnumerable contine metode paralele corespunzatoare celor din Enumerable, plus altele care permit constructii specifice ParallelQuery.

AsParallel(), este cheia utilizării PLINQ. Conversia unui IEnumerable intr-un ParallelQuery. Pentru a utiliza PLINQ, trebuie să creăm o instanță ParallelQuery apelând metoda AsParallel () pe o instanță a IEnumerable și folosindu-o ca bază pentru aplicarea caracteristicilor LINQ.

AsSequential() este opusa lui AsParallel() și ne permite să aplicăm selectiv paralelism într-o interogare complexă. Conversia unui ParallelQuery intr-un IEnumerable.

Urmatoarele metode configurează instanțele ParallelQuery produse de AsParallel() pentru a schimba modul în care o interogare este executată sau se comportă.

AsOrdered() modifica un ParallelQuery pentru a păstra ordinea de prelucrare a articolelor. **AsUnordered()** modifica un ParallelQuery pentru a renunța la ordinea de prelucrare a articolelor.

WithCancellation() modifica ParallelQuery pentru a monitoriza un token de anulare.

WithDegreeOfParallelism() modificați un ParallelQuery pentru a seta o limită superioară a numărului de taskuri utilizate pentru executarea unei interogări.

Creaza o instanță ParallelQuery pe o instanță a IEnumerable, ca bază pentru aplicarea caracteristicilor LINQ

Exemplul 8. AsParallel()

```
int[] sourceData = new int[10];
 for (int i = 0; i < sourceData.Length; i++) {
    sourceData[i] = i;
 }
// define a sequential linq query
 IEnumerable<double> results1 =
    from item in sourceData
    select Math.Pow(item, 2);
// define a parallel linq query
 IEnumerable<double> results2 =
    from item in sourceData.AsParallel()
```

```
    select Math.Pow(item, 2);
```

Exemplul 9. Filtrarea Datelor

```
// create some source data
    int[] sourceData = new int[100000];
    for (int i = 0; i < sourceData.Length; i++) {
      sourceData[i] = i;
    }
    // define a filtering query using keywords
    IEnumerable<double> results1
      = from item in sourceData.AsParallel()
        where item % 2 == 0
        select Math.Pow(item, 2);
    // define a filtering query using extension methods
    IEnumerable<double> results2
      = sourceData.AsParallel()
      .Where(item => item % 2 == 0)
      .Select(item => Math.Pow(item, 2));
```

Rezultatele obținute de interogarea paralelă nu sunt în aceeași ordine cu cele ale interogării secvențiale.

PLINQ partitioneaza datele sursă pentru a îmbunătăți eficiența taskurilor și acest lucru distruge ordinea naturală a elementelor.

Putem păstra ordinea într-o interogare PLINQ utilizând metoda de extensie AsOrdered(), care modifică instanța ParallelQuery.

Acest lucru reduce performanta. De aceia ar trebui să utilizam metoda AsOrdered() numai dacă ordinea rezultatelor în raport cu datele sursă este importantă.

Exemplul 10. AsOrdered()

```
// create some source data
```

```
int[] sourceData = new int[10];
for (int i = 0; i < sourceData.Length; i++) {
   sourceData[i] = i;
}
// preserve order with the AsOrdered() method
IEnumerable<double> results =
   from item in sourceData.AsParallel().AsOrdered()
   select Math.Pow(item, 2);
```

Putem controla ordonarea, combinând metodele de extensie

AsOrdered() și **AsUnordered()**.

Metoda AsUnordered() este exact opusa lui AsOrdered() și deci, spune lui PLINQ că ordinea nu trebuie păstrată.

Metoda **Take(n)** ia primele n elemente din sursa de date.

În exemplu, primele zece elemente sunt preluate din sursa de date (Take(10)) după ce este apelată metoda AsParallel(). Pentru această parte a interogării, elementele sunt ordonate.

Elementele luate sunt apoi utilizate ca bază pentru un apel la Select(), care poate fi efectuat fără a pastra ordinea, așa că apelam metoda AsUnordered().

Exemplul 11. AsUnordered()

```
// create some source data
int[] sourceData = new int[10000];
for (int i = 0; i < sourceData.Length; i++) {
   sourceData[i] = i;
}
// define a query that has an ordered subquery
var result =
   sourceData.AsParallel().AsOrdered()
   .Take(10).AsUnordered()
   .Select(item => new {
```

```
            sourceValue = item,
            resultValue = Math.Pow(item, 2)
        });
```

Puteți cere ca o interogare să fie efectuată paralel utilizând metoda de extensie WithExecutionMode (), care ia o valoare din enumerarea ParallelExecutionMode ca argument.

Valorile enumerării sunt:

Default - PLINQ va decide dacă executia interogari va fi secvențiala sau paralela.

ForceParallelism - Interogarea va fi paralelizată, chiar dacă executia paralelă va fi mai costisitoare decat exacutia secventiala.

Metoda WithExecutionMode() modifică o instanță ParallelQuery;

Exemplul12. WithExecutionMode()

```
// create some source data
    int[] sourceData = new int[10];
    for (int i = 0; i < sourceData.Length; i++) {
       sourceData[i] = i;
    }
    // define the query and force parallelism
    IEnumerable<double> results =
       sourceData.AsParallel()
     .WithExecutionMode(ParallelExecutionMode.ForceParallelism)
       .Where(item => item % 2 == 0)
       .Select(item => Math.Pow(item, 2));
```

Puteți solicita o limită superioară a numărului de Taskuri care vor fi utilizate pentru a efectua o interogare LINQ utilizând metoda extensiei WithDegreeOfParallelism().

Motorul PLINQ poate alege să utilizeze mai puține sarcini decât sunt specificate prin această metodă dar nu mai multe.

Exemplul urmator demonstrează utilizarea acestei metode pentru a specifica maxim două sarcini.

Exemplul13. WithDegreeOfParallelism()

```
// create some source data
    int[] sourceData = new int[10];
    for (int i = 0; i < sourceData.Length; i++) {
       sourceData[i] = i;
    }
    // define the query and force parallelism
    IEnumerable<double> results =
       sourceData.AsParallel()
       .WithDegreeOfParallelism(2)
       .Where(item => item % 2 == 0)
       .Select(item => Math.Pow(item, 2));
```

Metoda de extensie **AsSequential()** transforma ParallelQuery intr-o sarcina secventiala IEnumerable. Această metodă este opusa metodei AsParallel().

Putem utiliza AsParallel() alternativ cu AsSequential() pentru a activa și dezactiva paralelismul în subinterogari.

Această tehnică poate fi utilă dacă dorim să evitam în mod explicit costurile paralelismului pentru o parte a unei interogări.

Exemplul urmator demonstrează utilizarea acestei metode.

Prima parte a interogării păstrează valorile sursă în paralel, iar a doua parte a interogării, unde valorile pătratului sunt dublate, se efectuează secvențial.

Exemplul 14. AsSequential()

```
// create some source data
    int[] sourceData = new int[10];
    for (int i = 0; i < sourceData.Length; i++) {
       sourceData[i] = i;
    }
```

```
// define the query and force parallelism
IEnumerable<double> results =
  sourceData.AsParallel()
  .WithDegreeOfParallelism(2)
  .Where(item => item % 2 == 0)
  .Select(item => Math.Pow(item, 2))
  .AsSequential()
  .Select(item => item * 2);
```

Exercitii.

1. Studiati si testati exemplele din textul de mai sus.
2. Pushout :

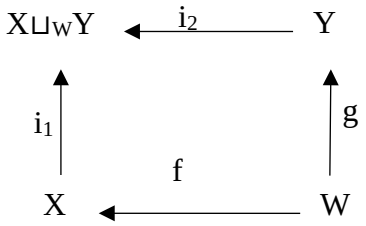
Daca avem trei multimi X,Y si W si doua functii :

f: W$\rightarrow$X si g: W$\rightarrow$Y

Atunci o multime P se numeste pushoutul lui X cu Y peste W (sau pushoutul lui f cu g) daca este izomorfa cu multimea:

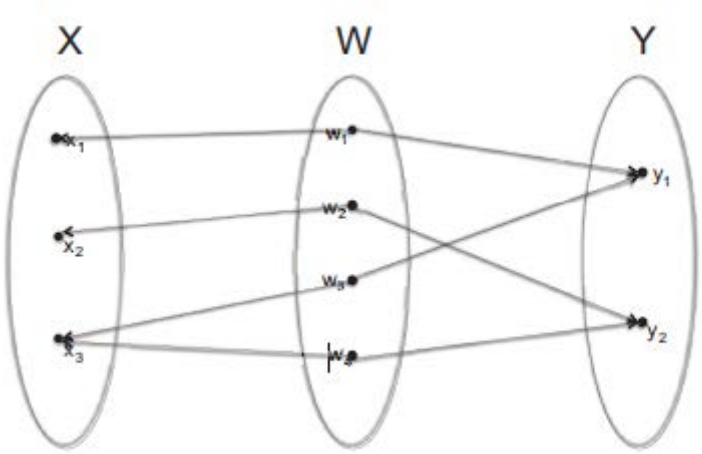
$X \sqcup_W Y = (X \sqcup W \sqcup Y)/\rho$, unde ρ este o relatie de echivalenta generata de functiile f si g dupa cum urmeaza: $x \rho y \Leftrightarrow y=f(x)$ sau $y= g(x)$.

Diagrama:



se numeste diagrama pushout.

De exemplu daca functiile f:W$\rightarrow$X si g:W$\rightarrow$Y sunt cele din diagrama



Atunci pushoutul lui f cu g are un singur element pentru ca:
$x_1 \rho w_1 \rho y_1 \rho w_3 \rho x_3 \rho w_4 \rho y_2 \rho w_2 \rho x_2$.
Pushoutul poate fi folosit pentru a caracteriza epimorfismele: Un morfism f: W$\rightarrow$ X este un epimorfism dacă și numai dacă pushoutul lui f și f există și este W.

Vom considera o clasa abstracta PushoutAbstract definite astfel.

```
public interface functie<Td, Tc>
{
    Tc Calcul(Td intrare);
}

public abstract class PushoutAbstract<T1, T2>
{
     public abstract HashSet<T2> getPushout(functie<T1, T2> fi, HashSet<T2> codomfi,
   functie<T1, T2> gi, HashSet<T2> codomgi, HashSet<T1> domi);
}
```

2.1. Sa se scrie o clasa Pushout care extinde clasa PushoutAbstract si implementeaza metodele clasei de baza.

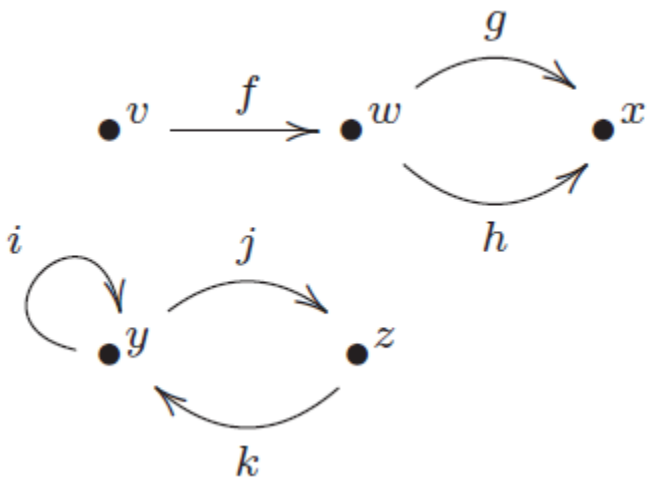
2.2. Sa se scrie un program care sa testeze clasa Pushout pentru exemplul de mai sus.

2.3. Implementati o metoda de extensie Pushout la interfata IEnumerable cu semnatura de mai jos si testati-o pe exemplu de mai sus.

```
public static class MyExtensionMethods
{
        public static HashSet<T2> Pushout<T1, T2>(this IEnumerable<T2> codomeniu,
HashSet<T1> domeniu, functie<T1, T2> fi, functie<T1, T2> gi)
        {
        }
    }
```

3. Grafuri:

Un graf G poate fi definit ca un tuplu G=(V, A, src, tgt), unde
V este o multime, numita multimea vârfurilor lui G;
A este o multime, numita multimea arcelor lui G;
src: A $\rightarrow$ V este o funcție, numită funcția sursă care asociaza fiecarui arc din A o sursa din V;
tgt: A $\rightarrow$ V este o funcție, numită funcția tinta care asociaza fiecarui arc din A o tinta din V;.
De exemplul pentru graful G=(V, A, src, tgt) din figura:



avem V ={v, w, x, y, z} și A= {f, g, h, i, j, k}. Funcțiile sursă și țintă src, tgt: A $\rightarrow$ V sunt exprimate în tabelul următor:

A	src	tgt
f	v	w
g	w	x
h	w	x
i	y	y
j	y	z
k	z	y

3.1. Utilizand clasa Pushout implementata mai sus calculati pushout -ul functiilor src si tgt.

3.2. Desenati grafu definit astfel:

A	src	tgt
f	v	w
g	v	w
h	v	w
i	x	w
j	z	w
k	z	z

V
u
v
w
x
y
z

3.3. Utilizand clasa Pushout implementata mai sus calculati Pushoutul functiilor src si tgt pentru acest graf.

3.4. Observati ca Pushoutul functiilor src si tgt in cazul grafurilor este izomorf cu multimea componentelor conexe ale grafului. Testati acest lucru adaugand componente conexe la graful de mai sus.