**Osnove LINQ tehnologije**

**00:14:02**

Jedinica: 4 od 19

**+Rezime**

Značajan deo vremena koji se troši na razvoj aplikacija odlazi na pisanje upita kojima se manipuliše podacima. Pritom, podaci mogu biti smešteni u relacionim bazama podataka, u fajlovima, u skladištima koja podatke čuvaju kao parove ključeva i vrednosti, ali i u radnoj memoriji računara. Kako bi se olakšao pristup podacima različitih skladišta, od .NET 3.5 verzije, sastavni deo platforme postaje jezik LINQ (*Language Integrated Query*).

**Šta je LINQ?**

LINQ predstavlja standardni, objedinjeni pristup za pretragu i modifikovanje podataka koji može da podrži praktično bilo koji tip podataka. Tako LINQ dozvoljava pretragu podataka u različitim oblicima, uključujući relacione baze, [XML](https://www.link-elearning.com/linkdl/opisPojma.php?id=142476) dokumenta, pa čak i strukture podataka smeštene u radnoj memoriji.

LINQ je prvi put predstavljen u verziji 3.5 .NET okvira, a Microsoft je i u novijim verzijama .NET okvira nastavio na unapređivanju ove tehnologije.

**Zašto LINQ?**

LINQ tehnologija poseduje brojne pozitivne strane, od kojih su najznačajnije sledeće:

* + LINQ obezbeđuje zajedničku sintaksu za pretragu bilo kojeg izvora podataka. Tako se pretraga podataka XML dokumenta obavlja na identičan način kao i pretraga SQL baze podataka, ADO.NET DataSeta, kolekcije u radnoj memoriji, ili podataka bilo kojeg drugog skladišta.
  + LINQ premošćava jaz između relacionog i objektnog modela.
  + LINQ ubrzava razvoj, hvatanjem grešaka u toku kompajliranja i obezbeđivanjem podrške za [IntelliSense](https://www.link-elearning.com/linkdl/opisPojma.php?id=142449) i [Debug](https://www.link-elearning.com/linkdl/opisPojma.php?id=142471).
  + LINQ izrazi za pretragu su, za razliku od tradicionalnih SQL izraza, strogo tipizirani, što osigurava da će bilo koje odstupanje u tipu očekivane vrednosti parametara biti detektovano tokom kompajliranja.

**LINQ prostori imena**

Sve pomenute funkcionalnosti smeštene su u nekoliko osnovih LINQ prostora imena. Ti prostori imena su prikazani tabelom 4.1.

|  |  |
| --- | --- |
| **Prostori imena** | **Opis** |
| **System.LINQ** | Klase i interfejsi koji omogućavaju kreiranje LINQ upita. |
| **System.Collections.Generic** | Omogućava kreiranje strogo tipiziranih kolekcija. |
| **System.Data.LINQ** | Funkcionalnosti koje omogućavaju korišćenje LINQ-a za pristup relacionim bazama podataka. |
| **System.XML.LINQ** | Funkcionalnosti koje omogućavaju korišćenje LINQ-a za pristup XML dokumentima. |
| **System.Data.Linq.Mapping** | Anotacije za dekorisanje klasa entiteta |

*Tabela 4.1 – LINQ prostori imena*

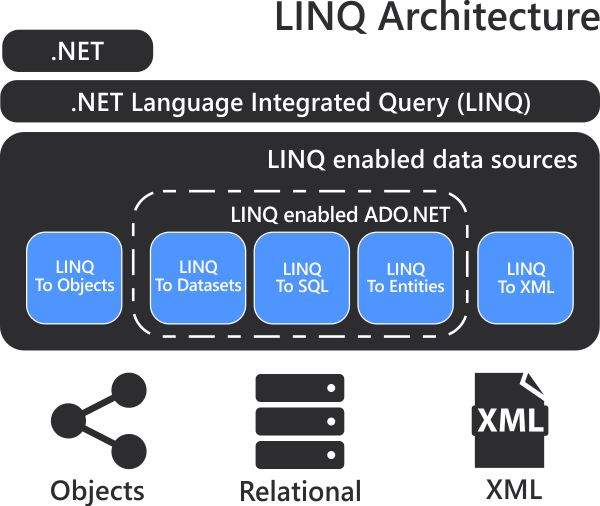
Sa navedenim prostorima imena susretaćemo su u ovoj i u narednoj lekciji na praktičnim primerima.

**Arhitektura LINQ-a**

LINQ se sastoji iz tri glavne komponente:

* + LINQ to Objects
  + LINQ to ADO.NET koji obuhvata:
    - LINQ to SQL
    - LINQ to DataSets
    - LINQ to Entities
  + LINQ to XML.

*Slika 4.1 dočarava LINQ arhitekturu.*



*Slika 4.1 – LINQ arhitektura*

**LINQ to Objects** operiše podacima u radnoj memoriji. Svaka klasa koja implementira [IEnumerable<T>](https://www.link-elearning.com/linkdl/opisPojma.php?id=142480) interfejs iz System.Collections.Generic prostora imena može biti korišćena kao objekat pretrage, upotrebom LINQ operatora za pretragu.

**LINQ to ADO.NET** operiše podacima iz eksternih skladišta, što mogu biti svi izvori sa kojima ADO.NET može da ostvari konekciju. Tako, bilo koja klasa koja implementira interfejse IEnumerable<T> ili [IOueryable<T>](https://www.link-elearning.com/linkdl/opisPojma.php?id=142481), može biti objekat pretrage LINQ izraza.

Na kraju, **LINQ to XML** je [API](https://www.link-elearning.com/linkdl/opisPojma.php?id=142482), koji se koristi za pojednostavljivanje komunikacije sa podacima u XML formatu.

|  |
| --- |
| ***Napomena:****LINQ to SQL biće predmet sledeće lekcije.* |

**LINQ to Objects**

Pojam *LINQ to Objects* se odnosi na LINQ upite nad strukturama podataka u radnoj memoriji. Tako svaki tip koji se zasniva na tipu IEnumerable<T> može biti predmet pretrage, pa se LINQ izrazi mogu koristiti nad korisnički definisanim listama, nizovima i [rečnicima](https://www.link-elearning.com/linkdl/opisPojma.php?id=142472), ali i nad bilo kojim drugim tipovima kolekcija. LINQ upiti nad podacima u memoriji su superiorniji u odnosu na tradicionalne pristupe, koji podrazumevaju upotrebu foreach u nekoliko tačaka:

* + konzistentniji su i čitljiviji, pogotovu kada je potrebno obaviti filtriranje;
  + omogućavaju moćne funkcionalnosti za filtriranje, sortiranje i grupisanje, što je moguće postići sa minimalnom količinom koda;
  + jednom napisani LINQ upiti mogu se koristiti i sa drugim tipovima skladišta sa veoma malo modifikacija.

Generalno pravilo koje važi je: što je kompleksnija operacija nad podacima, to će korist od korišćenja LINQ-a biti veća.

Već je rečeno da se za pretragu podataka smeštenih u radnoj memoriji mogu koristiti i tradicionalni pristupi. Stoga ću u nastavku, za postizanje identične operacije nad podacima, biti korišćena dva pristupa. Jedan će podrazumevati tradicionalne tehnike, a drugi upotrebu LINQ izraza.

Dat je sledeći niz string vrednosti:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | string[] names = { "Roman Trelkovsky",              "Roberto Miranda",              "Richard Walker",              "Sarah Shagal",              "Rosemary Woodhouse",              "Adam Lang",              "Dean Corso",              "Marion Crane",              "Rupert Cadell",              "Joseph Maloney"              }; |

Potrebno je pronaći i prikazati sve osobe čije ime i prezime zajedno imaju više od 12 karaktera. Da bi se to postiglo, može se napisati sledeće:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | IEnumerable<string> namesOfPeople = from name in names                                                  where name.Length > 12                                                  select name;              foreach (var name in namesOfPeople)              {                  Console.WriteLine(name);              } |

**Za pretraživanje objekata koji se nalazi u radnoj memoriji, potrebno je koristiti:**

 LINQ to Objects

 LINQ to ADO.NET

 LINQ to XML

 LINQ to SQL

**Sintaksa LINQ upita**

LINQ tehnologija omogućava dva načina za manipulaciju podacima. Jedan podrazumeva upotrebu lambda izraza, a drugi specijalnu sintaksu za pretragu. Jedan isti upit, realizovan na dva pomenuta načina, prikazan je u nastavku:

|  |  |
| --- | --- |
| 1  2 | var longWords = words.Where(w => w.length > 10);  var longwords = from w in words where w.length > 10; |

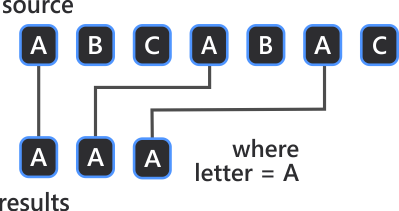
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Napomena:****lambda izrazi se koriste za kreiranje anonimnih, lokalnih funkcija koje se mogu prosleđivati kao parametri ili vratiti kao povratna vrednost funkcije. Lambda izrazi su naročito korisni za pisanje LINQ upita.*    *Da bi se kreirao lambda izraz, definišu se ulazni parametri na levoj strani specijalnog lambda operatora =>, dok se na desnoj strani definiše izraz ili blok izjava koji su potrebni da se izvrše. Tako jedan jednostavan lambda izraz može da izgleda ovako:*     |  |  | | --- | --- | | 1 | x => x \* x |     *U navedenom lambda izrazu ulazni parametar, koji je imenovan kao x, podiže se na kvadrat. Kompletan primer upotrebe ovakvog lambda izraza je sledeći:*     |  |  | | --- | --- | | 1  2 | del myDelegate = x => x \* x;  int j = myDelegate(5); //j = 25 | |

**LINQ operatori za pretragu**

U prethodnom primeru mogla se primetiti upotreba specijalnih [exstension metoda](https://www.link-elearning.com/linkdl/opisPojma.php?id=142483) koje predstavljaju LINQ operatore za pretragu. Jedan takav operator, upotrebljen u prethodnom primeru je operator where. Pored ovog operatora, LINQ poseduje bogat set različitih operatora kojima se obavlja filtriranje, sortiranje, agregacija i ostale operacije nad podacima. Takvi operatori biće prikazani u nastavku.

**Filtriranje**

Filtriranje je proces kojim se inicijalni skup podataka sužava samo na one koji zadovoljavaju određene kriterijume. Najpoznatiji takav operator je operator where. Slika 4.2 dočarava efekat koji operator za filtriranje where ima na skupu podataka.



*Slika 4.2 – Efekat operatora Where*

Definicija operatora where prikazana je tabelom.

|  |  |  |
| --- | --- | --- |
| **Metoda** | **Opis** | **Upitni izraz** |
| Where | Selektuje vrednosti na osnovu funkcije koja definiše kriterijum. | where |

*Tabela 4.2 – Operator where*

U nastavku će biti prikazan primer upotrebe operatora WHERE prilikom pretrage jednog niza stringova.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | string[] words = { "LINQ", "WCF", "WPF", "ADO.NET", "ASP.NET", "MVC" };              IEnumerable<string> query = from word in words where word.Length == 3 select word;              foreach (var item in query)              {                  Console.WriteLine(item);              } |

Prikazani primer rezultuje ispisivanjem sledećeg rezultata:

WCF  
WPF  
MVC

Unutar LINQ upita za pretragu ključna reč where je iskorišćena kako bi se u rezultujućem skupu elemenata dobili samo oni elementi iz izvornog skupa koji se sastoje iz tačno 3 karaktera.

**Spajanje (Join)**

Pojam *spajanje* (Join) se odnosi na proces spajanja informacija više različitih entiteta koji su međusobno povezani relacijom. Tako se informacije kombinuju i koriste za kreiranje novog objektnog entiteta.

Spajanje je karakteristična operacija kod relacionih baza podataka, te Vam je ovaj pojam verovatno poznat iz kursa o radu sa bazama. Operator join ima identičnu svrhu kao i join kod relacionih baza podataka. Tabela 4.3 prikazuje join operatore.

|  |  |  |
| --- | --- | --- |
| **Metoda** | **Opis** | **Upitni izraz** |
| Join | Spaja dve sekvence na osnovu deljenih ključeva. | join … in … on … equals … |
| GroupJoin | Spaja dve sekvence na osnovu deljenih ključeva i grupiše elemente. | join … in … on … equals … into … |

*Tabela 4.3 – Operatori za spajanje*

U nastavku će biti prikazan primer koji će ilustrovati upotrebu join operatora na primeru dve kolekcije. Kolekcije će baratati sledećim tipovima:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | class Address          {              public int Id { get; set; }              public string Country { get; set; }              public string City { get; set; }              public Address(int id, string country, string city)              {                  Id = id;                  Country = country;                  City = city;              }          }          class Person          {              public string FirstName { get; set; }              public string LastName { get; set; }              public int AdressId { get; set; }              public Person(string firstname, string lastname, int adressId)              {                  FirstName = firstname;                  LastName = lastname;                  AdressId = adressId;              }          } |

Reč je o tipovima Address i Person, koji su međusobno povezani relacijom. Svaka osoba ima samo jednu adresu, a jednu istu adresu može imati više osoba. Korišćenjem prikazanih tipova kreirane su sledeće kolekcije.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | List<Address> addresses = new List<Address>() {                  new Address(1, "England", "London"),                  new Address(2, "Serbia", "Belgrade"),                  new Address(3, "USA", "New York"),                  new Address(4, "Russia", "Moscow")              };              List<Person> people = new List<Person>() {                  new Person("John", "Brown", 1),                  new Person("Ivan", "Petrovic", 2),                  new Person("Milan", "Jovanovic", 2),                  new Person("Ben", "Sugar", 4),                  new Person("Vladimir", "Sostakov", 3)              }; |

Lako se može zaključiti da entiteti tipa Person ne sadrže direktno informaciju o imenu države ili grada, već su takve informacije sadržane u instancama tipa Address. Kako dobiti potpuno novu kolekciju, koja će kombinovati informacije iz jedne i iz druge kolekcije. Odgovor, naravno, leži u upotrebi join operatora, pa se može napisati nešto ovako:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | var list = (from p in people                          join a in addresses on p.AdressId equals a.Id                          select new                          {                              PersonName = p.FirstName + " " + p.LastName,                              PersonCountry = a.Country                          });              foreach (var item in list)              {                  Console.WriteLine("Person: " + item.PersonName + " Country: " + item.PersonCountry);              } |

Prikazani kod će kao svoj rezultat ispisati sledeće:

Person: John Brown Country: England  
Person: Ivan Petrovic Country: Serbia  
Person: Milan Jovanovic Country: Serbia  
Person: Ben Sugar Country: Russia  
Person: Vladimir Sostakov Country: USA

**Selektovaje (projektovanje)**

Projektovanje je operacija kojom se objekat transformiše u potpuno novu formu sa unapred definisanim odlikama. Operatori za projektovanje su prikazani tabelom 4.4.

|  |  |  |
| --- | --- | --- |
| **Metoda** | **Opis** | **Upitni izraz** |
| Select | Projektuje novu vrednost na osnovu primenjenih funkcija za transformaciju. | select |
| SelectMany | Projektuje sekvencu vrednosti na osnovu primenjenih funkcija transformacija u jedan niz. | Postiže se upotrebom više from izraza. |

*Tabela 4.4 – Operatori za projektovanje*

U nastavku slede primeri operatora za projektovanje.

Select:

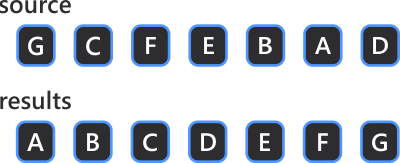
|  |  |
| --- | --- |
| 1  2  3  4 | List<string> words = new List<string>() { "ASP.NET", "WPF", "MVC", "JavaScript" };  var query = from word in words select word.Substring(0, 1);  foreach (string s in query)  Console.WriteLine(s); |

SelectMany:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | List<string> words = new List<string>() { "ASP.NET", "WPF", "MVC", "JavaScript" };  var query = from word in words from letters in word.ToCharArray() select letters;  foreach (char item in query)  {  Console.WriteLine(item);  } |

**Sortiranje**

Operacija sortiranja dozvoljava uticanje na raspored elemenata unutar jednog niza na osnovu jednog ili više atributa. Sortiranje se može obaviti na osnovu više kriterijuma sortiranja, pri čemu je prvi kriterijum primarni, a drugi sekundarni. Slika 4.3 dočarava proces sortiranja jednog skupa elemenata.



*Slika 4.3 – Efekat operatora za sortiranje*

Osnovni operatori za sortiranje prikazani su tabelom 4.5.

|  |  |  |
| --- | --- | --- |
| **Metoda** | **Opis** | **Upitni izraz** |
| OrderBy | Sortira vrednosti po rastućem redu. | orderby |
| OrderByDescending | Sortira vrednosti po opadajućem redu. | orderby ... descending |
| ThenBy | Definiše drugi kriterijum za sortiranje, koji će se primeniti nakon prvog, a sortiranje će biti obavljeno po rastućem redu. | orderby …, … |
| ThenByDescending | Definiše drugi kriterijum za sortiranje, koji će se primeniti nakon prvog, a sortiranje će biti obavljeno po opadajućem redu. | orderby …, … descending |
| Reverse | Okreće redosled elemenata u kolekciji. | \* |

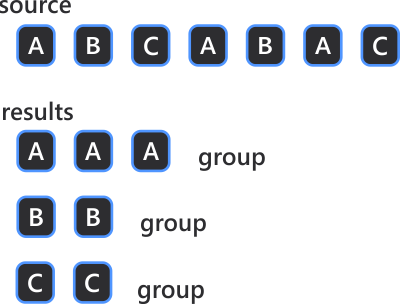
*Tabela 4.5 – Operatori za sortiranje*

Sledeći primer dočarava upotrebu operatora za sortiranje.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | List<string> words = new List<string>() { "ASP.NET", "WPF", "WCF", "MVC", "C#", "JavaScript" };  var ascendingWords = from word in words orderby word.Length select word;  var descendingWords = from word in words orderby word.Length descending select word;  var twoCriteria = from word in words orderby word.Length, word.First() select word;  var reverse = words.Reverse<string>();              Console.Write("\nAscending: \n");              foreach (string item in ascendingWords)              {                  Console.WriteLine(item);              }              Console.Write("\nDescending: \n");              foreach (string item in descendingWords)              {                  Console.WriteLine(item);              }              Console.Write("\nTwo criteria: \n");              foreach (string item in twoCriteria)              {                  Console.WriteLine(item);              }              Console.Write("\nReverse: \n");              foreach (string item in reverse)              {                  Console.WriteLine(item);              } |

**Grupisanje**

Ovi operatori vrše grupisanje podataka na osnovu neke zajedničke karakteristike, kao na slici 4.4.



*Slika 4.4 – Efekat operatora za grupisanje*

Tabela 4.6 prikazuje operatore za grupisanje.

|  |  |  |
| --- | --- | --- |
| **Metoda** | **Opis** | **Upitni izraz** |
| GroupBy | Organizuje sekvencu podataka u jednu grupu. | group … by -or- group … by … into … |

*Tabela 4.6 – Operatori za grupisanje*

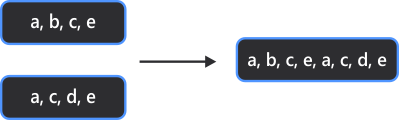
Sledeći primer ilustruje efekat koji grupisanje ima na jedan skup podataka.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | int[] numbers = { 34, 22, 1, 3, 56, 28, 98, 5, 7, 76, 45 };  var query = from num in numbers group num by num % 2;  foreach (var group in query)  {  foreach (var item in group)  {  Console.WriteLine(item);  }  } |

Primenom operatora za grupisanje dobija se dvodimenzionalna kolekcija, u kojoj je svaka grupa jedna zasebna kolekcija. Upravo zbog toga se za ispisivanje svih članova kolekcije koriste dve foreach petlje.

**Nadovezivanje (konkatenacija)**

Operatori za nadovezivanje obavljaju [konkatenaciju](https://www.link-elearning.com/linkdl/opisPojma.php?id=142450) dve sekvence podataka. Slika 4.5 dočarava efekat konkatenacije.



*Slika 4.5 – Efekat operatora za nadovezivanje*

Tabela 4.7 prikazuje LINQ operatore za nadovezivanje.

|  |  |  |
| --- | --- | --- |
| **Metoda** | **Opis** | **Upitni izraz** |
| Concat | Nadovezuje dve sekvence podataka. | \* |

*Tabela 4.7 – Operatori za nadovezivanje*

Sledeći primer ilustruje upotrebu operatora za nadovezivanje.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | int[] numbers = { 1, 2, 3, 4, 5 };  int[] numbers2 = { 6, 7, 8, 9, 10 };  var nums = numbers.Concat(numbers2);    foreach (var num in nums)  {    Console.Write(num + " ");  } |

**Agregacija**

Operatori za obavljanje agregacije se primenjuju nad nizom vrednosti i u zavisnosti od određenog kriterijuma najčešće formiraju jednu finalnu vrednost koja ima posebno značenje. Tako su neki od najznačajnijih ovakvih operatora oni za izračunavanje srednje, maksimalne ili minimalne vrednosti. Ovakvi operatori ne poseduju sintaksu po kojoj se mogu koristiti unutar izraza za pretraživanje, već samo kao [exstenion metode](https://www.link-elearning.com/linkdl/opisPojma.php?id=142483). Tabela prikazuje ovakve LINQ operatore.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Opis** | **Upitni izraz** |
| Average | srednja vrednost skupa vrednosti | \* |
| Count | ukupan broj elemenata | \* |
| Max | najveća vrednost iz skupa vrednosti | \* |
| Min | najmanja vrednost iz skupa vrednosti | \* |
| Sum | zbir svih vrednosti iz skupa vrednosti | \* |

*Tabela 4.8 – Operatori za agregaciju*

Sledeći blok koda ilustruje upotrebu operatora za agregaciju korišćenjem exstension metoda. Operatori za agregaciju se najčešće primenjuju nakon nekih drugih LINQ operatora i takav scenario je upravo ilustrovan prikazanim primerom, gde se agregatne metode upotrebljavaju nakon filtracije korišćenjem operatora where. Operatorom where vrši se filtriranje svih parnih vrednosti.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | int[] numbers = {1, 45, 3, 5, 18, 7, 8, 87, 35, 87, 22, 37, 98, 101};  double averageEvenValue = numbers.Where(num => num % 2 == 0).Average();  double evenNumsCount = numbers.Where(num => num % 2 == 0).Count();  double evenMax = numbers.Where(num => num % 2 == 0).Max();  double evenMin = numbers.Where(num => num % 2 == 0).Min();  double sumEvenValue = numbers.Where(num => num % 2 == 0).Sum();  Console.WriteLine("The mean value of all even numbers: " + averageEvenValue);  Console.WriteLine("Even numbers count: " + evenNumsCount);  Console.WriteLine("The maximum value of all even numbers: " + evenMax);  Console.WriteLine("The minimum value of all even numbers: " + evenMin);  Console.WriteLine("The sum of all even numbers: " + sumEvenValue); |

**Kvantifikatori**

Kvantifikatori su specijalna vrsta operatora koji vraćaju logičke vrednosti true ili false, u zavisnosti od toga da li je definisani uslov zadovoljen. Prikazani su tabelom.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Opis** | **Upitni izraz** |
| All | Vraća true, ako sve vrednosti skupa zadovoljavaju definisan uslov. | \* |
| Any | Vraća true, ako bilo koja od vrednosti zadovoljava definisan uslov. | \* |
| Contains | Vraća true, ukoliko skup sadrži definisan element. | \* |

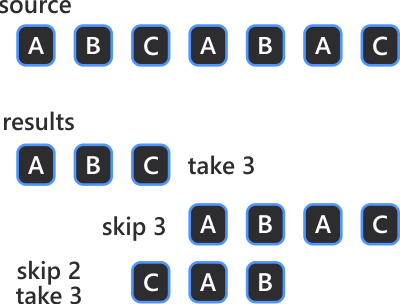
 *Tabela 4.9 – Kvantifikatori*

Sledeći primer ilustruje upotrebu kvantifikatora.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | int[] numbers = {1, 45, 3, 5, 18, 7, 8, 87, 35, 87, 22, 37, 98, 101};  bool anyEvenValue = numbers.Any(num => num % 2 == 0);  bool isThereAny = numbers.Where(num => num == 44).Any();  bool allEvenValue = numbers.All(num => num % 2 == 0);  bool containsSpecificValue = numbers.Contains(45);  Console.WriteLine("Any even number: " + anyEvenValue);  Console.WriteLine("All the numbers are even: " + allEvenValue);  Console.WriteLine("Whether array contains at least one item that meets criteria: " + isThereAny);  Console.WriteLine("Whether array contains specific value: " + containsSpecificValue); |

**Deljenje**

Deljanje (Partitioning) u LINQ-u jeste proces kojim se ulazni niz vrednosti deli na dve sekcije, bez promene rasporeda elemenata. Slika 4.6 ilustruje način na koji se deljanje obavlja, kao i efekat koji se njime postiže.



*Slika 4.6 – Efekat deljenja*

Tabela 4.10 prikazuje LINQ operatore kojima se postiže deljenje.

|  |  |  |
| --- | --- | --- |
| **Metoda** | **Opis** | **Upitni izraz** |
| Skip | Preskače elemente u nizu sve do određene pozicije. | \* |
| SkipWhile | Preskače elemente u nizu koji zadovoljavaju kriterijum, sve do prvog elementa koji ne zadovolji kriterijum; nadalje uzima sve elemente. | \* |
| Take | Uzima elemente iz niza sve do određene pozicije. | \* |
| TakeWhile | Uzima elemente iz niza redom, počev od onih koji zadovoljavaju definisan uslov, sve do prvog elementa koji ne zadovoljava definisan uslov. | \* |

*Tabela 4.10 – LINQ operatori za deljenje*

Sledeći blok koda ilustruje upotrebu operatora za deljenje.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | int[] numbers = { 1, 45, 3, 5, 18, 7, 8, 87, 35, 87, 22, 37, 98, 101 };              var takeDemo = numbers.Take(3);              var takeWhileDemo = numbers.TakeWhile(num => num < 80);              var skipDemo = numbers.Skip(3);              var skipWhileDemo = numbers.SkipWhile(num => num < 80);              Console.WriteLine("Take demo:");              takeDemo.ToList().ForEach(Console.WriteLine);              Console.WriteLine("TakeWhile demo:");              takeWhileDemo.ToList().ForEach(Console.WriteLine);              Console.WriteLine("Skip demo:");              skipDemo.ToList().ForEach(Console.WriteLine);              Console.WriteLine("SkipWhile demo:");              skipWhileDemo.ToList().ForEach(Console.WriteLine); |

**Generisanje**

Pojam generisanja odnosi se na proces kreiranja nove sekvence vrednosti. Metode za obavljanje ovakvog procesa prikazane su tabelom 4.11.

|  |  |  |
| --- | --- | --- |
| **Metoda** | **Opis** | **Upitni izraz** |
| DefaultIfEmpty | Ukoliko je skup vrednosti prazan, generiše podrazumevanu vrednost skupa. | \* |
| Empty | Generiše praznu sekvencu elemenata određenog tipa. | \* |
| Range | Generiše kolekciju koja se sastoji iz sekvence brojeva. | \* |
| Repeat | Generiše sekvencu koja se sastoji iz određenog broja članova iste vrednosti. | \* |

*Tabela 4.11 – LINQ metode za generisanje*

DefaultIfEmpty:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | int[] numbers = { 1, 45, 3, 5, 18, 7, 8, 87, 35, 87, 22, 37, 98, 101 };  int[] emptyNumbers = { };  var aNumber = numbers.First();  var bNumber = emptyNumbers.DefaultIfEmpty().Single();  var cNumber = emptyNumbers.DefaultIfEmpty(13).Single();  Console.WriteLine("A: " + aNumber);  Console.WriteLine("B: " + bNumber);  Console.WriteLine("C: " + cNumber); |

Empty:

U nekim situacijama može se javiti potreba za prosleđivanjem prazne kolekcije nekoj metodi ili za definisanjem prazne kolekcije kao povratne vrednosti neke metode. U takvim situacijama i više nego korisna može biti metoda empty. Ova metoda generiše prazne kolekcije određenog tipa. Tako nešto ilustruje sledeći primer:

|  |  |
| --- | --- |
| 1  2 | var empty = Enumerable.Empty<int>();  Console.WriteLine(empty.Count()); |

Range:

|  |  |
| --- | --- |
| 1  2 | IEnumerable<int> squares = Enumerable.Range(1, 5).Select(x => x \* x);  foreach (int num in squares) { Console.WriteLine(num); } |

Repeat:

|  |  |
| --- | --- |
| 1  2  3  4  5 | IEnumerable<string> strings = Enumerable.Repeat("I like LINQ.", 3);              foreach (string str in strings)              {                  Console.WriteLine(str);              } |

**Set operacije**

Set operacije u LINQ-u su operacije koje proizvode rezultat na osnovu prisustva ili odsustva jednakih elemenata u okviru tekućeg ili nekog drugog skupa vrednosti.

Tabela 4.12 prikazuje dostupne set operacije u LINQ-u.

|  |  |  |
| --- | --- | --- |
| **Metoda** | **Opis** | **Upitni izraz** |
| Distinct | Uklanja duplikate iz kolekcije vrednosti. | \* |
| Except | Vraća razlike, odnosno elemente jedne kolekcije koji se ne pojavljuju u drugoj. | \* |
| Intersect | Vraća elemente koji postoje u obe kolekcije. | \* |
| Union | Vraća sve jedinstvene elemente koji postoje u obe kolekcije. | \* |

*Tabela 4.12 – LINQ set operatori*

Slikama 4.7, 4.8, 4.8 i 4.10 prikazani su efekti koje set operatori imaju na podacima. Pored pomenutih slika, navedeni su i slikoviti primeri koda, u vezi sa upotrebom ovih operatora.

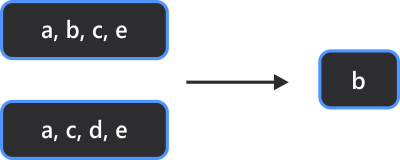
*Distinct*:

https://www.link-elearning.com/linkdl/coursefiles/1142/MDA_04_07.jpg

*Slika 4.7 – Operator Distinct*

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | int[] numbers = { 1, 45, 1, 3, 5, 45, 18, 7, 18, 8, 87, 87, 35, 87, 22, 37, 98, 101, 98 };              var distinctNumbers = numbers.Distinct();              Console.WriteLine("Original numbers: ");              numbers.ToList().ForEach(num => Console.Write(num + " "));              Console.WriteLine("\nDistinct numbers: ");              distinctNumbers.ToList().ForEach(num => Console.Write(num + " "));              Console.WriteLine("\n"); |

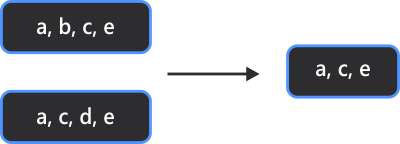
*Except*:



*Slika 4.8 – Operator Except*

|  |  |
| --- | --- |
| 1  2  3  4  5 | int[] numbers = { 3, 5, 45, 18, 7, 87, 35, 37, 98 };              int[] numbers2 = { 1, 45, 3, 5, 55, 18, 7, 8, 87, 35, 97, 22, 37, 98, 101 };              var exceptNumbers = numbers2.Except(numbers);              Console.WriteLine("Except numbers: ");              exceptNumbers.ToList().ForEach(num => Console.Write(num + " ")); |

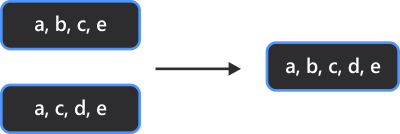
*Intersect*:



*Slika 4.9 – Operator Intersect*

|  |  |
| --- | --- |
| 1  2  3  4  5 | int[] numbers = { 3, 5, 45, 18, 7, 87, 35, 37, 98 };              int[] numbers2 = { 1, 45, 3, 5, 55, 18, 7, 8, 87, 35, 97, 22, 37, 98, 101 };              var intersectNumbers = numbers2.Intersect(numbers);              Console.WriteLine("Intersect numbers: ");              intersectNumbers.ToList().ForEach(num => Console.Write(num + " ")); |

*Union*:



*Slika 4.10 – Operator Union*

|  |  |
| --- | --- |
| 1  2  3  4  5 | int[] numbers = { 2, 4, 44, 17, 9, 87, 35, 37, 98 };              int[] numbers2 = { 1, 45, 3, 5, 55, 18, 7, 8, 87, 35, 97, 22, 37, 98, 101 };              var unionNumbers = numbers2.Union(numbers);              Console.WriteLine("Union: ");              unionNumbers.ToList().ForEach(num => Console.Write(num + " ")); |

**Jednakost**

LINQ poseduje jedan operator koji se koristi za utvrđivanje jednakosti dva skupa elemenata. Takav operator prikazan je tabelom 4.13.

|  |  |  |
| --- | --- | --- |
| **Metoda** | **Opis** | **Upitni izraz** |
| SequenceEqual | Utvrđuje jednakost dve sekvence vrednosti poređenjem pojedinačnih elemenata. | \* |

*Tabela 4.13 – LINQ operator za utvrđivanje jednakosti*

Sledeći primer ilustruje upotrebu operatora za utvrđivanje jednakosti.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | int[] numbers = { 2, 4, 44, 17, 9, 87, 35, 37, 98 };              int[] numbers2 = { 2, 4, 44, 17, 9, 87, 35, 37, 98 };              int[] numbers3 = { 1, 45, 3, 5, 55, 18, 7, 8, 87, 35, 97, 22, 37, 98, 101 };              bool areEqual = numbers.SequenceEqual(numbers2);              bool areEqual2 = numbers.SequenceEqual(numbers3);              Console.WriteLine("Are equal (1-2): " + areEqual);              Console.WriteLine("Are equal (1-3): " + areEqual2); |

**Operacije nad elementima**

Operatori za izvršavanje operacija nad elementima vraćaju jedan element iz sekvence. Takvi operatori prikazani su tabelom 4.14.

|  |  |  |
| --- | --- | --- |
| **Metoda** | **Opis** | **Upitni izraz** |
| ElementAt | Vraća element koji se nalazi na određenoj poziciji unutar kolekcije. | \* |
| ElementAtOrDefault | Vraća element koji se nalazi na određenoj poziciji unutar kolekcije ili podrazumevanu vrednost, ukoliko je navedeni indeks izvan granica opsega. | \* |
| First | Vraća prvi element kolekcije ili prvi element koji zadovoljava definisan kriterijum. | \* |
| FirstOrDefault | Vraća prvi element kolekcije ili prvi element koji zadovoljava definisan kriterijum. Ukoliko se element ne pronađe, vraća podrazumevanu vrednost. | \* |
| Last | Vraća poslednji element kolekcije ili poslednji element koji zadovoljava definisan kriterijum. | \* |
| LastOrDefault | Vraća poslednji element kolekcije ili poslednji element koji zadovoljava definisan kriterijum. Ukoliko se element ne pronađe, vraća podrazumevanu vrednost. | \* |
| Single | Vraća jedini element kolekcije ili jedini element koji zadovoljava kriterijum. | \* |
| SingleOrDefault | Vraća jedini element kolekcije ili jedini element koji zadovoljava kriterijum. Ukoliko traženi element ne postoji ili kolekcija ne sadrži tačno jedan element, vraća podrazumevanu vrednost. | \* |

*Tabela 4.14 – Operatori za izvršavanje operacija nad elementima*

Sledeći primer ilustruje upotrebu metoda za sprovođenje operacija nad elementima.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | int[] numbers = { 2, 4, 44, 17, 9, 87, 35, 37, 98 };  int[] emptyNumbers = { };  int[] number = { 2 };  int elementAt = numbers.ElementAt(2); //40  int elementAtOrDefault = numbers.ElementAtOrDefault(22); //0  int first = numbers.First(); //2  int firstWithCondition = numbers.First(num => num > 50); //87  int firstOrDefault = emptyNumbers.FirstOrDefault(); //0  int firstOrDefaultWithCondition = numbers.FirstOrDefault(num => num > 100); //0  int last = numbers.Last(); //98  int lastWithCondition = numbers.Last(num => num > 10 && num < 50); //37  int lastOrDefault = emptyNumbers.LastOrDefault(); //0  int lastOrDefaultWithCondition = numbers.LastOrDefault(num => num > 100); //0  int single = number.Single(); //2  int singleWithCondition = numbers.Single(num => num > 40 && num < 45); //44  int singleOrDefault = number.SingleOrDefault(); //2  int singleOrDefaultWithCondition = numbers.SingleOrDefault(num => num > 100); //0 |