**LINQ to SQL**

**00:17:16**

Jedinica: 5 od 19

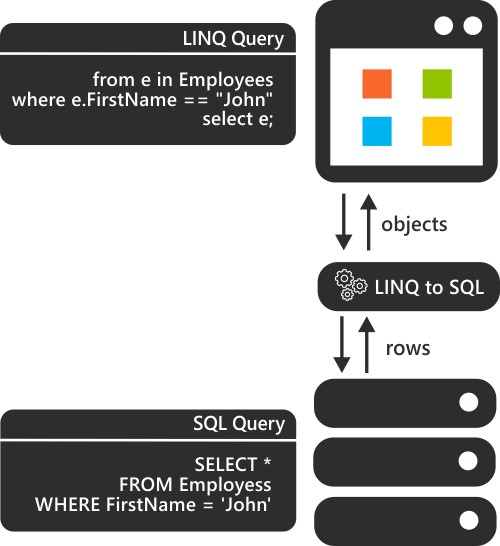
**Rezime**

* LINQ to SQL je tehnologija koja omogućava rukovanje podacima relacionih baza podataka.
* LINQ to SQL podatke tretira sa apstraktnije perspektive, što na kraju rezultira smanjenom kompleksnošću.
* Da bi se definisao jedan LINQ to SQL upit, potrebno je da postoji: klasa entiteta, data context i LINQ upit.
* LINQ to SQL obavlja konektovanje na bazu, konvertovanje LINQ konstrukcija u SQL naredbe, izvršavanje SQL naredbi, transformisanje rezultata u objekte i praćenje promena na objektima, kao i automatsko ažuriranje baze.
* Klase entiteta se koriste za kreiranje objekata u koje LINQ smešta podatke iz izvora podataka.
* Visual Studio, razvojno okruženje, poseduje funkcionalnost koja omogućava automatsko generisanje klasa entiteta.
* Object Relational Designer vrši mapiranje relacionih podataka, koji se nalaze u bazi, na objekte. Ovaj proces se naziva Object Relational Mapping i omogućava da se relacionim podacima rukuje upotrebom objekto orjentisanih principa.

LINQ to SQL je tehnologija koja omogućava rukovanje podacima relacionih baza podataka. Logički je slična tehnologiji [ADO.NET](https://www.link-elearning.com/linkdl/opisPojma.php?id=142492), ali podatke tretira sa apstraktnije perspektive, što na kraju rezultira smanjenom kompleksnošću. Korišćenjem ove tehnologije, u pozadini se sprovode mnoge operacije. Tako LINQ to SQL obavlja sledeće operacije za korisnika:

* + konektovanje na bazu,
  + konvertovanje LINQ konstrukcija u SQL naredbe,
  + izvršavanje SQL naredbi,
  + transformisanje rezultata u objekte,
  + praćenje promena na objektima i automatsko ažuriranje baze.

Opisane operacije prikazane su slikom 5.1.



*Slika 5.1 – LINQ to SQL*

Da bi se definisao jedan LINQ to SQL upit, potrebno je da postoji:

* + klasa entiteta,
  + data context,
  + LINQ upit.

**Klasa entiteta**

Klase entiteta se koriste za kreiranje objekata u koje LINQ smešta podatke iz izvora podataka. To su klase, kao i bilo koje druge klase, s tom razlikom što LINQ definiše atribute koji govore okruženju na koji način je potrebno koristiti klasu.

Primer jedne klase entiteta je sledeći:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | [Table(Name = "Employees")]      class Employee      {          [Column]          public string FirstName { get; set; }          [Column]          public string LastName { get; set; }      } |

|  |
| --- |
| ***Napomena:****za definisanje anotacija na klasi entiteta, neophodno je dodati referencu na prostor imena System.Data.Linq, koji se nalazi u sklopu System.Data.Linq.dll.* |

Atribut [Table] definiše klasu kao entitet, a poseduje opciono *Name* svojstvo, koje može biti korišćeno kako bi se definisao naziv tabele. Atribut *Column* markira polje klase kao polje koje će čuvati podatak iz tabele. Takođe, u klasi entiteta moguće je definisati i polja koja se ne mapiraju na kolone tabele, i u takvom slučaju njih će LINQ ignorisati.

**Definisanje izraza za pretragu podataka**

Prvi korak u čitanju podataka iz baze podrazumeva kreiranje data context objekta. Ovakav objekat obavlja identičnu funkciju koju obavlja objekat ADO.NET konekcije. On ne samo da upravlja konekcijom sa skladištem podataka, već vrši i prevođenje LINQ upita u SQL, prosleđivanje takvog SQL-a serveru i kreiranje objekata iz rezultata dobijenog od baze.

Sledeća linija koda ilustruje način na koji se kreira data context objekat.

|  |  |
| --- | --- |
| 1 | DataContext db = new DataContext(ConfigurationManager.ConnectionStrings["Northwind"].ConnectionString); |

|  |
| --- |
| ***Napomena:****za korišćenje ConfigurationManager klase neophodno je dodati referencu na System.Configuration prostor imena.* |

Kao što se može primetiti, konstruktor DataContext klase prihvata jedan parametar. Reč je o konekcionom stringu. Konekcioni string se čita iz konfiguracionog fajla, tako da je konfiguracioni fajl potrebno da izgleda ovako:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <configuration>    <connectionStrings>      <add name="Northwind" connectionString="Data Source=localhost;Initial Catalog=NORTHWND;Integrated Security=True" />    </connectionStrings>  ...  </configuration> |

Nakon kreiranja DataContext objekta može se pristupiti čitanju podataka. To se postiže kreiranjem specijalnog objekta koji predstavlja jednu tabelu u bazi.

|  |  |
| --- | --- |
| 1 | Table<Employee> employeeTable = db.GetTable<Employee>(); |

S obzirom na to da se u tabeli Employees, u bazi Northwind, nalaze upisane osobe, može se reći da je tabela kolekcija objekata koji su u nju mogu upisati. Tako je tip Table generička kolekcija čiji su elementi određenog tipa. U primeru, taj  tip je Employees. Tako se nad objektom tipa DataContextpoziva metoda GetTable, koja govori okruženju da je potrebno da pristupi zapisima iz tabele Employees i da referencu na njih smesti u instancu tipa Table<Employee>.

Poslednji korak u procesu definisanja pretrage nad podacima podrazumeva kreiranje samog upita.

Takva linija izgleda ovako:

|  |  |
| --- | --- |
| 1  2  3  4 | var employees =                  from e in employeeTable                  where e.FirstName.StartsWith("A")                  select e; |

Izraz upita sastoji se iz from ključne reči i tela upita. Ključna reć where se koristi za definisanje uslova po kome će podaci biti filtrirani. Tako će definisanim izrazom biti selektovani samo oni redovi u kojima vrednost kolone *FirstName* započinje karakterom *A*.Ključna reč from definiše i promenljivu koja se koristi za iteraciju kroz rezultate izraza i ona u primeru ima naziv e.

Kod kompletne demonstrativne aplikacije je sledeći:

Employee.cs:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | [Table(Name = "Employees")]      class Employee      {          [Column]          public string FirstName { get; set; }          [Column]          public string LastName { get; set; }      } |

Program.cs:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | class Program      {          static void Main(string[] args)          {              DataContext db = new DataContext(ConfigurationManager.ConnectionStrings["Northwind"].ConnectionString);              Table<Employee> employeeTable = db.GetTable<Employee>();              var employees =                  from e in employeeTable                  where e.FirstName.StartsWith("A")                  select e;              foreach (var employee in employees)              {                  Console.WriteLine("Firstname: " + employee.FirstName + " Lastname: " + employee.LastName);              }          }      } |

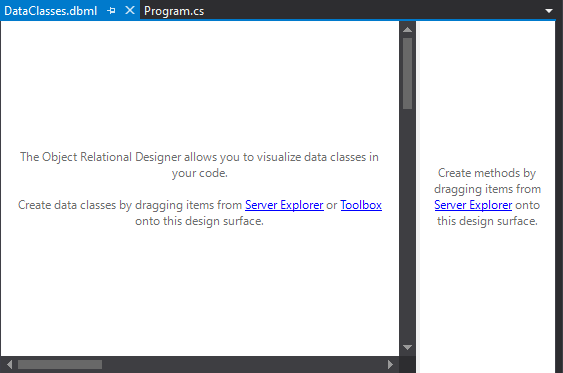
**Automatsko generisanje klasa entiteta**

Visual Studio, razvojno okruženje, poseduje funkcionalnost koja omogućava automatsko generisanje klasa entiteta, što je u dosadašnjem toku lekcije urađeno ručno. Sve što je potrebno uraditi jeste dodati novu stavku projektu, koja se temelji na šablonu *LINQ to SQL Classes*, kao na slici 5.2.

https://www.link-elearning.com/linkdl/coursefiles/1142/MDA_05_02.jpg

*Slika 5.2 – LINQ to SQL Classes šablon*

Prethodni korak će rezultovati otvaranjem *Object Relational Designer*a, koji će inicijalno biti prazan, kao na slici 5.3.



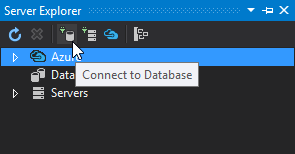
*Slika 5.3 – Object Relational Designer (ORD)*

Object Relational Designer sastoji se iz dva dela:

* + Entities deo (levo) – osnovna dizajn podloga koja prikazuje klase entiteta, asocijacije i nasleđivanja. Dozvoljava ručno kreiranje klasa, asocijacija i nasleđivanja ili kreiranje prevlačenjem is Server Explorera. Asocijacije između dve tabele automatski se dodaju u ovaj okvir, ukoliko su takve tabele povezane u bazi.
  + Methods deo (desno) – dizajn podloga koja prikazuje DataContext metode koje su mapirane sa uskladištenim procedurama i funkcijama.
  + Object Relational Designer vrši mapiranje relacionih podataka, koji se nalaze u bazi, na objekte. Ovaj proces se naziva *Object Relational Mapping* i omogućava da se relacionim podacima rukuje upotrebom objekto orjentisanih principa.
  + U Solution Exploreru Object Relational Designer se predstavlja fajlom sa ekstenzijom .dbml. Tako se Object Relational Designer može otvoriti duplim klikom na fajl sa ovom ekstenzijom.

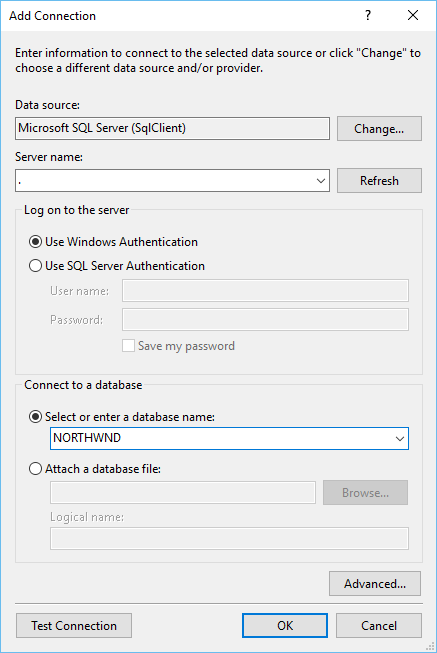
Da bi se pokrenuo proces generisanja klasa entiteta, u Entities odeljak Object Relational Designera, dovoljno je prevući tabele iz Server Explorer panela. Server Explorer se može dobiti odabirom opcije *View -> Server Explorer*.

Da bi tabele baze podataka mogle da se dodaju unutar Object Relational Designera, prethodno je potrebno uspostaviti konekciju sa bazom podataka. Za to je dovoljno odabrati opciju *Connect to Database* unutar Server Explorer panela, kao na slici 5.4.



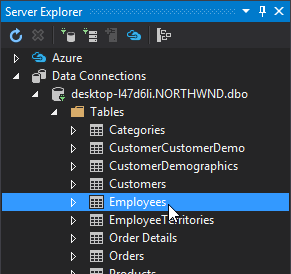
*Slika 5.4 – Kreiranje nove konekcije sa bazom*

Nakon odabira *Connect to Database* opcije otvara se prozor za konfigurisanje konekcije. Sa lokalnog SQL servera potrebno je odabrati Northwind bazu podataka, kao na slici 5.5.



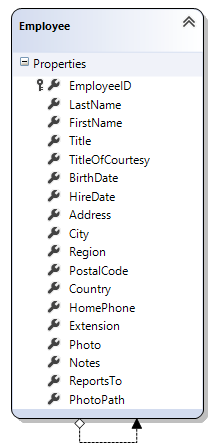
*Slika 5.5 – Konfigurisanje konekcije*

Nakon uspešno kreirane konekcije može se izvršiti dodavanje tabele unutar Object Relational Designera, kao na slici 5.6.



*Slika 5.6 – Tabele baze podataka Northwind unutar Server Explorera*

Prevlačenjem tabele Employess, unutar Object Relational Designera, biće kreiran klasni dijagram na osnovu dodate tabele, koji će izgledati kao na slici 5.7.

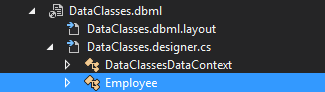


*Slika 5.7 – Klasni dijagram za Employee klasu*

Prevlačenjem tabele u OR dizajner, kreira se i odgovarajuća DataContext klasa. Visual Studio automatski kreira DataContext u sledećem formatu *<Name>DataContext*, gde je *<Name>* naziv koji je postavljen prilikom dodavanja stavke *LINQ to SQL Classes*.

DataContext klasa nalazi se unutar System.Data.Linq.DataContext prostora imena. Ova klasa odgovorna je za konektovanje aplikacije na bazu podataka. Objekti koji su kreirani za svaku tabelu, korišćenjem Object Relational Designer postaju svojstva ove klase.

Kada se fajl sa ekstenzijom .dbml, koji predstavlja Object Relational Designer sačuva, u pozadini se vrši generisanje klasa entiteta. Generisane klase se mogu pronaći unutar fajla sa ekstenzijom .designer.cs, kao na slici 5.8.



*Slika 5.8 – Generisane klase*

**Manipulacija podacima korišćenjem LINQ to SQL-a**

Korišćenjem LINQ to SQL tehnologije moguće je vršiti osnovne CRUD operacije, dodavanjem, modifikovanjem i uklanjanjem objekata iz objektnog modela. Takve operacije, LINQ to SQL transformišu u SQL upite, a podatke dobijene od servera nazad u podatke u objektnom obliku.

Za dodavanje novog objekta, DataContext klasa poseduje InsertOnSubmit, a za brisanje DeletOnSubmit metodu.

Ipak, efekat ovih metoda neće odmah biti vidljiv, odnosno navedene metode ne menjaju odmah stvarne zapise u bazi. Da bi promene uistinu bile reflektovane i na bazu podatka, potrebno je pozvati SubmitChanges metodu DataContext klase.

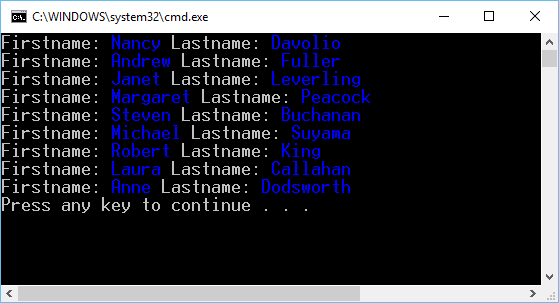
|  |
| --- |
| ***Napomena:****za pretragu podataka mogu se koristiti svi oni operatori koji su prikazani u prethodnoj lekciji.* |

**Čitanje podataka**

Vršenje bilo koje operacije nad podacima podrazumeva prethodno instanciranje DataContext klase, što je u primeru ove lekcije klasa sa nazivom DataClassesDataContext. Objekat DataContext klase sadrži kolekciju, koja predstavlja objektnu reprezentaciju svih podataka u tabeli. Stoga će se čitanje podataka sastojati iz čitanja ove kolekcije. Sledeći primer to i ilustruje.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | DataClassesDataContext dataContext = new DataClassesDataContext();              List<Employee> employees = dataContext.Employees.ToList();              foreach (var employee in employees)              {                  Console.Write("Firstname: ");                  Console.ForegroundColor = ConsoleColor.Blue;                  Console.Write(employee.FirstName);                  Console.ResetColor();                  Console.Write(" Lastname: ");                  Console.ForegroundColor = ConsoleColor.Blue;                  Console.Write(employee.LastName + "\n");                  Console.ResetColor();  } |

Prikazani kod ima efekat kao na slici 5.9.



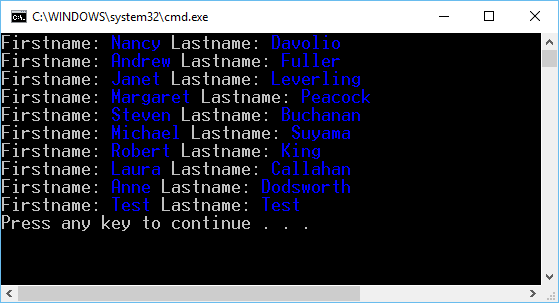
*Slika 5.9 – Pročitani podaci iz baze Northwind*

**Kreiranje zapisa**

Novi zapis tabele baze podataka može se dodati kreiranjem novog objekta i dodavanjem njega kreiranoj kolekciji objekata. Sledeći primer to ilustruje.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | DataClassesDataContext dataContext = new DataClassesDataContext();  Employee newEmployee = new Employee();  newEmployee.FirstName = "Test";  newEmployee.LastName = "Test";  dataContext.Employees.InsertOnSubmit(newEmployee);  dataContext.SubmitChanges();  List<Employee> employees = dataContext.Employees.ToList();  foreach (var employee in employees)  {      Console.Write("Firstname: ");      Console.ForegroundColor = ConsoleColor.Blue;      Console.Write(employee.FirstName);      Console.ResetColor();      Console.Write(" Lastname: ");      Console.ForegroundColor = ConsoleColor.Blue;      Console.Write(employee.LastName + "\n");      Console.ResetColor();  } |

Efekat prikazanog koda je kao na slici 5..



*Slika 5.10 – Kreiran zapis u bazi Northwind*

Na slici 5.9 se može primetiti da se sada na kraju kolekcije Employees nalaze i zaposleni sa imenom i prezimenom *Test Test,* koje smo samostalno dodali.

**Koju metodu je potrebno pozvati kako bi se obavila perzistencija promena na objektima, korišćenjem tehnologije LINQ to SQL?**

 Save

 SubmitChanges

 SaveChanges

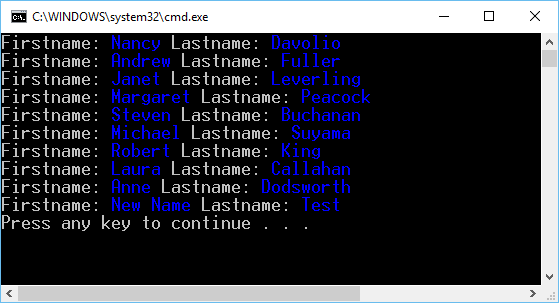
 Persist

**Ažuriranje podataka**

Procedura ažuriranja podataka podrazumeva prethodni dolazak do reference na objekat ili kolekciju objekata koju je potrebno ažurirati. Tako nešto se postiže definisanjem upita za pretragu, korišćenjem operatora predstavljanih u prethodnoj lekciji. U primeru koji sledi, prvo će biti dobavljana referenca na objekat koji predstavlja zaposlenog koga smo dodali u prethodnom koraku ove lekcije. Kada se dobavi referenca na ovog zaposlenog, biće izvršena promena njegovog imena.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | DataClassesDataContext dataContext = new DataClassesDataContext();              //First() returns one object rather than a collection. \*/              var existingEmployee = (from e in dataContext.Employees where e.FirstName.Equals("Test") where e.LastName.Equals("Test") select e).First();              /\* Change the first name of the employee. \*/              existingEmployee.FirstName = "New Name";              dataContext.SubmitChanges();              List<Employee> employees = dataContext.Employees.ToList();              foreach (var employee in employees)              {                  Console.Write("Firstname: ");                  Console.ForegroundColor = ConsoleColor.Blue;                  Console.Write(employee.FirstName);                  Console.ResetColor();                  Console.Write(" Lastname: ");                  Console.ForegroundColor = ConsoleColor.Blue;                  Console.Write(employee.LastName + "\n");                  Console.ResetColor();              } |

Nakon koda za pretragu i izmenu jednog objekta kolekcije, dodat je i kod za prikaz svih zaposlenih. Tako se možemo uveriti da je definisana promena stvarno i izvršena. Efekat koji ima prikazani kod može se videti na slici 5.11.



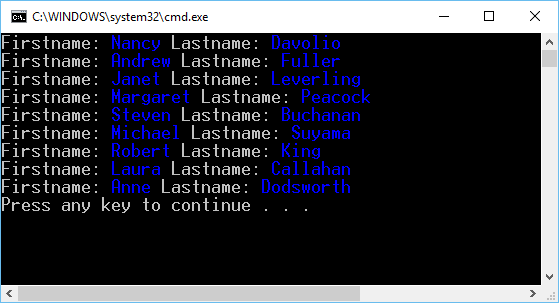
*Slika 5.11 – Ažuriran zapis u bazi Northwind*

**Brisanje podataka**

Brisanje podataka, slično ažuriranju, podrazumeva prethodni pronalazak podatka koji je potrebno obrisati, a onda i njegovo brisanje. Sledeći primer to ilustruje.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | DataClassesDataContext dataContext = new DataClassesDataContext();              var existingEmployee = (from e in dataContext.Employees where e.FirstName.Equals("New Name") where e.LastName.Equals("Test") select e).First();              dataContext.Employees.DeleteOnSubmit(existingEmployee);              dataContext.SubmitChanges();              List<Employee> employees = dataContext.Employees.ToList();              foreach (var employee in employees)              {                  Console.Write("Firstname: ");                  Console.ForegroundColor = ConsoleColor.Blue;                  Console.Write(employee.FirstName);                  Console.ResetColor();                  Console.Write(" Lastname: ");                  Console.ForegroundColor = ConsoleColor.Blue;                  Console.Write(employee.LastName + "\n");                  Console.ResetColor();              } |

Nakon koda za brisanje definisan je i kod za prikaz svih zaposlenih. Tako se možemo uveriti da se tabela za čuvanje informacija o zaposlenima vraća u oblik koji je imala na početku ove lekcije. To se može videti na slici 5.12.



*Slika 5.12 – Obrisan zapis u bazi Northwind*