**Pregled podataka**

**00:23:21**

Jedinica: 9 od 19

**Rezime**

* Tehnike upravljanja i pregleda podataka drugačije se nazivaju  CRUD operacije.
* Za rad sa podacima, EF omogućava tri različita pristupa: LINQ to Entities, Entity SQL i korišćenje sirovih SQL naredbi.
* Kada se LINQ koristi za rukovanje entitetima EF-a, tada je reč o tehnologiji LINQ to Entities.
* Entity SQL je SQL dijalekt koji se koristi za upravljanje EDM-om i veoma je sličan standardnom SQL-u, ali nema direktan uticaj na bazu podataka, već isključivo na EDM.
* EF ostavlja i mogućnost pisanja čistih SQL naredbi, koje će biti prosleđene serveru baze podataka, pa se tako podaci mogu dobiti u formi entiteta koji već postoje unutar EF-a, ali i u formi bilo kojeg drugog tipa.

Rad sa podacima uglavnom podrazumeva tri osnovna zadatka: pregled, upravljanje i manipulisanje. Da bi se sa podacima bilo šta radilo, potrebno je doći do njih. Tako se pregled podataka zasniva na kreiranju upita kojima se iz skladišta dobijaju podaci. Upravljanje podacima podrazumeva obavljanje unosa, ažuriranja i brisanja, pa se tehnike upravljanja i pregleda drugačije nazivaju i [CRUD](https://www.link-elearning.com/linkdl/opisPojma.php?id=142448) operacije. Na kraju, manipulacija podacima podrazumeva filtritranje, organizaciju i modifikovanje podataka unutar aplikacije koja takve podatke koristi, kako bi se dobio neki finalni proizvod obrade.

Lekcija pred Vama biće posvećena tehnikama koje se mogu koristiti za dobavljanje i pretragu podataka.

Za rad sa podacima, EF omogućava tri različita pristupa:

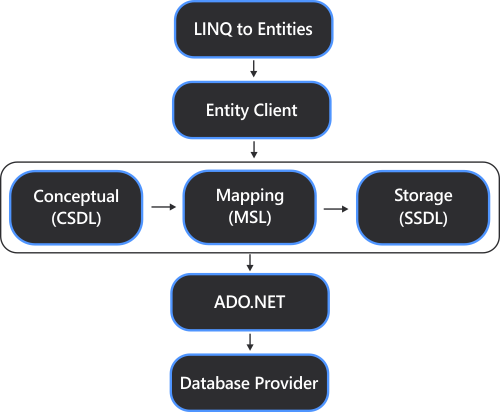
* 1. LINQ to Entities,
  2. Entity SQL,
  3. SQL.

**LINQ to Entities**

Language-Integrated Query (LINQ) je veoma moćan upitni jezik, kome je u potpunosti posvećen jedan od prethodnih modula. Kada se LINQ koristi za rukovanje entitetima EF-a, tada se govori o tehnologiji koja se zove LINQ to Entities.

LINQ to Entities omogućava brojne prednosti za developere, od kojih je jedna od najznačajnijih – jednostavnost. Tako je moguće kreirati relativno kompleksne upite, bez prethodnog detaljnog poznavanja baze podataka. Još jedna vrlo značajna prednost LINQ to Entities tehnologije je mogućnost detekcije grešaka u toku kompajliranja, ali i [IntelliSense](https://www.link-elearning.com/linkdl/opisPojma.php?id=142449) pomoć razvojnog okruženja prilikom sastavljanja upita.

Prilikom korišćenja tehnologije LINQ to Entities, developer se oslanja na komponentu koja se zove EntityClient. LINQ to Entities zapravo vrši transformisanje upita u objekat EntityClient i pozive metoda ovog tipa. EntityClient zatim povezuje napisan LINQ to Entities upit sa odgovarajućim ADO.NET provajderom. Tako EntityClient rukuje direktno konceptualnim modelom. Sve ovo je prikazano na slici 9.1.



*Slika 9.1 – Veza između LINQ to Entities upita i realnih podataka*

Na slici 9.1 se mogu videti i tri komponente koje do sada nisu pomenute. To su:

* + CSDL (Conceptual schema definition language),
  + MSL (mapping specification language),
  + SSDL (store schema definition language).

Sve tri komponente se zasnivaju na jezicima koji su derivati jezika XML, a koriste se za opis konceptulanog modela (CSDL), modela skladišta (SSDL) i za mapiranja između ova dva modela (MSL).

U jednoj EF aplikaciji metapodaci o modelu i mapiranju učitavaju se iz tri fajla sa ekstenzijama .csdl, .ssdl i .msl. EF koristi ove informacije kako bi izvršio prevođenje upita nad konceptulanim modelom u komande koje su specifične za skladište kojim se operiše.

|  |
| --- |
| ***Napomena:****u ovoj i u narednim lekcijama za demonstraciju manipulacije podacima biće korišćena baza podataka Northwind. Stoga je na početku potrebno kreirati tipove modela na osnovu ove baze podataka, na način na koji je to demonstrirano u prethodnim lekcijama.* |

U sledećem primeru preuzimaju se svi podaci Categories tabele uz pomoć LINQ-a.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | using (var nwe = new NorthwindEntities())              {  var query = from c in nwe.Categories where c.CategoryName.StartsWith("c") select c;                  List<Category> ctgs = query.ToList();                  foreach (Category c in query)    Console.WriteLine("Category name: " + c.CategoryName + " Description: " + c.Description);              } |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Napomena:****u primeru tip podatka kojim se predstavljaju podaci koji će se izvući iz DbContext objekta jeste interfejs IQueryable. Pored ovog interfejsa, u ovim situacijama može se koristiti još jedan interfejs. Reč je o interfejsu IEnumerable. Tako će na prvi pogled sledeći izrazi biti ekvivalentni:*     |  |  | | --- | --- | | 1  2 | IQueryable<Category> query = from c in nwe.Categories where  c.CategoryName.StartsWith("c") select c; |     I     |  |  | | --- | --- | | 1  2 | IEnumerable<Category> query = from c in nwe.Categories where  c.CategoryName.StartsWith("c") select c; |         *Razlika između IQueryable i IEnumerable interfejsa ogleda se u mestu na kojem se logika filtera izvršava. Korišćenjem tipa IEnumerable, svi podaci se čitaju iz baze i smeštaju u radnu memoriju. Filtriranje se zatim obavlja na klijentskoj strani.*    *Prilikom korišćenja interfejsa IQueryable filtriranje se obavlja na serveru, odnosno na bazi podataka korišćenjem SQL upita. Tako se zahtev upućuje serveru, gde se upit i izvršava, a klijentu se vraćaju samo zahtevani podaci.*    *Zaključak je da je prilikom rada sa podacima u radnoj memoriji dobar izbor IEnumerable interfejs. Ukoliko se podaci dobavljaju iz baze podataka, IQueryable je bolji izbor, pogotovu ukoliko se vrši i filtracija objekata.* |

LINQ omogućava različite pogodnosti za rukovanje podacima, kao što su filtracija, sortiranje, agregacija, spajanje. Sve ove operacije prikazane su u modulu *LINQ*.

U nastavku će biti prikazano nekoliko primera korišćenja LINQ upita za rukovanje podacima Northwind baze podataka.

*Sve narudžbine čija težina prelazi 500*:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | using (var nwe = new NorthwindEntities())              {  IEnumerable<Order> query = from o in nwe.Orders where o.Freight > 500 select o;                List<Order> orders = query.ToList();                foreach (Order order in orders)                Console.WriteLine("Ship name: " + order.ShipName);              } |

*Sve porudžbine, zajedno sa informacijama o kontakt osobi*:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | using (var nwe = new NorthwindEntities())              {  var query = from o in nwe.Orders join c in nwe.Customers on o.CustomerID equals c.CustomerID where o.Freight > 500 select new                  {                      ShipName = o.ShipName,                      ContactName = c.ContactName,                      ContactTitle = c.ContactTitle                  };                  var orders = query.ToList();                  foreach (var order in orders)  Console.WriteLine("Ship name: " + order.ShipName + " Contact Name: " + order.ContactName + " Contact Title " + order.ContactTitle);              } |

**Entity SQL**

Pored LINQ-a, podatke iz EDM-a je moguće preuzeti i putem Entity SQL-a. Entity SQL je SQL dijalekt koji se koristi za upravljanje EDM-om i veoma je sličan standardnom SQL-u. Ipak, ovaj jezik nema direktan uticaj na bazu podataka, već isključivo na EDM. Za razliku od LINQ upita, Entity SQL upiti su stringovi, zbog čega nije moguća njihova automatska sintaksna i logička korekcija uz pomoć IntelliSensea.

Da bi se kreirao i izvršio jedan Entity SQL upit, potrebno je njega kao parametar proslediti instanci ObjectQuery klase, a zatim aktivirati neki od metoda ove klase za vršenje materijalizacije.

U sledećem primeru biće kreiran jedan upit koji preuzima sve kategorije iz baze Northwind. Upit izvršava metod ToList, pretvarajući rezultat u generičku listu tipa Category. Upit je sličan standardnom SQL upitu, ali umesto da pristupa direktno podacima u bazi, on se obraća entitetima EDM-a.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | using (var nwe = new NorthwindEntities())              {                  var objctx = (nwe as IObjectContextAdapter).ObjectContext;                  ObjectQuery<Category> query = new ObjectQuery<Category>(                      "SELECT VALUE c from NorthwindEntities.Categories as c", objctx);                  List<Category> results = query.ToList();                  foreach (Category cats in results)                      Console.WriteLine(cats.Description);              } |

Entity SQL upiti podržavaju većinu funkcionalnosti standardnog SQL-a, pa je tako moguće izvršiti agregaciju, filtraciju, sortiranje i slično. Takođe, Entity SQL prepoznaje i hijerarhiju EDM-a (prepoznaje nasleđivanje). Parametri se u Entity SQL upitima označavaju prefiksom @. Na primer, ukoliko je potrebno filtrirati prethodni upit na osnovu parametra, napisali bismo ovako nešto:

|  |  |
| --- | --- |
| 1  2 | ObjectQuery<Category> query = new ObjectQuery<Category>(  "SELECT VALUE c from NorthwindEntities.Categories as c where c.CategoryId = @cid", objctx); |

U primeru parametar je predstavljen oznakom *@cid*. Vrednost ovog parametra se može postaviti korišćenjem Parameters svojstva ObjectQuery klase. Kolekcija Parameters sadrži objekte tipa ObjectParameter. U sledećem primeru postavlja se vrednost parametra sa nazivom *cid*.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | using (var nwe = new NorthwindEntities())              {                  var objctx = (nwe as IObjectContextAdapter).ObjectContext;                  ObjectQuery<Category> query = new ObjectQuery<Category>(  "SELECT VALUE c from NorthwindEntities.Categories as c where c.CategoryId = @cid", objctx);                  query.Parameters.Add(new ObjectParameter("cid", 5));                  List<Category> results = query.ToList();                  foreach (Category cats in results)                      Console.WriteLine(cats.Description);              } |

Korišćenje Parameters kolekcije nije neophodno i parametre je moguće pridružiti upitu prostom [konkatinacijom](https://www.link-elearning.com/linkdl/opisPojma.php?id=142450), kao u sledećem primeru:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | using (var nwe = new NorthwindEntities())              {                  var objctx = (nwe as IObjectContextAdapter).ObjectContext;                  int cid = 5;                  ObjectQuery<Category> query = new ObjectQuery<Category>(  "SELECT VALUE c from NorthwindEntities.Categories as c where c.CategoryId = " + cid, objctx);                  List<Category> results = query.ToList();                  foreach (Category cats in results)                      Console.WriteLine(cats.Description);              } |

U nastavku lekcije biće prikazano nekoliko primera manipulacije podacima, korišćenjem Entity SQL-a.

*Sortiranje narudžbina po težini, opadajućim redosledom*:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | using (var nwe = new NorthwindEntities())              {                  var objctx = (nwe as IObjectContextAdapter).ObjectContext;                  ObjectQuery<Order> query = new ObjectQuery<Order>(  "SELECT VALUE o from NorthwindEntities.Orders as o order by o.Freight desc", objctx);                  List<Order> results = query.ToList();                  foreach (Order order in results)   Console.WriteLine("Ship name: " + order.ShipName + " Freight: " + order.Freight);              } |

Često je u realnim aplikacijama potrebno postići efekat straničenja. Tako nešto podrazumeva prikaz podataka ne više stranica, pri čemu bi se na jednoj stranici prikazivao limitiran broj zapisa. To je moguće postići kombinovanjem ključnih reči skip i limit. Ukoliko pretpostavimo da se na jednoj stranici prikazuje 10 zapisa, sadržaj za četvrtu stranicu može da se dobije na sledeći način:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | using (var nwe = new NorthwindEntities())              {                var objctx = (nwe as IObjectContextAdapter).ObjectContext;                ObjectQuery<Order> query = new ObjectQuery<Order>(  "SELECT VALUE o from NorthwindEntities.Orders as o order by o.Freight desc skip 30 limit 10", objctx);                List<Order> results = query.ToList();                foreach (Order order in results)  Console.WriteLine("Ship name: " + order.ShipName + " Freight: " + order.Freight);              } |

**EntityConnection**

Kreiranje Entity SQL upita moguće je postići i korišćenjem EntityConnection i EntityCommand tipova. Manipulacija podacima na ovaj način odvija se u nekoliko koraka:

* + kreiranje konekcije na osnovu adekvatnog konekcionog stringa;
  + otvaranje konekcije;
  + kreiranje objekta komande i definisanje eventualnih parametara;
  + izvršavanje komade;
  + manipulacija podacima;
  + zatvaranje konekcije.

U nastavku biće opisani navedeni koraci.

**Kreiranje konekcije**

Prvi korak prilikom rada sa podacima jeste kreiranje objekta konekcije. Konekcija se ostvaruje upotrebom EntityConnection klase, odnosno upotrebom njenog metoda **Open**. Da bi klasa bila instancirana, mora joj biti prosleđen konekcioni string kroz konstruktor. Konekcioni string možemo uneti direktno u konstruktor, ali je daleko praktičnija varijanta da se unese samo referenca na konekcioni string koji se nalazi unutar konfiguracionog fajla aplikacije (*App.config ili Web.config*). Ukoliko je EDM generisan na osnovu postojećeg logičkog modela (baze), tada je najverovatnije generisan i konekcioni string. U suprotnom, konekcioni string je potrebno kreirati ručno.

Referenca na konekcioni string konstruktoru klase EntityConnection, prosleđuje se po nazivu. Ako je naziv konekcionog stringa *NorthwindEntities*, kao parametar potrebno je  proslediti sledeći string:

name=NorthwindEntities

Ako je naziv konekcionog stringa *AdventureWorksEntities*, tada je potrebno proslediti sledeći string:

name=AdventureWorksEntities

Kreiranje konekcije za NorthwindEntities može konačno izgledati ovako:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | EntityConnection conn = new EntityConnection("name=NorthwindEntities");  Ipak, konekcija se najčešće kreira na sledeći način:              using (EntityConnection conn = new EntityConnection("name=NorthwindEntities"))  {  ...  } |

U ovom slučaju aplikacija će se ponašati identično kao i u prethodnom, ali će konekcioni objekat biti automatski uništen nakon napuštanja bloka. Ovo je naročito korišćena tehnika kada se rukuje resursima (bazom ili fajlovima), jer automatski prekida konekciju sa resursom i samim tim smanjuje mogućnost [*curenja* konekcije](https://www.link-elearning.com/linkdl/opisPojma.php?id=142488).

**Otvaranje konekcije**

Kada je konekcija kreirana, ona se može otvoriti pozivom metode Open.

|  |  |
| --- | --- |
| 1  2  3  4 | using (EntityConnection conn = new EntityConnection("name=NorthwindEntities"))              {                  conn.Open();              } |

**Kreiranje komandnog objekta**

Nakon kreiranja i otvaranja konekcije moguće je preći na kreiranje komandnog objekta. Komanda se predstavlja tipom EntityCommand. Objekat komande zahteva konekcioni objekat i upit. U sledećem primeru kreirana je komanda koja vraća sve kategorije. Konstruktoru su prosleđena dva parametra. Upit (queryString) i konekcija.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | using (EntityConnection conn = new EntityConnection("name=NorthwindEntities"))              {                  conn.Open();                  string queryString =      "SELECT VALUE c from NorthwindEntities.Categories as c";                  EntityCommand cmd = new EntityCommand(queryString, conn);              } |

Ukoliko je upitu potrebno pridružiti i neke parametre, to je moguće uraditi uz pomoć Parameters kolekcije, baš kao što je to urađeno nešto ranije u ovoj lekciji. Parameters kolekcija sadrži **EntityParameter** objekte. Razlika između EntityParameter objekta i ostalih parametarskih objekata (nešto ranije su to bili objekti tipa ObjectParameter) je u tome što EntityParameter ne može da prihvati naziv i vrednost parametra direktno kroz konstruktor. Stoga se ova dva svojstva moraju pridružiti naknadno.

Sledećim primerom prikazuju se kategorije čiji je id 5. Ova filtracija realizuje se pomoću EntityParameter parametra.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | using (EntityConnection conn = new EntityConnection("name=NorthwindEntities"))              {                  conn.Open();                  string queryString =      "SELECT VALUE c from NorthwindEntities.Categories as c where c.CategoryID = @cid";                  EntityCommand cmd = new EntityCommand(queryString, conn);                  EntityParameter p = new EntityParameter();                  p.ParameterName = "cid";                  p.Value = 5;                  cmd.Parameters.Add(p);              } |

**Izvršavanje komande**

Kada je konekcija uspešno uspostavljena i komandni objekat kreiran, komanda može biti izvršena. Postoje tri načina na koja komanda može biti izvršena. Bez povratne vrednosti, sa skalarnom povratnom vrednošću (samo jedna vrednost) i sa povratnom vrednošću u obliku DataReader toka. Svaki od ovih načina aktivira se odgovarajućom metodom. Za aktivaciju komande bez povratne vrednosti koristi se ExecuteNonQuery metod. Ovaj metod se koristi kada je potrebno izvršiti ažuriranje ili brisanje podataka iz baze.

ExecuteScalar je metod koji vraća jednu vrednost. Ako izvršimo upit koji će vratiti više redova sa više kolona, ovaj metod će vratiti samo prvu kolonu prvog reda. Prema tome, najčešće se koristi za izvršavanje [agregatnih funkcija](https://www.link-elearning.com/linkdl/opisPojma.php?id=142451).

Najčešće korišćeni metod komandnog objekta je ExecuteQuery, koji vraća DataReader objekat kao rezultat, i svi podaci iz baze dalje se pregledaju uz pomoć dobijenog DataReader objekta. DataReader tretira dobijene podatke kao tok, i čita ih po [forward only/fast forward](https://www.link-elearning.com/linkdl/opisPojma.php?id=142489) principu.

Sledeći primer upravo ilustruje izvršavanje komande upotrebom metode ExecuteReader.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | using (EntityConnection conn = new EntityConnection("name=NorthwindEntities"))              {                  conn.Open();                  string queryString =      "SELECT VALUE c from NorthwindEntities.Categories as c where c.CategoryID = @cid";                  EntityCommand cmd = new EntityCommand(queryString, conn);                  EntityParameter p = new EntityParameter();                  p.ParameterName = "cid";                  p.Value = 5;                  cmd.Parameters.Add(p);                  EntityDataReader rdr = cmd.ExecuteReader(      System.Data.CommandBehavior.SequentialAccess);                  byte[] bfr = new byte[1024];                  while (rdr.Read())                  {                      Console.WriteLine("Category: ");                      Console.Write("id: " + rdr.GetInt32(0));                      Console.Write(" Name: " + rdr["CategoryName"]);                      Console.Write(" Description: " + rdr[2] + "\n");                  }                } |

EntityDataReader koristi metod Read za čitanje podatka. Metod Read vraća sadržaj sledećeg reda. Za kretanje kroz redove EntityDataReader objekta koristi se marker koji određuje njegovu poziciju. Na početku čitanja, ovaj marker se nalazi na praznom prostoru, ispred svih redova tabele. Prilikom prve upotrebe metode Read, marker se pomera na sledeći red (prvi red u rezultatima) i vraća njegov sadržaj (polja). Ova polja je moguće preuzeti na različite načine – kroz [asocijativni niz](https://www.link-elearning.com/linkdl/opisPojma.php?id=142452), [indeksirani niz](https://www.link-elearning.com/linkdl/opisPojma.php?id=142453) ili upotrebom različitih Get metoda izloženih od strane DataReader klase. U upravo prikazanom primeru upotrebljene su sve nabrojane varijante.

Metod Read vraća rezultat tipa boolean, zbog čega je vrlo pogodan za korišćenje unutar while petlje.

|  |
| --- |
| ***Napomena:****Metodi ExecuteReader je neophodno proslediti CommandBehavior.SequentialAccess enumeraciju kako bi čitanje moglo da bude ispravno izvršeno.* |

**SQL**

Na kraju, EF ostavlja mogućnost i pisanja čistih SQL naredbi koje će biti prosleđene serveru baze podataka. Podaci se mogu dobiti u formi entiteta koji već postoje unutar EF-a, ali i u formi bilo kojeg drugog tipa. Sledećih nekoliko primera ilustrovaće tehnike za upućivanje sirovih SQL upita korišćenjem EF-a.

**Čitanje entiteta**

Čitanje entiteta se može obaviti na sledeći način:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | using (var nwe = new NorthwindEntities())              {                  var query = nwe.Orders.SqlQuery("SELECT \* from dbo.Orders");                  List<Order> results = query.ToList();                foreach (Order order in results)  Console.WriteLine("Ship name: " + order.ShipName + " Freight: " + order.Freight);              } |

Izvorne SQL naredbe formiraju se korišćenjem SqlQuery metode, koja se poziva nad svojstvima DbContext klase. Metoda SqlQuery vraća odgovarajući tip entiteta, baš kao što je to bio slučaj i sa LINQ naredbama. Ipak, za uspešno kreiranje entiteta, neophodno je da selektovane kolone odgovaraju svojstvima koje entitet poseduje, inače će u protivnom doći do izbacivanja izuzetka. Tako će sledeći primer stvoriti izuzetak.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | using (var nwe = new NorthwindEntities())              {  var query = nwe.Orders.SqlQuery("SELECT ShipName, Freight from dbo.Orders");                  List<Order> results = query.ToList();                  foreach (Order order in results)  Console.WriteLine("Ship name: " + order.ShipName + " Freight: " + order.Freight);              } |

**Čitanje ostalih tipova podataka**

Upućivanjem sirovih SQL naredbi, nije neophodno dobijene podatke tretirati kao entitete. To ilustruje sledeći primer, u kome se pročitani podatak smešta u jednu promenljivu tipa string.

|  |  |
| --- | --- |
| 1  2  3  4  5 | using (var nwe = new NorthwindEntities())              {  string shipName = nwe.Database.SqlQuery<string>("SELECT ShipName from dbo.Orders").FirstOrDefault();                  Console.WriteLine("Ship name: " + shipName);              } |

SQL naredbe koje vraćaju instance bilo kojeg tipa, uključujući i primitivne tipove, kreiraju se korišćenjem SqlQuery metode, koja se ovog puta poziva nad Database klasom.

**Kreiranje sirovih komandi**

U svim prikazanim prethodnim primerima demonstrirano je kreiranje sirovih upita za dobavljanje podataka. Slede primeri koji ilustruju kreiranje sirovih komandi za unos, ažuriranje i brisanje.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | using (var ctx = new TestEntities())              {                  //Insert command  int noOfRowInserted = ctx.Database.ExecuteSqlCommand("INSERT INTO student(studentname) VALUES('New Student')");                  //Update command  int noOfRowUpdated = ctx.Database.ExecuteSqlCommand("UPDATE student SET studentname = 'changed student by command' WHERE studentid = 1");                  //Delete command  int noOfRowDeleted = ctx.Database.ExecuteSqlCommand("DELETE FROM student WHERE studentid = 1");              } |

Kako što se može videti, za kreiranje sirovih SQL naredbi, koristi se metoda ExecuteSqlCommand, klase Database.

|  |
| --- |
| ***Napomena:****upiti su demonstrativnog karaktera i ne odnose se na Northwind bazu podataka.* |

**Koji pristup za rad sa podacima omogućava proveru napisanih upita korišćenjem IntelliSensea?**

 LINQ to Entities

 Entity SQL

 SQL