**Osnove rada sa podacima korišćenjem EF-a**

**00:12:45**

Jedinica: 8 od 19

**Rezime**

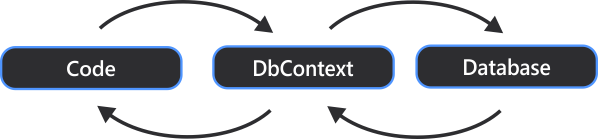
* Osnovna svrha EF-a jeste da omogući rukovanje podacima kao da su objekti.
* Osnovna klasa za rukovanje entitetima EF-a jeste DbContext.
* Klasa koja nasleđuje DbContext klasu naziva se kontekstna klasa ili jednostavno kontekst.
* DbContext klasa se nalazi u prostoru imena System.Data.Entity.
* DbContext je odgovoran za pregled i pretragu podataka, praćenje promena, perzistenciju, keširanje, upravljanje relacijama i materijalizaciju objekata.
* EF poznaje POCO i dinamičke proxy entitete.
* Entiteti mogu imati skalarna, navigaciona i kompleksna svojstva.
* Tokom životnog veka svaki entitet može da prođe kroz nekoliko različitih stanja, u zavisnosti od operacija koje su izvršene nad njim.
* Stanje entiteta se izražava enumeracijom tipa System.Data.Entity.EntityState.

U prethodnim lekcijama demonstrirano je kako se na osnovu postojeće baze podataka može izvršiti generisanje klasa za rukovanje podacima u objektom maniru. Prikazano je i to kako se na osnovu modela može kreirati i baza i klase domenskog modela. Takođe, u prethodnoj lekciji objašnjene su brojne tehnike za modelovanje različitih situacija prilikom rada sa podacima. Ipak, ni u jednoj od prethodnih lekcija nije izvršeno stvarno rukovanje podacima. Stoga će to biti predmet ovog modula.

**DBContext**

Osnovna svrha EF-a je da omogući rukovanje podacima kao da su objekti. Zbog toga EF tokom generisanja objektnog modela, kreira objektne ekvivalente pojmovima koji postoje u bazi podataka. Svim takvim objektima se rukuje centralizovano.

Osnovna klasa za rukovanje entitetima EF-a jeste DbContext. U prethodnoj lekciji ste mogli da vidite da se ova klasa ne koristi direktno, već kao osnovni tip za izvedenu klasu koja se generiše u procesu mapiranja. Takva klasa koja nasleđuje DbContext klasu naziva se kontekstna klasa ili jednostavno kontekst. Slika 8.1 ilustruje ulogu kontekstne klase u radu sa podacima.



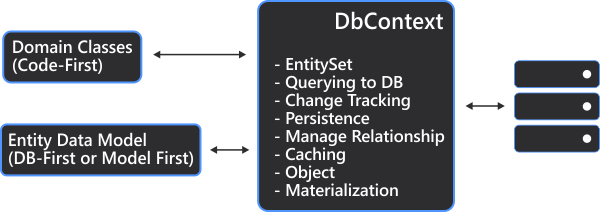
*Slika 8.1 – Uloga DbContext tipa prilikom rada sa podacima*

DbContext klasa se nalazi u prostoru imena System.Data.Entity.

U verziji pre EF 4.1, EDM je generisao kontekstne klase na osnovu tipa ObjectContext. U novijim verzijama za rukovanje podacima, kao osnovna klasa se koristi DbContext, jednostavnija i sveobuhvatnija alternativa. DbContext je konceptualno sličan ObjectContext tipu i može se reći da predstavlja omotač za funkcionalnosti definisane u starijem tipu.

**Koja je uloga DBContext tipa?**

DbContext je esencijalni sastojak Entity Frameworka, odgovoran za rukovanje podacima kao objektima. Ova klasa predstavlja most između domenskog modela i baze podataka. Slika 8.2 ilustruje takvu ulogu DbContext klase.



*Slika 8.2 – DbContext*

Sa slike 8.2 može se videti da je, bez obzira na pristup koji se koristi prilikom upotrebe EF-a, DbContext osnovni element za interakciju sa podacima.

DbContext je odgovoran za sledeće aktivnosti:

* + **pregled podataka** – DbContext poseduje kolekcije entiteta na osnovu podataka u bazi, koje predstavlja tipom DbSet;
  + **pretraga podataka** – DbContext pretvara LINQ to Entities i Entity SQL upite u izvorne SQL upite, koje upućuje bazi podataka;
  + **praćenje promena** – DbContext prati sve promene koje nastaju na entitetima koji se nalaze unutar kontekstnog objekta;
  + **perzistencija** – klasa DbContext je zadužena i za sprovođenje osnovnih operacija nad podacima, kao što su Insert, Update i Delete;
  + **keširanje** – DbContext obavlja keširanje podataka podrazumevano, tako što čuva entitete koji se nalaze u kontekstu unutar radne memorije;
  + **upravljanje relacijama**;
  + **materijalizacija objekata** – DBContext konvertuje sirove podatke iz tabela u objekte entiteta.

**Koja klasa za rukovanje podacima se može nazvati osnovnom, roditeljskom klasom za rukovanje podacima?**

 DbContext

 DbSet

 ObjectContext

 Nijedna od navedenih

**Instanciranje DbContext klase**

Funkcionalnosti DbContext klase se mogu koristiti instanciranjem kontekstne klase koju EF generiše prilikom mapiranja. Sledeći blok koda ilustruje primer instanciranja ove klase.

|  |  |
| --- | --- |
| 1  2  3  4 | using (var ctx = new NorthwindEntities())  {      // code for data manipulation goes here  } |

**Pristup ObjectContext API-u**

Iako je API koji klasa DbContext izlaže znatno jednostavniji za korišćenje za obavljanje svih osnovnih operacija, nekada se može javiti potreba za korišćenjem funkcionalnosti koje su definisane ObjectContext API-em. Zbog toga je moguće dobiti referencu na ObjectContext objekat iz objekta DBContext tipa. To se može postići korišćenjem interfejsa IObjectContextAdpter, kao u primeru:

|  |  |
| --- | --- |
| 1  2  3  4 | using (var ctx = new NorthwindEntities())              {                  var objectContext = (ctx as IObjectContextAdapter).ObjectContext;              } |

**Dodavanje entiteta u kontekst**

Već više puta je rečeno da se korišćenjem EF-a podacima rukuje kao objektima. Kada se neki podatak pročita iza baze, referenca na njega se smešta unutar konteksta. Sledeći kod to i ilustruje:

|  |  |
| --- | --- |
| 1  2  3  4 | using (var ctx = new NorthwindEntities())              {                  Employee employee = ctx.Employees.Find(1);              } |

U prikazanom primeru iskorišćena je metoda Find za čitanje entiteta sa id vrednosti 1. Na ovaj način je unutar konteksta dodat jedan entitet.

**Tipovi entiteta**

EF generalno poznaje dva tipa entiteta:

* + POCO entitete,
  + Dinamičke proxy entitete.

POCO je skraćenica koja se odnosi na pojam *Plain Old CLR Objects*, a označava klasu koja ne zavisi od bilo koje klase specifične za neki framework. Tako bi primer jedne POCO klase bio sledeći:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | public class Person      {          public int id { get; set; }          public string first\_name { get; set; }          public string last\_name { get; set; }          public System.DateTime date\_of\_birth { get; set; }          public Nullable<int> is\_deleted { get; set; }      } |

Pored POCO entiteta EF poznaje i Dynamic Proxy entitete. Oni su neka vrsta entiteta omotača za malopre opisane POCO entitete, a omogućavaju dodatne funkcionalnosti kao što su odloženo učitavanje (lazy loading) i automatsko praćenje promena. Svi entiteti u EF-u su podrazumevano Dynamic Proxy entiteti.

Sledeći primer ilustruje čitanje tipa entiteta koji je dodat unutar konteksta.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | using (var ctx = new NorthwindEntities())              {                  Employee employee = ctx.Employees.Find(1);                  var entityType = ObjectContext.GetObjectType(employee.GetType());                  Console.WriteLine("Entity type: " + employee.GetType().FullName);                  Console.WriteLine("Actual Entity type: " + entityType.Name);              } |

Prikazani kod rezultuje sledećim izlazom:

Entity type:  
System.Data.Entity.DynamicProxies.Employee\_52FC80EEBF7DC335DE27131A  
EB746C4A2B004F5849005FFB380F6D57805F8329  
Actual Entity type: Employee

Prilično je jasno da se prvom linijom za ispis dobija tip entiteta. Reč je naravno o Dynamic Proxy entitetu, što se jasno može videti po prostoru imena u kome se nalazi ovakav [dinamički tip](https://www.link-elearning.com/linkdl/opisPojma.php?id=142487).

Da bi se dobio stvarni tip entiteta, u primeru se koristi ObjectContext klasa i njena metoda GetObjectType. Tako druga linija za ispis emituje stvarni tip entiteta, što je naravno Employee.

**Svojstva entiteta**

Entiteti mogu imati tri vrste svojstava:

* + Skalarna (Scalar properties),
  + Navigaciona (Navigation properties),
  + Kompleksna (Complex properties).

Skalarna svojstva predstavljaju vrednosti koje su sadržane unutar entiteta. Tako entitet Person poseduje svojstvo kao što su *PersonId* ili *FirstName*, što su sve primeri skalarnih svojstava, koja odgovaraju kolonama u Person tabeli.

Navigaciona svojstva su pokazivači na povezane entitete. Tako entitet Author može imati svojstvo Titles, koje će omogućiti aplikaciji navigaciju sa Author entiteta na entitet Title.

Kompleksna svojstva su ona koja omogućavaju skalarnim svojstvima da budu organizovana unutar jednog entiteta. Baš kao i samo entiteti, i kompleksni tipovi se sastoje od skalarnih svojstava ili drugih kompleksnih tipova, ali ne i od navigacionih svojstava. Ona ne mogu ući u sastav jednog kompleksnog svojstva. Kompleksni tipovi ne mogu imati identifikacioni ključ, zato što EF njima ne može da rukuje van konteksta entiteta u kome se nalaze.

**Životni vek entiteta**

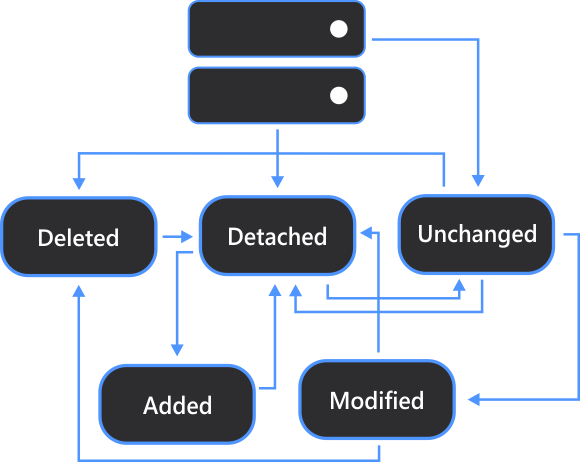
Pre nego što se pređe na demonstraciju obavljanja osnovih CRUD operacija korišćenjem EF-a, veoma je bitno razumeti životni vek entiteta i način na koji njima rukuje EF.

Tokom svog životnog veka svaki entitet može da prođe kroz nekoliko različitih stanja, u zavisnosti od operacija koje su izvršene nad njim. Stanje entiteta se izražava enumeracijom tipa System.Data.Entity.EntityState. Vrednosti ove enumeracije prikazane su u tabeli 8.1.

|  |  |
| --- | --- |
| **Stanje** | **Opis** |
| **Added** | Objekat je nov, dodat u kontekst, a metoda SavaChanges još nije pozvana. Nakon čuvanja promena stanje objekta se menja u Unchanged. |
| **Deleted** | Objekat je obrisan iz konteksta. Nakon čuvanja izmena stanje objekta prelazi u Detached. |
| **Detached** | Objekat postoji, ali se ne prati i nije u kontekstu. Postoji nekoliko različitih scenarija koji mogu objekat dovesti u ovakvo stanje. Objekat je u ovakvom stanju odmah nakon kreiranja, ali pre ubacivanja u kontekst. Objekat je takođe u ovakvom stanju nakon pozivanja metode Detach. |
| **Modified** | Kada se vrednost nekog od skalarnih svojstava objekta promeni, a metoda SaveChanges još nije pozvana, entitet je u ovom stanju. Kod POCO entiteta, stanje se menja u Modified kada se pozove metoda DetectChanges. U svakom slučaju, kada se napravljene promene sačuvaju, stanje entiteta se menja u Unchanged. |
| **Unchanged** | Objekat nije modifikovan od kada je dodat u kontekst ili od poslednjeg poziva metode SaveChanges. |

*Tabela 8.1 – Vrednosti EntityState enumeracije*

Opisana stanja i njihove međusobne povezanosti prikazane su slikom 8.3.



*Slika 8.3 –Stanja entiteta*

Kontekstni objekat čuva referencu na sve objekte pročitane iz baze podataka. Pored reference, unutar konteksta čuvaju se i neke dodatne informacije, kao što su stanje entiteta, ali i modifikacije koje su načinjene na vrednostima svojstava entiteta.

Promena stanja sa *Unchanged* na *Modified* je jedina promena stanja koja će automatski biti obavljena od strane okruženja. Sve ostale promene moraju se eksplicitno izazvati korišćenjem odgovarajućih metoda klasa DbContext i DbSet.

|  |
| --- |
| ***Napomena:****o DbSet klasi biće više reči u jednoj od narednih lekcija ovog modula.* |

**Informacije o entitetima**

Klasa DbEntityEntry omogućava čitanje različitih informacija o entitetima. Instanca ove klase za određeni entitet se može dobaviti korišćenjem metode Entry, DbContext klase. Sledeći primer to ilustruje:

|  |  |
| --- | --- |
| 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 | using (var ctx = new NORTHWNDEntities())              {                  Employee employeeEntity = ctx.Employees.Find(5);                  if (employeeEntity == null)                  {                      return;                  }                  //edit employee name                  employeeEntity.FirstName = "Edited First name";                  //get DbEntityEntry object for employeeEntity                  DbEntityEntry employeeEntry = ctx.Entry(employeeEntity);                  //get current EntityState                  Console.WriteLine("Entity State: {0}", employeeEntry.State);                  Console.WriteLine("\*\*\*\*\*\*\*\*Property Values\*\*\*\*\*\*\*\*");  foreach (var propertyName in employeeEntry.CurrentValues.PropertyNames)                  {                      Console.WriteLine("Property Name: {0}", propertyName);                      //get original value                      var orgVal = employeeEntry.OriginalValues[propertyName];                      Console.WriteLine("Original Value: {0}", orgVal);                      //get current values                      var curVal = employeeEntry.CurrentValues[propertyName];                      Console.WriteLine("Current Value: {0}", curVal);                  }              } |

Efekat prikazanog koda se može videti na slici 8.4.

