**Konkurencija**

**00:16:20**

Jedinica: 17 od 19

**Rezime**

* Problem konkurentnosti nastaje kada više korisnika pristupi istim podacima i pritom svaki od njih želi da ih modifikuje.
* Dva osnovna modela konkurencije su: pesimistički konkurentni model i optimistički konkurentni model.
* Po pesimističkom konkurentnom modelu, ukoliko je jednoj transakciji potreban pristup podacima kojima rukuje neka druga transakcija, podaci su zaključani sve dok prva transakcija ne bude završena ili opozvana.
* Kod optimističkog konkurentnog modela zaključavanje (lock) se ne koristi.
* U Entity Frameworku moguće je rešiti konflikte rukovanjem OptimisticConcurrencyException izuzetkom koji izaziva Entity Framework.
* Svojstvo ConcurrencyMode svojstva entiteta se koristi za markiranje svojstva za proveru konkurentnosti.
* Jedan od načina za rešavanje problema konkurentnosti jeste korišćenje metode Reload, koja obavlja prepisivanje vrednosti entiteta sa onima koje su pročitane iz baze.
* Kada se koristi pristup Client Wins, vrednosti u bazi podataka se prepisuju vrednostima koje postoje unutar entiteta.

Aplikacije koje rade sa bazama podataka i koje ujedno podržavaju višekorisnički pristup mogu se suočiti sa problemima konkurentnosti – odnosno problemom koji nastaje kada više korisnika pristupi istim podacima i pritom svaki od njih želi da ih modifikuje.

Entity Framework obezbeđuje optimistički konkurentni model kako bi omogućio prevazilaženje ovakvih problema. Stoga će cilj ove lekcije biti demonstracija procesa kojim se jedna transakcija izoluje i obezbeđuje.

**Konkurentni modeli**

U situacijama kada više korisnika može da pristupi jednom istom resursu, može se govoriti o modelima konkurencije. Dva osnovna modela konkurencije su sledeća:

* + *Pesimistički konkurentni model.* Ukoliko je jednoj transakciji potreban pristup podacima kojima rukuje neka druga transakcija, podaci su zaključani sve dok prva transakcija ne bude završena ili opozvana. Transakcija koja pokušava da pristupi zaključanim podacima mora da čeka sve dok se podaci ne otključaju, odnosno dok korisnik koji ih je zaključao ne završi sa modifikacijama. Ovo dalje znači da neke transakcije, za koje je potrebno više vremena, mogu narušiti performanse i skalabilnost.
  + *Optimistički konkurentni model.*U ovom modelu zaključavanje (lock) se ne koristi. Umesto toga se, prilikom ažuriranja podataka, prvo proverava da li došlo do promene vrednosti od poslednjeg čitanja. Ukoliko je do promena došlo, izbacuje se greška.
  + Kao što je već rečeno, Entity Framework obezbeđuje optimistički konkurentni model.

**Problem konkurencije**

Kada dva ili više korisnika modifikuju iste podatke u isto vreme, dolazi do konflikta. Takvim konfliktima je potrebno upravljati i omogućiti aplikaciji da se oporavi neprimetno i nastavi sa radom.

Bilo koja višekorisnička aplikacija koja rukuje podacima mora imati isplaniranu strategiju za rešavanje problema konkurentnosti. Entity Framework ima set definisanih pravila koja primenjuje za detekciju mogućih konflikata. Pre nego što se pređe na njihovu demonstraciju, dobro je podsetiti se postupka koji koristi Entity Framework za rad sa podacima:

* 1. podaci se učitavaju u kontekstni objekat tipa [DbContext](https://www.link-elearning.com/linkdl/opisPojma.php?id=142493),
  2. vrši se modifikacija podataka unutar konteksta i pritom objekat tipa DbContext prati sve promene koje se događaju nad podacima,
  3. promene napravljene nad podacima unutar konteksta se preslikavaju na bazu, u trenutku poziva metode SaveChanges.

Prilikom obavljanja poslednje, treće navedene stavke, može doći do konflikta, jer postoji mogućnost da je neki drugi korisnik u međuvremenu već izvršio modifikovanje podataka.

**EF za rešavanje konflikata nastalih usled problema konkurencije koristi:**

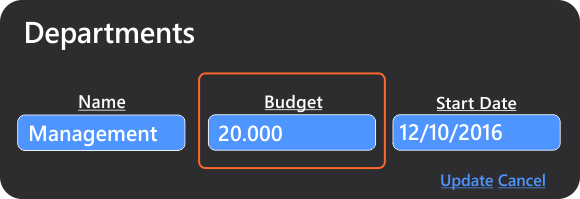
 pesimistički konkurentni model

 optimistički konkurentni model

**Optimistički konkurentni model**

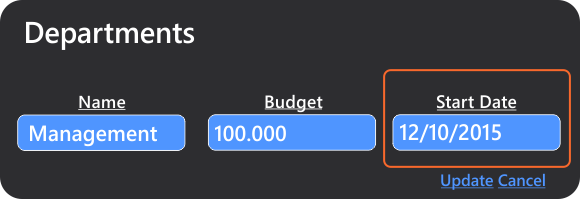
U narednim redovim biće objašnjeno kako optimistički konkurentni model rešava opisane probleme. Da bi sve bilo jasnije, biće demonstriran jedan primer.

Jovana otvara aplikaciju kako bi umanjila sumu budžeta sa 100.000 evra na 20.000 za odsek *Management*, kao na slici 17.1.



*Slika 17.1 – Promena koju je napravila Jovana*

Pre nego što Jovana klikne na *Update*, Milan pokreće istu aplikaciju, i menja *Start Date* polje sa 12.10.2016. na 12.10.2015 za odsek *Management*, kao na slici 17.2.



*Slika 17.2 – Promena koju je napravio Milan*

Jovana će sada kliknuti na Update, a tek posle nje i Milan će kliknuti na Update. Kada neki korisnik sada pristupi tim podacima, on će videti da je vrednost budžeta za ovaj odsek 100.000, što je netačno, jer ju je Jovana promenila.

Neke od akcija koje se mogu preduzeti da bi se rešio ovaj problem su sledeće:

* + Moguće je pratiti koje svojstvo je korisnik modifikovao, a onda ažurirati samo te kolone u bazi. Tako u ovom primeru ništa ne bi bilo izgubljeno, jer su ova dva korisnika modifikovala različita svojstva. Tako da, kada bi neko sledeći put pristupio ovom odseku, video bi i datum 12.10.2015, a i 20.000 evra.
  + Ovo je podrazumevano ponašanje Entity Frameworka i može smanjiti značajan broj konflikata koji rezultuju gubitkom podataka. Međutim, ovo ponašanje ne izbegava gubljenje podataka ako su promene napravljene nad istim svojstvom entiteta.
  + Takođe, ovo ponašanje nije uvek moguće. Kada se primenjuje uskladištena procedura nad nekim entitetom iz baze, sva svojstva entiteta će biti ažurirana, bez obzira na stvarni broj promenjenih polja.
  + Moguće je dozvoliti da Milanova promena prekorači Jovaninu promenu. Nakon što Milan klikne na Update, budžet će se vratiti na 20.000. Ovakav scenario se zove *Client Wins* ili *Last in Wins*.
  + Moguće je sprečiti da se Milanova promena ažurira u bazi podataka. Tipično, prikazala bi se poruka o grešci i trenutno stanje podataka, a zatim bi se omogućio ponovni unos promena, ukoliko i dalje postoji potreba za tim. Ovakav pristup se zove *Store Wins*, što znači da vrednosti iz izvora podataka imaju prednost nad vrednostima koje je prosledio klijent.

**Detektovanje konkurentnih konflikata**

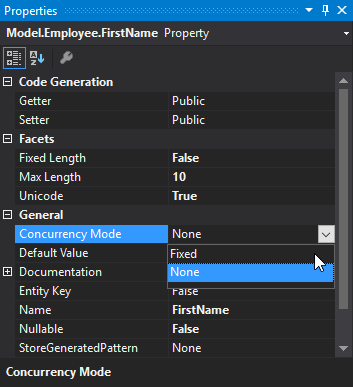
U Entity Frameworku moguće je rešiti konflikte rukovanjem [OptimisticConcurrencyException](https://www.link-elearning.com/linkdl/opisPojma.php?id=142495) izuzetkom koji izaziva Entity Framework. Da bi znao kada da izazove ovaj izuzetak, Entity Framework mora biti u mogućnosti da detektuje konflikt.

U Entity Frameworku omogućavanje optimističkog modela konkurentnosti zahteva sprovođenje sledeća dva koraka:

* 1. definisanje svojstva koje će biti korišćeno za proveru konkurentnosti;
  2. upravljanje OptimisticConcurrencyException izuzetkom koji se emituje kada provera ne uspe.

**Definisanje svojstva za proveru konkurentnosti**

Svojstvo ConcurrencyMode svojstva entiteta se koristi za markiranje svojstva za proveru konkurentnosti. Njegovu vrednost je moguće menjati u Properties prozoru, kao na slici 17.3.



*Slika 17.3 – Concurrency Mode svojstvo u Properties prozoru*

Svojstvo Concurrency Mode može imati dve vrednosti, i to:

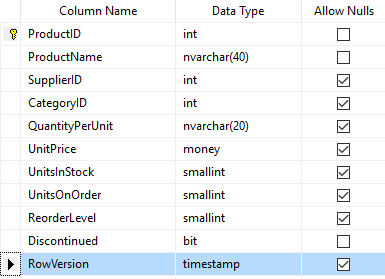
* + None – svojstvo nije uključeno u proveru konkurentnosti;
  + Fixed – svojstvo se proverava prilikom poziva metode za čuvanje entiteta.

Podrazumevano je da Entity Framework čuva promene bez provere konkurentnih konflikata, što znači da je podrazumevana vrednost Concurrency Mode svojstva *None*.

Najčešća je praksa da se za proveru konkurentnosti u tabeli definiše posebna kolona, čija vrednost će se uzimati u obzir prilikom provere konkurentnosti. Stoga je u tabeli potrebno dodati kolonu koja će biti tipa rowversion. Rowversion je specijalni tip podatka SQL servera, čije vrednosti se predstavljaju brojevima u binarnom obliku. Tako se za kolonu koja je tipa rowversion, vrednost automatski generiše prilikom svakog unosa ili ažuriranja reda sa podacima.

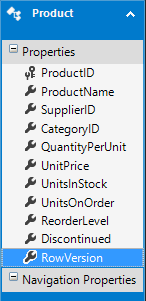
|  |
| --- |
| ***Napomena:****rowversion tip podataka i timestamp tip podataka su sinonimi.* |

Na slici 17.4 je prikazana tabela Products baze podataka Northwind u koju je dodata *RowVersion*kolona koja će se koristiti za proveru konkurentnosti.



*Slika 17.4 – RowVersion kolona u bazi podataka*

Nakon kreiranja kolone koja će se koristiti za proveru konkurentnosti, potrebno je izvršiti osvežavanje EF modela, kako bi novokreirana kolona ušla u sastav modela. Nakon kreiranja EF modela, izgled entiteta Product treba da bude kao na slici 17.5. (obratiti pažnju na postojanje svojstva RowVersion).



*Slika 17.5 – Entitet Product*

Za kreirano svojstvo RowVersion potrebno je postaviti i Concurrency Mode na Fixed, kao na slici 17.3.

**Rukovanje izuzecima nastalim u optimističkom modelu konkurentnosti**

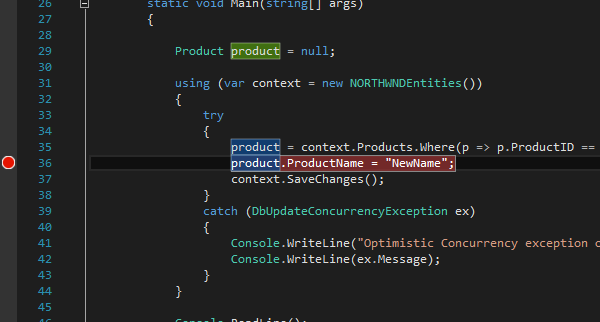
Kada Entity Framework prepozna konkurentni konflikt, vrši se emitovanje OptimisticConcurrencyException izuzetka, a zadatak programera jeste da upravlja tim izuzetkom u kodu. Stoga će u sledećem primeru biti demonstrirana jedna situacija koja će proizvesti OptimisticConcurrencyException izuzetak.

Dat je sledeći kod:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | Product product = null;  using (var context = new NORTHWNDEntities())  {      try      {          product = context.Products.Where(p => p.ProductID == 1).Single();          product.ProductName = "NewName";          context.SaveChanges();      }      catch (DbUpdateConcurrencyException ex)      {          Console.WriteLine("Optimistic Concurrency exception occured");          Console.WriteLine(ex.Message);      }  }  Console.ReadLine(); |

Prikazani kod je poznat iz prethodnih lekcija. U primeru se vrši instanciranje kontekstne klase, a zatim se unutar try bloka izvršava učitavanje proizvoda koji ima Id vrednost 1. Tako učitanom proizvodu se zatim menja ime i poziva metoda koja promene preslikava na bazu podataka. Opisan kod je napisan unutar jednog try bloka, kako bi mogao da se uhvati eventualni izuzetak koji se može pojaviti zbog problema konkurentnosti. Zbog toga je definisan i catch blok.

Ukoliko se prikazani kod pokrene, može se primetiti da će sve proći bez ikakvih problema. Ipak, kako bi se simulirao problem konkurentnog pristupa jednom resursu, u nastavku će biti urađeno sledeće. Biće potrebno postaviti tačku prekida na liniju u kojoj se menja naziv proizvoda, kao na slici 17.6, i aplikaciju pokrenuti u debug modu.

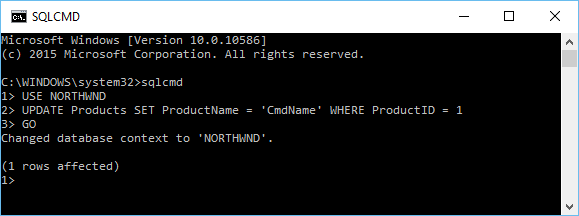


*Slika 17.6 – Postavljanje tačke prekida*

Nakon pokretanja aplikacije u debug modu, izvršavanje će biti pauzirano u liniji na kojoj je postavljena tačka prekida. U primeru sa slike 17.6 je to linija sa brojem 36.

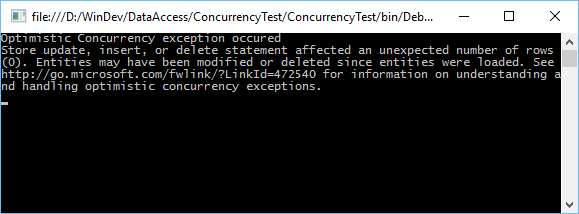
Ovo je linija u kojoj je učitavanje entiteta već obavljeno. Ovom istom zapisu sada može da pristupi i neki drugi korisnik, i da izvrši promenu imena proizvoda. Da bi se to simuliralo, dovoljno je otvoriti SQL Server Management Studio i izvršiti modifikaciju ili koristiti obični Command Prompt, i uputiti bazi sirovi komandu. Da bi se izvršila modifikacija imena proizvoda korišćenjem CMD-a, to je moguće postići korišćenjem alata *sqlcmd*, kao na slici 17.7.

|  |
| --- |
| ***Napomena:****obratite pažnju da je ovu izmenu potrebno obaviti dok je izvršavanje aplikacije pauzirano u debug modu.* |



*Slika 17.7 – Ažuriranje podataka korišćenjem CMD-a*

Nakon obavljenog ažuriranja imena proizvoda u bazi, potrebno je nastaviti izvršavanje aplikacije unutar Visual Studio razvojnog okruženja. Nastavak izvršavanja proizvešće izuzetak, kao na slici 17.8.



*Slika 17.8 – Emitovanje OptimisticConcurrencyException izuzetka*

**Upravljanje konfliktima konkurentnosti**

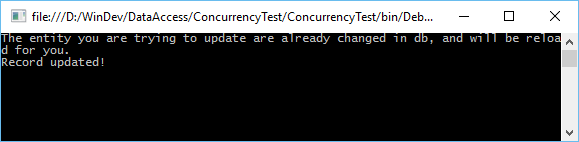
Kada se konflikt konkurentnosti detektuje, kao što je to bio slučaj u upravo prikazanom primeru, potrebno je odlučiti na koji način rešiti takvu situaciju. U nastavku će biti prikazane tehnike za rešavanje problema konkurentnosti korišćenjem dva najzastupljenija pristupa: *Database Wins* i *Client Wins*.

**Database Wins**

Jedan od načina za rešavanje problema konkurentnosti jeste korišćenje metode Reload, koja obavlja prepisivanje vrednosti entiteta sa onima koje su pročitane iz baze. Entitet se tako opet vraća korisniku na doradu, ali ovoga puta sa svežim vrednostima iz baze. Sledeći primer će ilustrovati najprostiji način za postizanje *Database Wins* scenarija.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | using (var context = new NORTHWNDEntities())       {           bool saveFailed;           do           {               try               {                   saveFailed = false;  Product product = context.Products.Where(p => p.ProductID == 1).Single();                   product.ProductName = "NewName";                   context.SaveChanges();                   Console.WriteLine("Record updated!");               }               catch (DbUpdateConcurrencyException ex)               {                   saveFailed = true;                   ex.Entries.Single().Reload();                   Console.WriteLine("The entity you are trying to update are already changed in db, and will be reload for you.");               }           }           while (saveFailed);       }       Console.ReadLine(); |

Ponovo je za testiranje prikazanog koda potrebno obaviti postavljanje tačke prekida nakon učitavanja entiteta, a pre njegovog čuvanja u bazu, i izvršiti modifikaciju proizvoda koji ima vrednosti Id-ja 1. Na taj način će se simulirati problem konkurentnosti. U takvoj situaciji prikazani kod neće emitovati izuzetak, niti će prestati sa radom, već će korisnika obavestiti o nastaloj situaciji, učitati vrednosti iz baze, i pokušati da ponovno napravi traženu izmenu. Tako će izgled konzolnog prozora da bude kao na slici 17.9.



*Slika 17.9 – Poruka koja se dobija nakon izvršavanja koda koji simulira Database Wins scenario*

**Client Wins**

Još jedan način na koji se može rešiti problem konkurentnosti jeste upotreba principa *Client Wins*. Prema ovom principu, vrednosti u bazi podataka se prepisuju vrednostima koje postoje unutar entiteta. Client Wins tehnika se postiže tako što se čitaju trenutne vrednosti iz baze koje se postavljaju kao originalne vrednosti entiteta. Sledi primer koji 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 context = new NORTHWNDEntities())       {           bool saveFailed;           do           {               try               {                   saveFailed = false;  Product product = context.Products.Where(p => p.ProductID == 1).Single();                   product.ProductName = "NewName";                   context.SaveChanges();                   Console.WriteLine("Record updated!");               }               catch (DbUpdateConcurrencyException ex)               {                   saveFailed = true;                   // Update original values from the database                   var entry = ex.Entries.Single();                   entry.OriginalValues.SetValues(entry.GetDatabaseValues());                   Console.WriteLine("The entity you are trying to update are already changed in db, and will be reload for you.");               }           }           while (saveFailed);       }       Console.ReadLine(); |