Fakultet elektrotehnike i računarstva

Zavod za primijenjeno računarstvo

**Objektno oblikovanje**

Laboratorij profila

Matea Bardun, 0036484568

Petar Jalušić, 0036484339

Toni Sredanović, 0036484318

Lucija Žbulj, 0036484115

Tanja Židić, 0036488247

Zagreb, 26.01.2018.

*Sustav za natjecateljsko plivanje*

*OR mapiranje*

# **1.** **Uvod**

U ovom seminarskom radu, kako bi se omogućila automatska i transparentna pohrana podataka iz baze podataka u objekte, korišteno je objektno relacijsko mapiranje. OR mapiranje je tehnika kojom se podaci transformiraju iz jednog oblika u drugi, između dva nekompatibilna sustava. Time se postiže veća produktivnost, jeftinije i lakše održavanje te nezavisnost o samoj tehnologiji SUBP-a. U ovom seminarskom radu, za OR mapiranje, koristili smo Fluent NHibernate, koji za razliku od običnog NHibernateane koristi XML jezik za konfiguraciju i mapiranje klasa, već strong-typed C# kod. FluentNHibernate nam je omogućio da umjesto pisanja SQL upita za rad s bazom podataka, koristimo njegovo jednostavno sučelje za CRUD operacije nad objektima te razredima i njihovim svojstvima. FluentNHibernate dakle uzima C# kod s konfiguracijskim parametrima te C# klase sa svojstvima koje treba pohraniti u bazu i dohvatiti kasnije, a kao izlaz daje automatski generirani SQL.

Samo korištenje Fluent NHibernate-a izvedeno je u par osnovnih koraka, a to su: definiranje objektnog modela i mapiranja, definiranje konfiguracijskih parametara, kreiranje session objekata i zatim kroz njih rad na samoj bazi podataka.

U sljedećim poglavljima prikazan je konkretni postupak korištenja Fluent Nhibernate-a u našem projektu.

2. Implementacija Fluent NHibernate

## **2.1.** **Kreiranje objektnog modela**

Na početku je potrebno kreirati objektni model. Za svaku relaciju u našoj bazi podataka definirali smo klasu istog imena, te joj pripisali svojstva istog imena kao atributi u relacijama u bazi podataka. Svaki model nasljeđuje klasu BaseEntity koja samo sadrži atribut Id. S obzirom da svaka tablica u bazi sadrži Id atribut, svaki će model nasljeđivati BaseEntity.

|  |
| --- |
| public class Race : BaseEntity  {  public virtual DateTime TimeStart { get; set; }  public virtual DateTime TimeEnd { get; set; }  public virtual Gender Gender { get; set; }  public virtual Pool Pool { get; set; }  public virtual Competition Competition { get; set; }  public virtual Length Length { get; set; }  public virtual Style Style { get; set; }  public virtual Category Category { get; set; }  public virtual Referee Refereee { get; set; }  } |

## 

Isječak koda 2.1. Definiranje objektog modela

**2.2.** **Definicija mapiranja**

Nakon stvaranja modela domene potrebno je definirati mapiranja. Mapiranja su jedan od važnijih koraka u implementaciji NHibernate-a jer nam ona pomažu da pravilno preslikamo relacije iz baze podataka u objekte u objektom modelu.

Za svaku klasu u našem objektom modelu smo definirali novu klasu koja definira mapiranje za istu.

|  |
| --- |
| public RaceMap()  {  Id(x => x.Id);  References(x => x.Pool).Column("idPool").Not.LazyLoad();  References(x => x.Competition).Column("idCompetition").Not.LazyLoad();  References(x => x.Length).Column("idLength").Not.LazyLoad();  References(x => x.Style).Column("idStyle").Not.LazyLoad();  References(x => x.Category).Column("idCategory").Not.LazyLoad();  References(x => x.Refereee).Column("idReferee").Not.LazyLoad();  Map(x => x.TimeStart);  Map(x => x.TimeEnd);  Map(x => x.Gender);  Table("Race");  } |

*Isječak koda 2.2. Prikaz mapiranja za klasu „Race“*

U prethodnom isječku koda vidimo jedan primjer mapiranja. U njemu su svi atributi mapirani u istovjetne u bazi podataka.

Oznaka „Id“ označava da se radi o primarnom ključu, dok oznaka „References“ označava da se radi o stranom ključu na neki drugi objekt modela domene. „Map“ označava obično preslikavanje atributa relacije, a s oznakom „Column“ zadajemo ime retka na koju se odnosi zadani atribut.

Oznaka „Not().LazyLoad()“ pomaže tomu da se atributi referencirane relacije učitaju odmah.

## **2.3.** **Postavljanje konfiguracije**

|  |
| --- |
| public static ISessionFactory CreateSessionFactory()  {  string connString = ConfigurationManager  .ConnectionStrings["DefaultConnection"]  .ConnectionString;  return Fluently.Configure()  .Database(  MsSqlConfiguration.MsSql2012  .ConnectionString(connString).ShowSql()  )  .Mappings(m=>m.FluentMappings.AddFromAssemblyOf<FluentNHibernateClass>())  .BuildSessionFactory();  }  *Isječak koda 2.3. Kreiranje ISessionFactory* |

Kako koristimo Fluent NHibernate konfiguracijska datoteka je pisana u C# kodu. U njoj je potrebno navesti connection string baze na koju se spajamo, te kreirati ISessionFactory iz zadanih konfiguracijskih parametara. Na sljedećem isječku koda prikazano je postavljanje konfiguracijskih parametara te instanciranje ISessionFactory-a za rad s SQL Server bazom podataka.

## **2.4.** **Rad s objektima**

Rad s objektima izveden je preko repozitorija, koji su kreirani za svaku bitniju komponentu u sustavu. U repozitorijma se nalaze metode koje na neki način komuniciraju s bazom podataka, bilo to dohvaćanje objekata, izmjena, spremanje ili brisanje.

Unutar svake metode repozitorija potrebno je prvo stvoriti novi session objekt te započeti transakciju nad njim. Na sljedećem isječku koda prikazana je jednostavna metoda repozitorija ClubRepository koja dohvaća sve klubove u bazi podataka, getClubs().

|  |
| --- |
| public List<Club> getClubs()  {  var result = new List<Club>();    var klasa = new FluentNHibernateClass();    using (var session = klasa.OpenSession())  {  using (var transaction = session.BeginTransaction())  {  result = (List<Club>)session.QueryOver<Club>().List<Club>();  transaction.Commit();  }  }  return result;  } |

*Isječak koda 2.4.*

Na početku je stvoren novi session, te je unutar njega pokrenuta transakcija. Nad session objektom obavljamo upit, u ovom slučaju dohvaćamo sve klubove iz klase Club, te ih spremamo u listu. Nakon što obavimo upit transakciju zatvaramo, čime se veza prema bazi podataka gasi.

Osim jednostavnih upita Fluent NHibernate omogućava i složenije upite. Primjer jednog takvog prikazan je u isječku 2.5.

|  |
| --- |
| public int getMyClubId(int CoachId)  {  Coach coach = null;  Season season = null;  int result;  var klasa = new FluentNHibernateClass();  using (var session = klasa.OpenSession())  {  using (var transaction = session.BeginTransaction())  {  result = (int)session.QueryOver<CoachSeason>().JoinAlias(x => x.Coach, () => coach).JoinAlias(x => x.Season, () => season).Where(() => season.TimeEnd > DateTime.Now && coach.Id == CoachId).Select(x => x.Club.Id).SingleOrDefault<int>();    transaction.Commit();  }  }    return result;  } |

*Isječak koda 2.5.*

U prethodnom isječku koristimo funkciju JoinAlias koja odgovara spajanju u relacijskim bazama podataka. Pomoću te funkcije spajamo relacije CoachSeason, Coach i Season, te gledamo postoji li n-torka čiji je Coach.Id jednak trenerovom id-u, te čija sezona traje još i sada.

Osim dohvaćanja objekata koristili smo i mogućnosti ažuriranja objekata metodom SaveOrUpdate(), te brisanja objekata metodom Delete().