SVEUČILIŠTE U ZAGREBU

**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

**Laboratorij profila**

Škola košarke

Ivana Vanjak

0036460333

Zagreb, veljača, 2015.

***Opis problema***

Administracija Škole košarke želi unaprijediti kvalitetu usluge za svoje članove tako što će izraditi informacijski sustav koji će im omogućiti praćenje treninga, utakmica i evidenciju članova. Škola košarke broji pedesetak aktivnih članova i treneru je postalo preteško samostalno voditi bilješke o svim igračima.

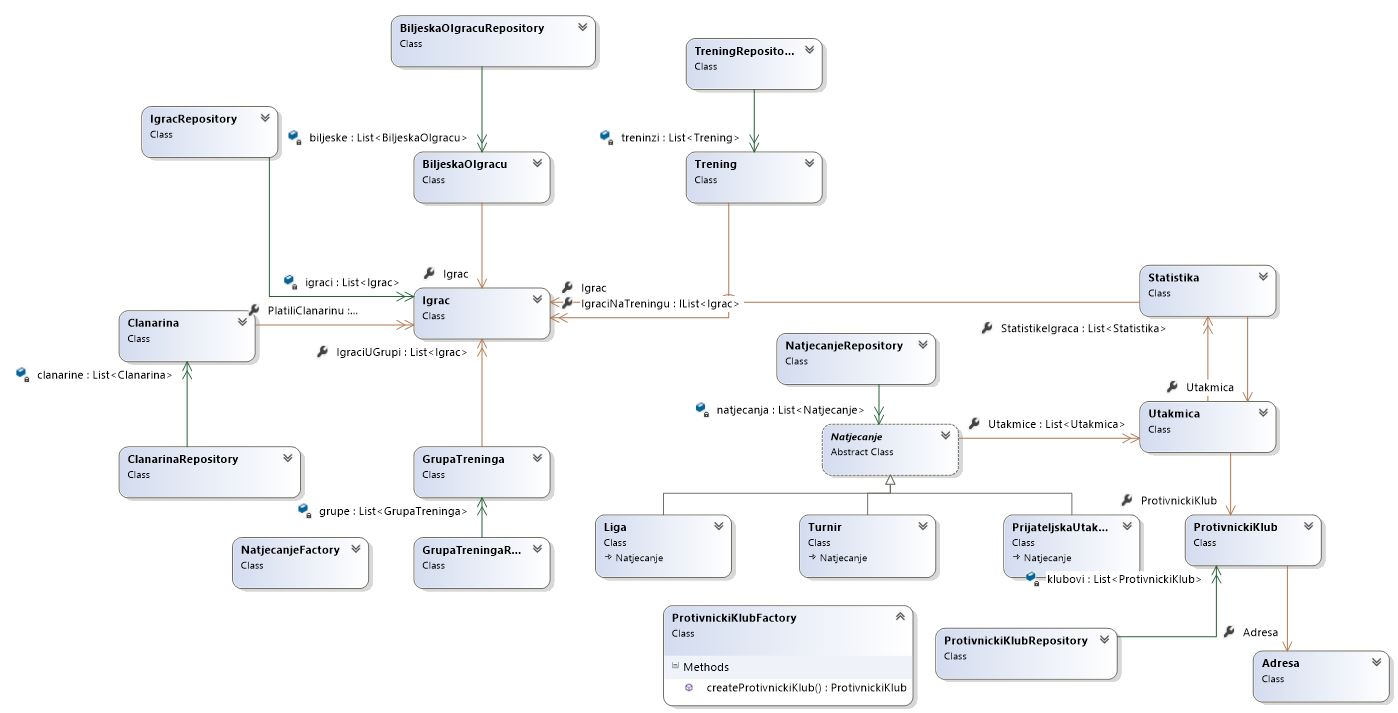
Članovi škole košarke su učenici osnovne škole. Prilikom upisa nekog djeteta uzimaju se njegovi podaci i uz to podaci o roditeljima/skrbnicima.

Škola košarke ima više različitih grupa za treninge. Ovisno o godištu igrača i njegovom napretku, trener odlučuje u koju grupu pripada. Na svakom treningu potrebno je provoditi evidenciju igrača i voditi zabilješke o njihovom napretku. To je potrebno da bi roditelji prilikom posjeta treneru mogli dobiti širi uvid u igrački razvoj svog djeteta, te kako bi trener prilikom savjetovanja igrača mogao imati pregled najbitnijih stavki na proteklim treninzima.

Igrači sudjeluju na raznim natjecanjima. Natjecanje može biti liga, turnir ili samo prijateljska utakmica. Potrebno je voditi evidenciju o rezultatima svih utakmica i statistici svih igrača na njima.

Također je važno imati podatke o protivničkim klubovima s kojima se igrači susreću, kao što su adresa i kontakt, kako bi sve potrebne stvari prilikom organizacije utakmice bile dostupne unutar aplikacije.

***Opis objektnog modela***

******

**Slika 1. Dijagram razreda**

Kao središte interesa u ovoj aplikaciji imamo igrače koji sudjeluju na raznim natjecanjima i pohađaju treninge. Igrači su predstavljeni entitetom u kojem su pohranjene sve osnovne informacije o samom igraču. Igračima se upravlja kroz repozitorij u kojem postoji lista svih igrača koji su trenutno u sustavu. On ima sve metode potrebne za kreiranje i dohvaćanje igrača. *IgracRepository* je *sigleton*, što znači da u svakom trenutku postoji samo jedna instanca te klase.

Kroz entitet *Trening* pratimo koji su igrači bili na pojedinom treningu. Treninzi se mogu odvijati u više grupa,, pa igrače na treningu biramo ovisno o grupi igrača koja ima određeni trening. U entitetu *GrupaTreninga* imamo listu igrača koji joj pripadaju. Za svakog igrača se mogu voditi bilješke koje će omogućiti prikaz napretka igrača. Treninzima, grupama treninga i bilješkama se također upravlja kroz istoimene repozitorije koji su realizirani isto poput repozitorija igrača.

Natjecanje je prikazano kroz hijerarhiju objekata. Kao apstraktnu klasu imamo entitet *Natjecanje* kojeg zatim nasljeđuju *Liga*, *Turnir* i *PrijateljskaUtakmica*. Uz atribute *Natjecanja*, *Liga* i *Turnir* definiraju dodatno svoje atribute.

*Natjecanje* ima listu objekata *Utakmica*. Svakom natjecanju se pridružuju utakmice koje su se održale na tom natjecanju, za koje se pamte ukupan rezultat i protivnički klub. Za svaku utakmicu definiramo skup igrača koji su na njoj prisustvovali kroz klasu *Statistika. Statistika* ima referencu na igrača za kojeg se piše statistika i referencu na utakmicu na kojoj je postigao statistiku te podatke o koševima, skokovima, asistencijama i osobnim pogreškama. Natjecanja, utakmice i statistike se prate u repozitoriju *NatjecanjeRepository* u kojem su defnirane sve metode potrebne za dodavanje i dohvaćanje svakog od njih. Tamo imamo listu natjecanja, preko koje se zatim traže utakmice, a preko utakmica statistike. Konkretno natjecanje se kreira preko *NatjecanjeFactory* klase, koja pruža statičke metode za izradu natjecanja. Pošalje joj se konkretni tip i potrebni parameri, a ona vraća objekt tipa *Natjecanje*.

U aplikaciji je potrebno pratiti i podatke o protivničkim klubovima. U entitetu *ProtivnickiKlub* pratimo sve podatke o klubovima, poput naziva, telefonskog broja te adrese koja je implemetirana kao vrijednosni objekt *Adresa.* Protivničkim klubovima se pristupa preko pripadnog repozitorija. Objekti se stvaraju preko *ProtivnickiKlubFactory* razreda.

U sustavu se još prati plaćanje članarina. Definiranjem mjeseca, godine i vrijednosti članarine opisana je jedna mjesečna članarina. U tom entitetu se još prate svi igrači koji su je platili. Članarine se prate kroz repozitorij.

***Mapiranje i pristup podacima***

Perzistenciju objekata iz modela domene postižemo pohranjivanjem objekata u relacijsku bazu podataka SQLite. Zbog neslaganja između relacijske i objektne tehnologije koristi se objektno-relacijski maper. On omogućava automatsku transformaciju iz jedne u drugu reprezentaciju. ORM ima API za CRUD, API za upite nad klasama i njihovim svojstvima, funkcionalnost za rad s metapodacima i tehnike za interakciju s transakcijskim objektima.

U ovoj aplikaciji, objektno-relacijsko mapiranje je ostvareno pomoću *framework*-a Fluent NHibernate. On kao ulaz uzima C# klasu sa svojstvima koje treba pohraniti u bazu i dohvatiti kasnije, te C# klasu s podacima za mapiranje. Kao izlaz da je automatski generirani SQL.

1) Prvo se kreira poslovni model. U modelu sva svojstva objekta moraju biti označena sa virtual,mora postojati *default* konstruktor koji ne prima parametre i sve kolekcije objekata moraju biti realizirane preko sučelja.

Primjer razreda iz modela:

public class BiljeskaOIgracu

{

private int id;

private Igrac igrac;

private DateTime datum;

private string biljeska;

public virtual int Id

{

get { return id; }

protected set { id = value; }

}

public virtual DateTime Datum

{

get { return datum; }

set { datum = value; }

}

public virtual Igrac Igrac

{

get { return igrac; }

set { igrac = value; }

}

public virtual string Biljeska

{

get { return biljeska; }

set { biljeska = value; }

}

public BiljeskaOIgracu(Igrac inIngrac, DateTime inDatum, string inBiljeska)

{

igrac = inIngrac;

datum = inDatum;

biljeska = inBiljeska;

}

public BiljeskaOIgracu() { }

}

2) Da bi NHibernate znao izvršiti mapiranje OO->SQL, mora imati podatke o mapiranju. Mapiranje se definira u *strong-typed* C# kodu. Veza *one-to-many* se ostvaruje ključnom riječi HasMany, *many-to-many* HasManyToMany i veza *many-to-one* References.

Veze se ostvaruju tako da se u tablici dodaje stupac sa stranim ključem prema tablici objekta kojeg se referencira.

Ukoliko želimo postići dvosmjernu vezu između entiteta, moramo to naglasiti NHibernate-u. To se postiže navođenjem ključne riječi Inverse(). *Parent-child* odnos postižemo tako što navodimo Cascade.All().

Prilikom učitavanja podataka NHibernate koristi *lazy loading*, ne učitava sve objekte koje trenutni objekt referencira, pa ukoliko tu funkcionalnost želimo zaobići moramo u mapiranju kod željenog svojstva navesti Not.LazyLoad().

Primjer mapiranja:

public class UtakmicaMap : ClassMap<Utakmica>

{

public UtakmicaMap() {

Id(p => p.Id).GeneratedBy.Native();

Map(p => p.Datum);

References(p => p.ProtivnickiKlub).Not.LazyLoad();

Map(p => p.BrojPostignutihKoseva);

Map(p => p.BrojPrimljenihKoseva);

HasMany(p => p.StatistikeIgraca).Inverse().Cascade.All().Not.LazyLoad();

}

}

Najveći problem kod objetno-relacijskog neslaganja je mapiranje hijerarhije nasljeđivanja. Imamo tri moguća pristupa:

– *Table per concrete class* — ”izbacujemo” polimorfizam i nasljeđivanje iz relacijskog modela

– *Table per class hierarchy* — omogućujemo polimorfizam denormalizacijom relacijskog modela uz korištenje diskriminatorskog stupca koji sadrži podatke o tipu

– *Table per subclass* — reprezentira odnos “*is a*”(nasljeđivanje) kao “*has a*” (strani ključ) odnos

U ovoj aplikaciji korišten je drugi pristup, *Table per class hierarchy,* iz razloga što izvedeni razredi ne dodaju puno novih svojstava naspram baznog razreda.

Svako svojstvo izvedenih klasa ima poseban stupac . Definira se diskriminator koji određuje kojeg je tipa objekt u tom retku. Kod ovog pristupa se i polimorfni i ne-polimorfni upiti efikasno izvršavaju. Potencijalni problem je što gubimo integritet podataka u bazi jer svojstva izvedenih klasa moraju biti *nullable*.

Primjer mapiranja hijerarhije objekata:

public class NatjecanjeMap : ClassMap<Natjecanje>

{

public NatjecanjeMap() {

Id(p => p.Id);

Map(p => p.Naziv);

Map(p => p.Godiste);

HasMany(p => p.Utakmice).Cascade.All().Not.LazyLoad();

DiscriminateSubClassesOnColumn("VRSTA\_NATJECANJA");

}

}

public class LigaMap : SubclassMap<Liga>

{

public LigaMap() {

DiscriminatorValue("LIGA");

Map(p => p.Rang);

Map(p => p.BrojKlubova);

Map(p => p.BrojBodova);

}

}

public class TurnirMap : SubclassMap<Turnir>

{

public TurnirMap() {

DiscriminatorValue("TURNIR");

Map(p => p.Rang);

Map(p => p.BrojKlubova);

}

}

U bazi nastaje jedna tablica sa svim atributima iz bazne i izvedenih klasa, te diskriminatorom. Na temelju vrijednosti diskriminatora, određuje se kojeg je tipa objekt redak tablice. Tako prilikom pohranjivanja objekata tipa Turnir vrijednost atributa BrojBodova ostaje null.

3) Globalna konfiguracija se definira u zasebnoj C# klasi.

Konfiguracijska datoteka:

public class FluentNHibernate

{

public static ISession OpenSession()

{

var nhConfig = Fluently.Configure()

.Database(SQLiteConfiguration.Standard

.ConnectionString("Data Source=SkolaKosarkeDB.db;Version=3")

.AdoNetBatchSize(100))

.Mappings(x => x.FluentMappings.AddFromAssembly(Assembly.GetExecutingAssembly()))

.BuildConfiguration();

var sessionFactory = nhConfig.BuildSessionFactory();

ISession sess = sessionFactory.OpenSession();

return sess;

}

}

4) Rad s objektima se definira kroz repozitorije. Ovisnosti su usmjerene prema sučeljima repozitorija. U Modelu su definirana sučelja IRepository, koja su implementirana u Data sloju. “Gornji” slojevi ovise o metodama sučeljima.

Kod stvaranja novog objekta u repozitoriju otvaramo sjednicu unutar koje započinjemo transakciju te spremamo novi objekt koji tad postaje perzistentan.

Primjer spremanja objekta u bazu:

using (ISession session = FluentNHibernate.OpenSession())

{

using (ITransaction transaction = session.BeginTransaction())

{

session.Save(igrac);

transaction.Commit();

}

}

Primjer dohvaćanja objekata iz baze:

using (ISession nSession = FluentNHibernate.OpenSession())

{

IQueryable<Igrac> query = (from i in nSession.Query<Igrac>()

where i.DatumRodenja.Year >= godiste

orderby i.Prezime

select i);

IList<Igrac> igraci = query.ToList<Igrac>();

return igraci;

}

Primjer ažuriranja:

using (ISession session = FluentNHibernate.OpenSession())

c.dodajIgraca(i);

using (ITransaction transaction = session.BeginTransaction())

{

session.Update(c);

transaction.Commit();

}

}