Uvod u ORM

Pristup podacima iz programskog koda 25/26

Ishod učenja 5 - ORM i migracije

ORM

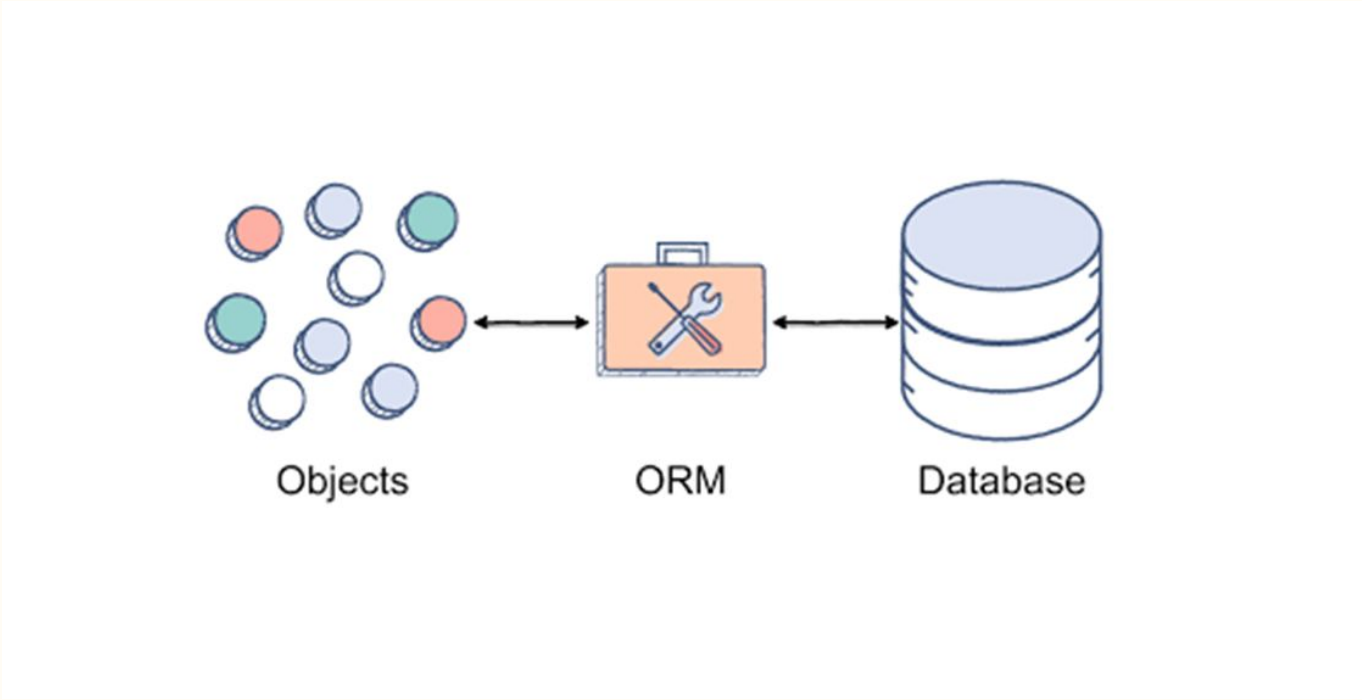
→ Object Relational Mapper

→ Međusloj između poslovne (aplikativne) logike i relacijske baze

→ Rad s bazom podataka pojednostavnljen je na rad s klasama/objektima dobro poznatim OOP paradigmi

→ Mapiranje podatak iz redova (n-torki) u objekte, upravljanje konekcijom i prijevodom idiomatskih OOP poziva u SQL upite

ORM



Naši primjeri

Entity Framework Core i njegova arhitektura!

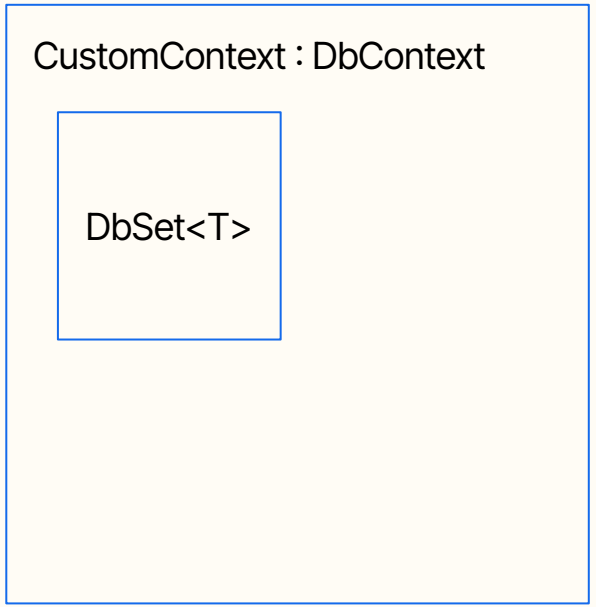
Entity Framework

→ Glavna klasa je klasa koja nasljeđuje DbContext

→ DbSet<T> članska varijabla omotava tip entitetskog objekta (eng. *wrapper*)

→ POCO klasa označava 1:1 mapiranje između tablice i klase

→ Povezanim entitetima se pristupa kroz vritualna svojstva



DbContext

→ Instanca DBContext podklase se koristi kako bi se pristupilo bazi podataka

→ Kombinira Unit of Work i Repository obrazac

→ Instanca se koristi u **Dependency Injection** okruženjima

Unit of Work

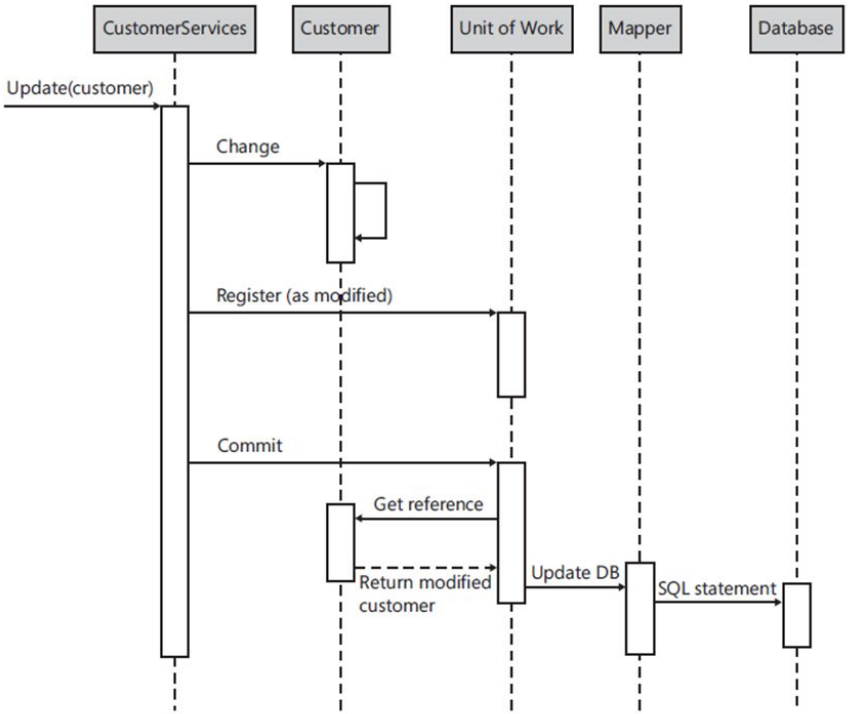


FIGURE 6-4 An overall schema for a unit of work

Primjer DbContext klase

```
public class DataContext : DbContext {
        public DbSet<T> Entities { get; set; }

    public DataContext(DbContextOptions options) : base(options) { }

        protected override void OnModelCreating(
                    ModelBuilder builder) {
            base.OnModelCreating(builder);
    }
}
```

DbContextOptionsBuilder

→ `UseInMemoryDatabase`

postavlja in-memory bazu podataka

→ `UseLazyLoadingProxies`

Postavlja *lazy loading* proxy objekte

→ `UseNpgsql` (`UseSqlLite`, `UseSqlServer`)

Postavlja koja će se baza koristiti i s kojim podacima pristupa (*connection string)*

→ `EnableSensitiveDataLogging`

Omogućava logging i sensitivnih podataka

DbSet<T>

→ generički parametar T predstavlja entitetski tip (klasu koja mapira tablicu iz baze podataka)

→ Metode na instanci:

- `Add`
- `AsNoTracking`
- `Find / FindAsync`
- `Include`
- `Remove`
- `FromSQL`
- `Update`

Change tracking

→ `DBContext` instanca prati promijene nastale na instancama entiteta

→ Instanca `DbContexta` stvara transakciju koja izvršava odgovarajuće SQL upite za svako stanje instance entiteta čije se promjene prate:

`Added`

`Modified`

`Deleted`

→ Transakcija se izvršava nakon poziva `SaveChanges` metode

Change tracking

Po defaultu, praćenje promjena je uključeno ukoliko su instance entitetskih objekta

→ vraćene kao rezultat izvršavanja upita

→ eksplicitno dodane u context korištenjem `Add`, `Attach`, `Update` ili sličnih metoda

→ prepoznate kao nove instance povezane s postojećim objektima čije se promjene prate

Primjer

Potrebno je razviti programsko rješenje za upravljanje studentskim rješenjima. Studente definira ime, prezime, JMBAG. Studenti pripadaju studijskim grupama na trenutnoj godini studija određene akademske godine. Studenti su upisani na kolegije (npr. Programiranje). Na kolegiju su definirane školske zadaće. Školske zadaće sadrže tekst zadataka, vrijeme pisanja i označene su ishodom učenja za koje se pišu. Studenti prenose kod rješenja te im asistenti po završetku pisanja zadaće, unose bodove.

Razviti:

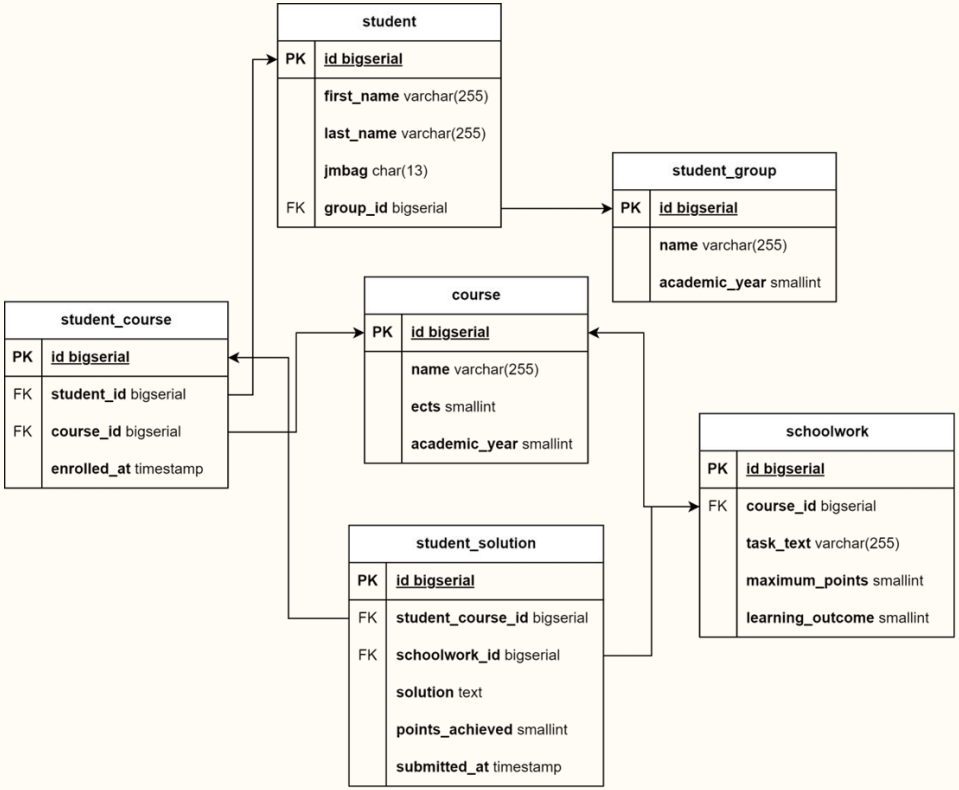
→ER diagram

→DDL upite (Postgres)

→Scaffold (*database first*)

→Isprobati CRUD operacije

Primjer - ER dijagram



Primjer - DDL

Vježbe-04-SolutionManager-DDL.sql

Primjer - kod

Potrebno je instalirati sljedeće biblioteke:

```
Microsoft.EntityFrameworkCore
Microsoft.EntityFrameworkCore.Tools
Npgsql.EntityFrameworkCore.PostgreSQL
```

Ako EF alati nisu instalirani, potrebno ih je instalirati:

```
dotnet tool install --global dotnet-ef
```

Primjer - kod

Potom, potrebno je izvršiti scaffold:

```
    dotnet ef dbcontext scaffold <cs>
Npgsql.EntityFrameworkCore.PostgreSQL
```

Kako bi scaffold proces bio uspješan, potrebno je uspješnog izgraditi projekt (eng. *build project)*

Scaffold - dodatak

Prilikom izdavanja scaffold naredbe, moguće je definirati neke korisne opcije:

`--output-dir (-o) <PATH>`
definira lokaciju generiranih klasa
`--schema <SCHEMA_NAME>`
definira koja shema će biti uključena
`--context (-c) <NAME>`
definira ime generirane DbContextklase
`--no-pluralize`
spriječava pluralizaciju imena entiteta za generiranje DBSet-ove
`--table <TABLE_NAME>... (-t)`
specifizira listu tablica

Primjer - nastavak

Koristeći generiranu DbContext klasu dohvatite i ispišite sva ime, prezime i JMBAG studenata iz tablice `student`

Potom, uz prijašnje navedene podatke, pokušajte ispisati ime grupe kojoj student pripada (zašto ne radi?)

Primjer - nastavak - rješenje lazy loading

→ EF po defaultu dohvaća podatke tek kada su potrebni

→ EF generira jedan upit koji dohvaća podatke o entitetu bez povezanih entiteta

N + 1 problem!

dotnet add package `Microsoft.EntityFrameworkCore.Proxies`

→U metodu OnConfiguring potrebno je dodati
`optionsBuilder.UseLazyLoadingProxies()`

Alternativno: https://learn.microsoft.com/en-us/ef/core/querying/related-data/lazy

Primjer - nastavak - eager loading

Dohvaća povezane podatke zajedno s ciljanim entitetom u jednom generiranom upitu

`Include()` metoda na `DBSet<T>`

```
_context.Students.Include(s => s.Group);
```

Informativni logging

Omogućite opciju informativnog ispisa u DBContext klasi i demonstrirajte razliku u generiranim upitima za eager loading i lazy loading pristup

Zadatak

Isprobajte ostale CRUD operacije (create, update, delete) nad danim entitetima!

- Dodajte novog studenta.
- Dodajte novo studentsko rješenje (entitet student_solution).
- Izmijenite studentovu grupu.
- Izbrišite jedan kolegij.

Gdje nastaju problemi?

Zadatak - objašnjenje

Kao što je spomenuto prije, kako biste spremili promjene nad entitetima, potrebno je omogućiti praćenje promjena nad istim

Navedeno je moguće ostvariti korištenjem različitih metoda:

Eksplicitnim pozivom `DbContext.Attach`

Implicitnim dodavanjem entiteta u povezani entitet kojemu je uključeno praćenje promjena

Pozivom metode `DbSet.Add`

```
course.Schoolworks.Add(newlyCreatedSchoolwork);
```