ORM i povezani podaci

Pristup podacima iz programskog koda 25/26

Ishod učenja 5 - ORM i migracije

Praćenje promjena - change tracking

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

→ to je moguće ostvariti korištenjem različith metoda:

→ Eksplicitnim pozivom DbContext.Attach

→ Implicitnim dodavanjem entiteta u povezani entitet kojemu je uključeno praćenje promjena

→ Pozivom metode DbSet.Add

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

Migracije

→ Skup promjena na shemi baze podataka

→ Potrebno je pratiti koje su promjene izvršene, kako bi shema uvijek bila ažurna

→ Često ju je moguće generirati automatski, putem definicija klasa i promjena

Primjer

→ Napravite entitet **studenta** koji pripada jednoj **grupi**

→ Stvorite migraciju i primjenite je nad instancom baze podataka

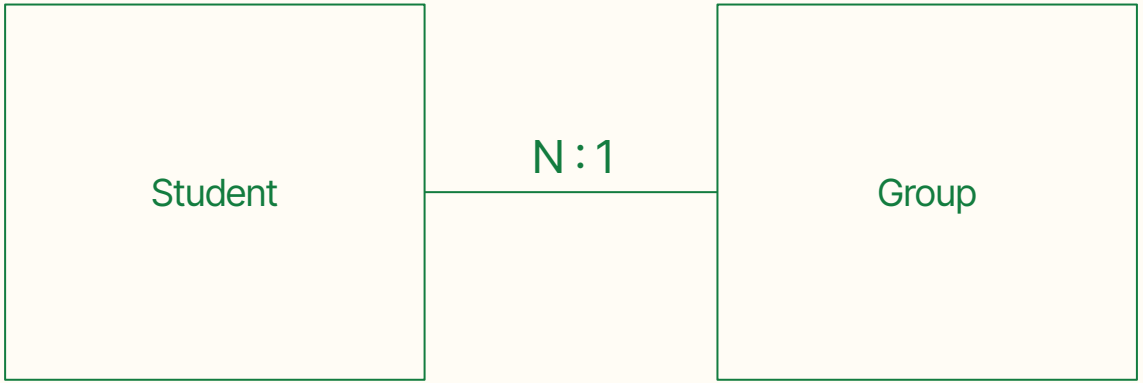
→ Kroz SQL dodajte podatke o nekoliko grupa i studenata

→ U kodu, pokušajte dohvatiti određenog studenata i ispisati podatke o povezanoj grupi

Definicija povezanih entiteta u *code first*

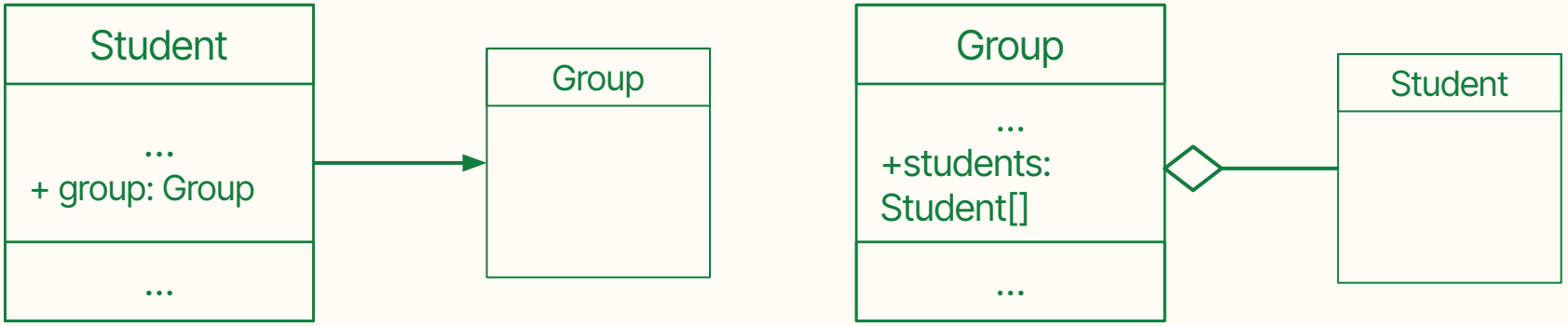
Prvo je potrebno definirati model entiteta i njihovih poveznice, a onda ih definiramo kroz klase

Recimo entitet student koji je povezan s grupom u odnosu N:1



Primjer - definicija entiteta

Htjeli bismo dohvatiti podatke o grupi kroz instancu studenta, ali isto tako želimo dohvatiti listu studenta za danu grupu.



Primjer - definicija entiteta

```
public class Group
{
    public int Id { get; set; }
    public string Name { get; set; }
    public virtual ICollection<Student> Students { get; set; }
}
```

Primjer - definicija entiteta

```
public class Student
{
    public int Id { get; set; }
    public string FirstName { get; set; }
    public string FirstName { get; set; }
    public int GroupId { get; set; }
    public virtual Group Group { get; set; }
}
```

Primjer - definicija entiteta

U `DbContext::OnModelCreating` metodi potrebno je definirati:

```
modelBuilder.Entity<Student>()
    .HasOne(s => s.Group)
    .WithMany(g => g.Students)
    .HasForeignKey(s => s.GroupId);
```

Primjer - definicija putem atributa

U klasi `Student`:

```
[Column("group_id")]
public long GroupId { get; set; }

[ForeignKey("GroupId")]
[InverseProperty("Students")]
public virtual StudentGroup Group { get; set; } = null!;
```

Primjer - definicija putem atributa

U klasi `Group`:

```
[InverseProperty("Group")]
public virtual ICollection<Student> Students { get; set; }
```

Mapiranje

Primijetite način na koje se navode svojstva i inverzna svojstva, kako bi se otvarilo **bidirekcijsko mapiranje**.

Dohvat povezanih podataka

→ Za dohvat povezanih podataka koristimo **navigacijska svojstva**

→ Dohvat povezanih podataka može se izvršavati:

EAGER LOADING

dohvat povezanih podataka odamah prilikom dohvata glavnih podataka

LAZY LOADING

dohvat povezanih podataka tek kada im pristupamo

EAGER LOADING

→ dohvat povezanih podataka zajedno s ciljanim entitetskim podacima, u jednom generiranom upitu
→ `Include()` metoda na `DBSet<T>`

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

=

```
SELECT * FROM student LEFT JOIN group ON group.id = student.group_id
```

LAZY LOADING

→ Dohvaća podatke tek kada su potrebni
→ ORM izvršava jedan upit za dohvat svih podataka traženog entiteta
→ kada je potreban povezani podataka, izvršava se zaseban odgovarajući upit

Ovakvo ponašanje uzrokuje **N + 1 problem**!

N + 1 problem

Primjer: imamo 25 studenata u tablici i ako želimo ispisati podatke o grupi (povezanog entiteta u N:1 kardinalitetu) za svakog studenata trebalo bi nam 26 upita:

1 upit za dohvat svih studenata (`SELECT * FROM student`)

+

25 x pojedinačni upit za dohvat podataka o povezanoj grupi za danog studenta (`SELECT * FROM group WHERE group.id = student.group_id`)

---

26 upita

Postavke lazy loading pristupa u EF

1. Instalirati paket `Microsoft.EntityFrameworkCore.Proxies`
2. U metodi `OnConfiguring`, u builderu dodati poziv `UseLazyLoadingProxies()`

Alternativno - Lazy Loading of Related Data

eager vs *lazy* loading

Na obrani je potrebno racionalizirati odluku korištenja *eager* ili *lazy* loading pristupa.

Za željeni ishod, možete implementirati dohvat povezanih podataka putem *eager* ili *lazy* loading pristupa.

Problemi

Pokušajte izbrisati grupu. Koji problem nastaje?