Spring Boot en Hibernate

Verder bouwen aan de Back-End

Leerdoelen

- Relationele databases
- One To One
- Many To One & One To Many
- Many To Many

Relaties

Entiteiten in een database zijn opzich leuk, maar we gebruiken niet voor niets een relationele database.

- Met relaties kunnen we entiteiten aan elkaar koppelen
- Wanneer we een van de twee entiteiten ophalen, krijgen we ook informatie terug over de ander
- Bv. een auto is gekoppeld aan een mens

Soorten relaties

Er zijn verschillende soorten relaties

- One to One
  - bv. Een auto heeft een eigenaar
- Many to One & One to Many
  - bv. Een school heeft meerdere leerlingen en meerdere leerlingen hebben een school
- Many to Many
  - bv. Een student kan meerdere vakken volgen en een vak heeft meerdere studenten

Spring boot

We kunnen deze relaties ook modeleren in Spring. We maken gebruik van annotaties om deze relaties aan te geven.

- @OneToOne
- @OneToMany
- @ManyToOne
- @ManyToMany

@OneToOne

- We hebben een class Kat en een class Chip. In beide classes maken we een veld van de andere kant van de relatie
- Door de @OneToOne annotatie maken we de relatie
- Aan de eigenaars-kant van de relatie zetten we @JoinColumn met de naam van de kolom van de foreign key
- Let op: Kans op infinite recursion -> @JsonIgnore

```
@OneToOne
@JoinColumn(name = "chip_id")
private Chip chip;
```

```
@OneToOne(mappedBy = "chip")
@JsonIgnore
private Kat kat;
```

@ManyToOne @OneToMany

- Net als bij @OneToOne maken we velden voor de relatie aan
- Aan de @OneToMany kant maken we een verzameling aan, in dit geval de interface List. Let op, deze moeten we instantiëren!
- Om infinite recursion te voorkomen, gebruiken we hier @JsonManagedReference aan de eigenaars kant, en @JsonBackReference aan de bezitte kant
- @JsonIgnore werkt ook

```
@OneToMany
@JsonManagedReference
private List<Kitten> kittens = new ArrayList<>();
```

```
@ManyToOne
@JsonBackReference
private Kat kat;
```

@ManyToMany

- Voor een many to many relatie hebben we een tussentabel nodig
- Daarin worden dan de id’s van beide kanten van de relatie opgeslagen
- Deze zullen we moeten specificeren in onze annotatie.
- Dit doen we doormiddel van de @JoinTable annotatie

```
@ManyToMany
@JoinTable(
        name = "katten_mensen",
        joinColumns = @JoinColumn(name = "kat_id"),
        inverseJoinColumns = @JoinColumn(name = "mens_id")
)
private List<Mens> mensen = new ArrayList<>();
```

```
@ManyToMany(mappedBy = "mensen")
@JsonIgnore
private List<Kat> katten = new ArrayList<>();
```

@JoinTable

- Allereerst geven we de naam van de tussentabel op
- Vervolgens geven we de naam van de id van de bezittende kant op.
- Het patroon is *classname_idname*
- In dit geval is de class *Kat* en de id *id*.
- De kolom naam is dan kat_id
- Met de InverseJoinColumns geven we het id van de bezitte kant aan.

```
@ManyToMany
@JoinTable(
        name = "katten_mensen",
        joinColumns = @JoinColumn(name = "kat_id"),
        inverseJoinColumns = @JoinColumn(name = "mens_id")
)
private List<Mens> mensen = new ArrayList<>();
```

@ManyToMany

- Vervolgens geven we aan de bezitte kant van de relatie aan welk veld van de bezittende kant de relatie beheerd
- Dit doen we met mappedBy
- We gebruiken hier @JsonIgnore om infinite recursie te voorkomen

```
@ManyToMany(mappedBy = "mensen")
@JsonIgnore
private List<Kat> katten = new ArrayList<>();
```

Vragen?

- E-mail mij op voornaam.achternaam@code-cafe.nl!

- Join de Code-Café community op discord!