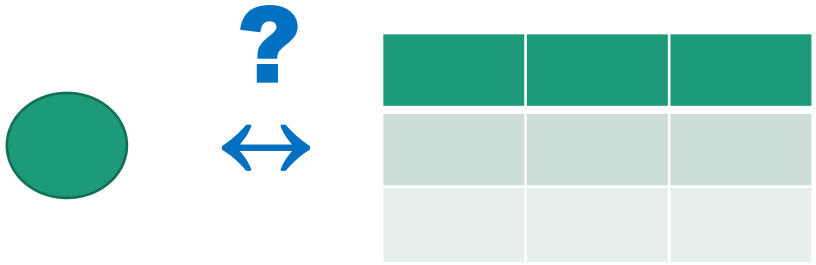
ORM (Object Relational Mapping)

Veerle Ongenae

ORM (Object Relational Mapping)

UNIVERSITEIT GENT

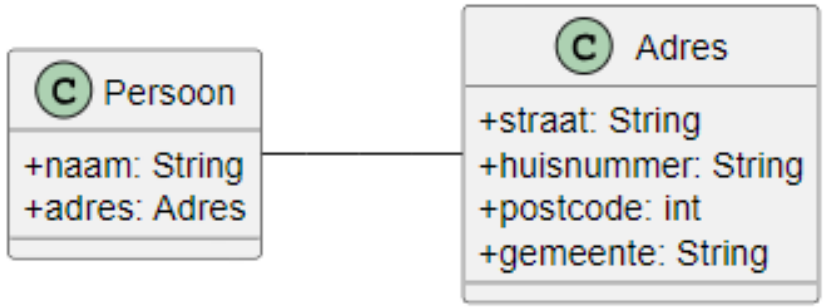
- Applicatie
  - Software
    - ➢ Objecttechnologie (bv. Java, C#, …)
  - Data
    - ➢ Relationele gegevensbanken
- Afbeelding (mapping) tussen
  - Data in objecten
  - Data in tabellen
- Hoe realiseren?



Verschil OO-concepten en relationele databanken: types versus tabellen

UNIVERSITEIT GENT

Klassen

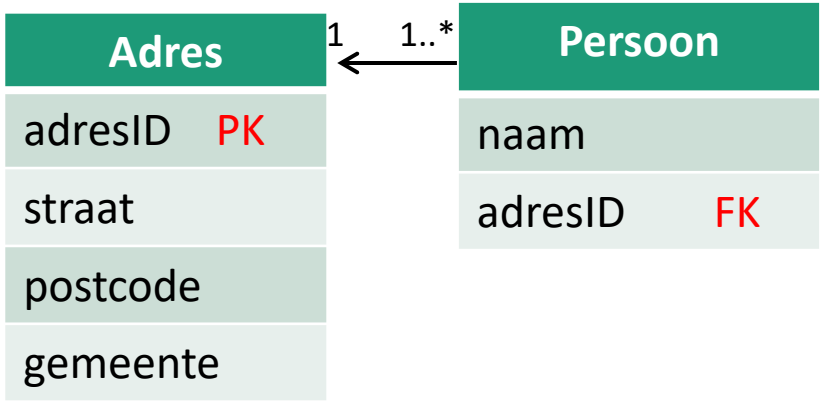


↔

Tabellen

Persoon
naam
straat
postcode
gemeente

of



Object-relational impedance mismatch

- Verschil OO-concepten en relationele databanken
  - Identiteit
  - Relaties
  - Overerving

Verschil OO-concepten en relationele databanken: identiteit

UNIVERSITEIT GENT

- Identiteit?
  - Wanneer zijn twee entiteiten hetzelfde?
- DB
  - Bepaald door primaire sleutel
- OO
  - Zelfde referentie (verwijzen naar zelfde geheugenplaats)
  - Resultaat equals-methode
    - ➢ Bv. een aantal attributen zijn dezelfde

Identiteit

- Primaire sleutel (DB)
  - Geen corresponderende eigenschap in domeinmodel
    - ➢ Besmetten domeinmodel met extra attribuut
- Automatische gegenereerde sleutel
  - Vergelijken objecten enkel mogelijk indien toegevoegd aan DB
- Object-identiteit is moeilijk te mappen naar record-identiteit

Schaduwinformatie

- Shadow Information
- Data
  - Nodig om informatie te bewaren (in DB)
    - ➢ Bv. Primaire sleutel
  - Geen deel van business model

ORM in Java

- JPA (Java Persistence API)
- Mapping tussen klassen en tabellen met annotaties

Dataklassen

- Moeten JavaBeans zijn
  - Defaultconstructor
  - Getters en setters voor eigenschappen


```
import javax.persistence.*;

@Entity
@Table(name = "sportclub")
public class Sportclub {
    private Long id;
    private String naam;

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    public Long getId() { return id;  }
    public void setId(Long id) { this.id = id;   }

    //@Basic
    //@Column(name = "NAAM")
    public String getNaam() { return naam;  }
    public void setNaam(String naam) { this.naam = naam; }
}
```

Entity

- Entiteit
- Beheerd door JPA-framework
- Klasse ↔ Tabel
  - Attribuut name: tabelnaam
  - Optioneel
- Object ↔ Record/rij

```
import javax.persistence.*;

@Entity
@Table(name = "sportclub")
public class Sportclub { … }
```

ID

- ID vereist
  - Samengestelde sleutels kan ook → weinig gebruikt
- name-attribuut → kolom in tabel


```
@Entity
@Table(name = "sportclub")
public class Sportclub {
    private Long id;
    private String naam;

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    public Long getId() { return id;  }
    public void setId(Long id) { this.id = id;   }
```

ID

- Generator
  - Hoe id aangemaakt bij nieuwe objecten?
    - AUTO
    - IDENTITY
    - SEQUENCE
    - TABLE

Object ↔ Tabel ??

```
//@Basic
//@Column(name = "NAAM")
public String getNaam() { … }
public void setNaam(String naam) { … }}
```

- Eigenschap v.e. object → nul of meerdere kolommen
  - Nul?
    - Sommige informatie moet niet bewaard worden
    - Vb. Gemiddelde
      - Berekend op basis van informatie uit DB
  - Meer dan één?
    - Eigenschap = object
      - Heeft op zijn beurt eigenschappen
    - Vb. Persoon-object heeft als eigenschap een Adres-object

Eigenschap ↔ kolom

- @Basic
  - Primitieve types (of wrappers)
  - Optioneel
- @Column
  - name-attribuut → kolom in tabel
  - Optioneel
- Niet bewaren
  - @Transient

```
//@Basic
//@Column(name = "NAAM")
public String getNaam() { … }
public void setNaam(String naam) { … }}
```

ORM-framework

- API basisbewerkingen
  - CRUD (Create Read Update Delete)
- Taal queries
  - Maakt gebruik van attributen en klassen
- Metadata om mapping te definiëren
- Technieken om transactionele objecten te gebruiken
  - Dirty checking: object gewijzigd?
  - Lazy association fetching: attributen pas ophalen als ze nodig zijn
  - Andere optimalisaties