Programming Advanced java

JPA

deel2

HOGESCHOOL PXL

JPA

1. **Domeinmodel**
2. Extra configuratie
3. Mapping van datatypes
4. Relaties
5. Zoekopdrachten

JPA - domeinmodel

Object Relational Mapping = relationele tabellen <-> java objecten

Domeinmodel mapping	
vanuit het domeinmodel	• code → databaseschema • nieuwe toepassingen • naamgeving entity-klasse en properties • meestal geen extra configuratie nodig
vanuit het relationeel model	• databaseschema → code • bestaande database • naamgeving entity-klasse en properties (of extra configuratie)
vanuit het midden	• domeinmodel != relationeel model • extra configuratie nodig

JPA - domeinmodel

```
<property name="javax.persistence.schema-generation.database.action"
value="drop-and-create" />
```

Waarde	***Betekenis***
`none`	Er wordt geen actie ondernomen.
`create`	De tabellen worden automatisch aangemaakt.
`drop-and-create`	De tabellen worden verwijderd en opnieuw aangemaakt.
`drop`	De tabellen worden verwijderd.

JPA

1. Domeinmodel
2. **Extra configuratie**
3. Mapping van datatypes
4. Relaties
5. Zoekopdrachten

JPA – extra configuratie

Tabel configuratie

```
@Entity
@Table(name="PERSONS")
public class Person {
    ...
}
```

Element	*Omschrijving*
`catalog`	De *catalog* van de tabel.
`name`	De naam van de tabel.
`schema`	Het schema van de tabel
`uniqueConstraints`	Reeks van *unique constraints*.
`indexes`	De lijst van indexen die voor deze tabel gemaakt moeten worden.

JPA – extra configuratie

Tabel configuratie

```
@Entity
@Table(name = "PERSONS" ,
        indexes={@Index(name="LAST_NAME_INDEX",
                        columnList="LAST_NAME")})
public class Person {
    ...
}

@Entity
@Table(name="PERSONS",
        uniqueConstraints = {
            @UniqueConstraint(columnNames="firstName")
        })
```

JPA – extra configuratie

Kolom configuratie

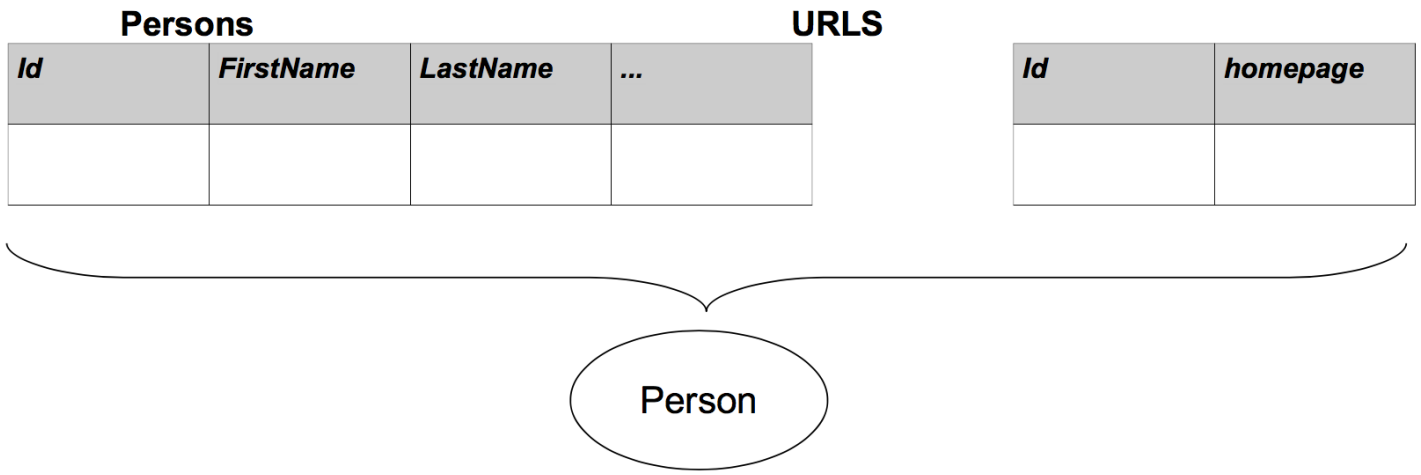
```
@Id
@GeneratedValue
private long id;
@Column(name="FIRST_NAME")
private String firstName;
@Column(name="LAST_NAME")
private String lastName;
```

Element	*Omschrijving*
`columnDefinition`	SQL-fragment dat gebruikt wordt bij de DDL voor de kolom.
`insertable`	Geeft aan of dit veld wordt opgegeven bij het invoegen van een nieuw record. Standaard is `true`.
`length`	De lengte van de kolom.
`name`	De naam van de kolom.
`nullable`	Geeft aan of de kolom `NULL`-waarden mag bevatten. Standaard is `true`.
`precision`	De nauwkeurigheid van getallen in geval van een decimale waarde.
`scale`	De schaal van getallen in geval van een decimale waarde.
`table`	De naam van de (secundaire) tabel die dit veld bevat.
`unique`	Geeft aan of dit veld uniek moet zijn. Standaard `false`.
`updatable`	Geeft aan of dit veld aangepast moet worden bij een *update*. Standaard `true`.

JPA – extra configuratie

Secundaire tabellen

```
@Entity
@Table(name="PERSONS")
@SecondaryTable(name="URLS")
public class Person {
   ...
   @Column(table="URLS")
   private String homepage;

   ...
}
```



JPA

1. Domeinmodel
2. Extra configuratie
3. **Mapping van datatypes**
4. Relaties
5. Zoekopdrachten

JPA – mapping van datatypes

Mapping enkelvoudige datatypes

Annotatie	***Omschrijving***
`@Basic(optional)`	Hiermee geven we aan of deze waarde optioneel is; met andere woorden of een lege waarde is toegestaan. Mogelijke waarden zijn `true` en `false`. De standaardwaarde is `true`. Voorbeeld: `@Basic(optional=false)`
`@Basic(fetch)`	Hiermee geven we aan of deze waarde onmiddellijk of vertraagd gelezen dient te worden. Bij vertraagd lezen, zullen de gegevens maar uit de databank gelezen worden zodra ze voor de eerste keer worden opgevraagd door middel van een *getter*. Mogelijke waarden zijn `FetchType.LAZY` en `FetchType.EAGER`. De standaardwaarde is `EAGER`. Merk op dat dit slechts een aanbeveling en geen onvoorwaardelijke verplichting is voor de implementatie van het persistentiemechanisme. Het *fetch*-type heeft de bedoeling performantieoptimalisaties mogelijk te maken. Voorbeeld: `@Basic(fetch=FetchType.LAZY)`

Hibernate → @basic(optional=false) == @Column(nullable=false)

→ fetchtype = eager

JPA – mapping van datatypes

Datatypes

→ Aanpassen gegenereerd datatype

- Primitieve datatypes
  → @Column(columnDefinition=*java-type*)
- Datum en tijden
  → @Temporal(TemporalType.DATE)
- Het opsommingstype
  → @Enumerated(EnumType.STRING)
- Grote objecten
  → @Lob
- Speciale datatypes
  → @Transient : uitsluiten van persistentie

JPA – mapping van datatypes

- Primitieve datatypes
  → @Column(columnDefinition=*java-type*)

Java-type
`boolean - Boolean`
`byte - Byte`
`int - Integer`
`long - Long`
`short - Short`
`float - Float`
`double - Double`
`BigDecimal`
`BigInteger`
`String`

JPA – mapping van datatypes

- Datum en tijden
  → @Temporal(TemporalType.DATE)

Java-type
`java.util.Date`
`java.util.Calendar`
`java.sql.TimeStamp`

Annotatie	*Omschrijving*
`@Temporal(TemporalType.DATE)`	Enkel de datum.
`@Temporal(TemporalType.TIME)`	Enkel de tijd.
`@Temporal(TemporalType.TIMESTAMP)`	Datum en tijd.

JPA – mapping van datatypes

- Het opsommingstype
  → @Enumerated(EnumType.STRING)

```
package persons;

public enum GenderType {
    MALE,FEMALE
}
```

```
@Enumerated(EnumType.STRING)
private GenderType gender;
```

Annotatie	***Omschrijving***
`@Enumerated(EnumType.ORDINAL)`	Het volgnummer. Standaardwaarde.
`@Enumerated(EnumType.STRING)`	De *string*-representatie.

JPA

1. Domeinmodel
2. Extra configuratie
3. Mapping van datatypes
4. **Relaties**
5. Zoekopdrachten

JPA – relaties

Soorten:

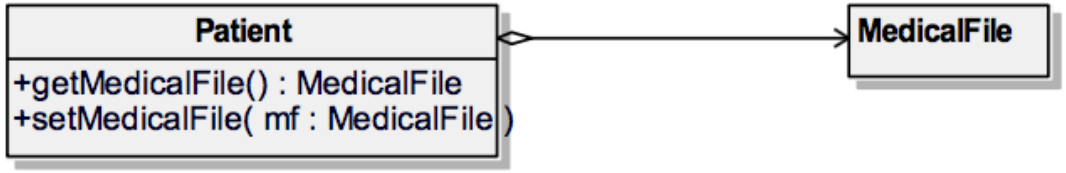
- One to one
- One to many
- Many to one
- Many to many

Type Navigeerbaarheid:

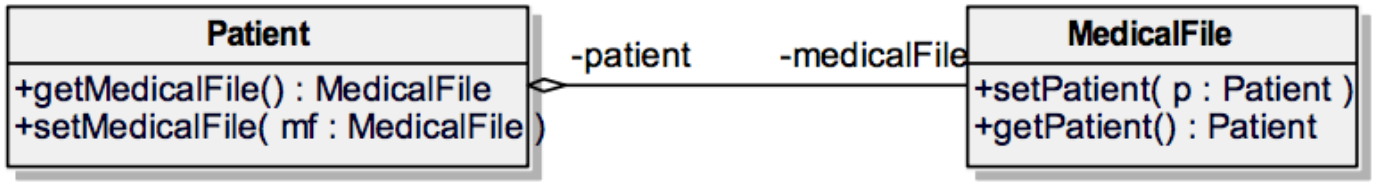
- unidirectional
- bidirectional

JPA – relaties – one to one

```
package medical;
import javax.persistence.*;

@Entity
public class Patient {
   @Id
   @GeneratedValue
   private long id;
   private String name;

   @OneToOne
   private MedicalFile medicalFile;

   public MedicalFile getMedicalFile() {
      return medicalFile;
   }

   public void setMedicalFile(MedicalFile medicalFile) {
      this.medicalFile = medicalFile;
   }

   public long getId() {
      return id;
   }

   public String getName() {
      return name;
   }
```



JPA – relaties – one to one bidirectional




```
@OneToOne(mappedBy="medicalFile")
private Patient patient;

public Patient getPatient() {
    return patient;
}

public void setPatient(Patient patient) {
    this.patient = patient;
}
```

JPA – relaties – one to one annotation elementen

Element	***Omschrijving***
`cascade`	Geeft het *cascade*-gedrag van de relatie aan.
`fetch`	Geeft het *fetch*-type van de relatie aan.
`mappedBy`	Geeft het veld aan dat bij de eigenaar de relatie vertegenwoordigt.
`optional`	Geeft aan of deze relatie optioneel is. Standaard `true`.
`orphanRemoval`	Geeft aan of objecten die losgekoppeld worden van de relatie automatisch verwijderd moeten worden. Standaard `false`.
`targetEntity`	Geeft de klasse aan van het gerelateerde object. Dit is enkel nodig indien dit niet duidelijk is aan de hand van de variabele of verzameling; bijvoorbeeld bij een *raw collection*.

JPA – relaties – one to one cascadetype

Cascade type	*Omschrijving*
`CascadeType.ALL`	Alle operaties.
`CascadeType.MERGE`	Enkel *merge*-operaties.
`CascadeType.PERSIST`	Enkel *persist*-operaties.
`CascadeType.REFRESH`	Enkel *refresh*-operaties.
`CascadeType.REMOVE`	Enkel *remove*-operaties.
`CascadeType.DETACH`	Enkel *detach*-operaties.

@OneToOne(cascade={CascadeType.PERSIST, CascadeType.REMOVE})

Default: geen cascading

JPA – relaties – one to one fetch-type

Fetch type	Omschrijving
`FetchType.LAZY`	Het gerelateerde object wordt vertraagd geladen wat wil zeggen pas op het moment dat het object wordt opgevraagd
`FetchType.EAGER`	Het gerelateerde object wordt onmiddelijk geladen. Default

```
@OneToOne(fetch=FetchType.LAZY)
```

JPA – relaties
One to many - Many to one

Verzameling referentie	Omschrijving
`Collection`	Algemene verzameling
`List`	Geordend Gesorteerd → @OrderBy
`Set`	Geen duplicaten
`Map`	Elementen opvragen via een veld

JPA – relaties
One to many - Many to one

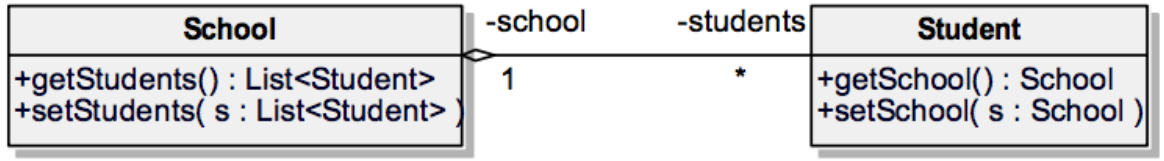
@OneToMany

Element	*Omschrijving*
`cascade`	Geeft het cascadegedrag van de relatie aan. Standaard niet ingesteld.
`fetch`	Geeft het *fetch*-type van de relatie aan. Standaard `LAZY`.
`mappedBy`	Geeft het veld aan dat aan de andere kant de relatie vertegenwoordigt.
`orphanRemoval`	Geeft aan of objecten die losgekoppeld worden van de relatie automatisch verwijderd moeten worden. Standaard `false`.
`targetEntity`	Geeft de klasse aan van het gerelateerde object.

@ManyToOne

Element	*Omschrijving*
`cascade`	Geeft het cascadegedrag van de relatie aan. Standaard niet ingesteld.
`fetch`	Geeft het *fetch*-type van de relatie aan. Standaard `EAGER`.
`optional`	Geeft aan of deze relatie optioneel is. Standaard `true`.
`targetEntity`	Geeft de klasse aan van het gerelateerde object.

JPA – relaties
One to many - Many to one




```
@Entity
public class School {
   @Id
   @GeneratedValue
   private long id;

   private String name;

   @OneToMany(mappedBy="school")
   private List<Student> students = new ArrayList<>();

   public List<Student> getStudents() {
      return students;
   }

   public void setStudents(List<Student> students) {
      this.students = students;
   }
```

```
@Entity
public class Student {
   @Id
   @GeneratedValue
   private long id;

   private String name;

   @ManyToOne
   private School school;

   public School getSchool() {
      return school;
   }

   public void setSchool(School school) {
      this.school = school;
   }
```

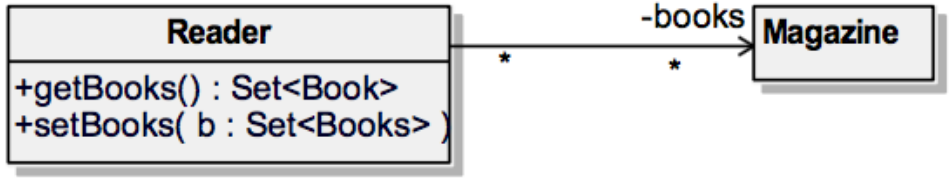
HOGESCHOOL PXL

JPA – relaties
Many to many

@JoinTable

Element	***Omschrijving***
`catalog`	De *catalog* van de tabel.
`inverseJoinColumns`	De kolomnaam van de *foreign key* in de tabel die niet de eigenaar is van de relatie.
`joinColumns`	De kolomnaam van de *foreign key* in de tabel die de eigenaar is van de relatie.
`name`	De naam van de tabel.
`schema`	Het schema van de tabel.
`uniqueConstraints`	*Unique constraints* die opgelegd moeten worden aan de tabel.

JPA – relaties
Many to many




```
@Entity
public class Reader {
   @Id
   @GeneratedValue
   private long id;

   private String name;

   @ManyToMany
   private Set<Magazine> magazines = new HashSet<>();
```

HOGESCHOOL PXL

JPA

1. Domeinmodel
2. Extra configuratie
3. Mapping van datatypes
4. Relaties
5. **Zoekopdrachten**

JPA – zoekopdrachten

- **Query API en JPQL** (Java Persistence Query Language)
  - Query / TypedQuery
  - Named queries

- Criteria API
- Native SQL queries

JPA – zoekopdrachten
Query / TypedQuery

Query

```
Query query = em.createQuery("select p from Person as p");
List<Person> persons = (List<Person>) query.getResultList();
```

TypedQuery

```
TypedQuery<Person> query =
        em.createQuery("select p from Person as p",Person.class);
List<Person> persons = query.getResultList();
```

JPA – zoekopdrachten

NamedQuery - static

annotation

```
@NamedQuery(name="findAll", query="select p from Person as p")
@Entity
```

orm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<entity-mappings version="2.1"
   xmlns="http://xmlns.jcp.org/xml/ns/persistence/orm"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence/orm
http://xmlns.jcp.org/xml/ns/persistence/orm_2_1.xsd ">
   <named-query name="findAll">
      <query>select p from Person p</query>
   </named-query>
</entity-mappings>
```

usage:

```
   EntityManagerFactory emf = Persistence
            .createEntityManagerFactory("course");
   EntityManager em = emf.createEntityManager();
   Query query = em.createQuery("select p from Person p");
   emf.addNamedQuery("findAll",query);
```

HOGESCHOOL PXL

JPA – zoekopdrachten
NamedQuery - dynamic

```
public class FindPerson {
    ...
        TypedQuery<Person> query =
                        em.createNamedQuery("findAll",Person.class);
        List<Person> persons = query.getResultList();
        return persons;
    ...
}
```

JPA – zoekopdrachten parameters

```
Query query = entityManager.createQuery
              ("select p from Person as p where p.lastName=:last");
query.setParameter("last", "Simpson");
```

```
Query query = entityManager.createQuery
              ("select p from Person as p where p.lastName=?1");
query.setParameter(1, "Simpson");
```

Beers JPA

- CRUD acties + unit tests

```
public Beer getBeerById(int id) throws BeerException ;
public List<Beer> getBeers() throws BeerException ;

public Beer updateBeer(Beer beer) throws BeerException ;

public Beer addBeer(String name, float price) throws BeerException;
public Beer addBeer(Beer beer) throws BeerException;

public void deleteBeer(Beer beer) throws BeerException;
```

- Add Brewer model met one-to-many relation + unit tests

- Extra
  - Gebruik MySql
  - queries
    - getBeerPriceByName (parameter)
    - getBeersByBrewerName (join)
    - Oefening p288