Relationales Datenbankpraktikum

Dr. A. Groß, M.Junghanns, V. Christen, Z. Sehili

Aufgabe 3 – Agenda

- Überblick zur Aufgabenstellung
- Hibernate
- Verwendung des Hibernate- Mapping und HQL innerhalb einer Konsolenapplikation

Aufgabe 3

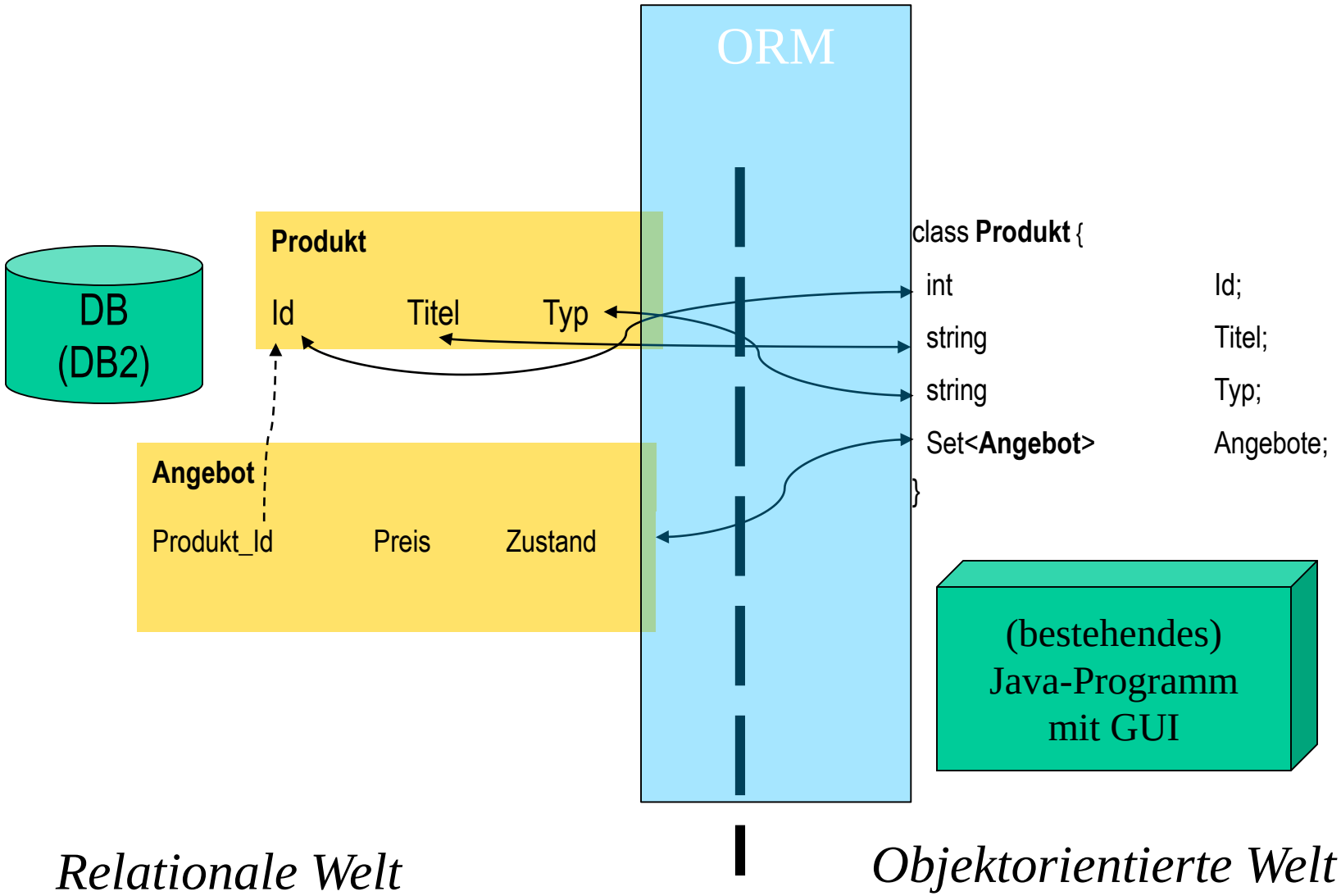
Aufgabe 3.1.: OR Mapping

- Definition eines Objekt Relationalen Mappings zwischen den Relationen des DB- Schemas und den Klassendefinitonen des UML- Modells

Aufgabe 3.2 Anwendung

- Implementierung einer Java-Konsolenapplikation, die eine definierte Menge von Funktionalitäten umfasst
  - Verwendung des definierten Hibernate- Mappings, HQL → **KEINE** Verwendung von SQL Statements

Aufgabe 3.1: ORM mit Hibernate



Hibernate

- Objekt-Relationales Persistenz-Framework
  - Open-Source-Projekt: www.hibernate.org
  - "Java-Objekt in relationaler Datenbank speichern und laden"

- Objekt-Relationales-Mapping (ORM)
  - Beschreibung durch XML-Konfigurationsdateien oder **Annotationen**
  - Trennung von DB-Anfragen und Java-Code

Hibernate: Beispiel

Datenbank
relational

```
class Kunde {
  int          id;
  string       name;
  int          plz;
  string       ort;
  date         datum;
  Set<Speise>  speisen;

}
```

```
class Speise {
  int          id;
  string       gericht;
  string       zutaten;
  Set<Kunde>   kunden;

}
```

Java
objektorientiert

Hibernate: Beispiel .

Kunde.java mit Annotationen

```
@Entity
@Table(name = "Kunde",
catalog = "dbprak16", schema =
"public")
public class Kunde {
  @Id
  @Column(name = "ID")
  private  int id;
  @Column(name = "NAME")
  private String name;
  @Column(name = "PLZ")
  private int plz;
  @Column(name = "ORT")
  private String ort;
  @Column(name = "DATUM")
  private Date datum;
```

```
@ManyToMany
 @JoinTable(name = "BESTELLUNG",
 catalog = "dbprak_16", schema
="public",
 joinColumns =
@joinColumn(name"ID_KUNDE"),
 inverseJoinColumns =
 @joinColumn(name="ID_SPEISE"))
 private Set<Speise> speisen;

 /*empty Constructor*/
 public Kunde(){}


 /*getter & setters */
}
```

Hibernate: Beispiel ..

Kunde.java mit Annotationen

```
@Entity
@ Table(name = “Speise”,
catalog = dbprak16, schema =
 “public”)
public class Speise{
  /*ID und atomare Attribute analog*/

@ManyToMany(mappedBy=“speisen”)
Set<Kunde> kunden;

/*Empty Constructor*/

/*getter & setters*/
}
```

Hibernate: Beispiel ...

Hibernate.cfg.xml

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
  "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <property name="hibernate.connection.driver_class"> ... </property>
    <property name="hibernate.connection.url"> ... </property>
    <property name="hibernate.connection.username"> ... </property>
    <property name="hibernate.connection.password"> ... </property>
    ...
    <mapping class="Kunde "/>
    <mapping class="Speise"/>
  </session-factory>
</hibernate-configuration>
```

Hibernate: Beispiel ....

Verbindung herstellen

```
import org.hibernate.*;
import org.hibernate.cfg.Configuration;

private SessionFactory sessionFactory;

try {
  System.out.println( "Initializing Hibernate" );
  sessionFactory = new Configuration().configure().buildSessionFactory();
  System.out.println( "Finished Initializing Hibernate" );
} catch( HibernateException ex ) {
    ex.printStackTrace();
    System.exit( 5 );
}
```

Hibernate: Beispiel .....

Speichern

```
try {
  Session sess = sessionFactory.openSession();
  Transaction trx = sess.beginTransaction();

  Kunde kunde = new Kunde();   /* + Werte setzen */
  sess.save( kunde );
  /* + Java-Objekt Speise erzeugen */
  kunde.speisen.add( speise );

  trx.commit();
} catch( HibernateException ex ) {
    if (trx != null)
      try { trx.rollback(); } catch( HibernateException exRb ) {}
    throw new RuntimeException( ex.getMessage() );
} finally {
    try { if( sess != null ) sess.close(); } catch( Exception exCl ) {}
}
```

Hibernate: Beispiel .....

Laden

```
List kunden = sess.createQuery( "from Kunde" ).list();
for( int i=0; i<kunden.size(); i++ ) {
  Kunde kunde = (Kunde)kunden.get( i );
  System.out.println( "Kunde:  " + kunde.name );
  ...
  Iterator itr = kunde.speisen.iterator();
  while( itr.hasNext() ) {
    System.out.print( ((Speise) itr.next()).gericht );
  }
}
```

Hibernate Hinweise

- Abbildung von Generalisierungen

Obligatorisch:University/Company und City/Country/Continent

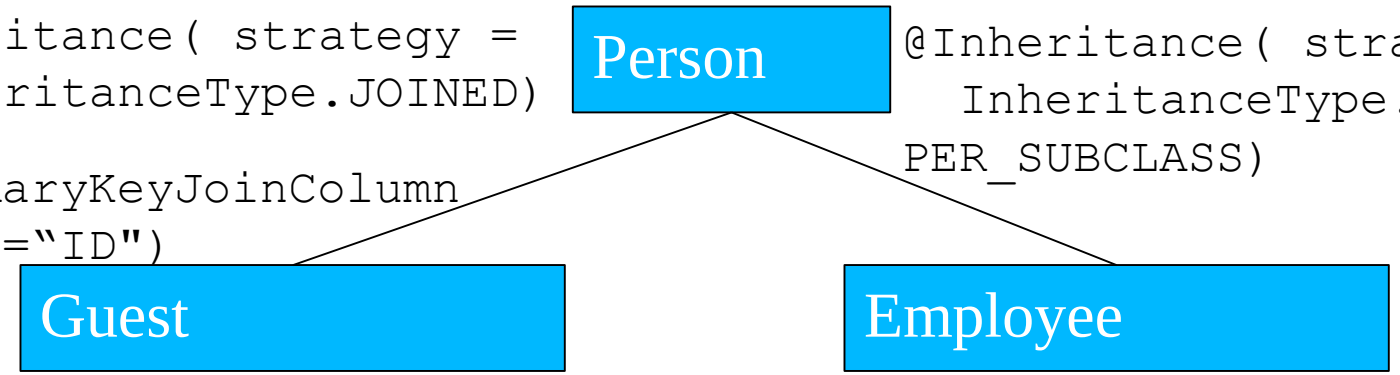
- Vertikal

```
@Inheritance( strategy =
  InheritanceType.JOINED)

 @PrimaryKeyJoinColumn
 (name="ID")
```

Horizontal

```
@Inheritance( strategy =
  InheritanceType.TABLE_
PER_SUBCLASS)
```



- Assoziationsklassen
  - Mapping zu Assoziationsklasse mit externer oder innerer Id-Klasse der involvierten Relationen & Beziehungsattribute
    - Zusammengesetze Id & n:1 Beziehung von Assoziationsklasse zu Klasse
- **Obligatorisch** für **workAt/studyAt** sonst optional
- Bidirektional vs. Unidirektional bei n:m Beziehungen
  - Nicht immer sinnvoll, dass beide Klassen Sets halten
  - Hoher Änderungsaufwand

3.2 Java Konsolenanwendung

Implementierung einer Konsolenanwendung unter Verwendung des definierten Mappings und HQL mit folgenden Funktionalitäten

Personenbezogen (Angabe der Id)

a) *Profil*: Ausgabe des Profils einer Person

b) *Gemeinsame Interessen*: Ausgabe von überlappenden Interessen (tag ID+Name) für eine Person bzgl. seiner Freunde.

c) *Überlappende Freundesmengen*: Ausgabe von gemeinsamen Freunden (Id+Name) für zwei Personen (Id)

d) *Ähnlichste Interessen*: Ausgabe der Personen (Id) mit der maximalen Überlappung (absolute Anzahl) bzgl. der Interessen für eine Person

e) *Job-Empfehlung*: Empfehlung von Unis/Firmen deren Lokation dem Wohnort entspricht und mindestens ein Freund bei dieser angestellt ist.

3.2 Java Konsolenanwendung

f) *Kürzester Pfad zu einer anderen Person*: Bestimmung des kürzesten Pfades zwischen zwei Personen (ID oder Name) bzgl. der Freundschaftsbeziehung.

Eingabe der StoredProcedure Ids und Rückgabe der Personen (Id,Name,Pfadlänge) die Element des kürzesten Pfades sind

3.2 Java Konsolenanwendung

Statistik

a) Ausgabe der TagClass Hierarchie in Form einer Taxonomie

0 Thing

0.1 Agent

0.1.1 Person

0.1.2 Organization

0.2 Work …

b) Beliebteste Kommentare: Bestimmung der Kommentare (Id, Creator Name) deren Anzahl an “Likes” größer als k ist

c) Land mit häufigsten Kommentaren und Posts: Ermittlung des Landes mit der höchsten Anzahl an Posts und Kommentaren

Vorgehen

1. Definition der Interfaces **PersonRelatedAPI** und **StatisticAPI** mit den entsprechenden Methoden und notwendigen Paramatern
2. Implementierung der Klassen **PersonRelatedImpl** und **StatisticImpl** die jeweilige Interfaces implementieren
3. Implementierung der Konsolenmenüs

```
while(!exit){
    printMenu();
    int selection = readSelection();
    performAction(selection);
}
public int readSelection(){
    Scanner scanner = new Scanner(System.in);
    return Integer.parseInt(scanner.readLine());
}
public void performAction( int selection){
    switch(selection){case 1: …break; case 2:… break;…}
}
```

Zusammenfassung

- Aufgabenstellung
  - Definition eines Hibernate- Mappings
  - Beispiel und Hinweise zur Erstellung des Hibernate- Mappings
  - Java- Konsolenanwendung mit personenbezogenen- und statistischen Funktionalitäten