Relationales Datenbankpraktikum

V. Christen, M. Franke, Z. Sehili, Dr. J. Zschache, M. Nentwig

Aufgabe 3 – Agenda

- Überblick zur Aufgabenstellung
- Hibernate Intro
- Verwendung des Hibernate- Mapping und HQL innerhalb einer GUI/Konsolen- Applikation

Aufgabe 3

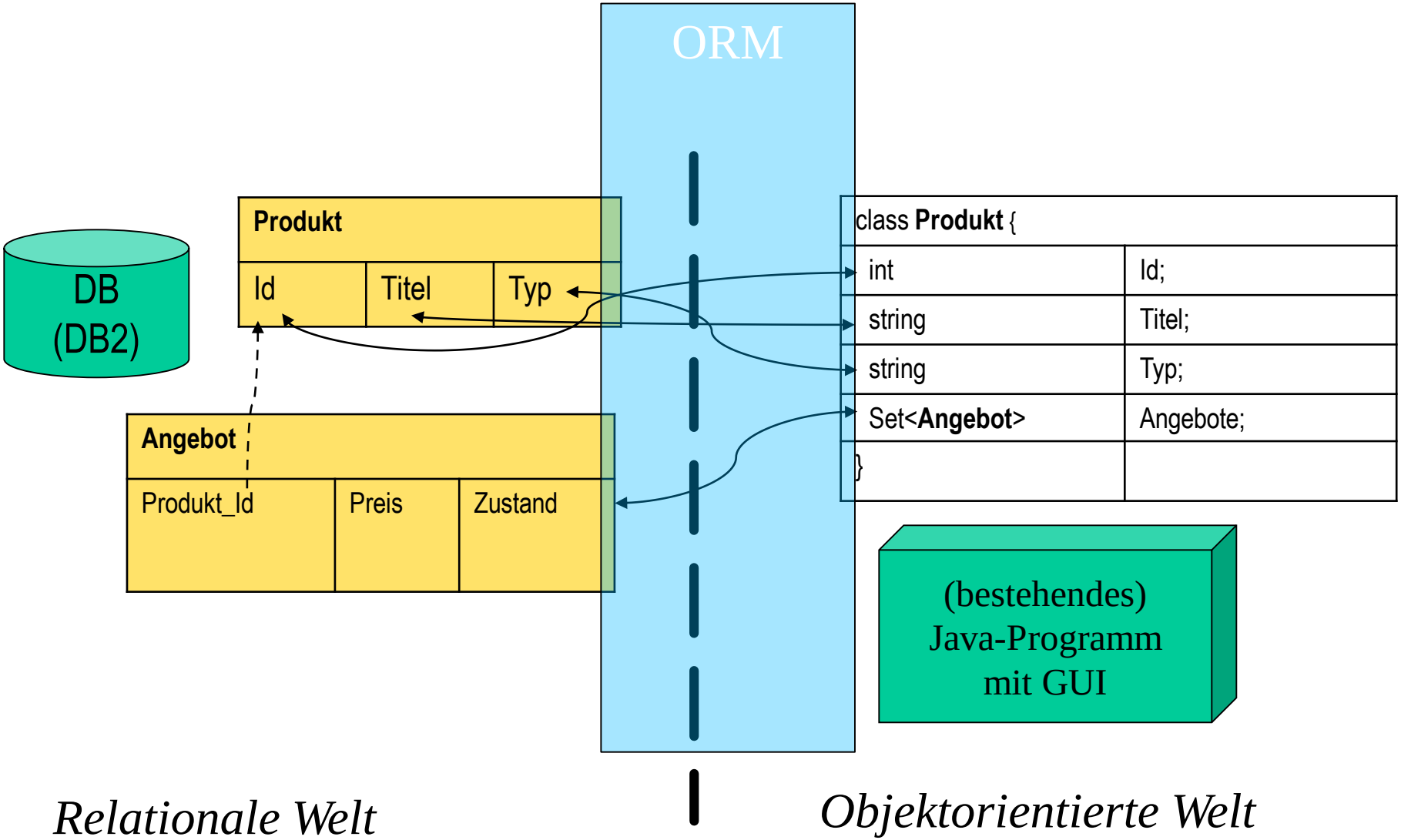
Aufgabe 3.1.: OR Mapping

- 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 oder programmatisch → **KEINE** Verwendung von SQL Statements

Aufgabe 3.1: ORM mit Hibernate

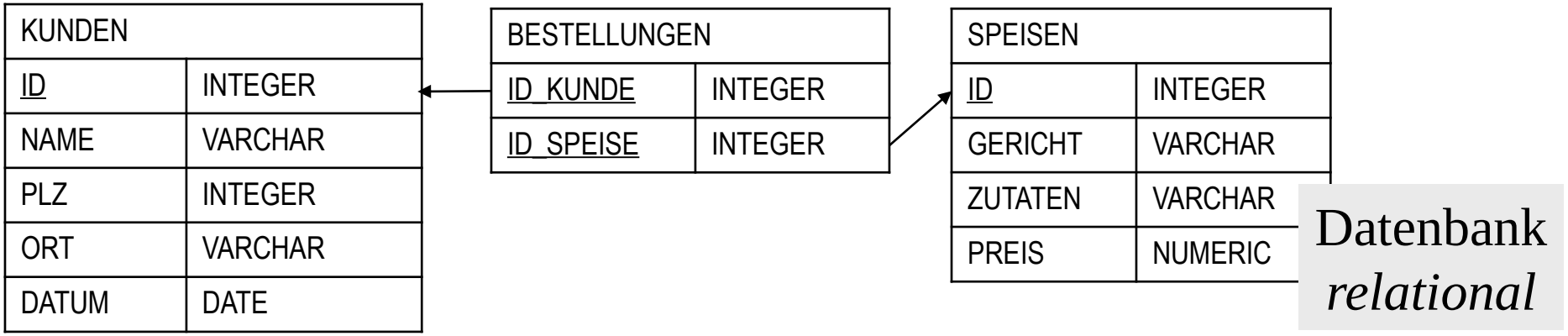


Hibernate

- 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



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

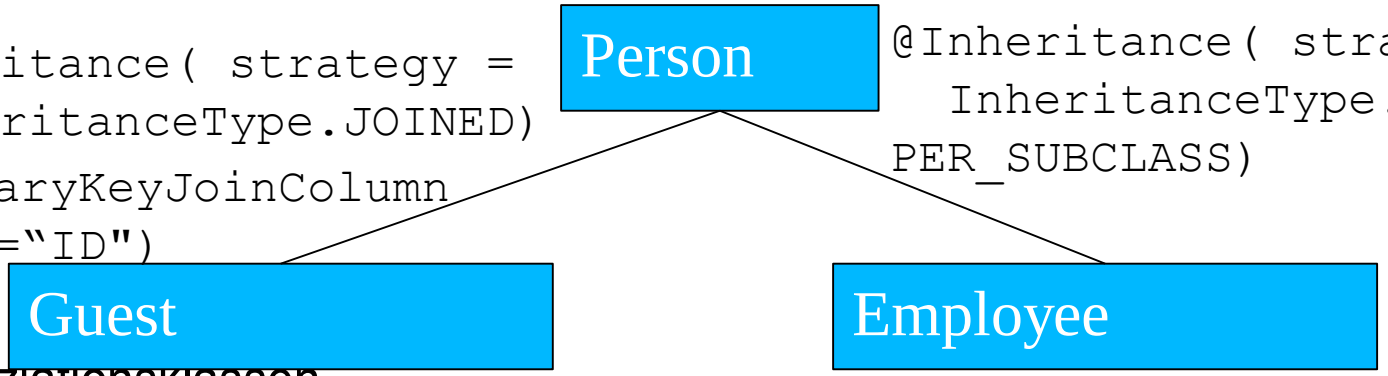
Obligatorisch: University/Company und City/Country/Continent

- 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
- Bidirektional vs. Unidirektional bei n:m Beziehungen
  - Nicht immer sinnvoll, dass beide Klassen Sets halten
  - Hoher Änderungsaufwand

3.2 Java Konsolenanwendung

- Implementierung einer API sowie Konsolen/GUI -Anwendung unter Verwendung des definierten Mappings und HQL mit folgenden Funktionalitäten

API

Use Case I: Klausurergebnisse eintragen

1. Liste aller Klausuren anzeigen, geordnet nach Datum
2. Nutzer wählt Klausur aus
3. Eingabe Matrikelnummer
4. DB-Zugriff -> Ist Student in DB vorhanden und für Klausur angemeldet?
5. Ja: Eingabe der Punkte
6. Test, ob Eingabe korrekt + Ausgabe Gesamtpunktzahl
7. Gehe zu 3. bis alle Studenten verarbeitet sind
8. Anzeige aller abwesenden Studenten

3.2 Java Konsolenanwendung

Use Case II: Klausurergebnisse (Notenschnitt) einsehen und anpassen

1. Liste aller Klausuren anzeigen, geordnet nach Datum
2. Nutzer wählt Klausur aus
3. Anzeige/PDF Generierung und Anpassung
   - Noten-Verteilung (Histogram)
   - Punkte-Noten-Verteilung
   - geordnete Liste aller Studenten mit Note und Punkte
   - Anpassung Punkte-Noten-Verteilung
4. Gehe zu 3. solange nicht exit

Use Case III: Top-Studenten

1) Eingabe der Gewichtungen
2) Anzeige der Liste aller Top-Studenten (Nutzung View)

3.2 Java Konsolenanwendung

Use Case II: Klausurergebnisse (Notenschnitt) einsehen und anpassen

1. Liste aller Klausuren anzeigen, geordnet nach Datum
2. Nutzer wählt Klausur aus
3. Anzeige(Konsole,wenn intuitive ;D)/PDF Generierung und Anpassung
   - Noten-Verteilung (Histogram)
   - Punkte-Noten-Verteilung
   - geordnete Liste aller Studenten mit Note und Punkte
   - Anpassung Punkte-Noten-Verteilung
4. Gehe zu 3. solange nicht exit

Vorgehen

Programm- Teil A: API

- Regelt den Zugriff auf die Datenbank
- Stellt notwendige Operationen für die obigen Use Cases bereit
  - Trennung von Logik und View

Programm- Teil B: GUI/Konsolen-Anwendung (View)

- Implementierung GUI bzw. Konsolen-Logik zur Abarbeitung/Ausführung der obigen Use Cases
- MVC Pattern

Zusammenfassung

- Aufgabenstellung
  - Definition eines Hibernate- Mappings
  - Beispiel und Hinweise zur Erstellung des Hibernate- Mappings
  - Java- Konsolenanwendung mit 3 Use Cases