Objektorientierte Datenbanken

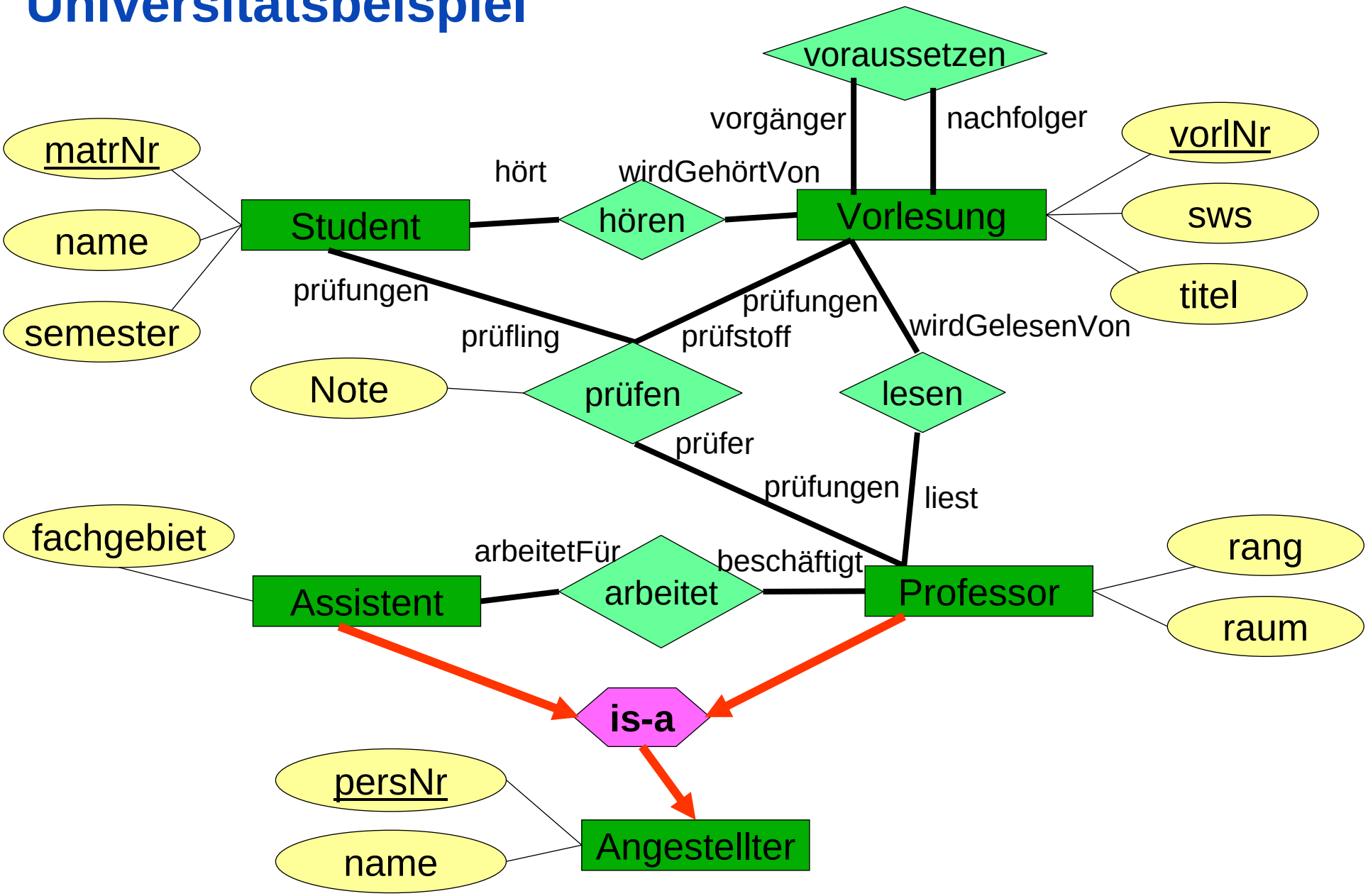
Vorlesung 11
Sebastian Iwanowski
FH Wedel

Hibernate: 2. Teil

Einiges des folgenden Materials stammt von Gavin King (Initiator von Hibernate).

Seine Beispiele wurden größtenteils auf unser Universitätsbeispiel angepasst.

Universitätsbeispiel



Wesentliche Eigenschaften von Hibernate

- Transparente Persistenz
- Transitive Persistenz (Persistenz per Erreichbarkeit)
- Inheritence mapping strategies
- **Automatic dirty object checking**
- **Detached Object Support**
- **Intelligent fetching and caching**
- **Unterschiedliche Anfragekonzepte (Queries und Criteria)**

Automatic Dirty Object Checking

- **Neue Werte für persistente Felder werden automatisch in die Datenbank geschrieben.**
  Im Prinzip wie bei JDO, aber mit SQL-update-Mechanismus für Wahrung der Objektidentität.

- **Unnötige Aktualisierungen der Datenbank werden vermieden.**
  Bei Zuweisung eines neuen Objekts wird zuerst nachgesehen, ob es tatsächlich unterschiedliche Werte enthält, bevor eine Aktualisierung gemacht wird.

nicht ganz durchgängig !

Detached Object Support

For applications using servlets + session beans:

- You don't need to `select` a row when you only want to `update` it!
- You don't need DTOs anymore!
- You may serialize objects to the web tier, then serialize them back to the EJB tier in the next request
- Hibernate lets you *selectively* reassociate a subgraph! (essential for performance)

aus Gavin King: *Object / Relational Mapping with Hibernate*

Detached Object Support

Step 1: **Retrieve some objects in a session bean:**

```
session
    .createQuery(“from Student student where student.semester > 1 ”)
    .list();
```

Step 2: **Collect user input in a servlet / action:**

```
student.setSemester(student.getSemester()-1);
```

Step 3: **Make the changes persistent, back in the session bean:**

```
session.update(student);
```

nach Gavin King: *Object / Relational Mapping with Hibernate*

Detached Object Support

Even transitive persistence!

```
Session session = sf.openSession();
Transaction tx = session.beginTransaction();
Student student =
   (Student) session.get(Student.class, itemId);
tx.commit();
session.close();

Prüfung prüfung = student.getPrüfungen.getFirst();
prüfung.setNote(1);

Session session2 = sf.openSession();
Transaction tx = session2.beginTransaction();
session2.update(student);
tx.commit();
session2.close();
```

nach Gavin King: *Object / Relational Mapping with Hibernate*

Detached Object Support

The Big Problem

Detached objects + Transitive persistence:

- How do we distinguish between newly instantiated objects and detached objects that are already persistent in the database?

Solution

- Version property (if there is one)
- Identifier value e.g. `unsaved-value="0"` (only works for generated surrogate keys, not for natural keys in legacy data)
- Write your own strategy, implement `Interceptor.isUnsaved()`

aus Gavin King: *Object / Relational Mapping with Hibernate*

Intelligent fetching and caching

Fetching strategies:

- **Immediate**
  Alles im cache wird sofort mit der Datenbank aktualisiert.

- **Lazy**
  Aktualisiert wird erst beim ersten Zugriff.

- **Eager (Outer Join)**
  Aktualisiert werden alle Daten, die zu einem Objekt gehören, das aktualisiert wird (Klassen und Navigationstiefe einstellbar).

- **Batch**
  wie eager, aber erst nach erstem Zugriff
  und auch nur bis zu einer einstellbaren Navigationstiefe

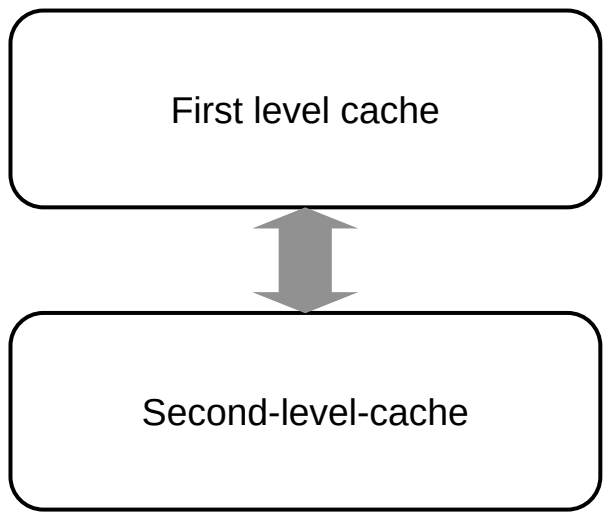
Intelligent fetching and caching

Caching:

Aufgaben:

- **Verringerung der Häufigkeit des tatsächlichen Zugriffs auf die Datenbank**
- **Arbeiten mit identischen Objekten für gleiche Werte**

Hibernate-Cache-Architektur:



Session

Session factory

First Level cache:

- **obligatorisch**
- **nicht konfigurierbar**

Second Level cache:

- **optional**
- **vielfältig konfigurierbar: Art der Objekte, Vorhaltungszeit, ...**

Anfragekonzepte

Was gab es vor Hibernate?

- **SQL-Anfragen** — ***JDBC***
- **SQL-ähnliche Anfragen mit Objekten statt Tabellen** — ***OQL (ODMG)***
- **Objektfilter** — ***JDOQL (JDO)***

Hibernate bietet alle drei Anfrageformen:

- **SQL**
- **HQL**
- **Criteria**

Hibernate Query Language (HQL)

Make SQL be object oriented:

- Classes and properties instead of tables and columns
- Polymorphism
- Associations
- *Much* less verbose than SQL

Full support for relational operations:

- Inner/outer/full joins, cartesian products
- Projection
- Aggregation (max, avg) and grouping
- Ordering
- Subqueries
- SQL function calls

aus Gavin King: *Object / Relational Mapping with Hibernate*

Hibernate Query Language (HQL)

HQL is a language for talking about “sets of objects”:

It unifies *relational operations* with *object models*

Simplest HQL Query:

```
from Student
```

i.e. get all the students:

```
List allStudents = session.createQuery(“from Student”).list();
```

nach Gavin King: *Object / Relational Mapping with Hibernate*

Hibernate Query Language (HQL)

HQL is a language for talking about “sets of objects”:

It unifies *relational operations* with *object models*

More realistic example:

```
select professor
from Professor professor
     join professor.liest vorlesung
where vorlesung.titel like ‘Grundlagen%’
     and vorlesung.sws > 1
```

i.e. get all the professors with a vorlesung of > 1 sws and title beginning with “Grundlagen”

nach Gavin King: *Object / Relational Mapping with Hibernate*

Hibernate Query Language (HQL)

HQL is a language for talking about “sets of objects”:

It unifies *relational operations* with *object models*

Projection:

```
select vorlesung.vorlNr, professor.name
from Professor professor
     join professor.liest vorlesung
where vorlesung.titel like ‘Grundlagen%’
     and vorlesung.sws > 1
order by vorlesung.sws desc
```

i.e. get the name and respective vorlNr of all the professors with a vorlesung of > 1 sws and title beginning with “Grundlagen” ordered by sws.

nach Gavin King: *Object / Relational Mapping with Hibernate*

Hibernate Query Language (HQL)

HQL is a language for talking about “sets of objects”:

It unifies *relational operations* with *object models*

Sorting functions:

```
select max (vorlesung.sws), vorlesung.vorlNr, professor.name
from Professor professor
     join professor.liest vorlesung
where vorlesung.sws > 1
order by max(vorlesung.sws) desc
```

i.e. get the maximum sws of a vorlesung of a professor, the professor’s name and the respective vorlNr of all the professors with a vorlesung of > 1 sws ordered by this max.

nach Gavin King: *Object / Relational Mapping with Hibernate*

Hibernate Query Language (HQL)

HQL is a language for talking about “sets of objects”:

It unifies *relational operations* with *object models*

Runtime fetch strategies (outer join option, eager fetching):

```
from Professor professor
     left join fetch professor.liest vorlesung
where vorlesung.titel like ‘Grundlagen%’
     and vorlesung.sws > 1
order by vorlesung.sws desc
```

lädt die Vorlesungsattribute gleich mit zusammen mit den Professorenattributen, aber nur diejenigen, die zu den Professoren der Lösungsmenge gehören.

nach Gavin King: *Object / Relational Mapping with Hibernate*

Criteria Queries

HQL query:

```
session.createQuery(“from Student”).list();
```

Criteria query:

```
session.createCriteria(Student.class).list();
```

Criteria Queries

HQL query:

```
session.createQuery
        (“from Professor professor
                join professor.liest vorlesung
                where vorlesung.titel like ‘Grundlagen%’
                        and vorlesung.sws > 1”).list();
```

Criteria query:

```
session.createCriteria(Professor.class)
        .setAlias (“liest”, vorlesung)
        add(Expression.like (vorlesung.titel, “Grundlagen”))
        add(Expression.gt (vorlesung.sws, new Integer (1))).list();
```

Criteria Queries with Examples

```
AuctionItem item = new AuctionItem();
item.setDescription("hib");
Bid bid = new Bid();
bid.setAmount(1.0);
List auctionItems =
   session.createCriteria(AuctionItem.class)
        .add( Example.create(item).enableLike(MatchMode.START) )
        .createCriteria("bids")
               .add( Example.create(bid) )
        .list();
```

Equivalent HQL:

```
   from AuctionItem item
        join item.bids bid
   where item.description like 'hib%'
        and bid.amount > 1.0
```

aus Gavin King: *Object / Relational Mapping with Hibernate*

Beim nächsten Mal:

Ausblick auf Persistenzstrategien der Zukunft
Zusammenfassung der Vorlesung