Objektorientierte Datenbanken

Vorlesung 11
Sebastian Iwanowski
FH Wedel

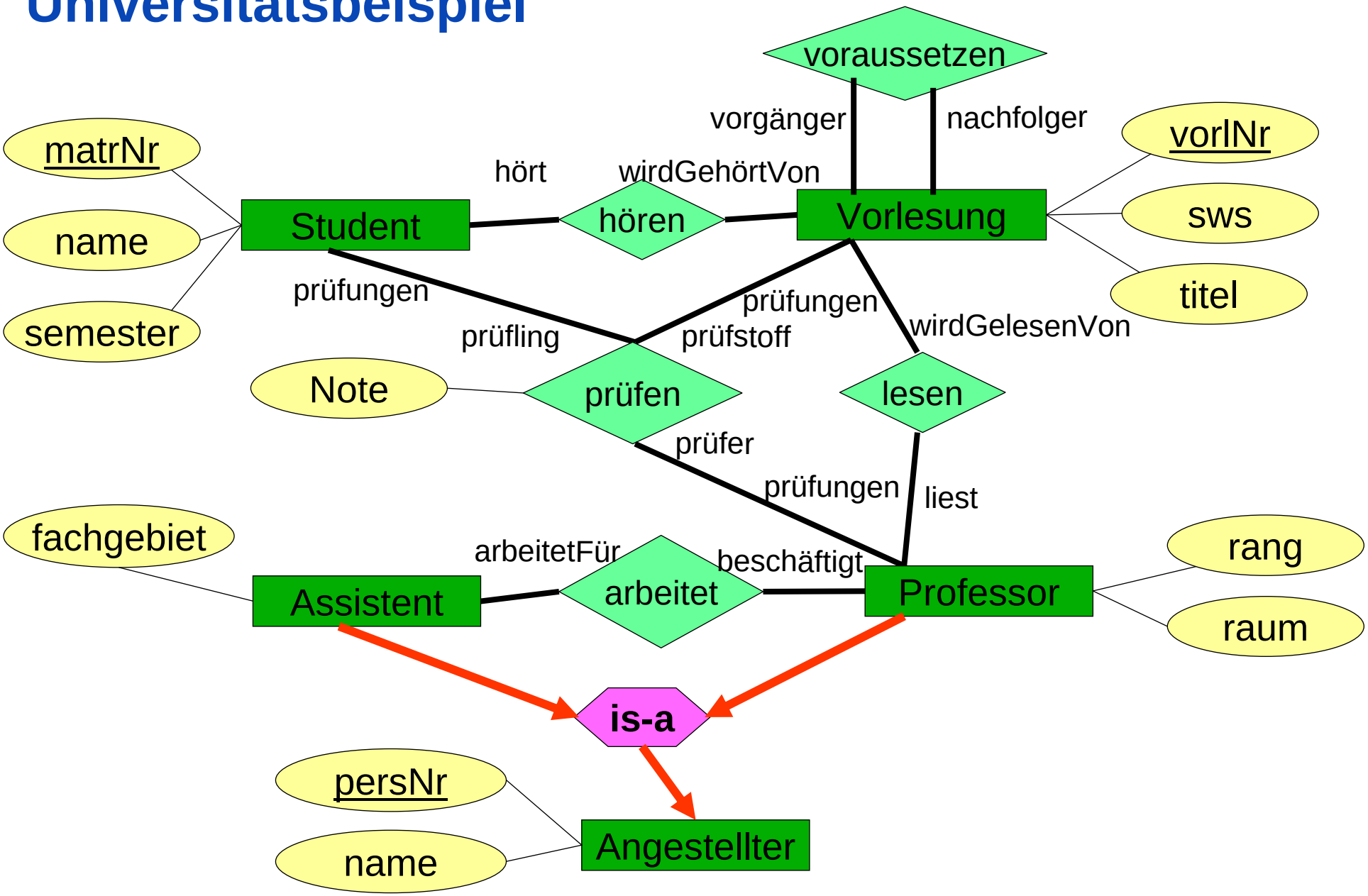
Wesentliche Eigenschaften von Hibernate

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

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

Seine Beispiele wurden größtenteils auf unser Universitätsbeispiel angepasst.

Universitätsbeispiel



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**

JDO-2 auch:

- **SQL**
- **JDOQL (single string)**
- **JDOQL (filter)**

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 + Sorting:

```
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*

Aggregation functions (e.g. max):

geht nur mit grouping:

```
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

Simple queries

HQL query:

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

Criteria query:

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

Criteria Queries

Joins and selections

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

Sorting and eager fetching

HQL query:

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

Criteria query:

```
session.createCriteria(Professor.class)
        .setFetchMode (“liest”, FetchMode.EAGER)
        .setAlias (“liest”, vorlesung)
        .add(Expression.like (“vorlesung.titel”, “Grundlagen%”))
        .add(Expression.gt (“vorlesung.sws”, new Integer (1)))
        .addOrder( Order.desc (“vorlesung.sws”)).list();
```

JDOQL

Eager fetching using fetch groups

Fetch group definition in XML metadata:

```
<class name = “Professor”>
        …
        <fetch-group name = “liestIncluded”>
                <field name = “liest”/>
        </fetch-group>
        …
</class>
```

Filter query:

```
pm.newQuery (…)
        .getFetchPlan()
        .addGroup (“liestIncluded”);
```