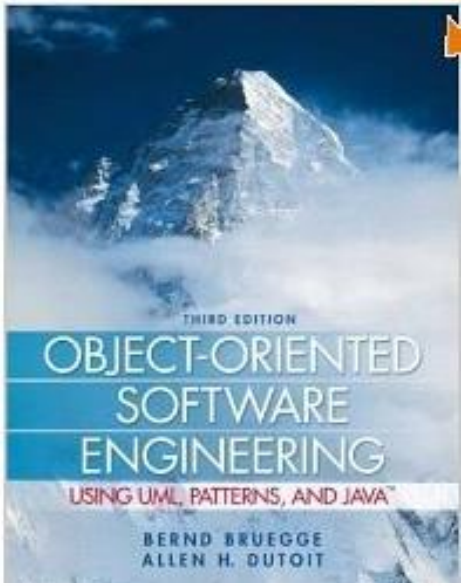
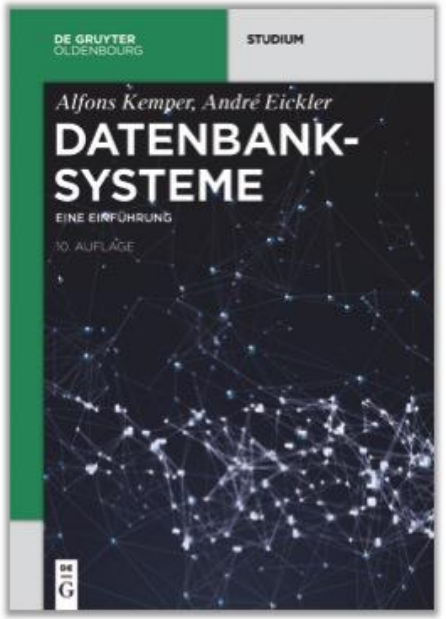
Einführung in die Informatik II für Ingenieurwissenschaften (MSE)

- Prof. Alfons Kemper, Ph.D.
- Simon Ellmann
- Christoph Anneser

- Teil 1:
  - Objektorientierte Modellierung (in UML) und
  - Programmierung in Java



- Teil 2:
  - Datenbanksysteme: Eine Einführung
  - Alfons Kemper und Andre Eickler
  - Oldenbourg Verlag, 10. Auflage, 2016



Vorlesung: Prof. Alfons Kemper

alfons.kemper@in.tum.de

Übungsbetrieb

- Übungsleitung:
  - Simon Ellmann (ellmann@in.tum.de)
  - Christoph Anneser (anneser@in.tum.de)

- Tutor:
  - Sebastian Reichbauer

Organisatorisches I

- Vorlesungs-Website: https://db.in.tum.de/teaching/ss24/ei2
- Moodle-Kurs: https://www.moodle.tum.de/course/view.php?id=96067

- Vorlesung:
  - Montags 10:00 – 11:15
  - Raum: BC2 0.01.17 (Garching-Hochbrück)
  - Aufzeichnungen des SS20 auf der Vorlesungs-Website

- Zentralübung:
  - Montags 11:30 – 12:30
  - Raum: BC2 0.01.17 (Garching-Hochbrück)
  - Aufzeichnungen des SS21 auf Panopto (need to sign in first!)

Organisatorisches II

- Fragestunde:
  - Donnerstags 15:30 – 16:30
  - Raum: BC2 0.01.17 (Garching-Hochbrück)

- Alle Materialen sind auf der Vorlesungs-Website zu finden

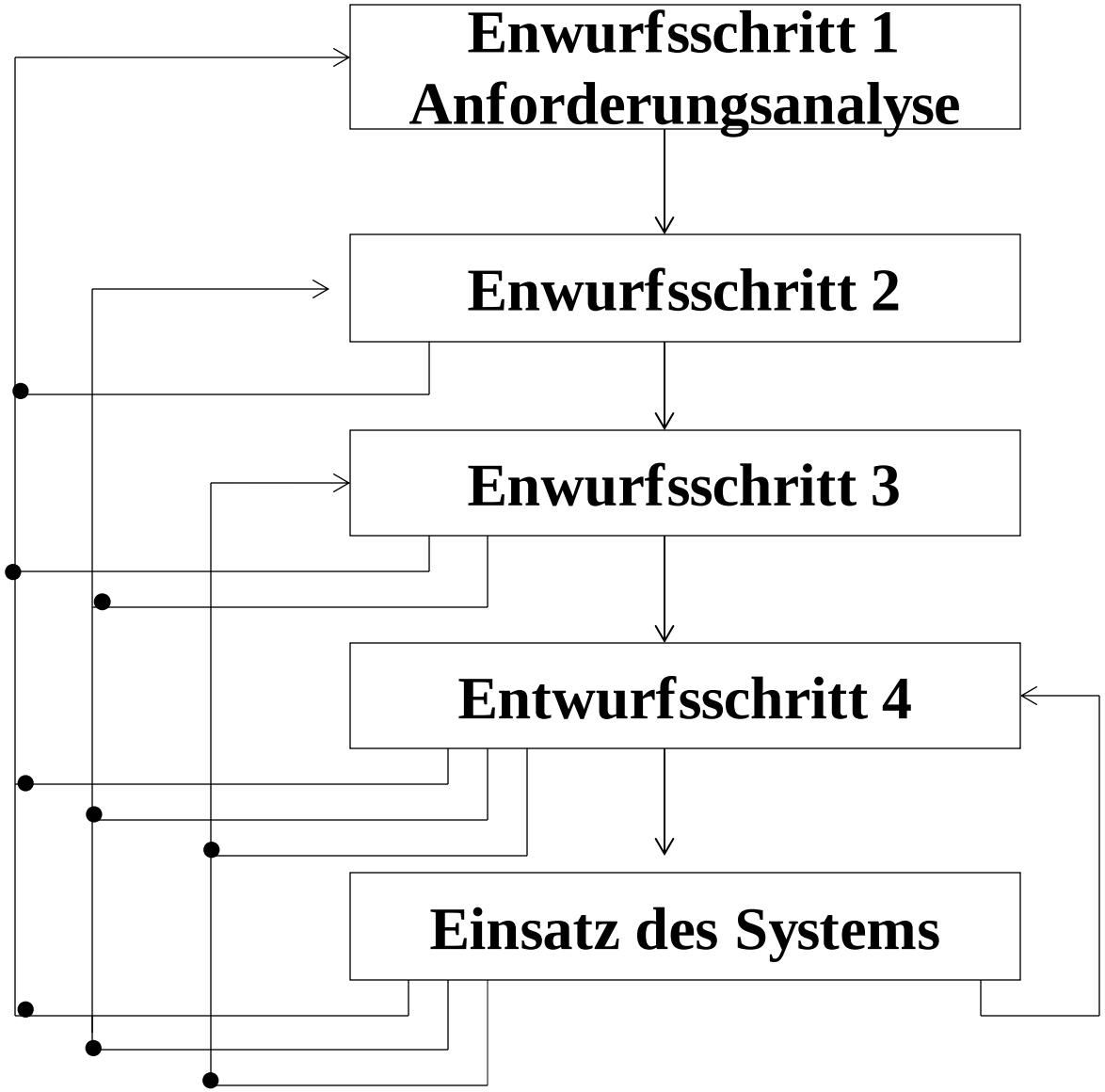
- Prüfungstermin:
  - Voraussichtlich Anfang September (ohne Gewähr)

Datenbankentwurf

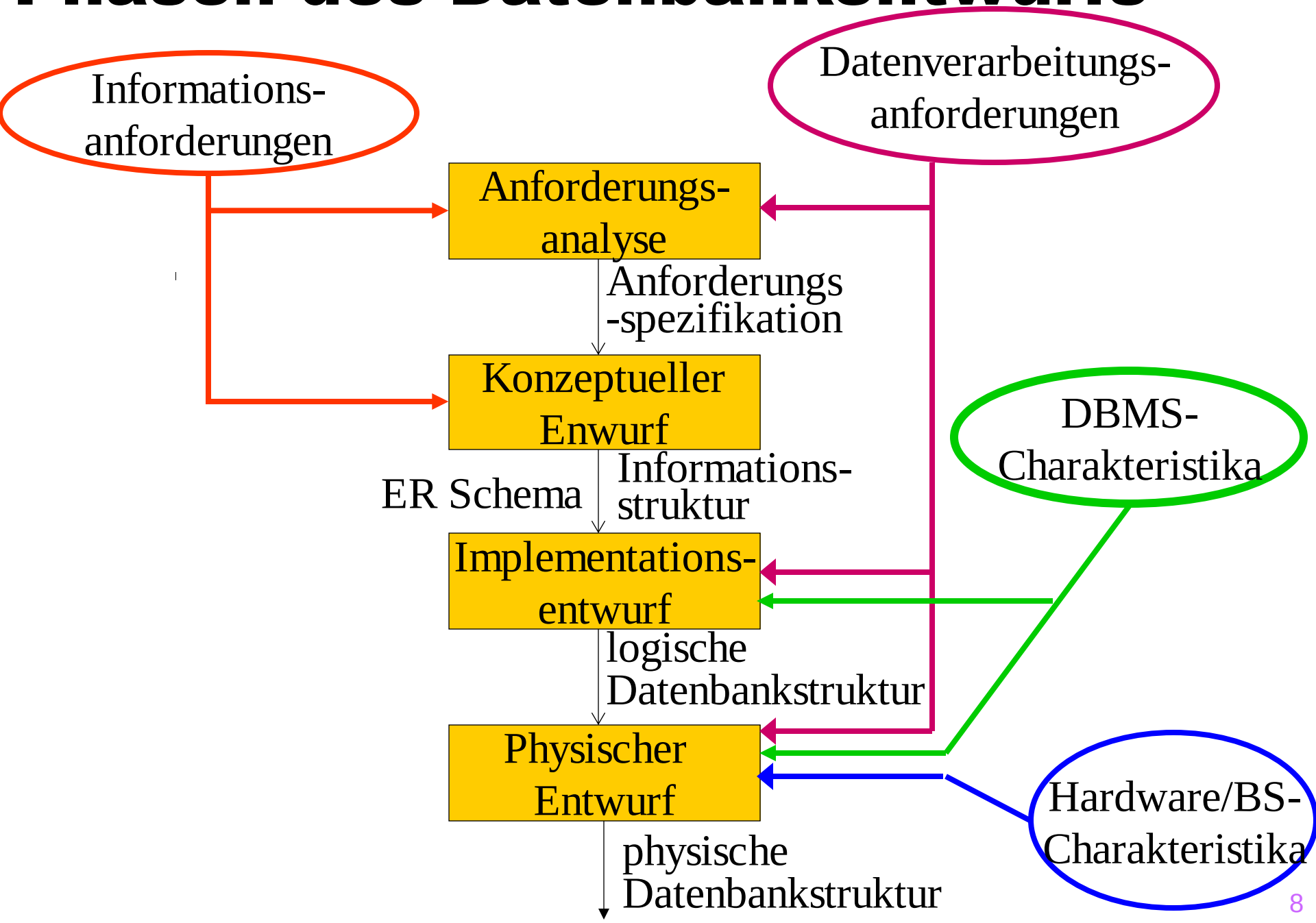
Abstraktionsebenen des Datenbankentwurfs

1. Konzeptuelle Ebene
2. Implementationsebene
3. Physische Ebene

Allgemeiner „top-down Entwurf“



Phasen des Datenbankentwurfs



Anforderungsanalyse

1. Identifikation von Organisationseinheiten
2. Identifikation der zu unterstützenden Aufgaben
3. Anforderungs-Sammelplan
4. Anforderungs-Sammlung
5. Filterung
6. Satzklassifikationen
7. Formalisierung

Objektbeschreibung

- Uni-Angestellte
  - Anzahl: 1000
  - Attribute
    - ❖PersonalNummer
      - Typ: char
      - Länge: 9
      - Wertebereich: 0...999.999.99
      - Anzahl Wiederholungen: 0
      - Definiertheit: 100%
      - Identifizierend: ja
    - ❖Gehalt
      - Typ: dezimal
      - Länge: (8,2)
      - Anzahl Wiederholung: 0
      - Definiertheit: 10%
      - Identifizierend: nein
    - ❖Rang
      - Typ: String
      - Länge: 4
      - Anzahl Wiederholung: 0
      - Definiertheit: 100%
      - Identifizierend: nein

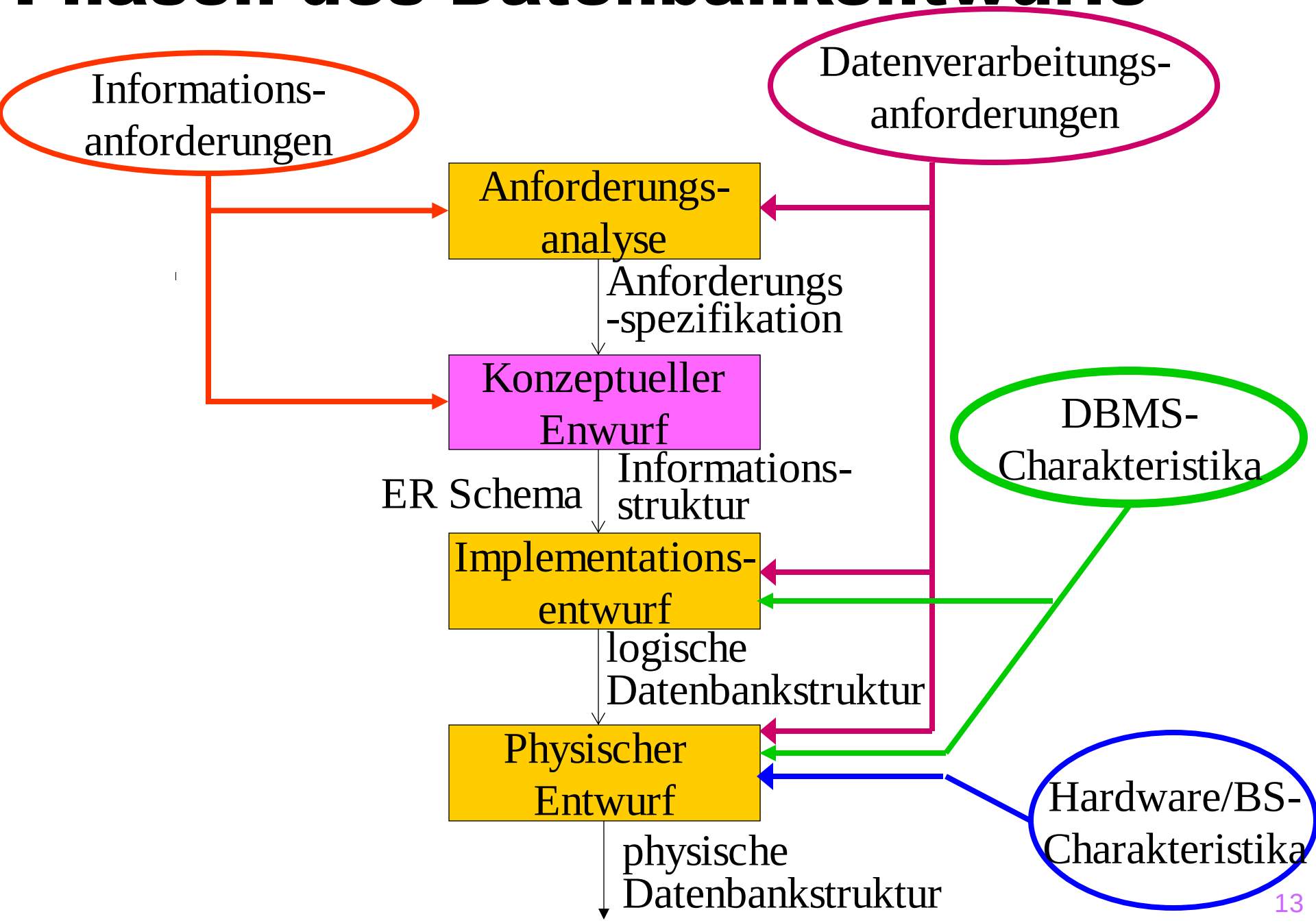
Beziehungsbeschreibung: *prüfen*

- Beteiligte Objekte:
  - Professor als Prüfer
  - Student als Prüfling
  - Vorlesung als Prüfungsstoff
- Attribute der Beziehung:
  - Datum
  - Uhrzeit
  - Note
- Anzahl: 100 000 pro Jahr

Prozeßbeschreibungen

- **Prozeßbeschreibung:** *Zeugnisausstellung*
  - Häufigkeit: halbjährlich
  - benötigte Daten
    - Prüfungen
    - Studienordnungen
    - Studenteninformation
    - ...
  - Priorität: hoch
  - Zu verarbeitende Datenmenge
    - 500 Studenten
    - 3000 Prüfungen
    - 10 Studienordnungen

Phasen des Datenbankentwurfs



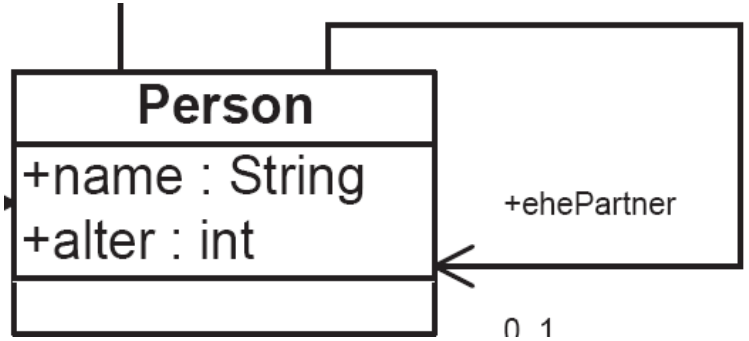
Datenmodellierung mit UML

- Unified Modelling Language UML
- De-facto Standard für den objekt-orientierten Software-Entwurf
- Zentrales Konstrukt ist die Klasse (class), mit der gleichartige Objekte hinsichtlich
  - Struktur (~Attribute)
  - Verhalten (~Operationen/Methoden)

  modelliert werden
- Assoziationen zwischen Klassen entsprechen Beziehungstypen
- Generalisierungshierarchien
- Aggregation

Klassen/Objekttypen in Java

```
class TypName {
        Typ_1 Attr_1 ;
        ...;
        Typ_n Attr_n ;
    // Operationen folgen hier
        ...
} // end class TypName;
```




```
class Person {
    public String name;
    public int alter;
    public Person ehePartner;
}
```

Werte versus Objekte

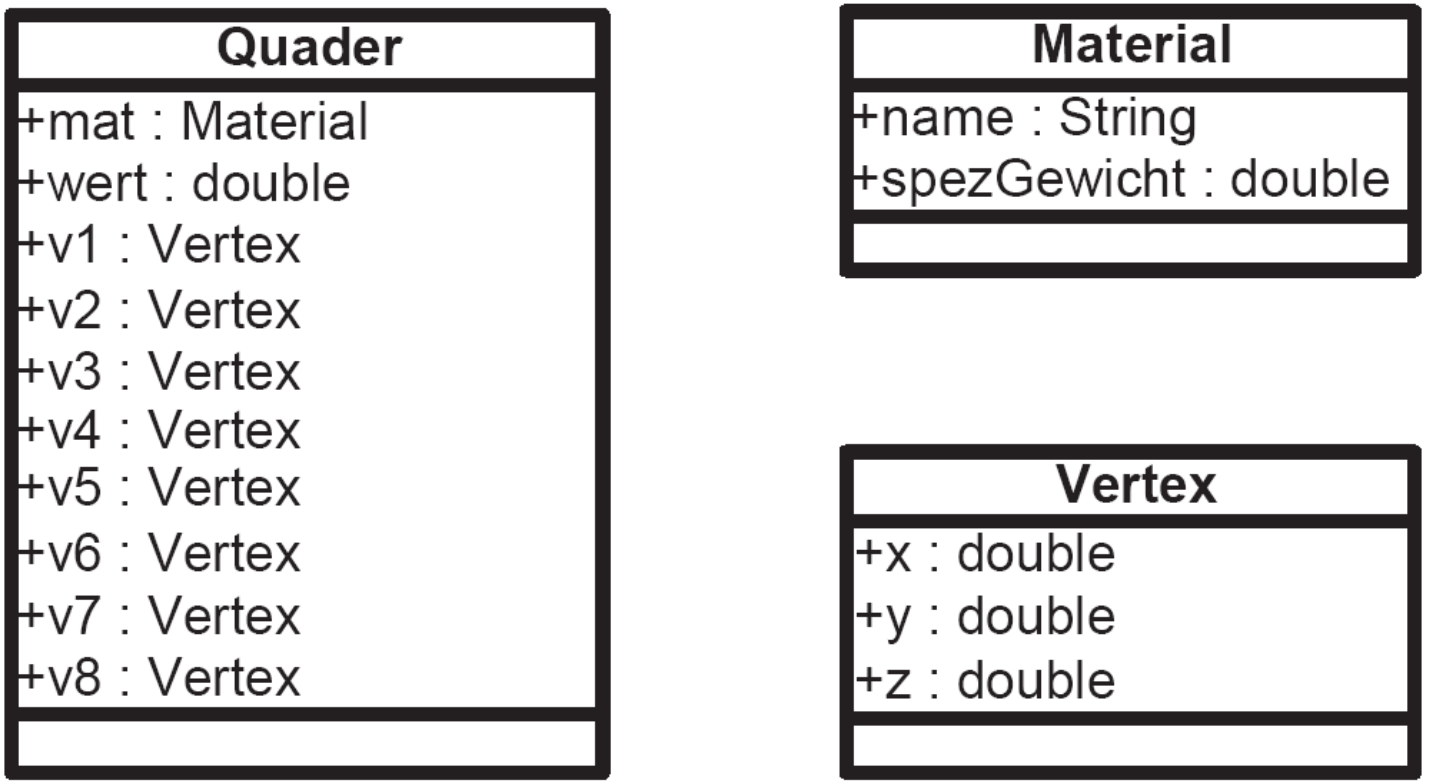
Typen		Instanzen
primitive Typen	$\longrightarrow$	Werte
Objekttypen (Klassen)	$\longrightarrow$	Objekte

Sorte	Wertebereich
boolean	{*true, false*}
byte	sehr kleine Integer-Zahlen
short	kleine Integer-Zahlen
int	Integer-Zahlen
long	große Integer-Zahlen
float	Fließkomma-Zahlen
double	Fließkomma-Zahlen doppelter Präzision
char	Character

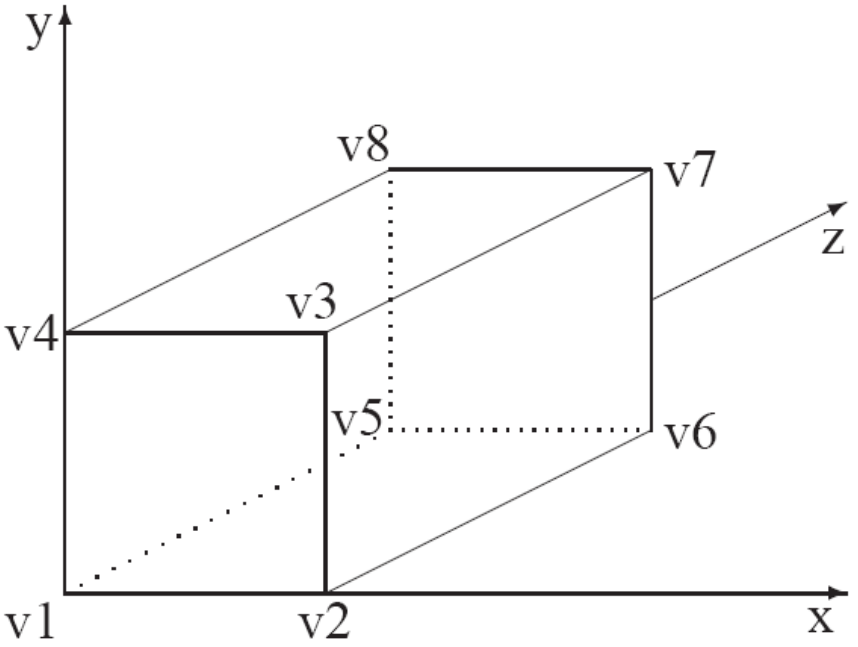
Java Klassendefinition: Syntax

```
[public] [abstract] class A
    [extends B] [implements Schnittstellen] {
        Instanz-Variable;
        ...
        Instanz-Variable;

        Konstruktor;
        ...
        Konstruktor;

        Operation/Methode;
        ...
        Operation/Methode;
}
```

Klassen/Objekttypen in UML



Klassen in Java



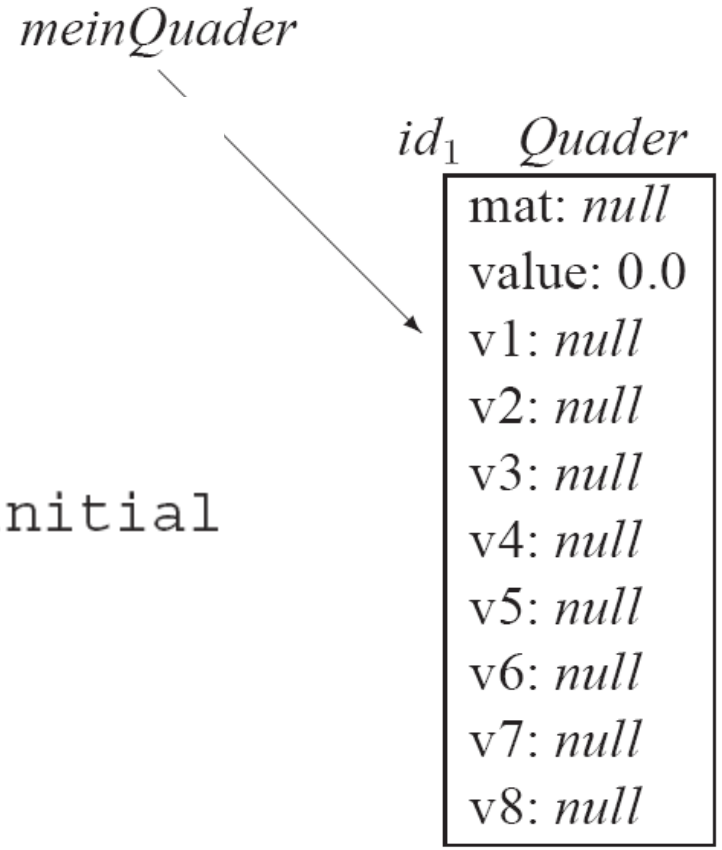

```
class Vertex {
    public double x;
    public double y;
    public double z;
}

public class Material {
    public String name;
    public double spezGewicht;
}

class Quader {
    public Vertex v1, v2, v3, v4, v5, v6, v7, v8;
    public Material mat;
    public double wert;
}
```

Instanziierung

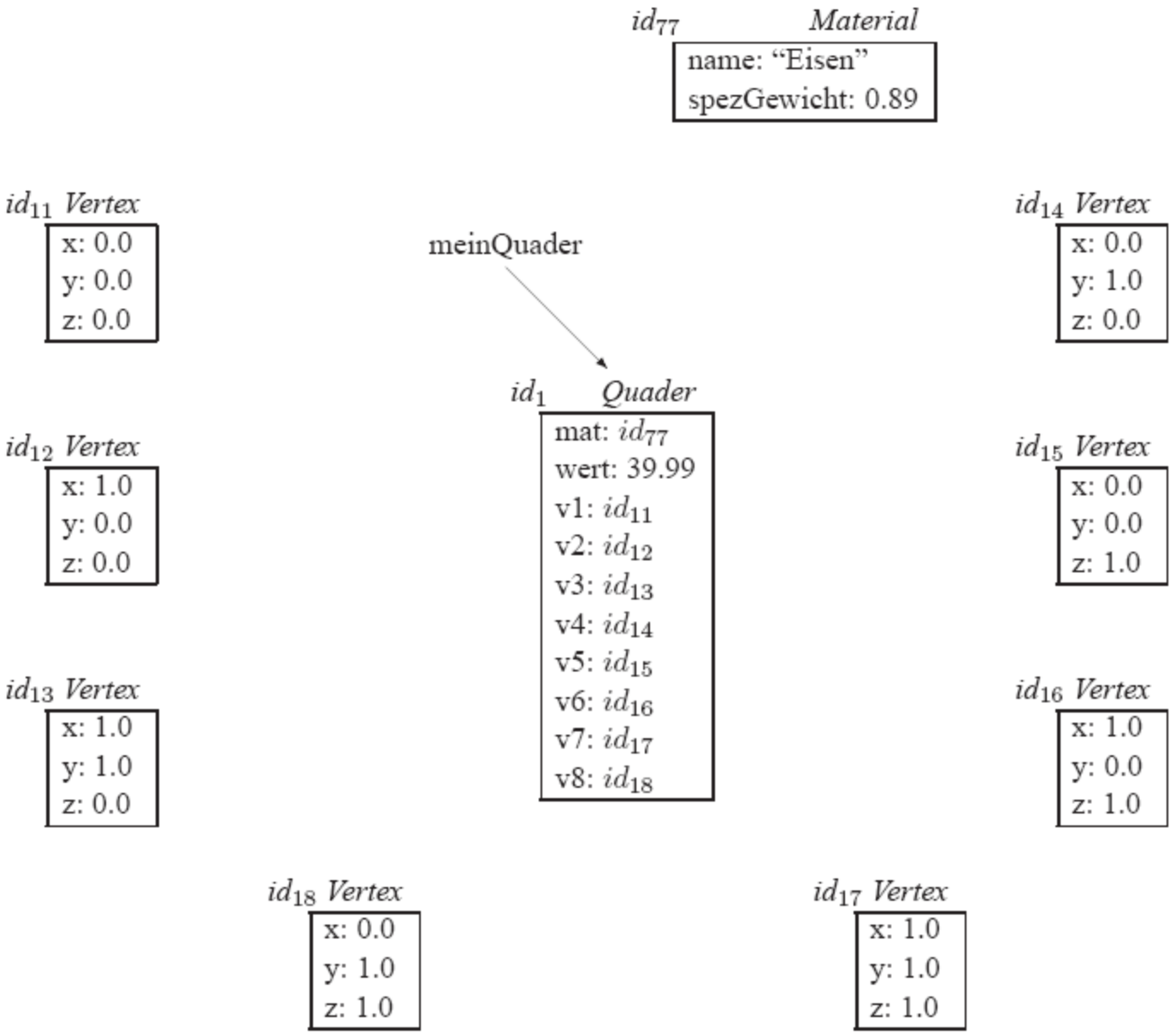
- *int, short, byte, long*-Attribute werden auf den Wert 0 initialisiert.
- *double, float*-Attribute werden initial auf den Wert 0.0 gesetzt.
- *char*-Instanzvariablen werden auf den Wert „$\backslash u0000$“ initialisiert.
- *boolean*-Attribute werden auf *false* gesetzt.
- Attribute, die auf einen Objekttyp eingeschränkt sind, werden auf *null* gesetzt. Dies ist ein spezieller Wert, der angibt, dass (noch) keine Referenz auf ein Objekt existiert.
- *String*-Attribute werden initial auch auf *null* gesetzt, da Strings in Java als Objekttyp definiert sind.

Instanziierung eines Quaders

```
Quader meinQuader;
meinQuader = new Quader();

meinQuader.v1 = new Vertex();
meinQuader.v1.x = 0.0;
meinQuader.v1.y = 0.0;
meinQuader.v1.z = 0.0;
meinQuader.v2 = new Vertex();
...     // instanziiere und initial
meinQuader.v8 = new Vertex();
meinQuader.v8.x = 0.0;
meinQuader.v8.y = 1.0;
meinQuader.v8.z = 1.0;
meinQuader.wert = 39.99;

meinQuader.mat = new Material();
meinQuader.mat.name = "Eisen";
meinQuader.mat.spezGewicht = 0.89;
```



Resultierendes Objektnetz



Objekt besteht aus (OID, Typ, Rep)

Jedes Objekt o kann man demnach als Tripel der folgenden Form auffassen:

$$o = (id_{\#}, Typ, Rep)$$

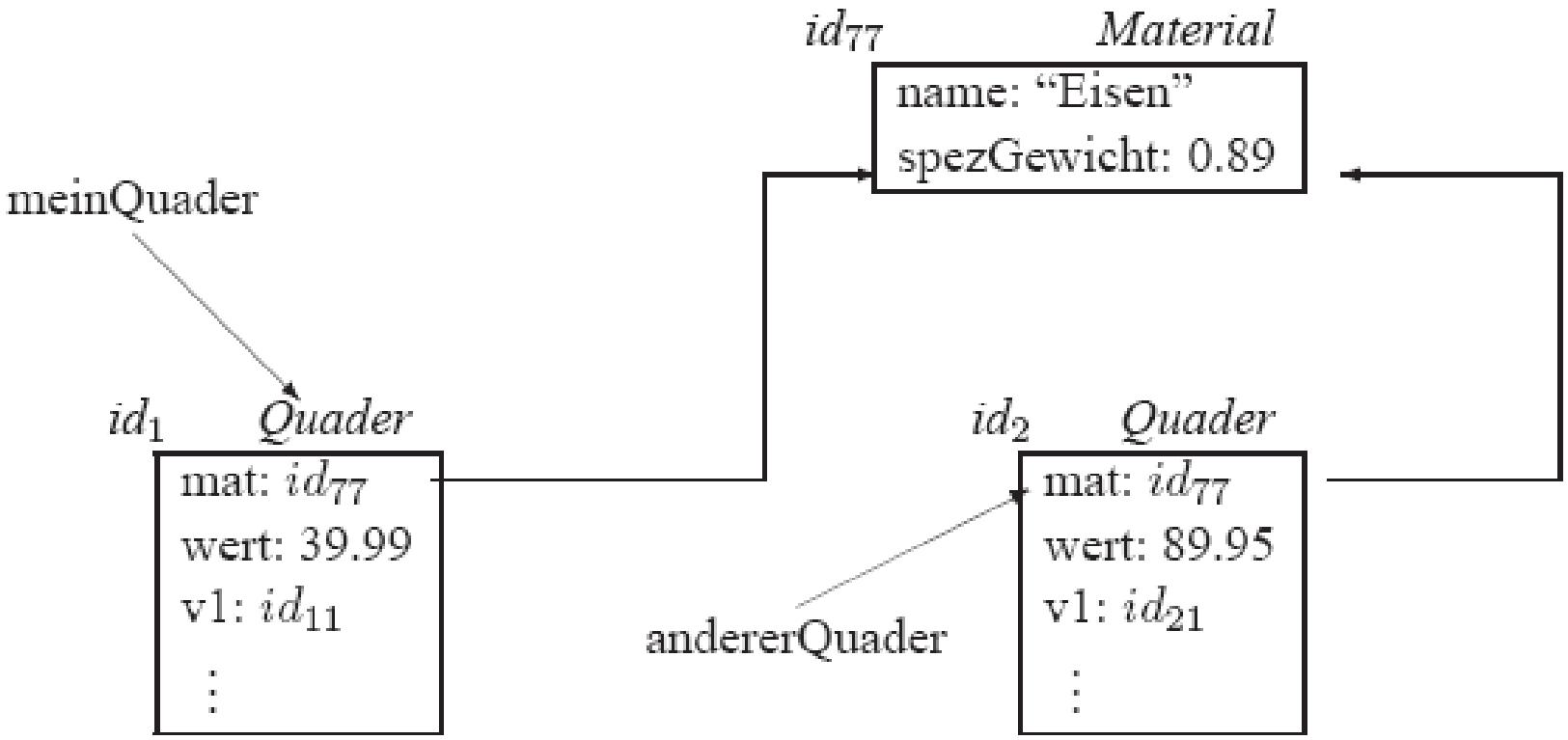
Die drei Teile haben die folgende Bedeutung:

- $id_{\#}$ stellt den Objektidentifikator des Objekts o dar.
- *Typ* spezifiziert den Objekttyp, von dem das Objekt instanziiert wurde.
- *Rep* entspricht dem internen Zustand (der derzeitigen strukturellen Repräsentation) des Objekts o.

- Als OID dient in Java die (virtuelle) Speicheradresse
  - Nennt man physische OID
- In Datenbanken verwendet man auch logische OIDs
  - Damit Objekte sich „bewegen“ können

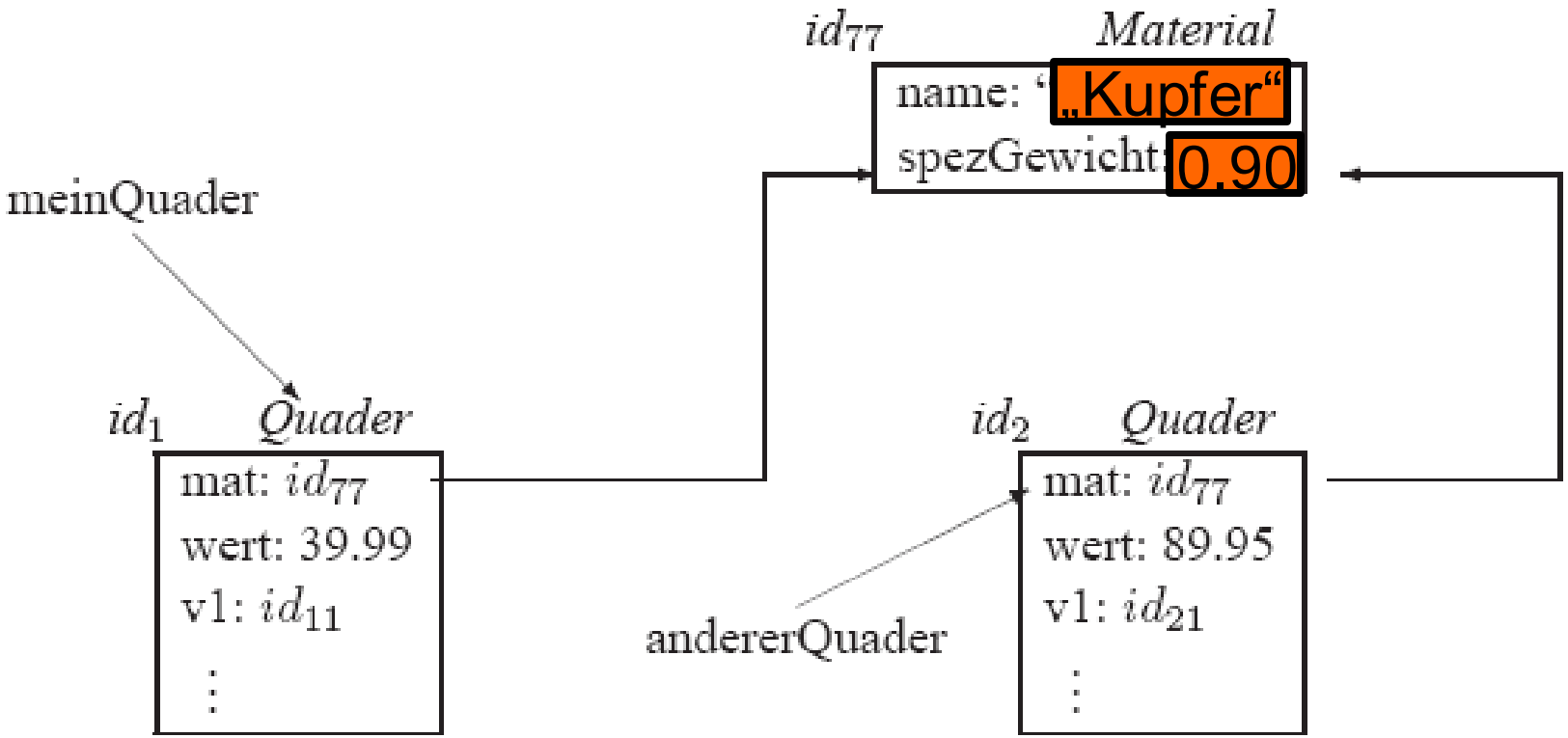
Shared Subobjects/Gemeinsame Unterobiekte

```
Quader andererQuader;
    ...
andererQuader = new Quader();
andererQuader.mat = meinQuader.mat;
```



Shared Subobjects/Gemeinsame Unterobjekte

```
meinQuader.mat.name = "Kupfer";
meinQuader.mat.spezGewicht = 0.90;
```

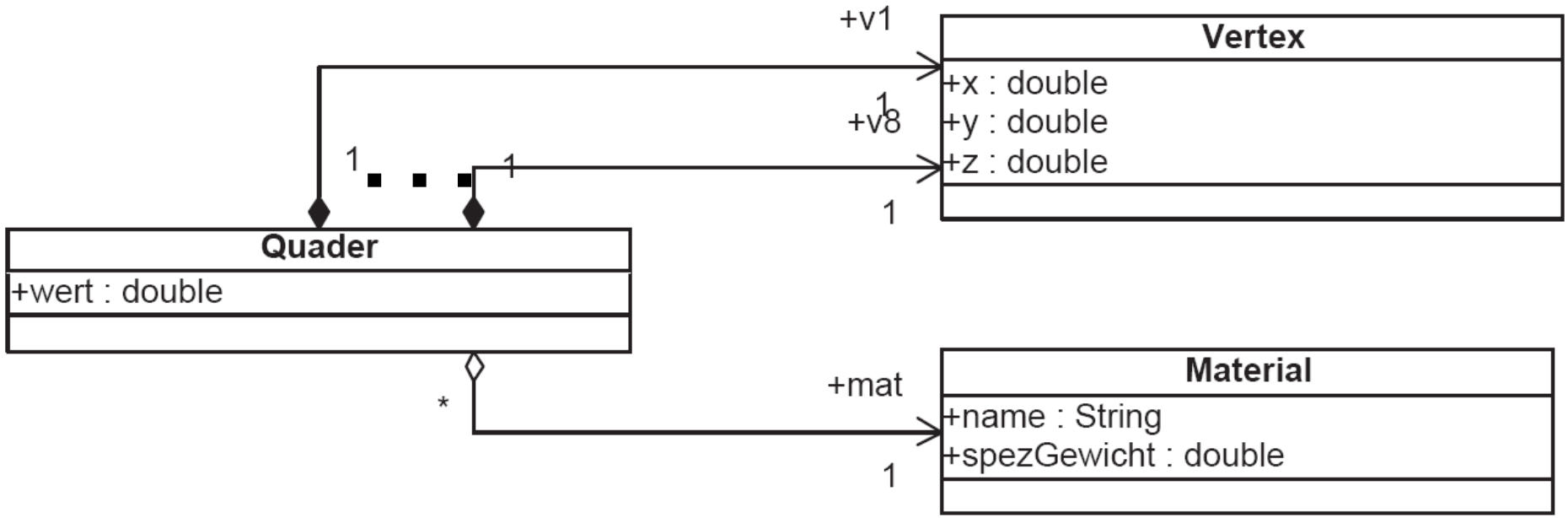


Wertvergleich versus Objektvergleich

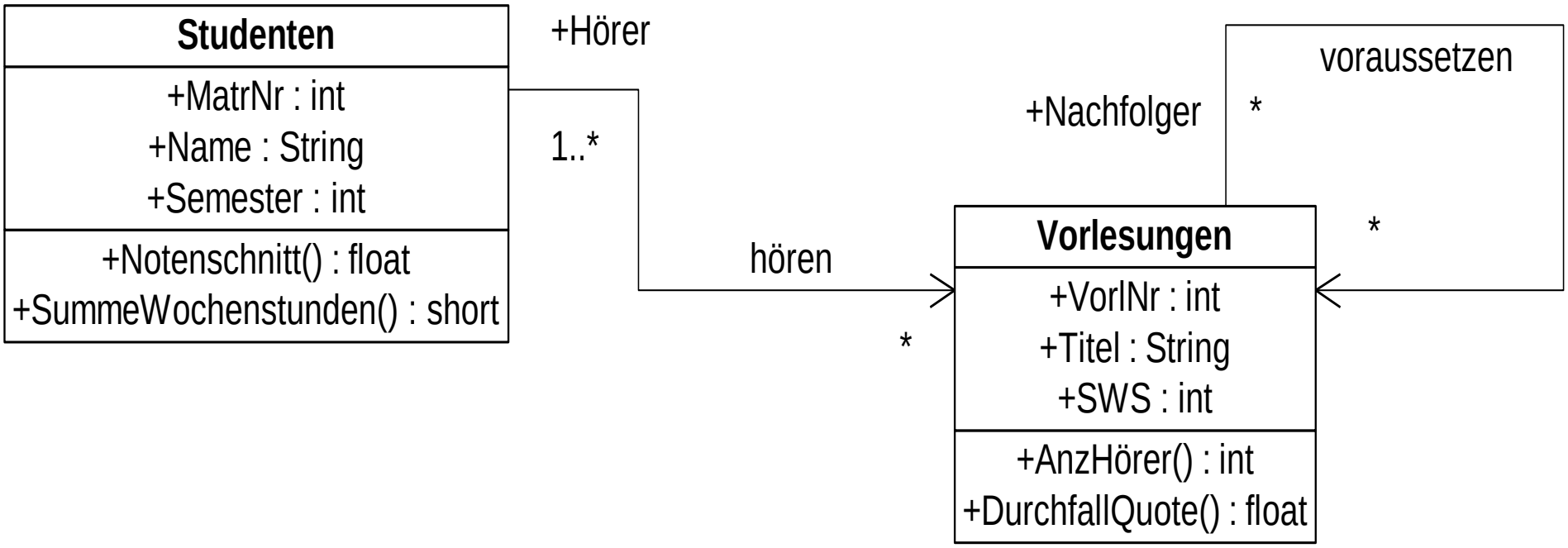
```
andererQuader.mat.name.equals("Kupfer");    // Objekt-Vergleich
andererQuader.mat.spezGewicht == 0.90;      // Wert-Vergleich
```

- Dasselbe ist nicht dasgleiche!
- Im Restaurant sollte man nie „dasselbe“ sondern „dasgleiche“ wie ein anderer bestellen

Beziehungen/Assoziationen in UML

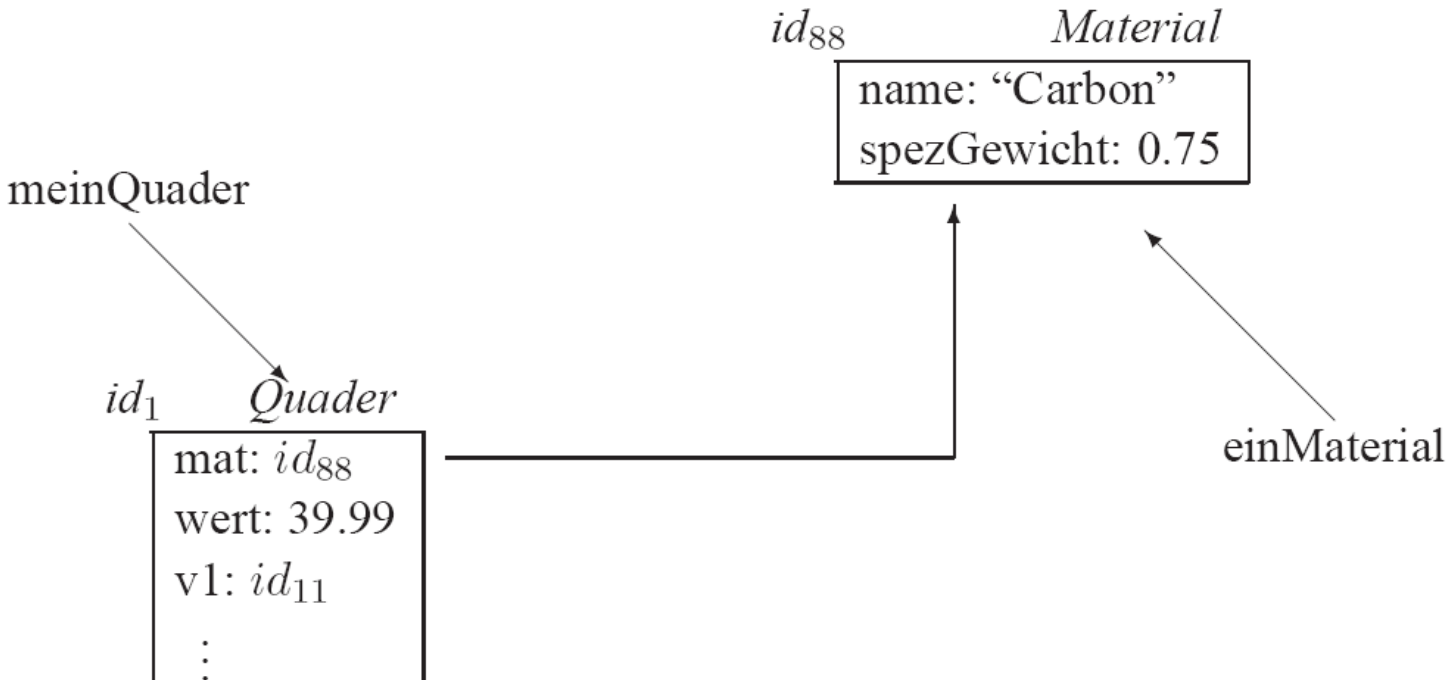


Klassen und Assoziationen



Referenzierung/Dereferenzierung

```
   Quader meinQuader = new Quader();
   Material einMaterial;
   double w;
   ...
(1)einMaterial = new Material();     // einMaterial speichert die OID id88
(2)einMaterial.name = "Carbon";
(3)einMaterial.spezGewicht = 0.75;
(4)meinQuader.mat = einMaterial;     // meinQuader hat die OID id1
(5)w = meinQuader.mat.spezGewicht;
```



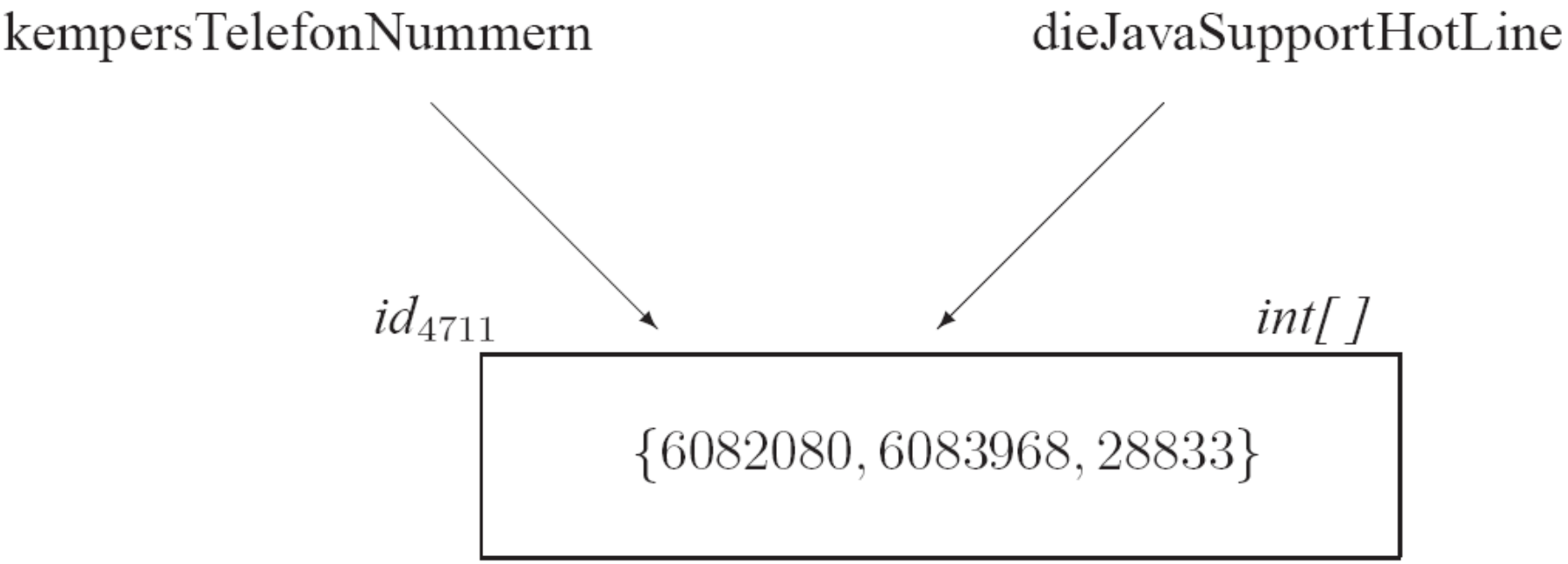
Kollektionen mit Arrays

```
ElementTyp[] arrayName = new ElementTyp[25];
ElementTyp einElem;
arrayName[0] = einElem;
```

```
int[] kempersTelefonNummern = {6082080,0,28833};
   // äquivalent zu:
   // kempersTelefonNummern = new int[3];
   // kempersTelefonNummern[0] = 6082080;
   // kempersTelefonNummern[1] = 0;
   // kempersTelefonNummern[2] = 28833;

int privNummer = kempersTelefonNummern[0];  // pr
                                  // privNummer == 6082080
```

Kollektion als shared subobject

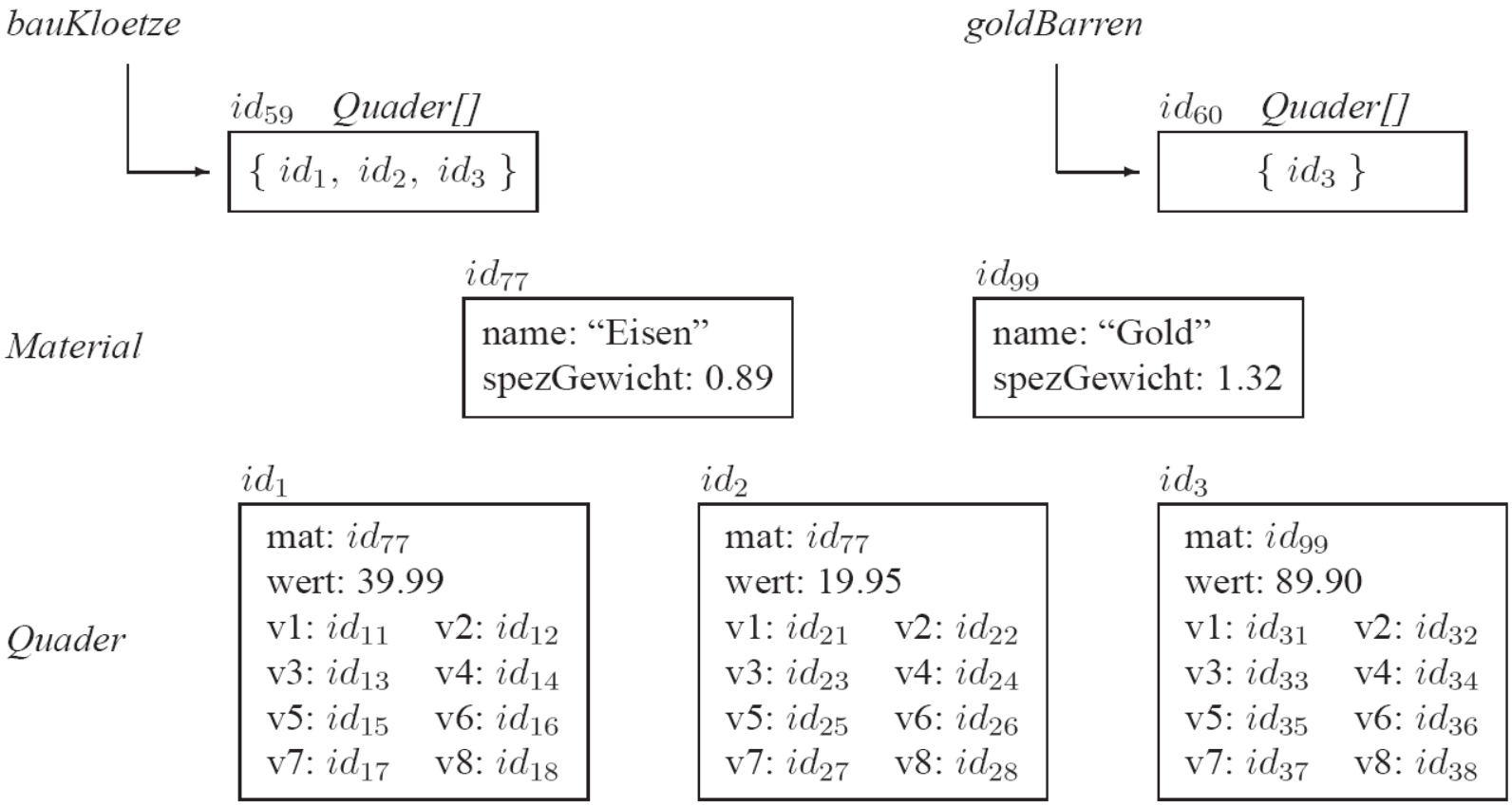
```
int[] dieJavaSupportHotLine;
dieJavaSupportHotLine = kempersTelefonNummern;
```




```
dieJavaSupportHotLine[1] = 6083968;
```

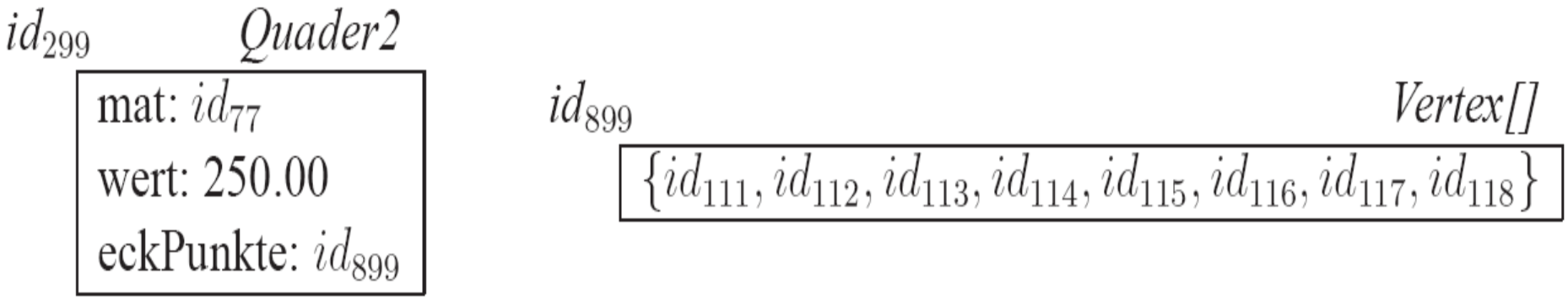
Kollektionen sind „first class citizens“

```
Quader[] bauKloetze = new Quader[3];
Quader[] goldBarren = new Quader[1];

Quader meinQuader = new Quader();
bauKloetze[0] = meinQuader;
goldBarren[0] = ...;
```



Kollektion natürlich auch als Typ einer Instanzvariablen möglich ...

```
class Quader2 {
    public Vertex[] eckPunkte;
    public Material mat;
    public double wert;
}

einQuader.eckPunkte = new Vertex[8];
```



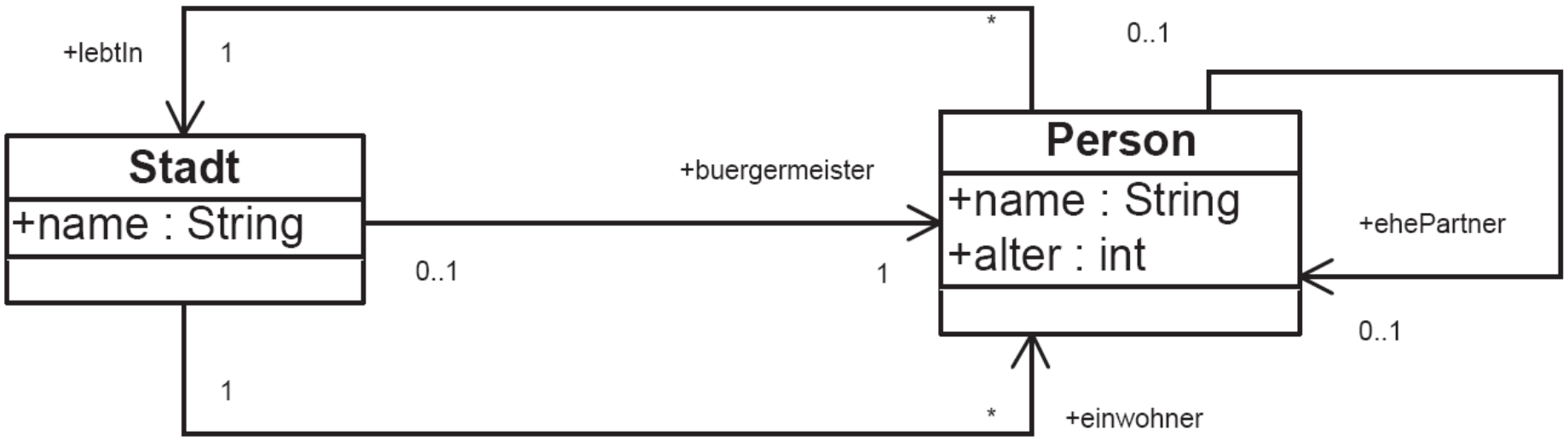
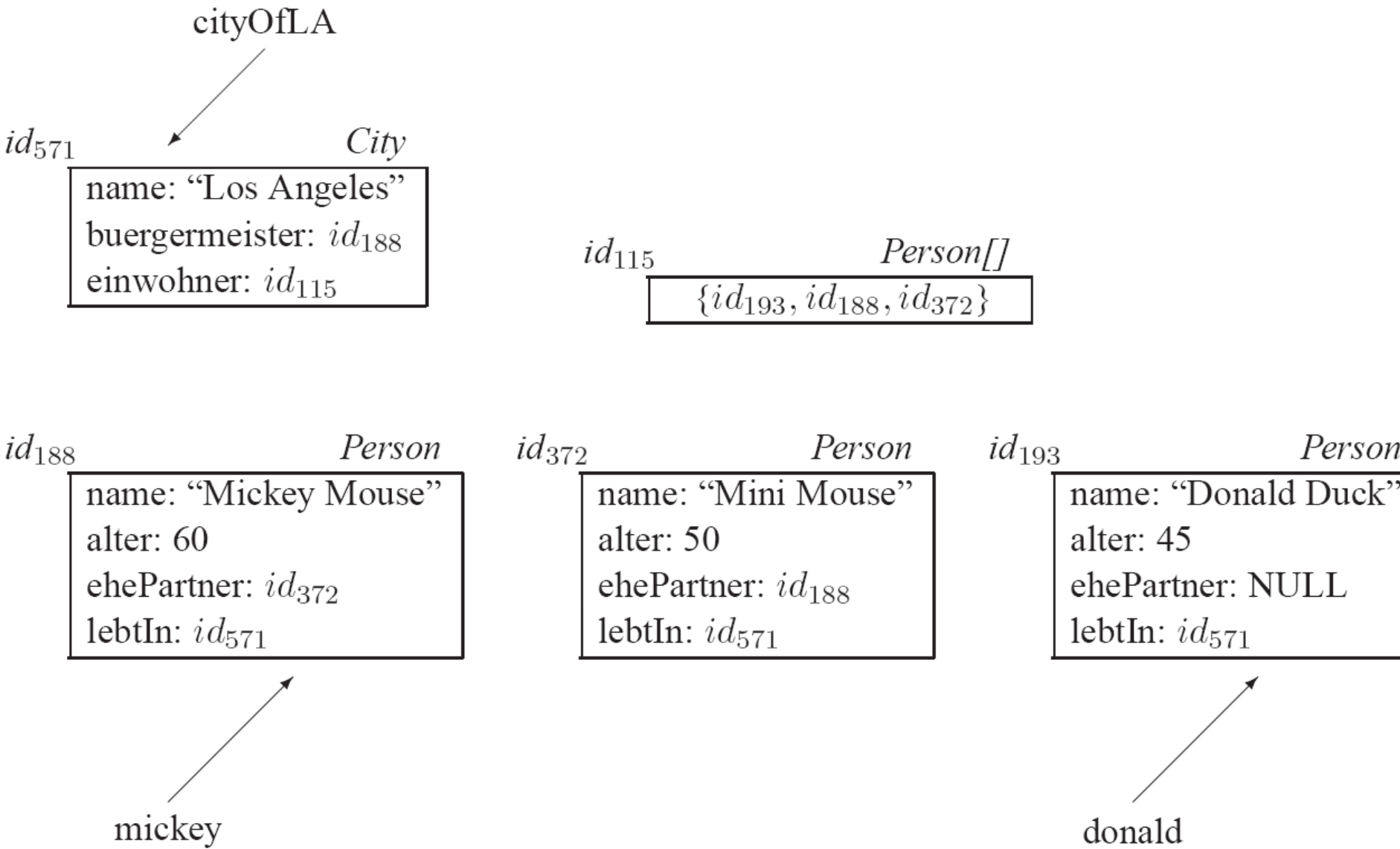


Abbildung 1.12: Die Assoziationen zwischen Person und Stadt

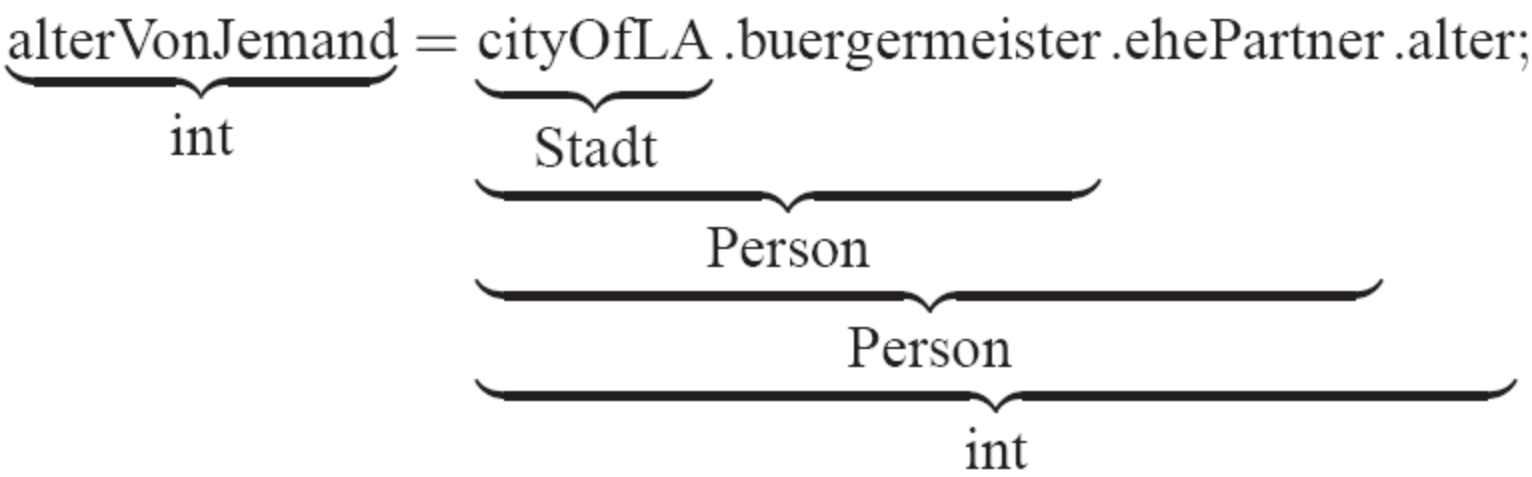
```
class Person {
    public String name;
    public int alter;
    public Person ehePartner;
    public Stadt lebtIn;
}

class Stadt {
    public String name;
    public Person buergermeister;
    public Person[] einwohner;
}
```

Objekt-Netz


cityOfLA
id_{571}
City
name: "Los Angeles"
buergermeister: id_{188}
einwohner: id_{115}
id_{115}
Person[]
$\{id_{193}, id_{188}, id_{372}\}$
id_{188}
Person
name: "Mickey Mouse"
alter: 60
ehePartner: id_{372}
lebtIn: id_{571}
id_{372}
Person
name: "Mini Mouse"
alter: 50
ehePartner: id_{188}
lebtIn: id_{571}
id_{193}
Person
name: "Donald Duck"
alter: 45
ehePartner: NULL
lebtIn: id_{571}
mickey
donald

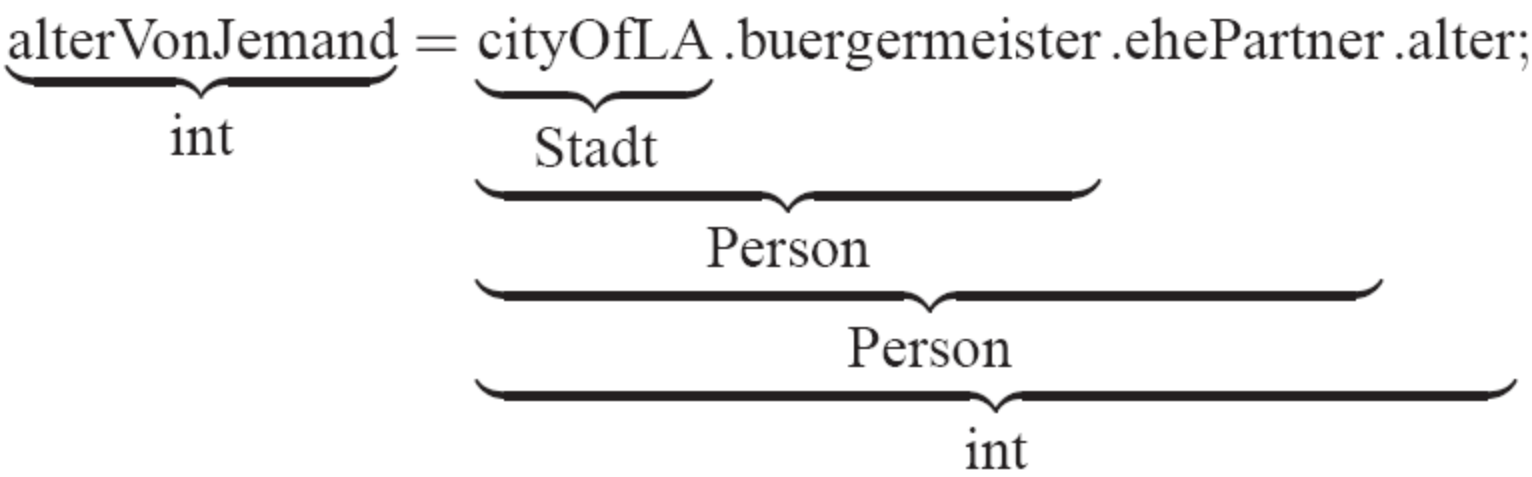
Typisierung von (Pfad-)Ausdrücken

```
   int gesamtAlter, alterVonJemand;
   Person jemand;
   String name;
   ...
(1)alterVonJemand = cityOfLA.buergermeister.ehePartner.alter;
(2)for (int i = 0; i < cityOfLA.einwohner.length; i++) {
      jemand = cityOfLA.einwohner[i];
      gesamtAlter = gesamtAlter + jemand.alter;
   }
```

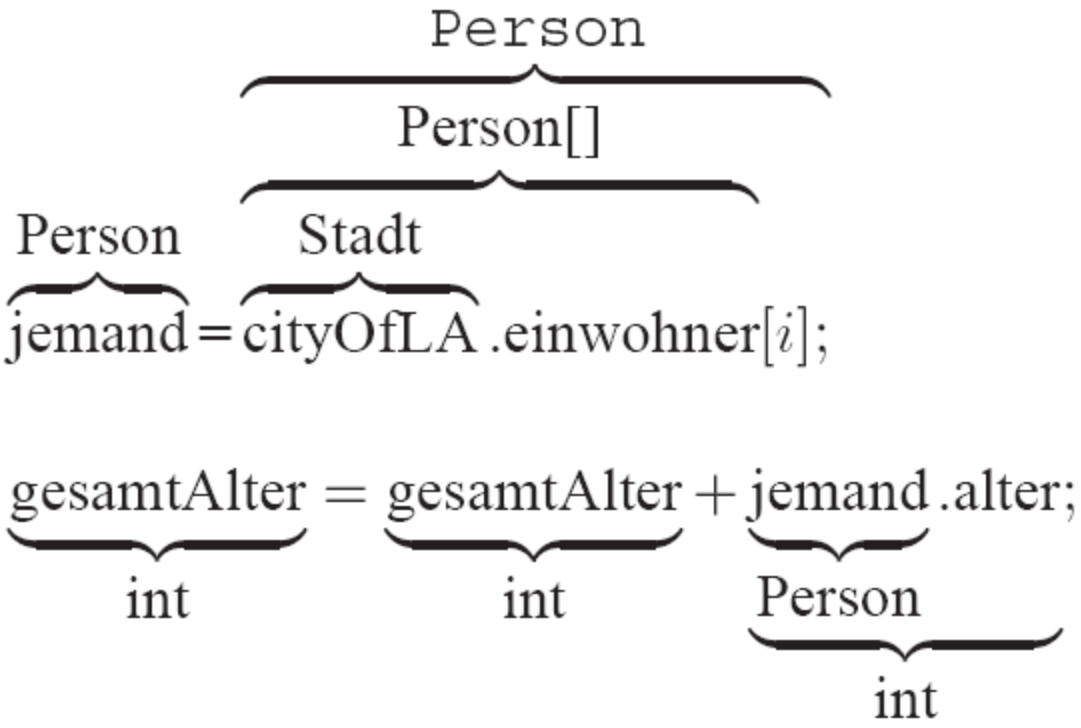


Typisierung von (Pfad-)Ausdrücken

```
   int gesamtAlter, alterVonJemand;
   Person jemand;
   String name;
   ...
(1)alterVonJemand = cityOfLA.buergermeister.ehePartner.alter;
(2)
   for (Person jemand : cityOfLA.einwohner) {
      gesamtAlter = gesamtAlter + jemand.alter;
   }
```

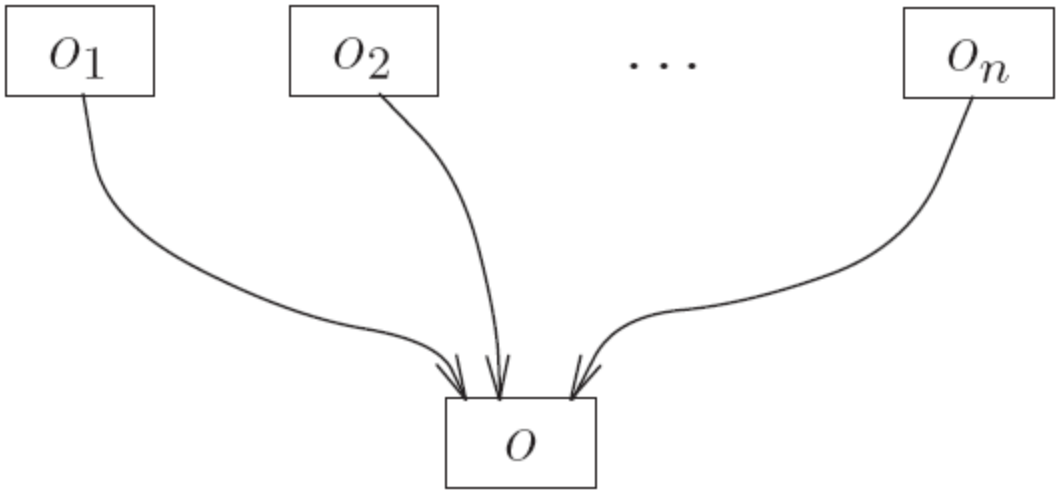


Typisierung ... cont'd



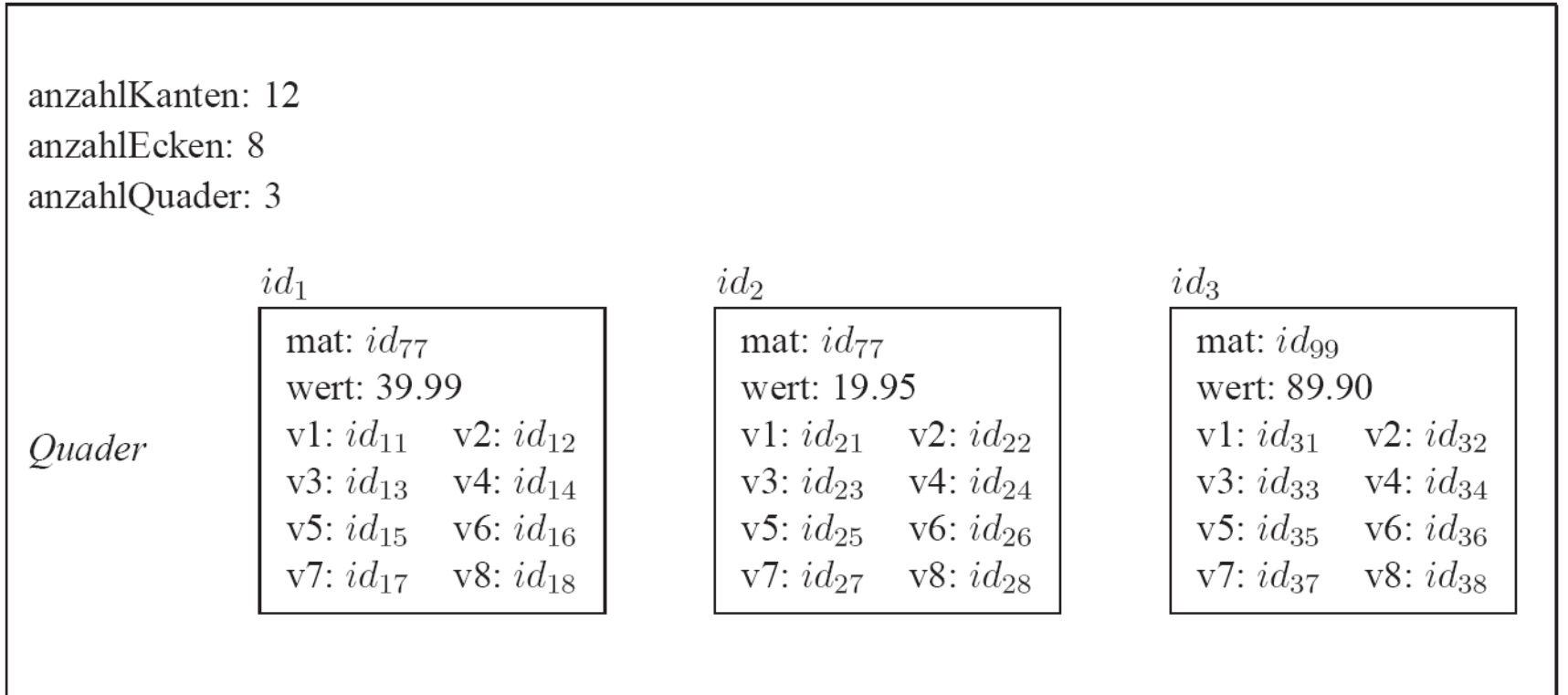
Speicherbereinigung / Garbage Collection

- Automatisch in Java
- Nur unerreichbare Objekte dürfen gelöscht werden
- Erst wenn die letzte Referenz auf ein Objekt entfernt wurde, darf der garbage collector „zuschlagen“



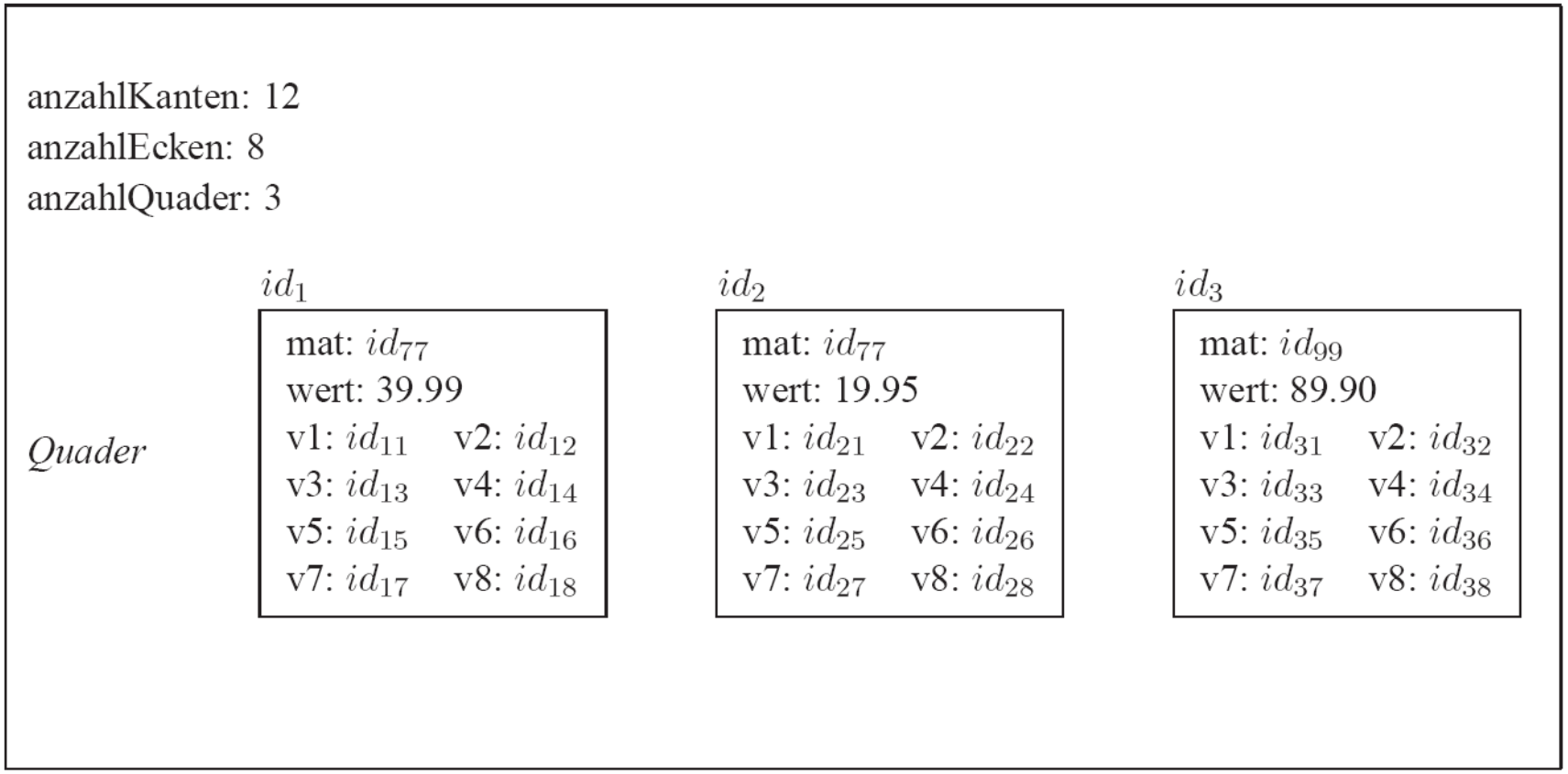
Klassen-Attribute

```
class Quader {
    public static final int anzahlKanten = 12;
    public static final int anzahlEcken = 8;
    public static int anzahlQuader = 0;
    public Vertex v1, v2, v3, v4, v5, v6, v7, v8;
    public Material mat;
    public double wert;
}
```

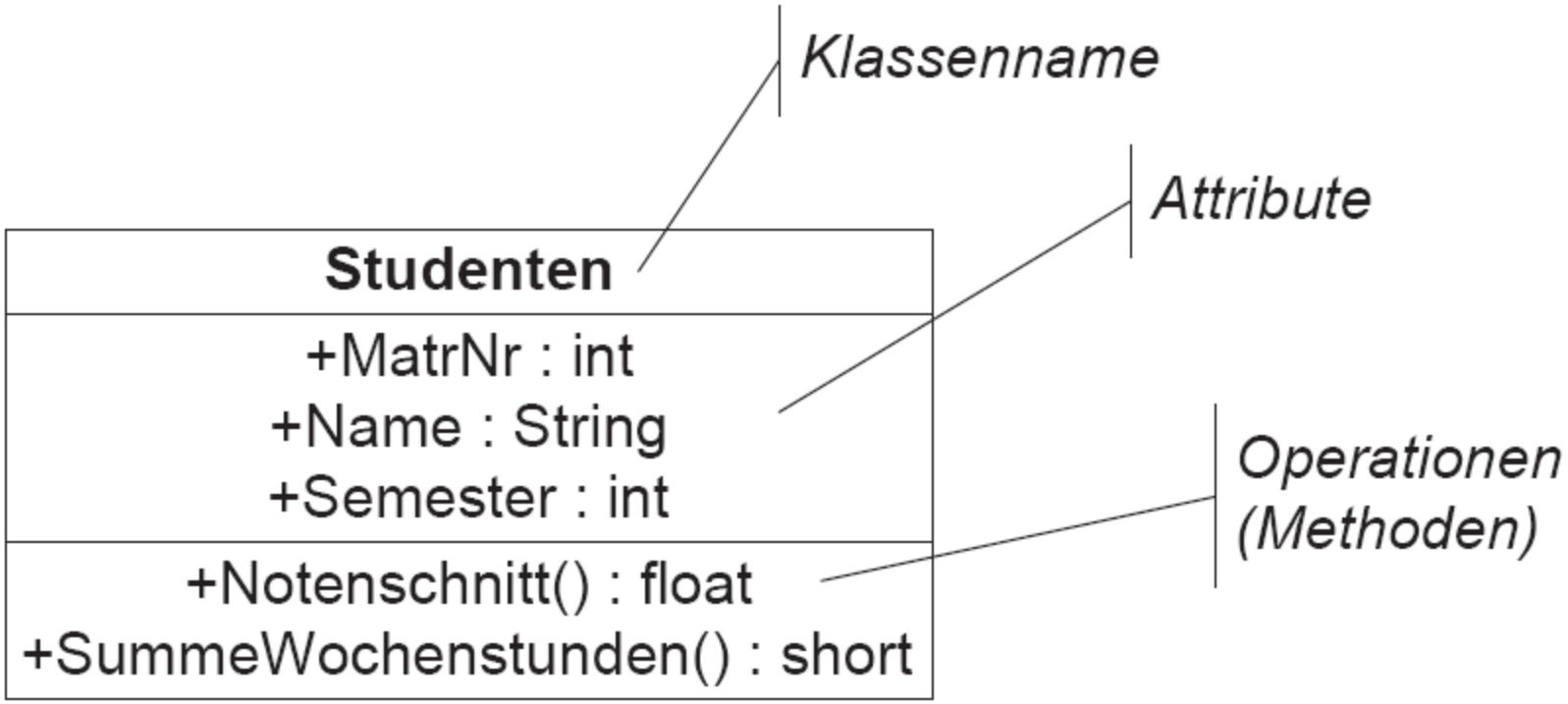


Klassen-Attribute: Zugriff und Modifikation

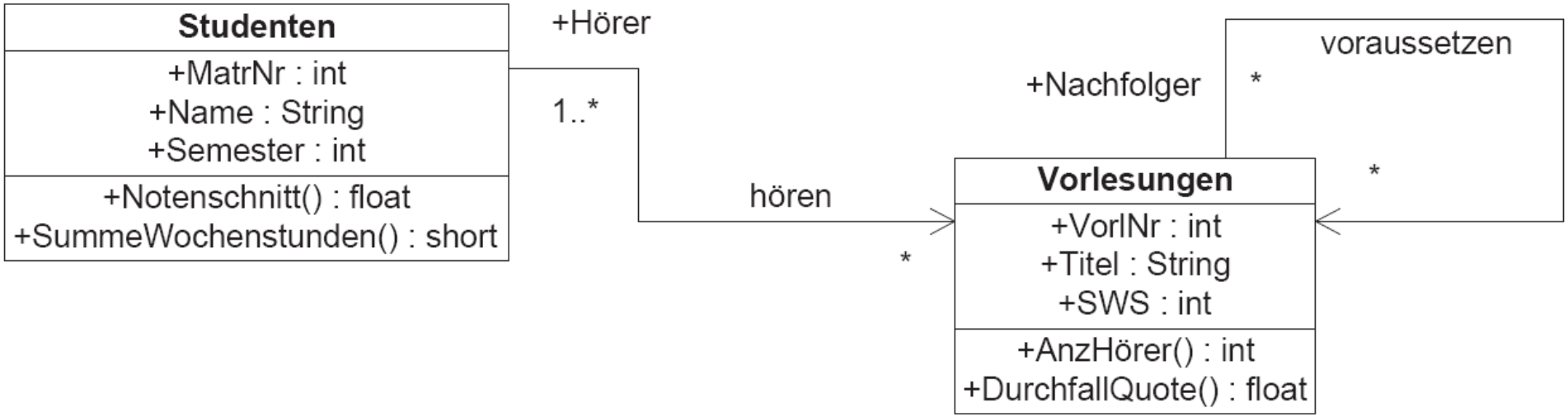
```
System.out.println(Quader.anzahlKanten);
Quader.anzahlQuader = Quader.anzahlQuader + 1;
```



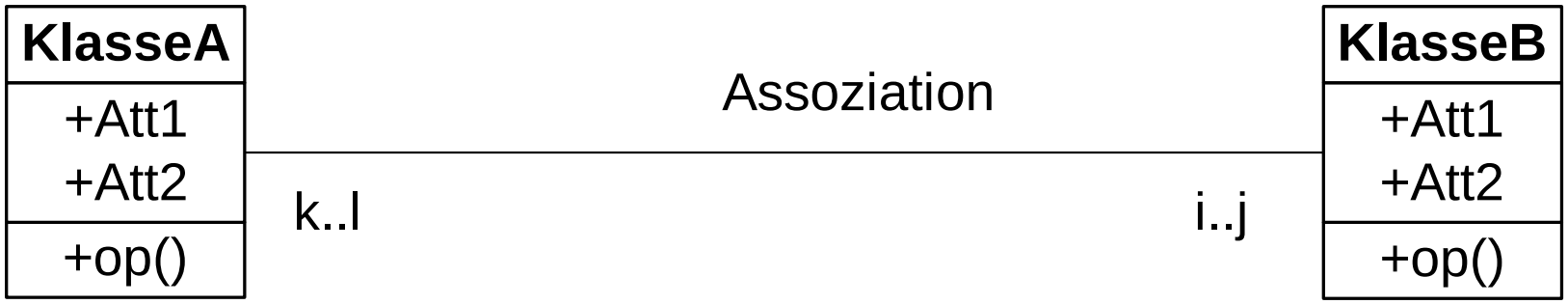
Systematische Modellierung mit UML und Umsetzung in Java



Assoziationen



Multiplizität



- Jedes Element von KlasseA steht mit mindestens i Elementen der KlasseB in Beziehung
- ... und mit maximal j vielen KlasseB-Elementen

- Analoges gilt für das Intervall k..l

- Multiplizitätsangabe ist analog zur Funktionalitätsangabe im ER-Modell
  - **Nicht** zur (min,max)-Angabe: **Vorsicht!**

Multiplizität/Funktionalität einer Assoziation

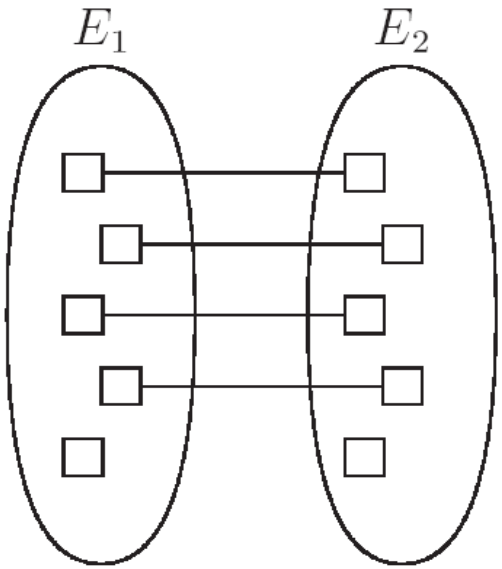
Betrachten wir das abstrakte Beispiel in Abbildung 1.19: Wenn man in UML an einer Seite der binären Assoziation die Multiplizitätsangabe $i..j$ macht, so bedeutet dies, dass jedes Objekt der Klasse auf der anderen Seite mit mindestens i und höchstens j Objekten der Klasse auf dieser Seite in Beziehung stehen muss. Bezogen auf unser Beispiel bedeutet dies, dass jedes Objekt der Klasse E_1 mit mindestens i und mit maximal j Objekten der Klasse E_2 in Beziehung stehen muss/darf. Analog muss jedes Objekt der Klasse E_2 mit mindestens k Objekten der Klasse E_1 in Beziehung stehen und es darf maximal mit l Objekten der Klasse E_1 in dieser Beziehung stehen. Wenn die minimale und die maximale Anzahl übereinstimmt, also $m..m$ gilt, so vereinfacht man dies in UML zu einem einzigen Wert m. Dies gilt auch für $*..*$ was immer als $*$ angegeben wird.

Funktionalitäten

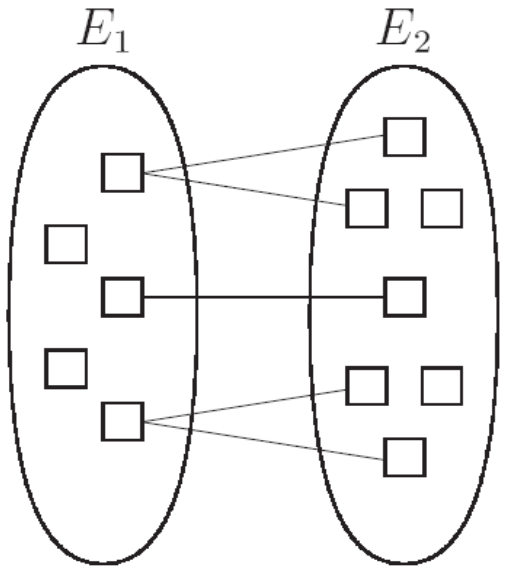
Ehemann : Frauen → Männer

Ehefrau : Männer → Frauen

1:1

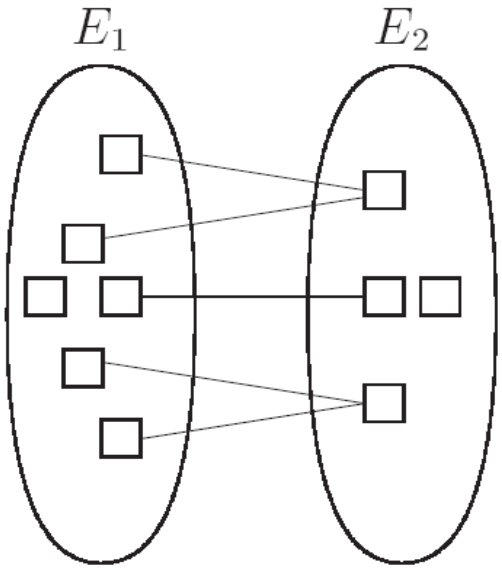


1:N

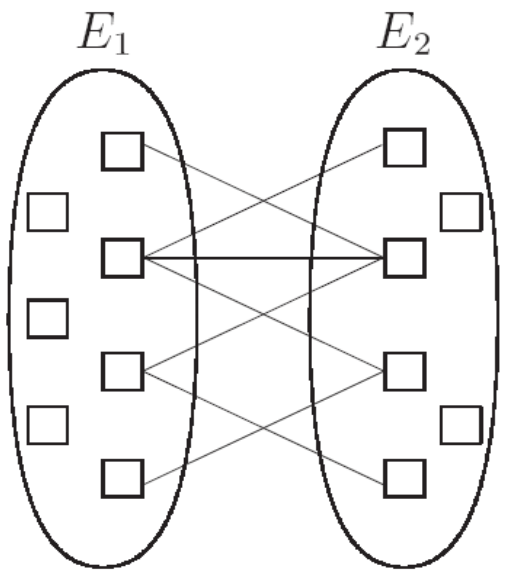


angestelltBei : Personen → Firmen

N:1



N:M



- *Eins-zu-eins* bzw. *1:1-Beziehung*, falls jedem Objekt e_1 aus E_1 höchstens ein Objekt e_2 aus E_2 zugeordnet ist und umgekehrt jedem Objekt e_2 aus E_2 ebenfalls maximal ein Objekt e_1 aus E_1. Man beachte, dass es auch Objekte aus E_1 (bzw. E_2) geben kann, denen kein „Partner“ aus E_2 (bzw. E_1) zugeordnet ist.

  In der UML-Notation aus Abbildung 1.19 ist eine Eins-zu-eins-Assoziation dadurch gekennzeichnet, dass $l = j = 1$ gilt.

  Ein Beispiel einer „realen“ 1:1-Beziehung ist *verheiratet* zwischen den Objekttypen *Männer* und *Frauen* – zumindest nach europäischem Recht.

- *Eins-zu-viele* bzw. *1:N-Beziehung*, falls jedem Objekt e_1 aus E_1 beliebig viele (also mehrere oder auch gar keine) Objekte aus E_2 zugeordnet sein können, aber jedes Objekt e_2 aus der Menge E_2 mit maximal einem Objekt aus E_1 in Beziehung steht.

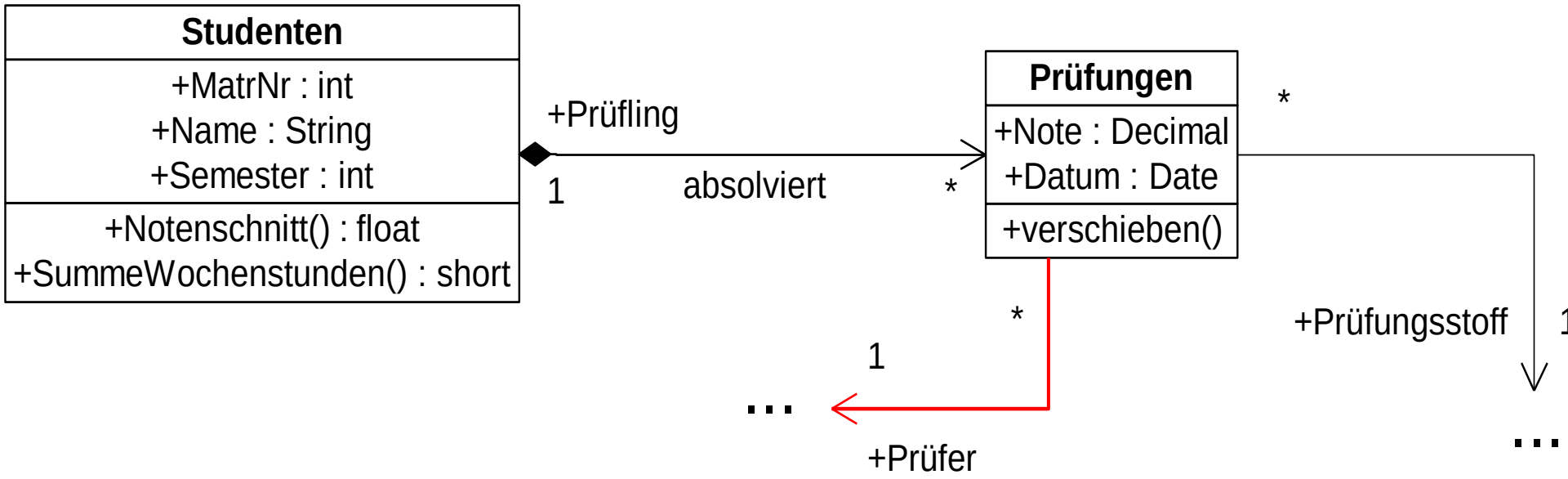
  In der UML-Notation aus Abbildung 1.19 ist eine Eins-zu-viele-Assoziation dadurch gekennzeichnet, dass $l = 1$ und $j > 1$ gilt. Insbesondere kann $j = *$ gelten.

  Ein anschauliches Beispiel für eine $1{:}N$-Beziehung ist *angestelltBei* zwischen *Personen* und *Firmen*, wenn wir davon ausgehen, dass eine Firma i.a. mehrere Personen beschäftigt, aber eine Person nur bei einer (oder gar keiner) Firma angestellt ist.

- *N:1-Beziehung*, falls analoges zu obigem gilt.

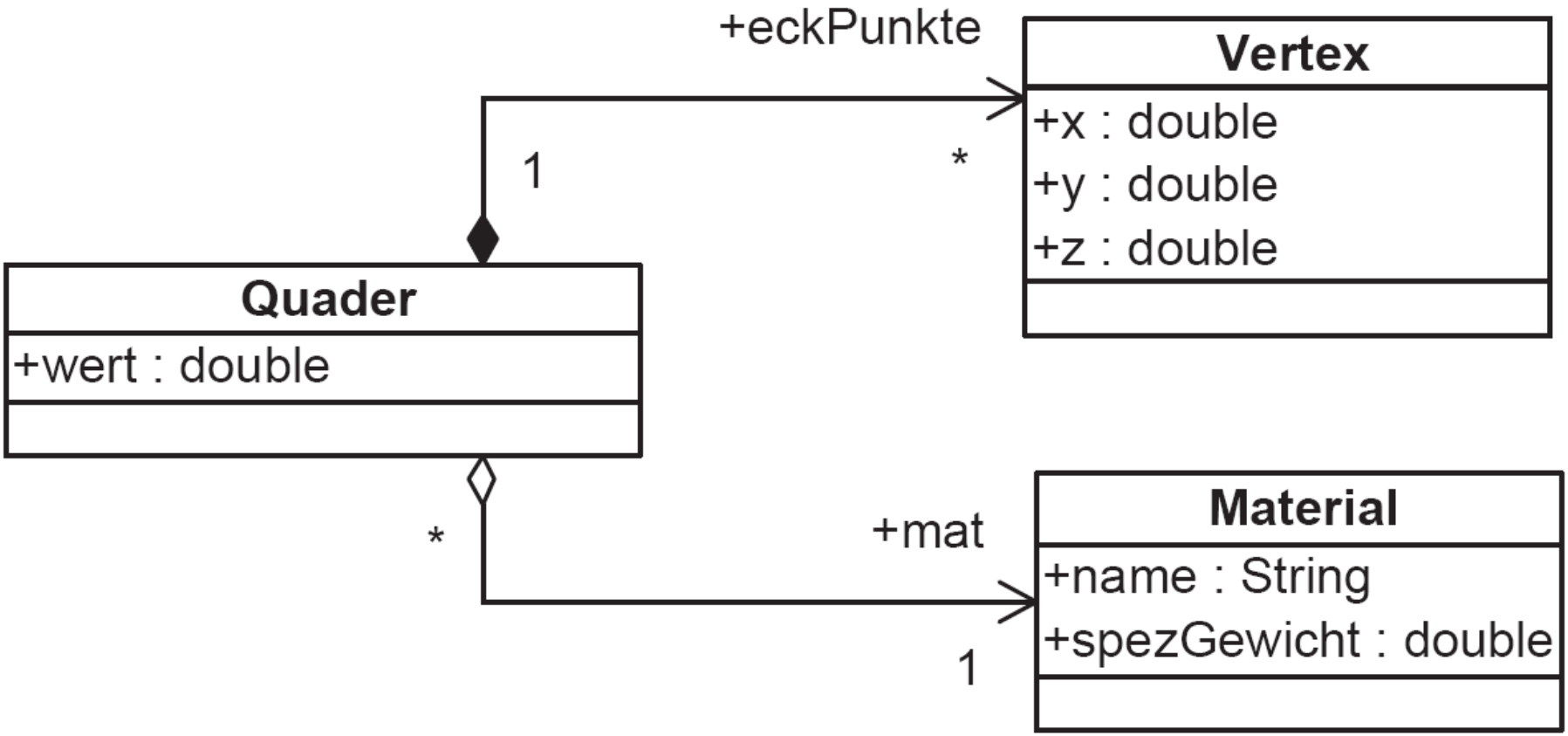
- $N{:}M$-*Beziehung*, wenn keinerlei Restriktionen gelten müssen, d.h. jedes Objekt aus E_1 mit mehreren Objekten aus E_2 in Beziehung stehen kann und umgekehrt jedes Objekt aus E_2 mit mehreren Objekten aus E_1 assoziiert werden darf.

Aggregation/Komposition

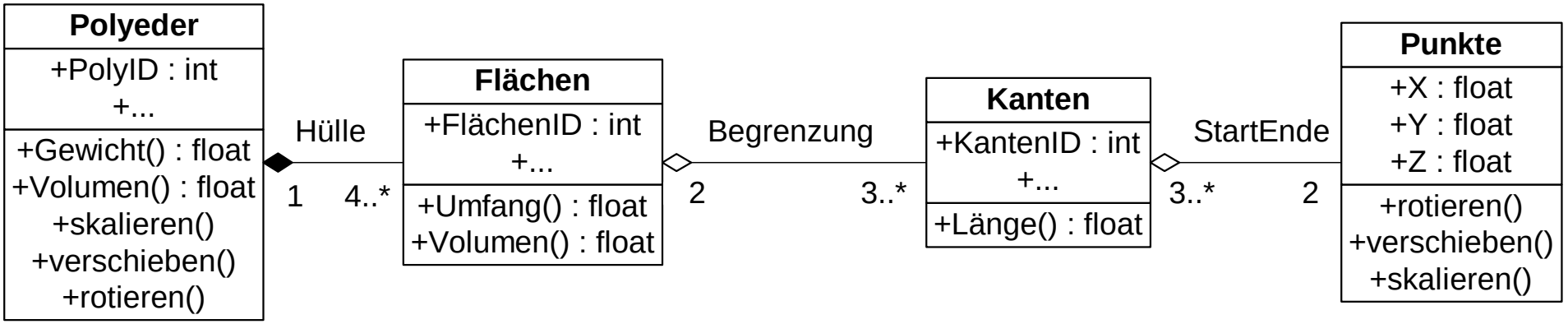


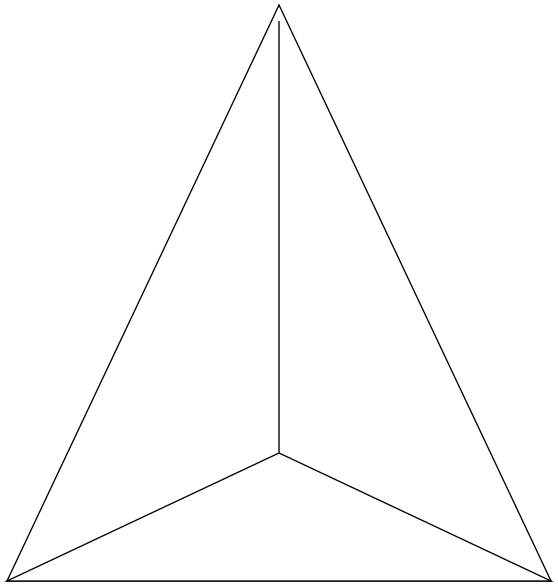
- Komposition (ausgefüllter Diamant)
  - Exklusive Zuordnung
  - Existenzabhängig
- Aggregation („leerer" Diamant)
  - Nicht-exklusive
  - Nicht-existenzabhängige Teil/Ganzes-Beziehung

Aggregation/Komposition

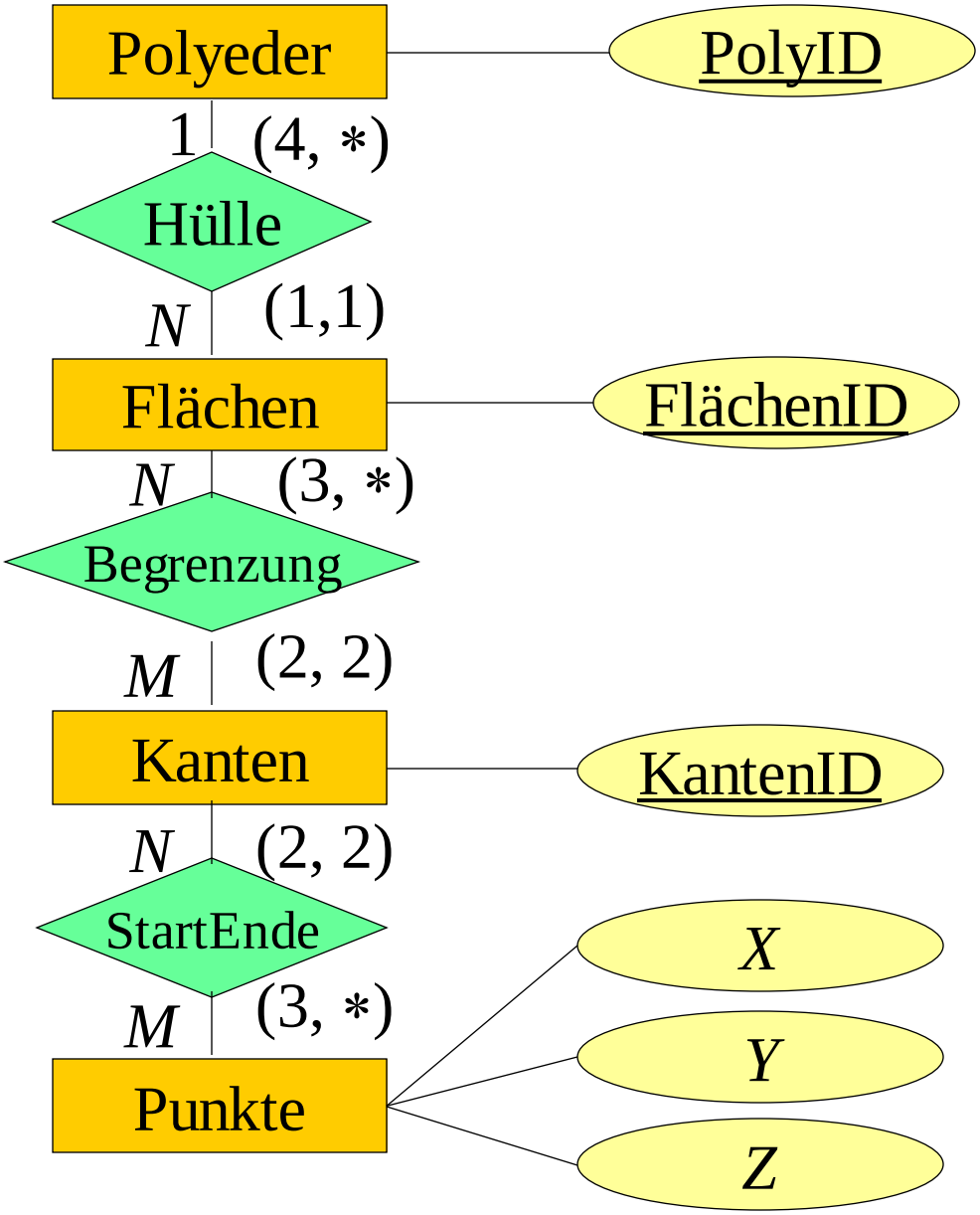


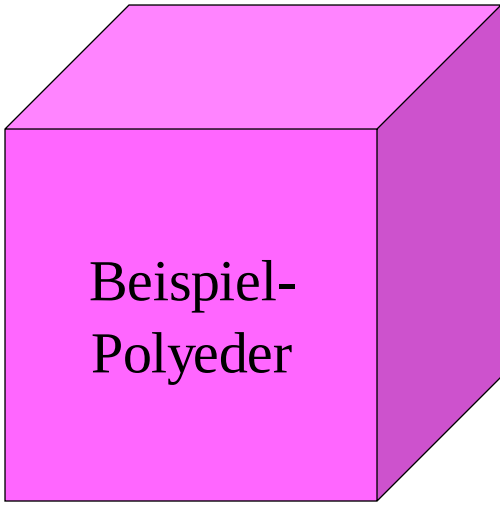
Begrenzungsflächenmodellierung von Polyedern in UML

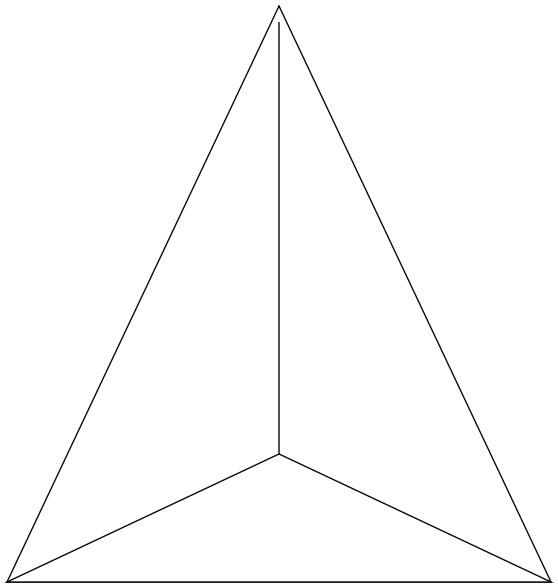




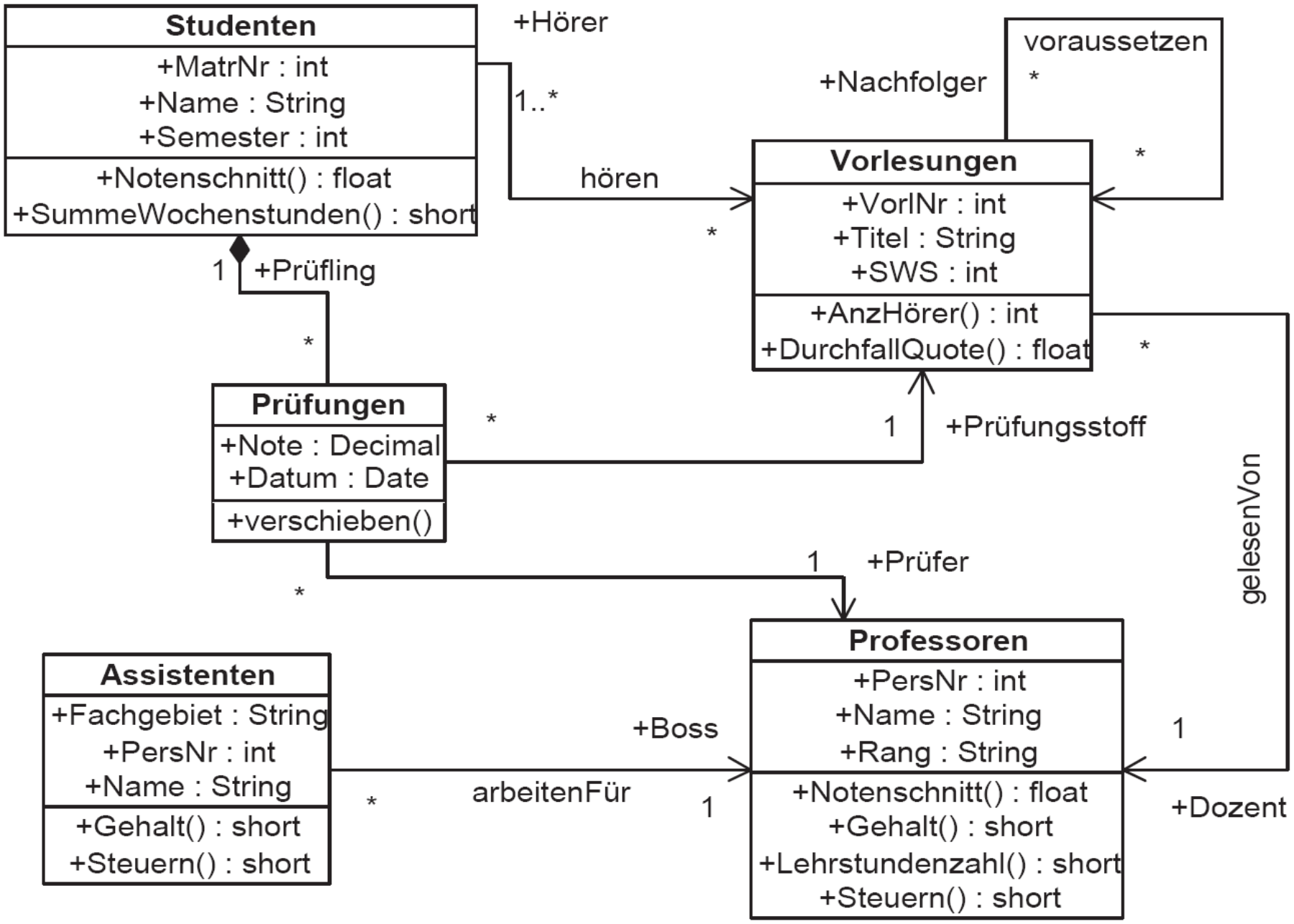
Begrenzungsflächendarstellung

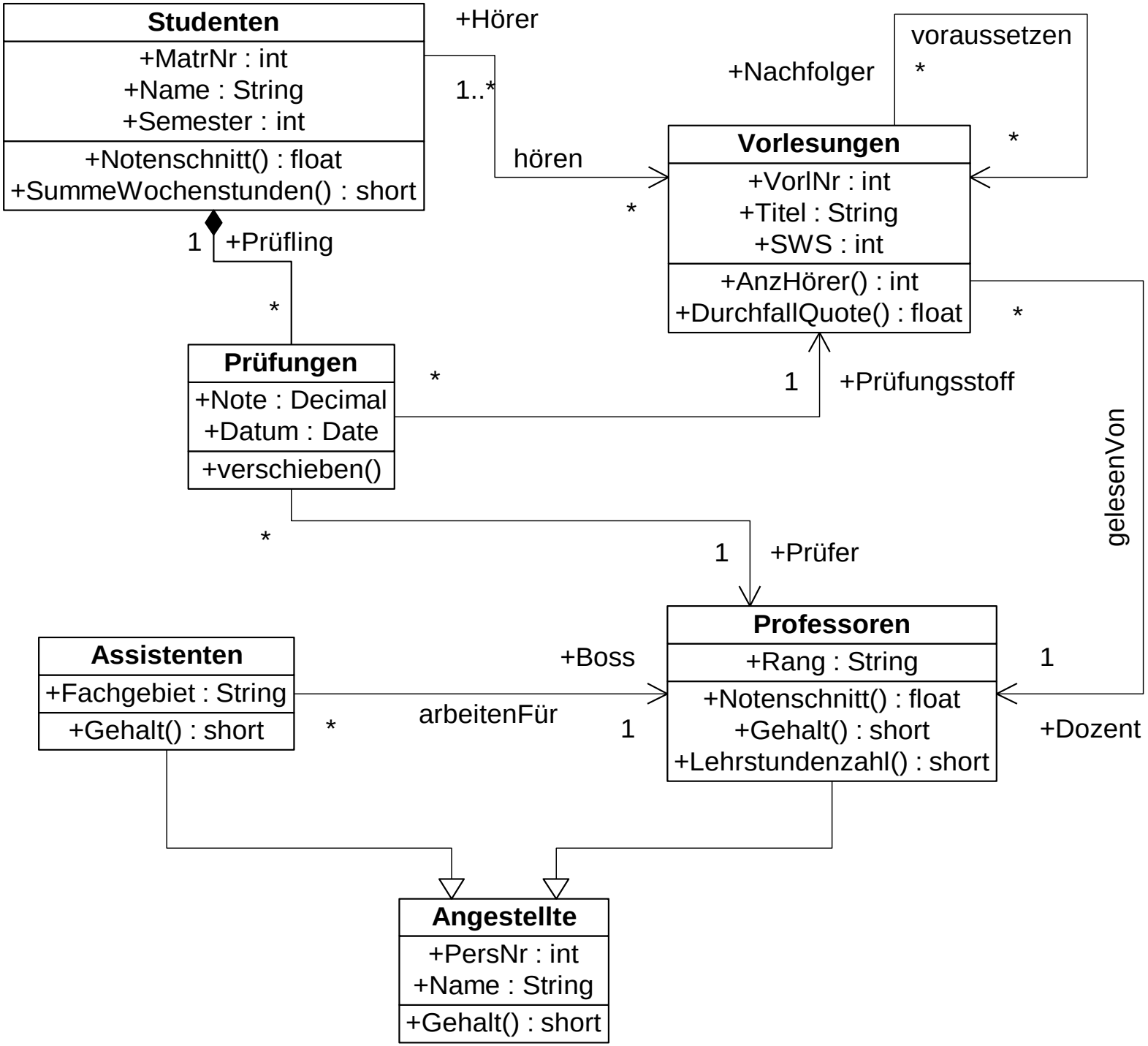






Universitäts-Modell




Studenten
+MatrNr : int
+Name : String
+Semester : int
+Notenschnitt() : float
+SummeWochenstunden() : short
+Hörer
1..*
hören
*
Vorlesungen
+VorlNr : int
+Titel : String
+SWS : int
+AnzHörer() : int
+DurchfallQuote() : float
voraussetzen
*
+Nachfolger
*
1
+Prüfling
*
Prüfungen
+Note : Decimal
+Datum : Date
+verschieben()
*
1
+Prüfungsstoff
*
gelesenVon
*
1
+Prüfer
Assistenten
+Fachgebiet : String
+Gehalt() : short
*
arbeitenFür
+Boss
1
Professoren
+Rang : String
+Notenschnitt() : float
+Gehalt() : short
+Lehrstundenzahl() : short
1
+Dozent
Angestellte
+PersNr : int
+Name : String
+Gehalt() : short

Umsetzung in Java
1:1-Assoziation


```
class E1 {
    public ... att11;
    public ... att12;
    public E2 zugeordnetesE2;
    public ... op1() {}
}

class E2 {
    public ... att21;
    public ... att22;
    public E1 zugeordnetesE1;
    public ... op2() {}
}
```

Umsetzung in Java
1:N-Assoziation


```
class E1 {
    public ... att11;
    public ... att12;
    public E2[] zugeordneteE2;
    public ... op1() {}
}

class E2 {
    public ... att21;
    public ... att22;
    public E1 zugeordnetesE1;
    public ... op2() {}
}
```

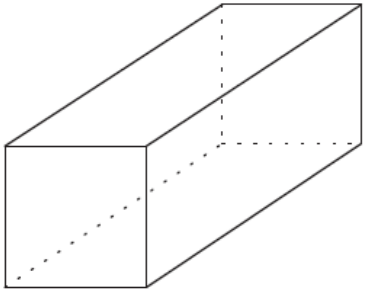
```
class Quader2 {
    public Vertex[] eckPunkte;
    public Material mat;
    public double wert;
}
```

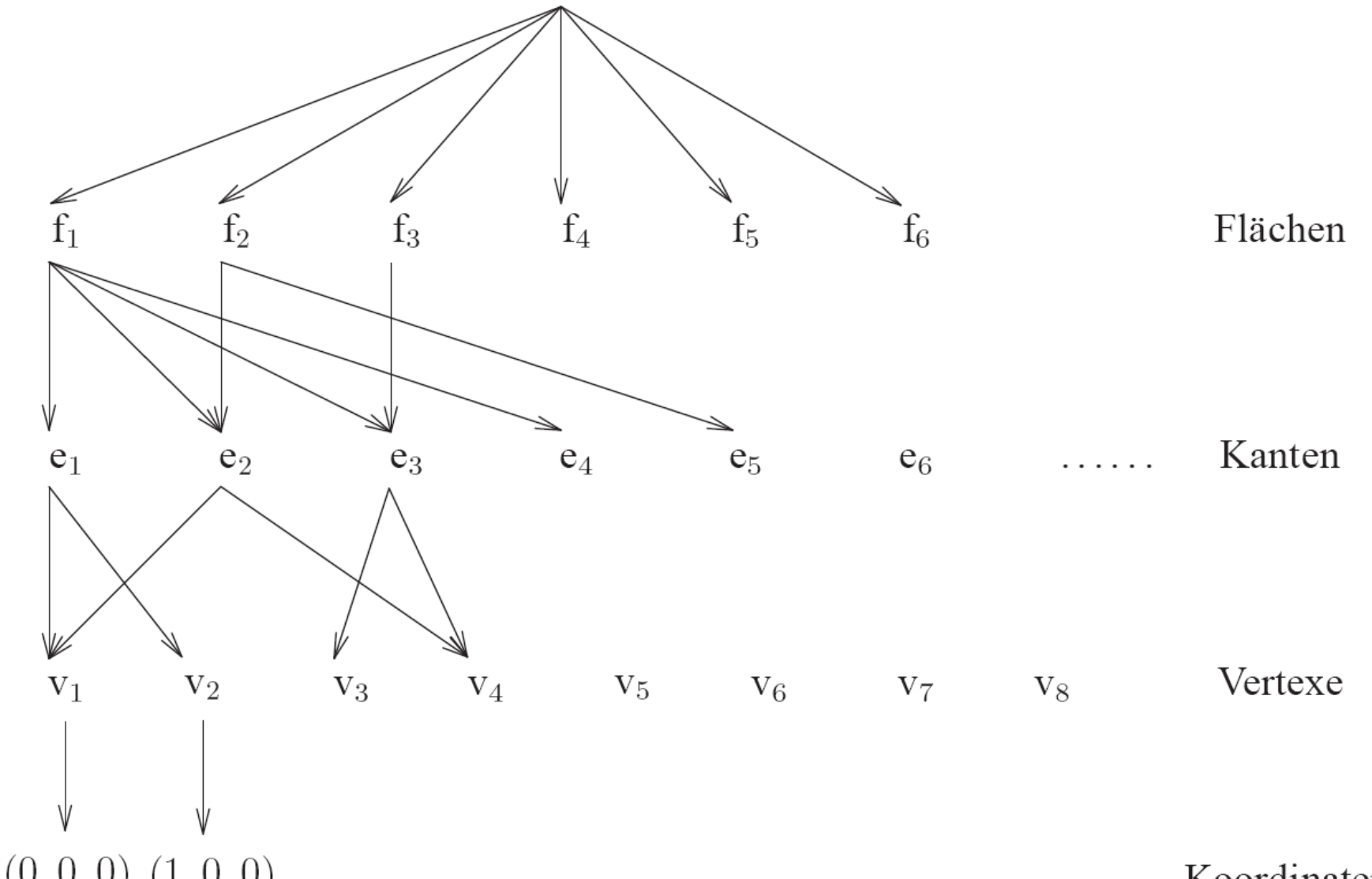
Umsetzung einer Assoziation in Java viele-viele (N:M)


```
class E1 {
    public ... att11;
    public ... att12;
    public E2[] zugeordneteE2;
    public ... op1() {}
}
```

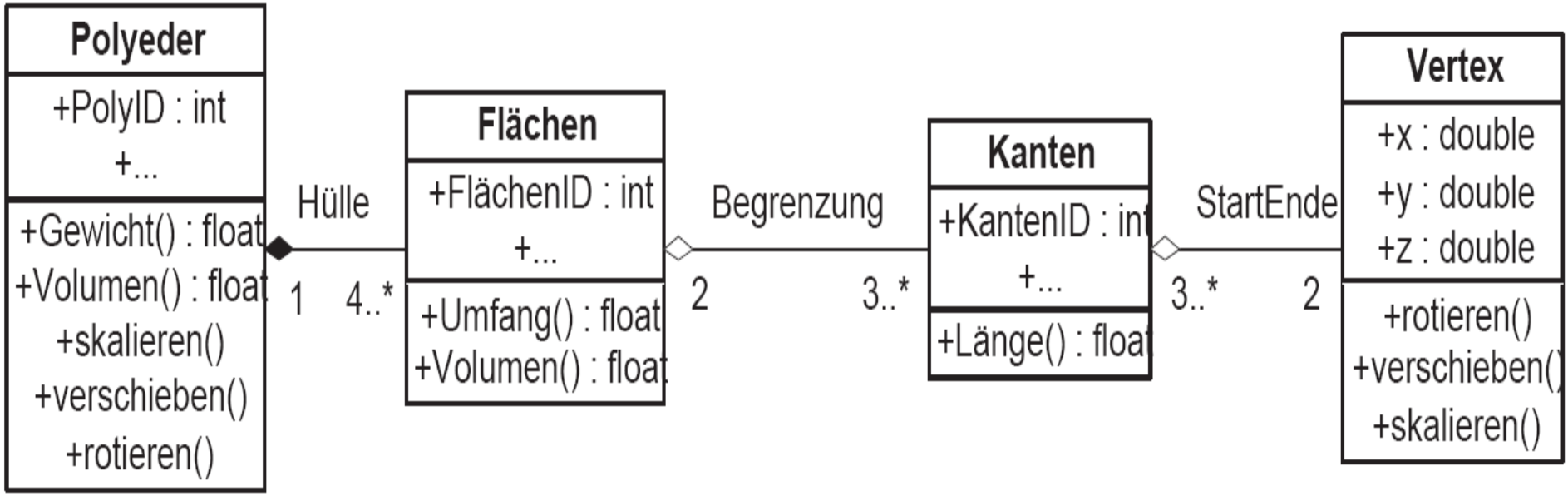
```
class E2 {
    public ... att21;
    public ... att22;
    public E1[] zugeordneteE1;
    public ... op2() {}
}
```

Begrenzungs-Flächen-Modell

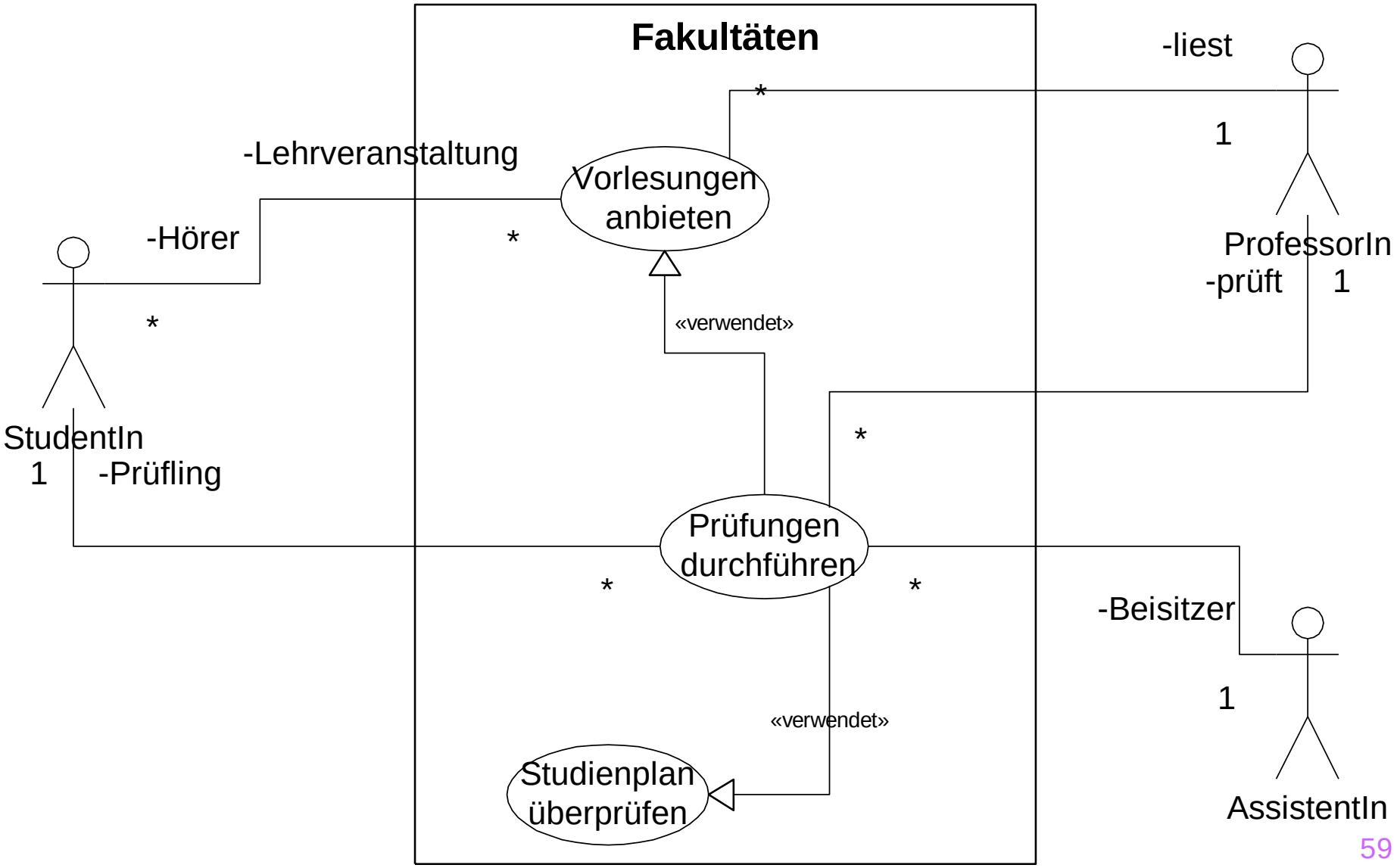




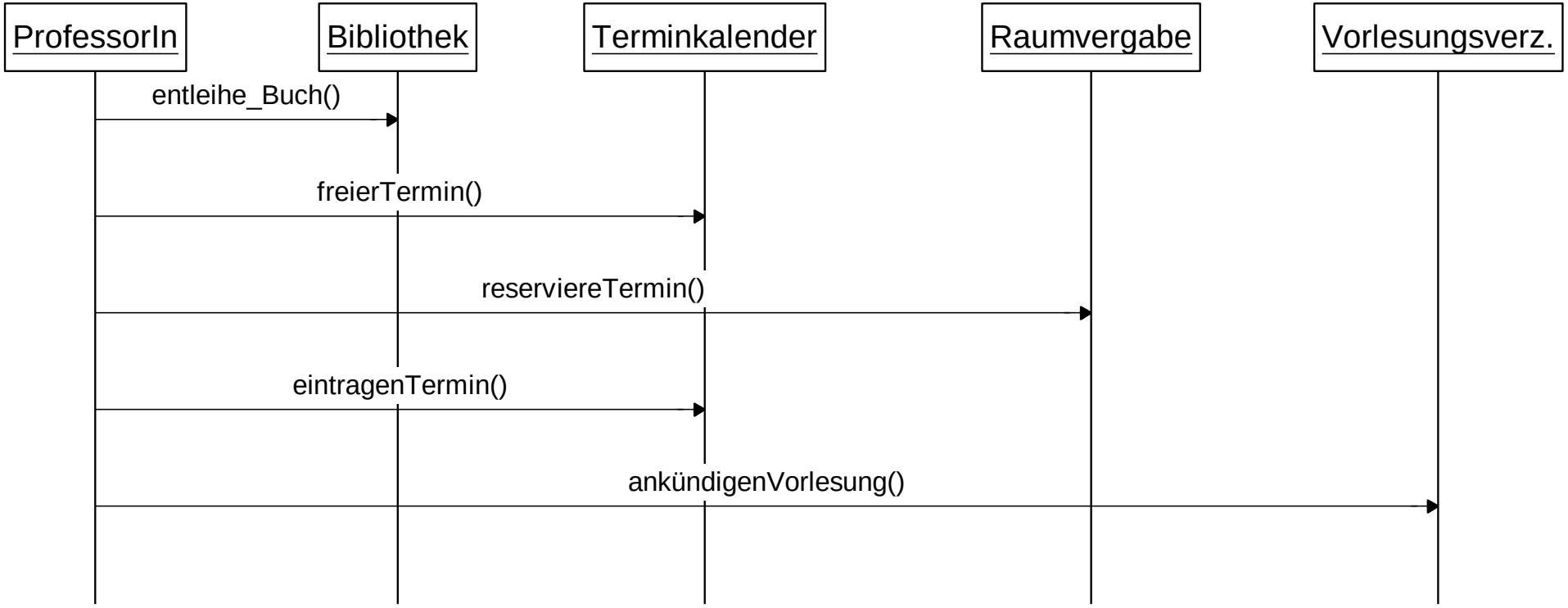
Polyeder in UML



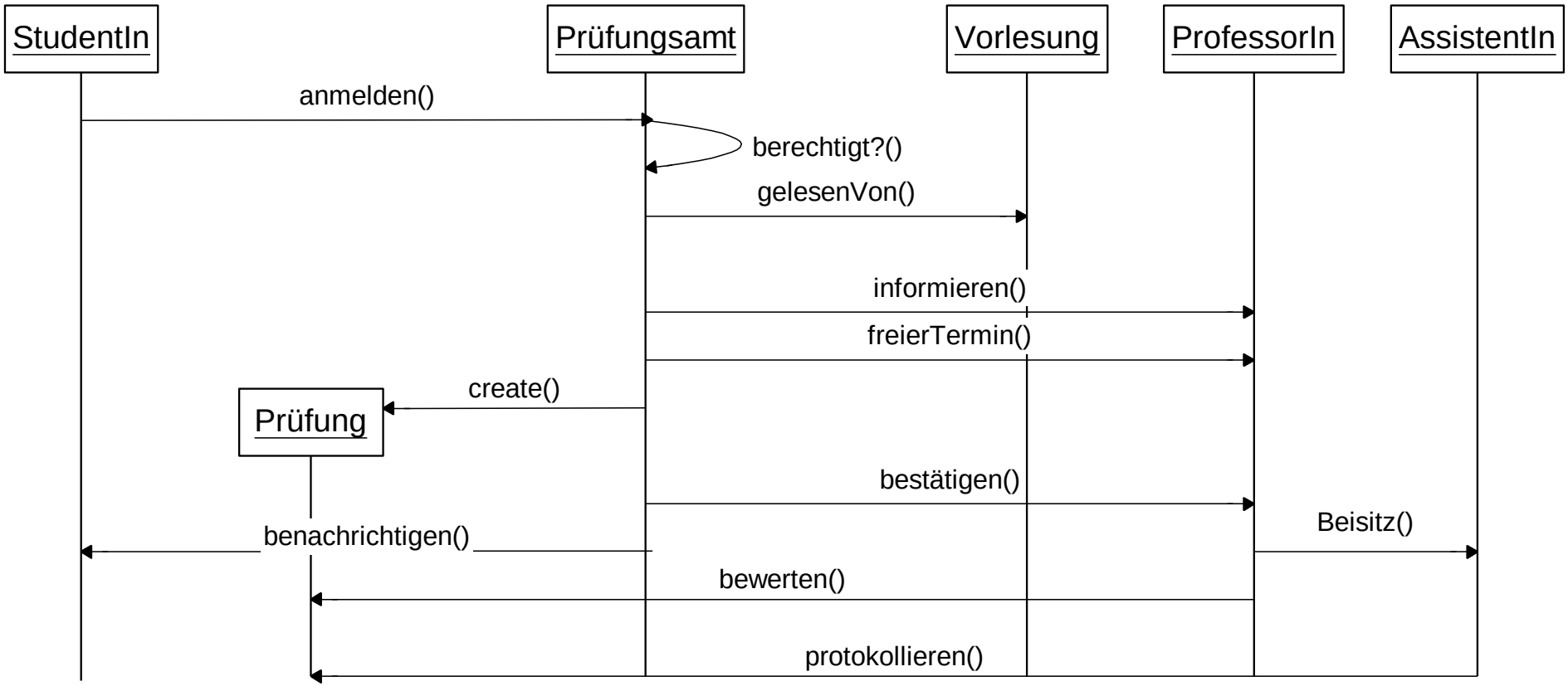
Anwendungsfälle (use cases)



Interaktions-Diagramm: Modellierung komplexer Anwendungen



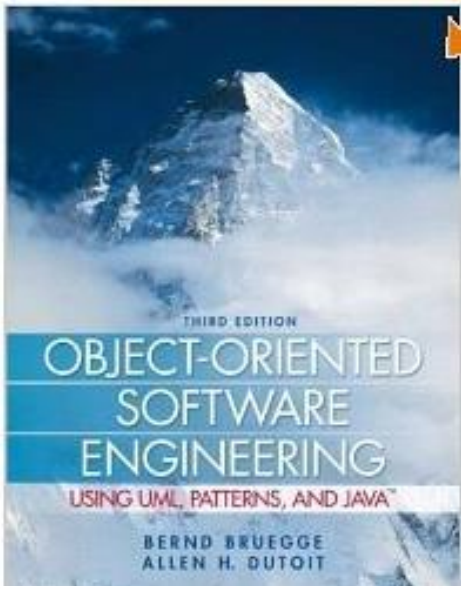
Interaktions-Diagramm:
Prüfungsdurchführung



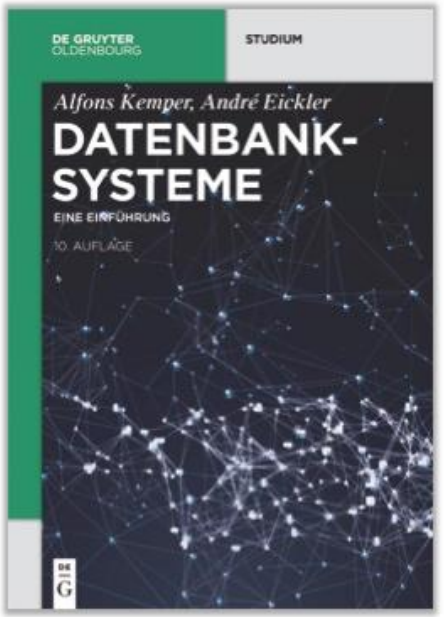
Einführung in die Informatik II für Ingenieurwissenschaften (MSE)

- Prof. Alfons Kemper, Ph.D.
- Christoph Anneser

- Teil 1:
  - Objektorientierte Modellierung (in UML) und
  - Programmierung in Java



- Teil 2:
  - Datenbanksysteme: Eine Einführung
  - Alfons Kemper und Andre Eickler
  - Oldenbourg Verlag, 10. Auflage, 2016



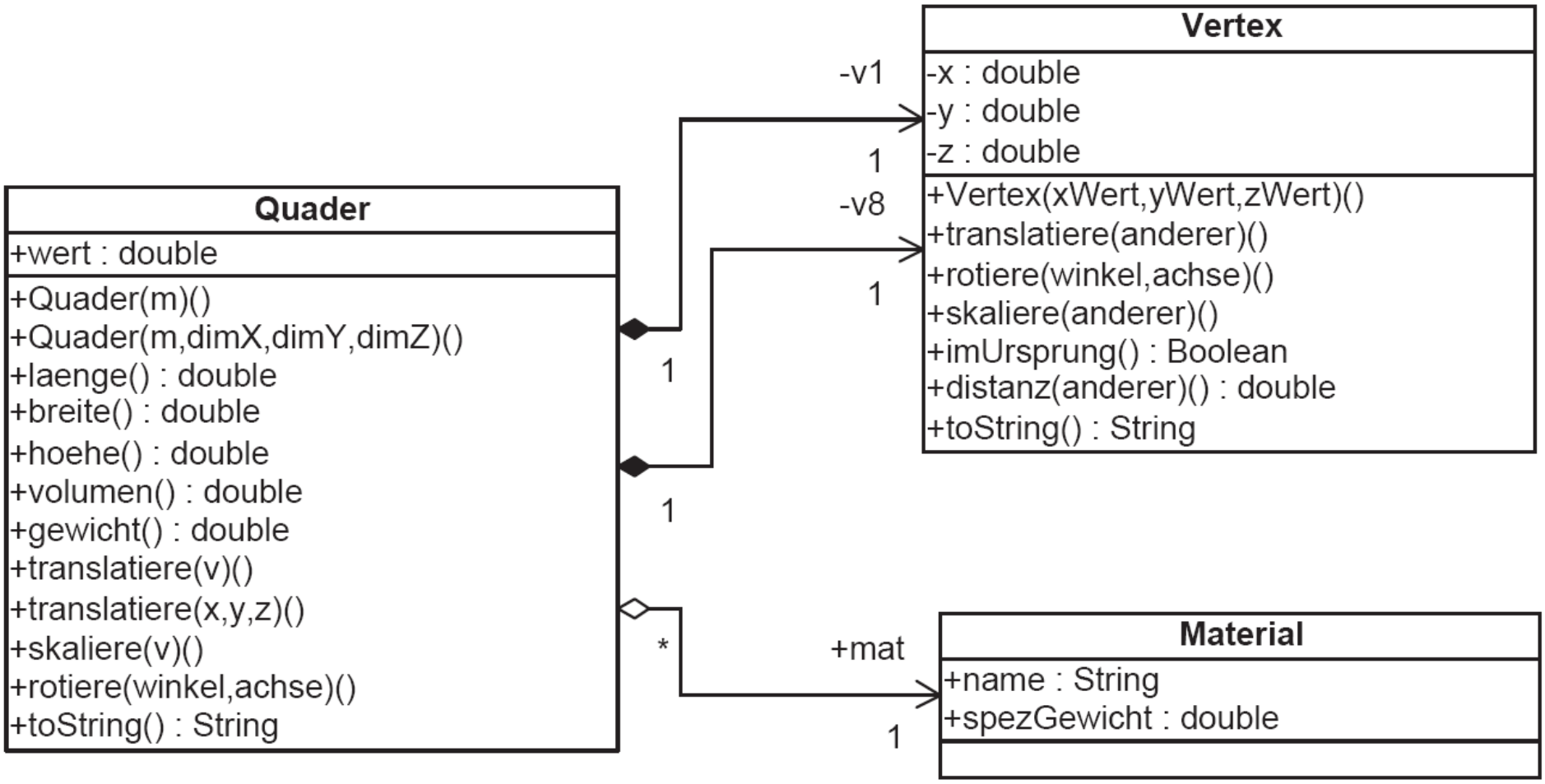
Verhalten der Objekte: Operationen

- Werden in den Klassen definiert
- Werden (i.d.R.) auf einem Objekt aufgerufen
  - Wird das Empfängerobjekt genannt
- Weitere Objekte können „mitspielen“
  - Werden als Parameter übergeben
- Ein Objekt oder ein Wert kann als Rückgabe-Parameter definiert werden
  - Oft werden Operationen aber nichts zurückgeben was als void gekennzeichnet wird

Klassifikation der Operationen

- Konstruktoren
  - Dienen der Initialisierung des Objekts
  - Oft wird in dem Zuge ein ganzes Objektnetz aufgebaut, indem untergeordnete Objekte gleich mit initialisiert werden, indem man im Konstruktor deren Konstruktoren mit aufruft
- Observer/Beobachter
  - Diese Operationen geben den internen Zustand (bzw. einen Teil davon) zurück
  - Haben also immer einen Rückgabe-Parameter
- Mutatoren
  - Ändern den internen Zustand des Objekts
  - Verursachen also Seiteneffekte
  - Haben meist keine Rückgabe: void

Verhalten von Quader- und Vertex-Objekten



...Java

```java
public class Vertex {
    double x;
    double y;
    double z;

    public Vertex(double xWert, double yWert, double zWert) {
        this.x = xWert; this.y = yWert; this.z = zWert;
    }

    public void translatiere(Vertex anderer) {  // Mutator
        this.x = this.x + anderer.x;   // kürzer: x += anderer.x;
        this.y = this.y + anderer.y;   //         y += anderer.y;
        this.z = this.z + anderer.z;   //         z += anderer.z;
    }

    public void skaliere(Vertex anderer) {
        // ...
    }

    public void rotiere(double winkel, char achse) {
        // ...
    }

    public boolean imUrsprung() {  // liegt der Wert sehr nahe bei (0,0,0) ?
        double epsilon = 0.000000001;  // Präzisionsgrenze
        return ((this.x < epsilon) && (this.x > -epsilon) &&
                (this.y < epsilon) && (this.y > -epsilon) &&
                (this.z < epsilon) && (this.z > -epsilon));
    }

    public double distanz(Vertex anderer) { // Distanz zwischen this und anderer
        double dx, dy, dz;
        dx = this.x - anderer.x;
        dy = this.y - anderer.y;
        dz = this.z - anderer.z;
        return Math.sqrt(dx * dx + dy * dy + dz * dz);
    }    // sqrt ist in Math definiert

    public String toString() {
        return ("x:␣" + this.x + ",␣y:␣" + this.y + ",␣z:␣" + this.z);
    }

} // public class Vertex
```

```
[public|private|protected] <ErgebnisTyp> <Name>(<ParameterTypListe>) {
   // Implementierung
}
```

Die einzelnen Bestandteile haben die folgende Bedeutung:

- Der *<Name>* identifiziert die Operation innerhalb der Klasse. Man beachte, dass jede Klasse ihren eigenen Namensraum hat, so dass die gleichnamigen Operationen von *Vertex* bzw. *Quader* sich nicht gegenseitig ins Gehege kommen. Sie sind zwar logisch verwandt, haben aber aus der Sicht des Java-Compilers nichts miteinander zu tun. Anders verhält es sich da bei gleichnamigen Operationen innerhalb derselben Klasse. Auch hierfür haben wir ein Beispiel: Den Konstruktor *Vertex* gibt es zweimal – einmal mit drei Parametern und einmal ohne Parameter. Dies bezeichnen wir als Überladung (engl. *overloading*) der Operation. Es handelt sich hierbei um zwei unterschiedliche Operationen, die vom Compiler anhand der Parameter beim Aufruf auseinander gehalten werden können. Mehr dazu erklären wir in dem Abschnitt 2.6.

- Der *<ErgebnisTyp>* legt den Typ des Rückgabe-Objekts bzw. des Rückgabewerts fest. Falls es keine Rückgabe gibt, wird **void** angegeben.

- Die *<ParameterTypListe>* gibt die Anzahl und die Typen der Parameter an.

- Die Implementierung erfolgt zum Schluss – innerhalb der geschweiften Klammern. Man beachte, dass ein guter objektorientierter Entwurf im Allgemeinen dafür sorgt, dass die einzelnen Operationen sehr einfach und knapp realisiert werden können.

- Die so genannten *access modifier* [**public**|**private**|**protected**] legen die Sichtbarkeit einer Operation fest. Eine öffentliche Operation ist von überall aufrufbar und wird mit dem Schlüsselwort **public** gekennzeichnet. Die privaten Operationen sind nur innerhalb der Klassendefinition aufrufbar – es handelt sich hierbei um Hilfsroutinen zur Implementierung der öffentlichen Operationen. Die mit **protected** angegebenen Operationen können auch in einer Unterklasse verwendet werden. Diese Sichtbarkeitsregeln werden in Abschnitt Abschnitt 2.4 nochmals diskutiert, da sie essentiell

Aufruf der Operationen

```
Vertex meinVertex = new Vertex(1.0,0.0,0.0);;
Vertex translationsVertex = new Vertex(0.0,2.0,2.0);
...
meinVertex.translatiere(translationsVertex);
```

Die Dot-Notation

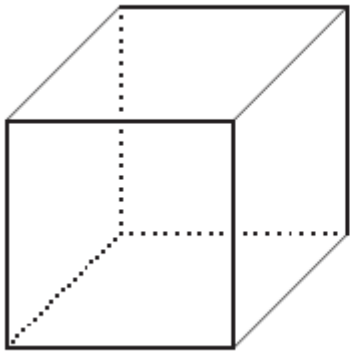
```
meinQuader.v1.translatiere(translationsVertex);
```

Pfadausdrücke (Dot-Notation) mit Operatoren „mitten drin“

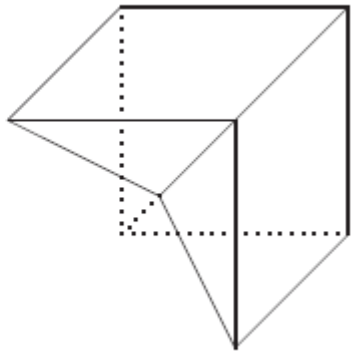
```
public class Person {
    public int alter;
    public Person ehePartner;
    public Person mutter;
    // ...
    public Person schwiegerMutter() {
       return ehePartner.mutter;
    }
    // ...
}
...
Person mickey;
...
mickey.schwiegerMutter();
mickey.schwiegerMutter().ehePartner;
```

Information Hiding: Geheimnisprinzip/Verkapselung

```
   Quader meinKomischerQuader;
   Material gold = new Material();   // neues Material
   gold.name = "Gold";
   ...
   meinKomischerQuader = new Quader(gold); // goldener Quader
   ...
(1)meinKomischerQuader.v1.x = 0.5;
(2)meinKomischerQuader.v1.y = 0.5;
(3)meinKomischerQuader.v1.z = 0.5;
```



before



after

```
1  public class Quader {
2      Vertex v1, v2, v3, v4, v5, v6, v7, v8;
3      Material mat;
4      double wert;
5
6      public Quader(Material m) {  // Konstruktor: Einheitswürfel wird kreiert
7          this.v1 = new Vertex(0.0,0.0,0.0);  this.v2 = new Vertex(1.0,0.0,0.0);
8          this.v3 = new Vertex(1.0,1.0,0.0);  this.v4 = new Vertex(0.0,1.0,0.0);
9          this.v5 = new Vertex(0.0,0.0,1.0);  this.v6 = new Vertex(1.0,0.0,1.0);
10         this.v7 = new Vertex(1.0,1.0,1.0);  this.v8 = new Vertex(0.0,1.0,1.0);
11         this.mat = m;
12         this.wert = 39.99;  // fiktiver Wert
13     }
14     public double laenge() {                // Observer-Funktion
15         return this.v1.distanz(this.v5);
16     }
17     public double breite() {
18         return this.v1.distanz(this.v2);
19     }
20     public double hoehe() {
21         return this.v1.distanz(this.v4);
22     }
23     public double volumen() {
24         return this.laenge() * this.hoehe() * this.breite();
25     }
26     public double gewicht() {
27         return this.volumen() * this.mat.spezGewicht;
28     }
29     public void translatiere(Vertex v) {    // Mutator
30         this.v1.translatiere(v);    this.v2.translatiere(v);
31         this.v3.translatiere(v);    this.v4.translatiere(v);
32         this.v5.translatiere(v);    this.v6.translatiere(v);
33         this.v7.translatiere(v);    this.v8.translatiere(v);
34     }
35     public void translatiere(double xWert, double yWert, double zWert) {
36         this.translatiere(new Vertex(xWert,yWert,zWert));
37                     // delegieren an obige translatiere Op.
38     }
39     public void skaliere(Vertex v) {
40         // ...
41     }
42     public void rotiere(double winkel, char achse) {
43         // ...
44     }
45     public String toString() { // observer
46        return("Quader␣der␣Dimension␣" + this.laenge() + "␣X␣" +
47                                         this.breite() + "␣X␣" + this.hoehe());
48     }
49
50 } // public class Quader
```

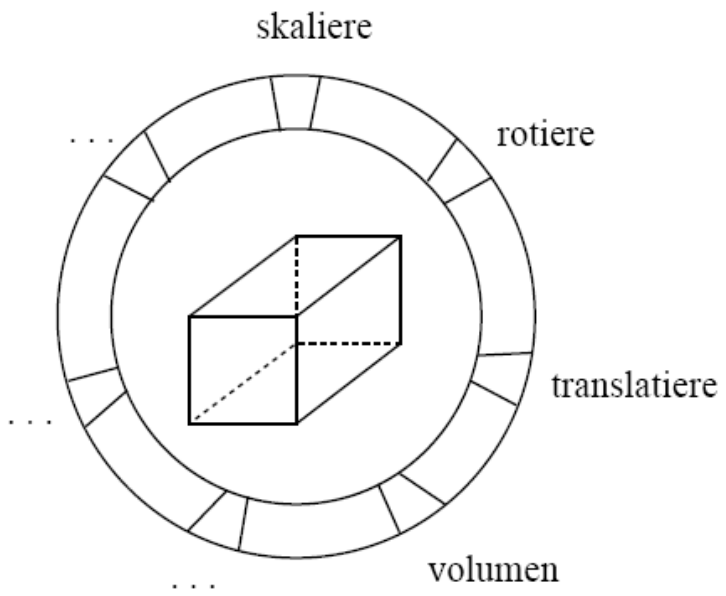
Access Modifier beschränken den Zugriff

- **public:** Öffentliche Komponenten sind für alle Klienten zugreifbar und werden als *public* angegeben.
- **private:** Die verborgenen Komponenten werden als *private* spezifiziert und sind nur für den Code innerhalb der Klasse sichtbar.
- **protected:** Die mit *protected* angegebenen Komponenten sind wie bei *private* nicht allgemein sichtbar. Im Unterschied zu *private* können Unterklassen aber auf die als *protected* gekennzeichneten Komponenten ihrer Oberklassen zugreifen.
- **package:** Der Default – wenn also nichts angegeben ist – ist der access modifier *package*. Code innerhalb desselben Packages hat Zugriff auf die Komponenten.

Quader-Definition

```
public class Quader {
    private Vertex v1, v2, v3, v4, v5, v6, v7, v8;
    public Material mat;
    public double wert;

    public Quader(Material m) {  // Einheitsquader kreieren
        // ...
    }
    public double laenge() {
        // ...
    }
    public double breite() {
        // ...
    }
    public double hoehe() {
        // ...
    }
```

Quader – cont‘d




```
    public double volumen() {
        // ...
    }
    public double gewicht() {
        // ...
    }
    public void translatiere(Vertex v) {
        // ...
    }
    public void translatiere(double xWert, double yWert, double zWert){
        // ...
    }
    public void skaliere(Vertex v) {
        // ...
    }
    public void rotiere(double winkel, char achse) {
        // ...
    }
    public String toString() {
        // ...
    }
} // public class Quader
```

Best Practice: Verbergen von Instanzvariablen

```
class Quader {
    private double wert;
    ...
    public double wert() {
       return this.wert;
    }
}
```

```
double wieViel;
Quader meinQuader;
...
wieViel = meinQuader.wert();
```

Später kann die Berechnung geändert werden

Initialisierung eines Objekts

```
public class Quader {
    private Vertex v1, v2, v3, v4, v5, v6, v7, v8;
    public Material mat;
    public double wert;

    public Quader(Material m) {  // Konstruktor: Einheitswürfel
        this.v1 = new Vertex(0.0,0.0,0.0);
        this.v2 = new Vertex(1.0,0.0,0.0);
        this.v3 = new Vertex(1.0,1.0,0.0);
        this.v4 = new Vertex(0.0,1.0,0.0);
        this.v5 = new Vertex(0.0,0.0,1.0);
        this.v6 = new Vertex(1.0,0.0,1.0);
        this.v7 = new Vertex(1.0,1.0,1.0);
        this.v8 = new Vertex(0.0,1.0,1.0);
        this.mat = m;
        this.wert = 39.99;  // fiktiver Wert
    }
    // ...
} // public class Quader
```

Initialisierung eines Vertex‘es

```
class Vertex {
    // ...
    public Vertex() {
       this.x = 0.0;  // Initialisierung könnte auch an den
       this.y = 0.0;  // anderen Konstruktor delegiert werden:

       this.z = 0.0;  // this(0.0,0.0,0.0);
    }
    // ...
}
```

```
Quader meinQuader;
Material gold;
gold = new Material();
    ...
meinQuader = new Quader(gold);
    ...
```

Overloading: Mehrere Operationen gleichen Namens

```
public class Quader {
    private Vertex v1, v2, v3, v4, v5, v6, v7, v8;
    public Material mat;
    public double wert;

    public Quader(Material m) {  //  Konstruktor: Einheitswürfel
        ... // Implementierung wie vorher


    }

    public Quader(Material m, double dimX, double dimY, double dimZ) {
        this.v1 = new Vertex(0.0,0.0,0.0);
        this.v2 = new Vertex(dimX,0.0,0.0);
        this.v3 = new Vertex(dimX,dimY,0.0);
        this.v4 = new Vertex(0.0,dimY,0.0);
        this.v5 = new Vertex(0.0,0.0,dimZ);
        this.v6 = new Vertex(dimX,0.0,dimZ);
        this.v7 = new Vertex(dimX,dimY,dimZ);
        this.v8 = new Vertex(0.0,dimY,dimZ);
        this.mat = m;
        this.wert = 39.99;  // fiktiver Wert
    }
    // ...
} // end class Quader
```

Aufruf unterscheidet sich entweder in Anzahl oder Typ der Parameter

Die Nutzung wird dann in folgendem Programmfragment illustriert:

```
   Quader meinQuader;
   Quader deinQuader;
   Material gold;
      ...
(1)meinQuader = new Quader(gold);                  // goldener Einheitswürfel
(2)deinQuader = new Quader(gold, 2.0, 3.0, 5.0); // Goldbarren Quader
   ...                                             // der Dimension 2 × 3 × 5
```

Translatiere unterschiedlich aufgerufen ...

```
public class Quader {
    // ...
    public void translatiere(Vertex v) {
        this.v1.translatiere(v);    this.v2.translatiere(v);
        this.v3.translatiere(v);    this.v4.translatiere(v);
        this.v5.translatiere(v);    this.v6.translatiere(v);
        this.v7.translatiere(v);    this.v8.translatiere(v);
    }
    public void translatiere(double x, double y, double z) {
        // delegiere die Arbeit an die andere translatiere-Operation
        this.translatiere(new Vertex(x,y,z));
    // ...
} // public class Quader
```

```
Quader meinQuader = new Quader(...);;
Vertex translatiereVertex = new Vertex(0.0,3.0,2.0);
   ...
meinQuader.translatiere(translatiereVertex);
meinQuader.translatiere(30.0, 2.0, 1.75);
   ...
```

Statische Operationen

```
Vertex p1, p2;
double d;
    ...
d = p1.distanz(p2);
    ...
```

Ungewohnt?

```
    ...
 d = distanz(p1, p2);
    ...
```

besser?

Realisierung ...

```
public class Vertex {
    ...
    public static double distanz(Vertex erster, Vertex zweiter) {
        return erster.distance(zweiter);
    }
}
```

Um dann die *distanz* zwischen zwei Punkten zu berechnen, ruft man die statische Methode wie folgt auf:

```
Vertex a = new Vertex(0.0,0.0,0.0);
Vertex b = new Vertex(3.0,0.0,0.0);
System.out.println(Vertex.distanz(a,b));
```

main() ... als statische Operation zum Testen

```
public class Quader {
   ... // wie zuvor
   public String toString() { // observer
       return("Quader der Dimension " + this.laenge() + " X " +
              this.breite() + " X " + this.hoehe());
    }

    public static void main(String args[]) {
       Material eisen = new Material("Eisen", 0.89);
       Quader meinQuader = new Quader(eisen);
       Quader andererQuader = new Quader(eisen);
       andererQuader.translatiere(new Vertex(3.5,2.5,4.5));
       andererQuader.translatiere(2.0,3.0,4.0);
       System.out.println(meinQuader);   // impliziter Aufruf von toString
       System.out.println("meinQuader hat das Gewicht: "
                          + meinQuader.gewicht());
       System.out.println(andererQuader);
       System.out.println("andererQuader hat das Gewicht: "
                          + andererQuader.gewicht());
       Material carbon = new Material("Carbon", 0.75);
       meinQuader.mat = carbon;
       System.out.println("meinQuader hat jetzt das Gewicht: "
                          + meinQuader.gewicht());
    }
```

Nutzung von main

```
javac Quader.java
java Quader
```

Die Ausgabe ist dann wie folgt:

```
Quader der Dimension 1.0 X 1.0 X 1.0
meinQuader hat das Gewicht: 0.89
Quader der Dimension 1.0 X 1.0 X 1.0
andererQuader hat das Gewicht: 0.89
meinQuader hat jetzt das Gewicht: 0.75
```

Parameter-Übergabe

Nun muss man zwei verschiedene Arten der Parameterübergabe unterscheiden: *call by value* und *call by reference*. *Call by value* kommt zum Einsatz, wenn primitive Typen als Argument verwendet werden. In diesem Fall werden die Argumente kopiert. Bei Objekttypen bezeichnet man die Semantik der Parameterübergabe als *call by reference*, da in diesem Fall nur eine Referenz auf das Objekt übergeben wird und keine Kopie des Objekts. Ändert nun die Operation Eigenschaften beim übergebenen Objekt wirkt sich dies auch außerhalb der Operation auf das Objekt aus – es wurde ja gerade keine automatische Kopie erstellt. Möchte man ein übergebenes Objekt nur in der Operation verändern, muss man es selbst kopieren.

Ausnahmen ... abfangen

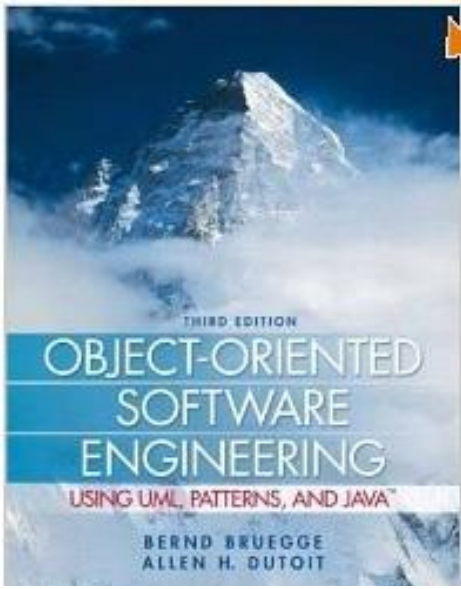
- Try ... catch

```
public class TesteAusnahme {
    public static void main(String[] args) {
        Material eisen = new Material("Eisen", 0.89);
        Quader meinQuader;
        try {
            meinQuader = new Quader(eisen,0.0,3.0,9.0);
        }
        catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

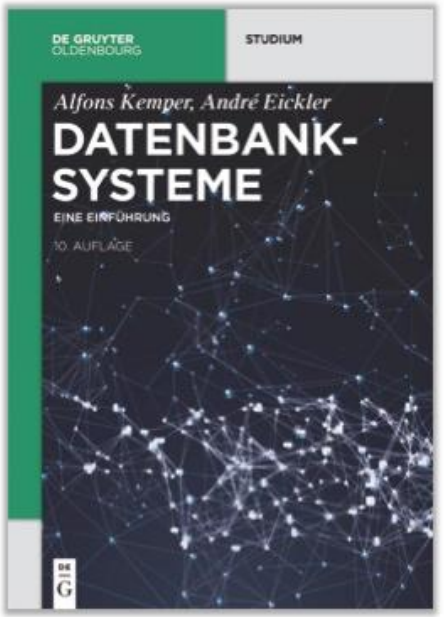
Einführung in die Informatik II für Ingenieurwissenschaften (MSE)

- Prof. Alfons Kemper, Ph.D.
- Christoph Anneser

- Teil 1:
  - Objektorientierte Modellierung (in UML) und
  - Programmierung in Java



- Teil 2:
  - Datenbanksysteme: Eine Einführung
  - Alfons Kemper und Andre Eickler
  - Oldenbourg Verlag, 10. Auflage, 2016



Generalisierung/Spezialisierung
Subtypisierung/Vererbung

- Bringt Struktur in die Klassen-Diagramme
- Erhöht die Wiederverwendbarkeit
- Erlaubt die schrittweise Verfeinerung

Motivation: Problem der Wiederverwendung

```
public class Person {
    public String name;
    public int alter;
    public Person ehePartner;
    // ...
    public Person(String n, int a) {
        this.name = n; this.alter = a;
    }
    public void heiraten(Person partner) {
        this.ehePartner = partner;
    }
    // ...
} // public class Person
```

Motivation; cont'd

```
public class Angestellter {
    public String name;
    public int alter;
    public Person ehePartner;
    public int steuerNr;
    public double gehalt;
    public Angestellter boss;

    public Angestellter(String n, int a, int s, double g) {
        this.name = n;
        this.alter = a;
        this.steuerNr = s;
        this.gehalt = g;
    }

    public void heiraten(Person partner) {
        this.ehePartner = partner;
    }

    public boolean istPensioniert() {
        return (this.alter > 64);
    }
} // public class Angestellter
```

Keine Wiederverwendung

Angestellte können nur Personen heiraten???

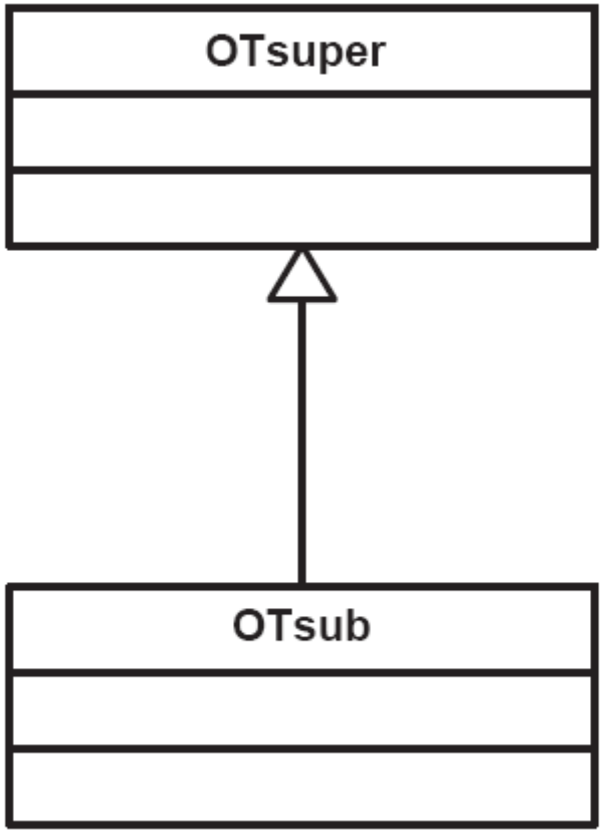
Zwei schwerwiegende Problem gibt es mit diesen beiden Klassendefinitionen:

1. *Mangelnde Wiederverwendbarkeit*
   Die Klasse *Person* enthält viele Komponenten, die in der Klasse *Angestellter* in gleicher Form nochmals repliziert wurden. Es wäre vorteilhafter gewesen, die Definition der Klasse *Angestellter* auf der Definition der Klasse *Person* aufzubauen.

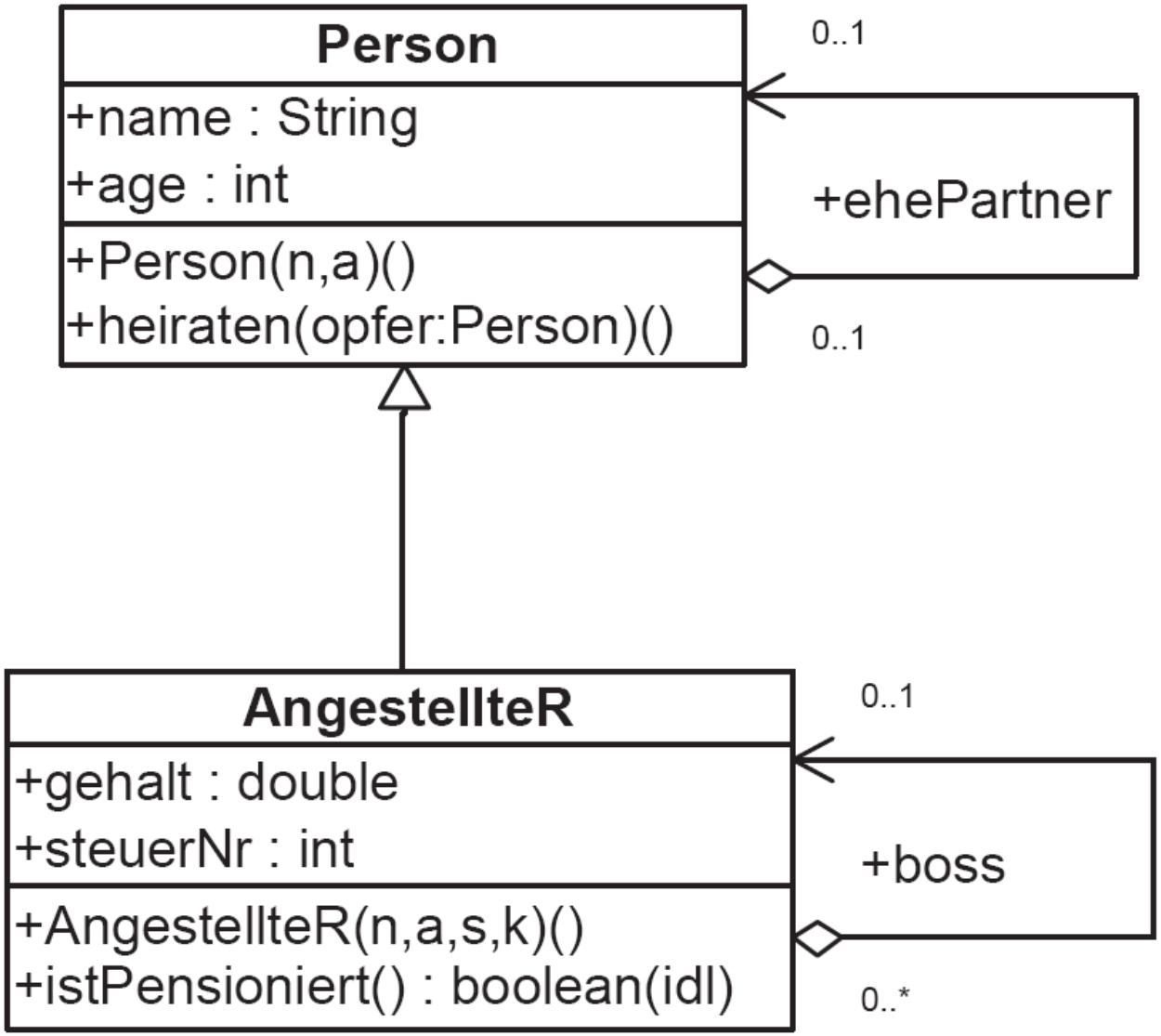
2. *Mangelnde Flexibilität*
   Diese Problem der mangelnden Flexibilität ist noch schwerwiegender. Es wird dadurch verursacht, dass die beiden Klassen völlig isoliert voneinander definiert sind, und die Instanzen der beiden Klassen sich nicht gegenseitig vertreten können. Wir wollen dies an dem Attribut *ehePartner* der Klasse *Person* illustrieren. Dieses Attribut ist genau wie das Argument *partner* der Operation *heiraten* auf Objekte vom Typ *Person* eingeschränkt. Dies bedeutet, dass niemand eine/n Angestellte/n heiraten kann. Das folgende Programmfragment illustriert dieses Problem:

```
Angestellter miniMouse;
Person mickeyMouse;
...
miniMouse.heiraten(mickeyMouse);    // okay, mickeyMouse ist eine Person
mickeyMouse.heiraten(miniMouse);    // illegal, miniMouse ist keine Person
```

Subtypisierung: Overtyp/Untertyp



Generalisierung/Spezialisierung



extends

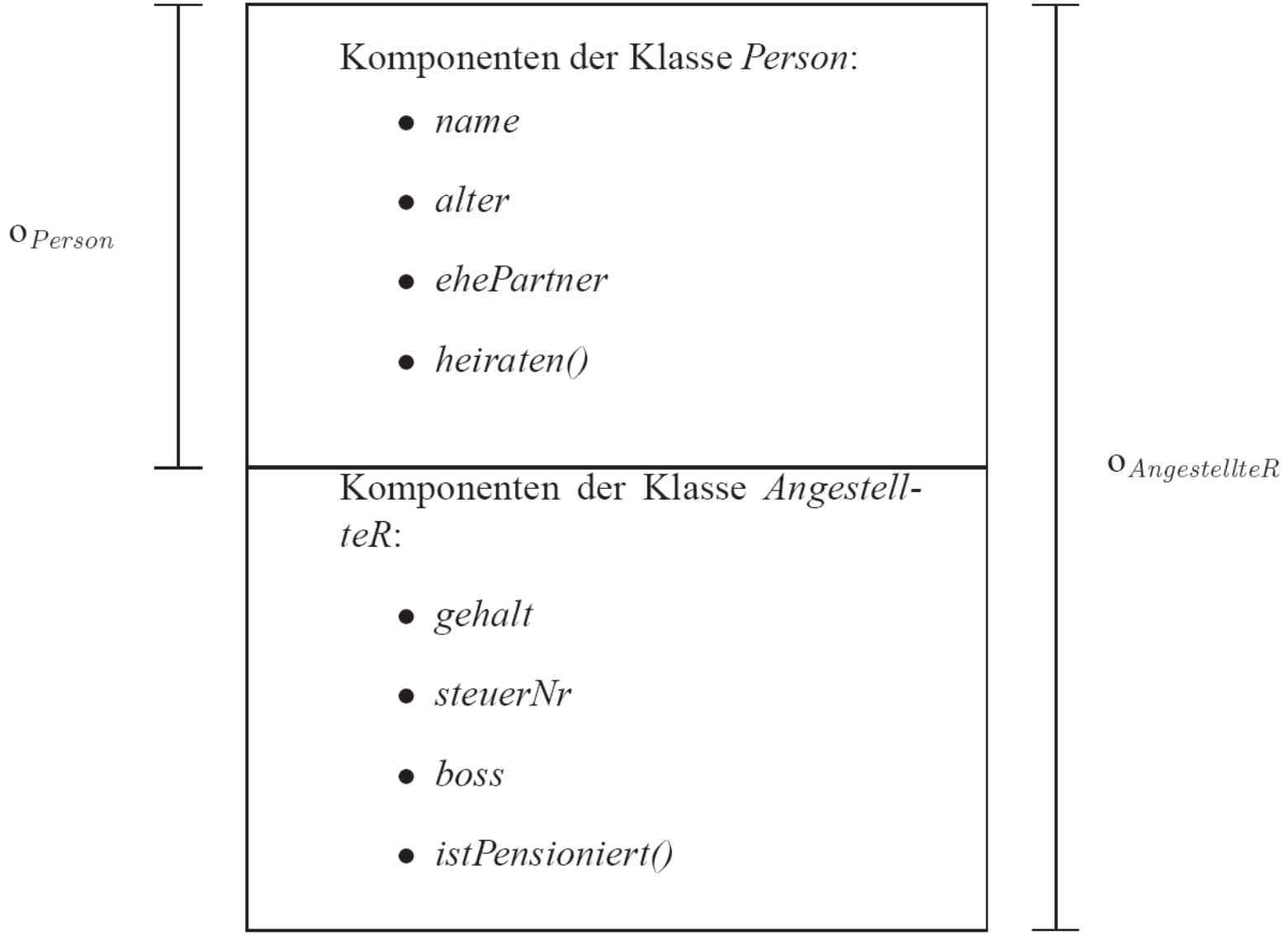
```
class Angestellter extends Person {
    public int steuerNr;
    public double gehalt;
    public Angestellter boss;

    Angestellter(String n, int a, int s, double knete) {
        super(n, a);
        this.steuerNr = s;
        this.gehalt = knete;
    }

    public boolean istPensioniert() {
        return (this.alter > 64);
    }

} // class Angestellter
```

Substituierbarkeit: Typ-Sicherheit immer noch gewährleistet

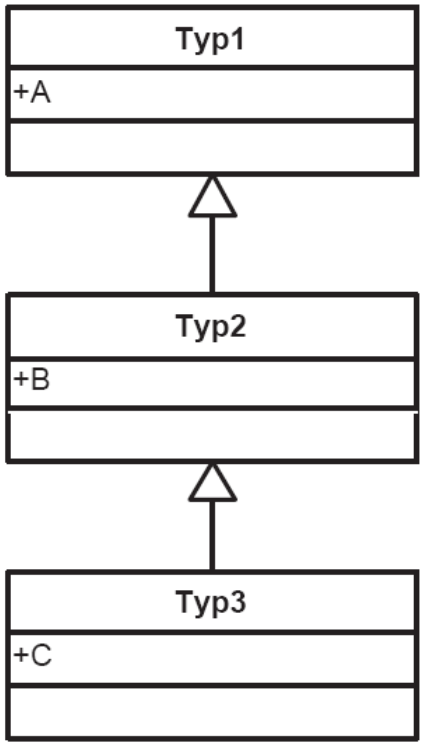
```
miniMouse = new Angestellter("Mini Mouse", 60, 007, 90000.0);
mickeyMouse = new Person("Mickey Mouse", 50);
```

```
Angestellter miniMouse;
Person mickeyMouse;
...
miniMouse.heiraten(mickeyMouse);    // okay, mickeyMouse ist eine Person
mickeyMouse.heiraten(miniMouse);    // jetzt okay, miniMouse ist
                                    als Angestellter auch eine Person
```

Vererbung: am Beispiel erläutert (eine Subtyp-Instanz „kann mehr“)



Generalisierung-Hierarchie



Instanzen

Typ1

id_1

A: . . .

Typ2

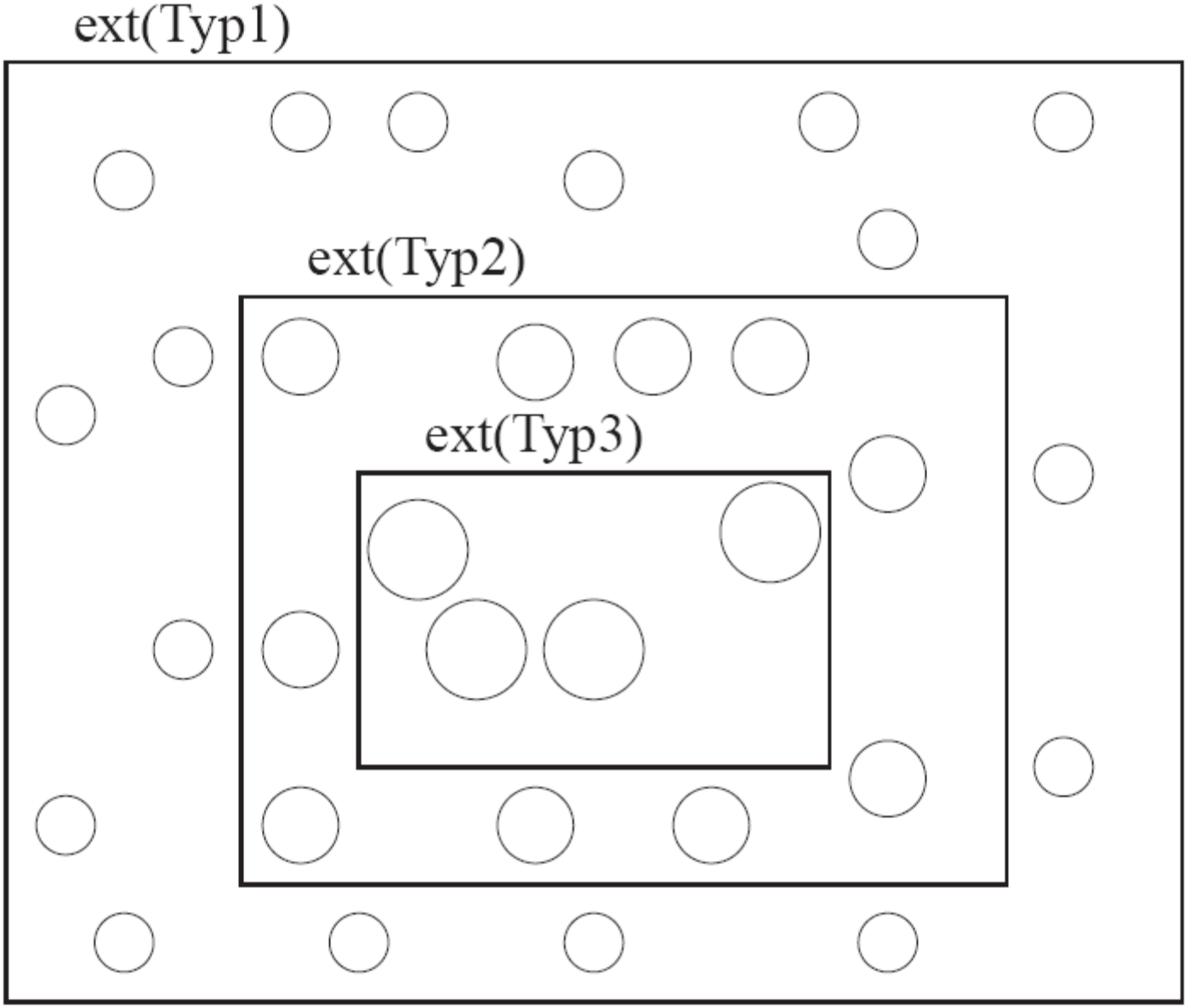
id_2

A: . . .
B: . . .

Typ3

id_3

A: . . .
B: . . .
C: . . .

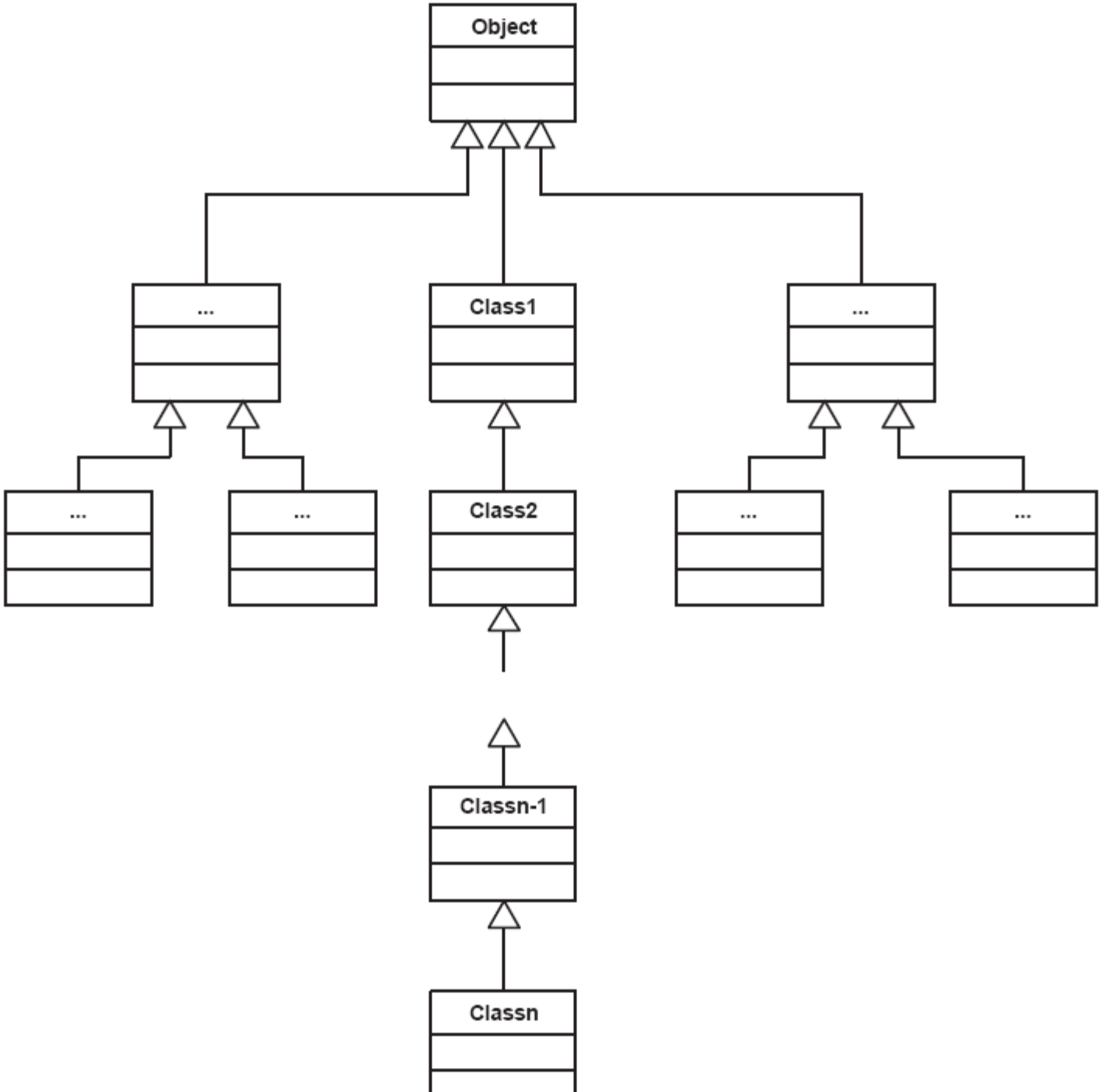


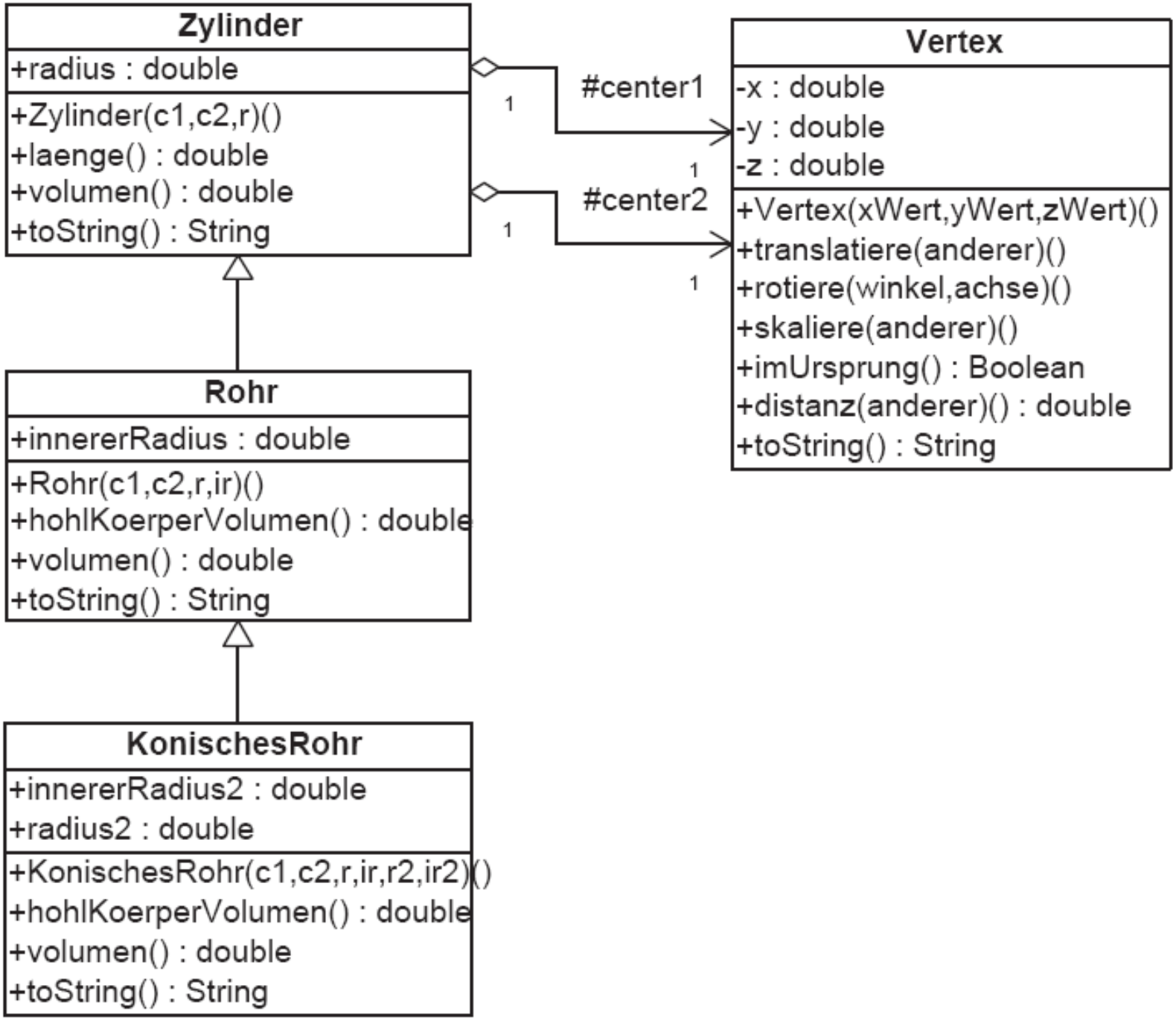
Visualisierung der Subtypisierung/Mengeninklusion

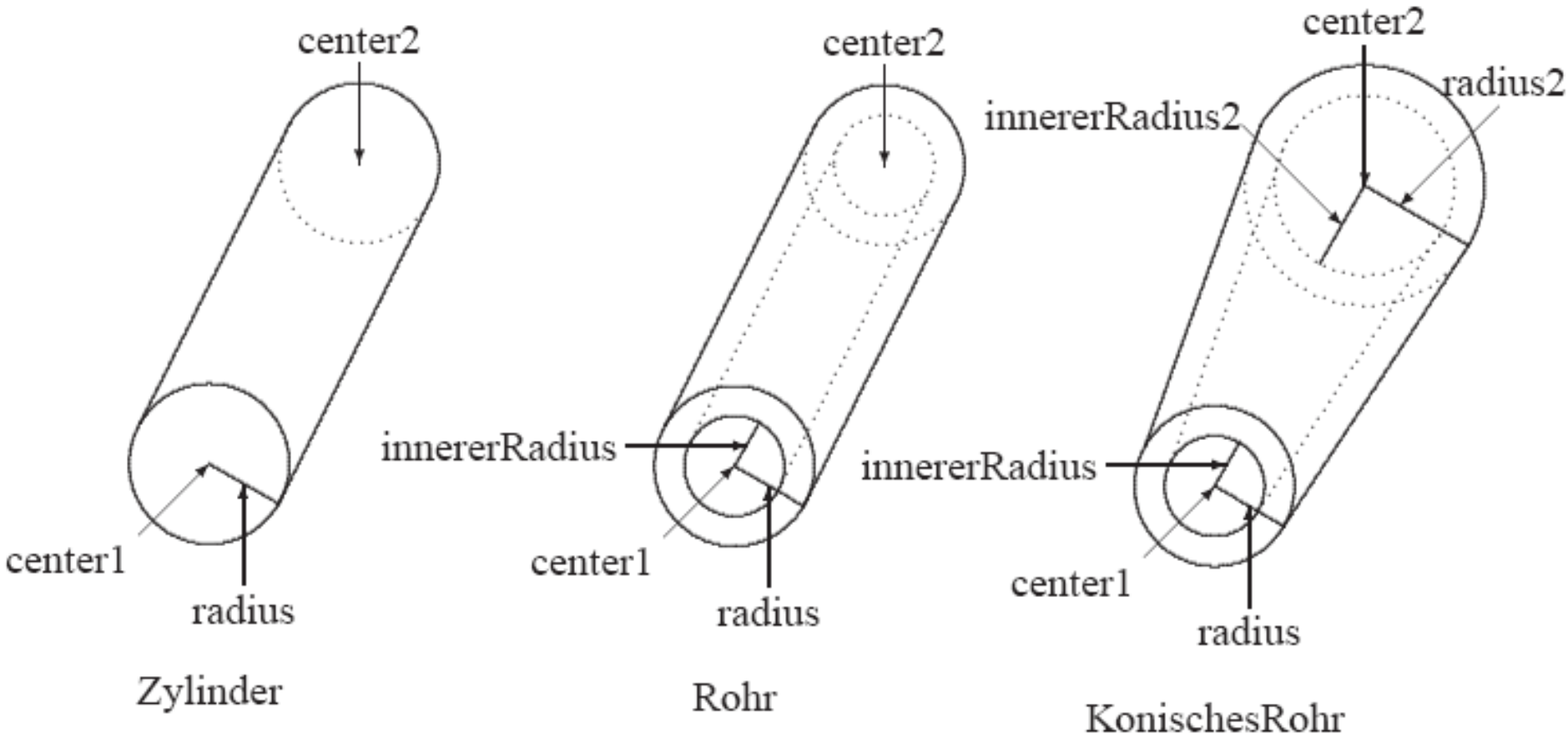
Wurzeltyp: Object

```
class OT {
   public ...;
}
```

```
class OT extends Object {
   public ...;
}
```


Object
...
Class1
...
...
...
Class2
...
...
Classn-1
Classn


Zylinder
+radius : double
+Zylinder(c1,c2,r)()
+laenge() : double
+volumen() : double
+toString() : String
Vertex
-x : double
-y : double
-z : double
+Vertex(xWert,yWert,zWert)()
+translatiere(anderer)()
+rotiere(winkel,achse)()
+skaliere(anderer)()
+imUrsprung() : Boolean
+distanz(anderer)() : double
+toString() : String
#center1
#center2
1
1
1
1
Rohr
+innererRadius : double
+Rohr(c1,c2,r,ir)()
+hohlKoerperVolumen() : double
+volumen() : double
+toString() : String
KonischesRohr
+innererRadius2 : double
+radius2 : double
+KonischesRohr(c1,c2,r,ir,r2,ir2)()
+hohlKoerperVolumen() : double
+volumen() : double
+toString() : String


center2
center1
radius
Zylinder
center2
innererRadius
center1
radius
Rohr
center2
radius2
innererRadius2
innererRadius
center1
radius
KonischesRohr

```
1  public class Zylinder extends Object {
2      public double radius;
3      protected Vertex center1;
4      protected Vertex center2;
5
6      public Zylinder(Vertex c1, Vertex c2, double r) {
7          this.center1 = c1;
8          this.center2 = c2;
9          this.radius = r;
10     }
11
12     public double laenge() {
13         return this.center1.distanz(this.center2);
14     }
15
16     public double volumen() {
17         return this.radius * this.radius * Math.PI * this.laenge();
18     }
19
20     public String toString() {
21         return "Radius:␣" + this.radius + "␣Center1:" + this.center1 +
22             "␣Center2:␣" + this.center2 + "␣Länge:␣" + this.laenge() +
23             "␣Volumen:␣" + this.volumen();
24     }
25
26     public static void main(String args[]) {
27         Zylinder c = new Zylinder(new Vertex(1,2,3), new Vertex(2,3,4), 5.5);
28         System.out.println(c);
29     }
30 }
```

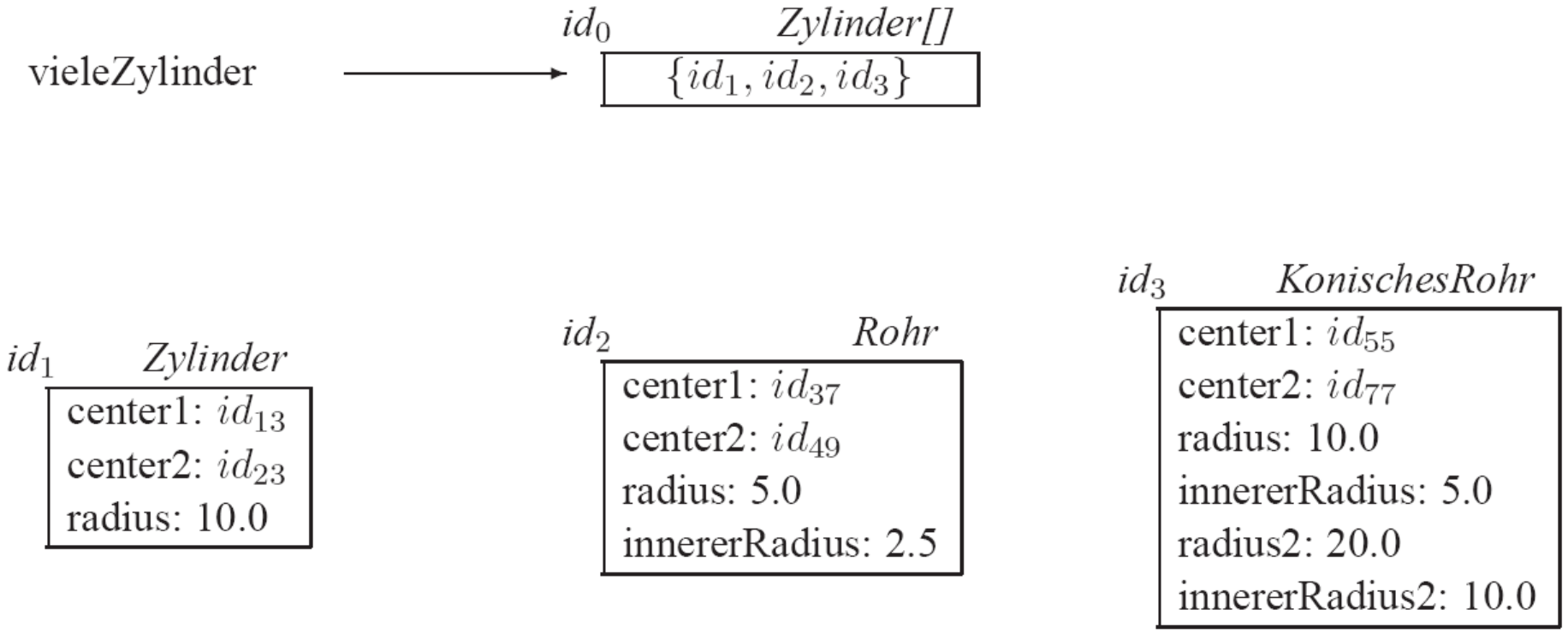
```
1  public class Rohr extends Zylinder {
2      public double innererRadius;
3
4      public Rohr(Vertex c1, Vertex c2, double r, double ir) {
5          super(c1, c2, r);
6          this.innererRadius = ir;
7      }
8
9      public double hohlKoerperVolumen() {
10         return this.innererRadius * this.innererRadius * Math.PI *
11                 this.laenge();
12     }
13
14     public double volumen() {
15         return super.volumen() - this.hohlKoerperVolumen();
16     }
17
18     public String toString() {
19         return super.toString() + "␣Hohlkörper-Volumen:␣" +
20                 this.hohlKoerperVolumen();
21     }
22
23     public static void main(String args[]) {
24         Rohr p = new Rohr(new Vertex(1,2,3), new Vertex(2,3,4), 7, 6);
25         System.out.println(p);
26     }
27 }
```

Verfeinerung / refinement

```
1  public class KonischesRohr extends Rohr {
2      public double innererRadius2;
3      public double radius2;
4
5      public KonischesRohr(Vertex c1, Vertex c2, double r,
6                           double ir, double r2, double ir2) {
7          super(c1, c2, r, ir);
8          radius2 = r2;
9          innererRadius2 = ir2;
10     }
11
12     public double hohlKoerperVolumen() {
13         return ((Math.PI * this.laenge() / 3) *
14                 (Math.pow(this.innererRadius, 2) +
15                  this.innererRadius * this.innererRadius2 +
16                  Math.pow(this.innererRadius2, 2)));
17     }
18
19     public double volumen() {  // Math.pow(x,n) berechnet x hoch n
20         return (((Math.PI * this.laenge() / 3) *
21                  (Math.pow(this.radius, 2) + this.radius * this.radius2 +
22                   Math.pow(this.radius2, 2))) - this.hohlKoerperVolumen());
23     }
24
25     public String toString() {
26         return super.toString() + "␣Hohlkörper-Volumen:␣" +
27                this.hohlKoerperVolumen();
28     }
29
30     public static void main(String args[]) {
31         KonischesRohr p = new KonischesRohr(new Vertex(1,2,3),
32                                             new Vertex(2,3,4), 7, 6, 5, 4);
33         System.out.println(p);
34     }
35 }
```

Verfeinerung / refinement nochmals

Dynamisches Binden



- Das Objekt mit der OID id_1 ist vom Typ *Zylinder*
- Das Objekt mit der OID id_2 ist vom Typ *Rohr*
- Das Objekt mit der OID id_3 ist vom Typ *KonischesRohr*

```
1  class DynBindingTest {
2      public static void main(String args[]) {
3          Zylinder cyl = new Zylinder(new Vertex(1,2,3),
4                                      new Vertex(2,3,4), 10.0);
5          Rohr p = new Rohr(new Vertex(1,2,3),
6                            new Vertex(2,3,4), 5.0, 2.5);
7          KonischesRohr cp = new KonischesRohr(new Vertex(1,2,3),
8                                               new Vertex(2,3,4),
9                                               10.0, 5.0, 20.0, 10.0);
10         Zylinder[] vieleZylinder = {cyl,p,cp};
11         double gesamtVolumen = 0.0;
12         Zylinder c;
13         for (int j = 0; j < vieleZylinder.length; j++) {
14             c = vieleZylinder[j];
15             gesamtVolumen = gesamtVolumen + c.volumen();
16           }
17           System.out.println("Gesamtvolumen: " + gesamtVolumen);
18       }
19   }
```

Substituierbarkeit: Eine Untertyp-Instanz kann eine Obertyp-Instanz substituieren – nicht umgekehrt

```
   Person einePerson;
   Angestellter einAngestellter;
   ...
(1)einePerson = einAngestellter;
(2)...
(3)einAngestellter = einePerson;          // nicht erlaubt!
```

$$\underbrace{\text{einAngestellter}}_{Angestellter} = \underbrace{\text{einePerson}}_{Person};$$

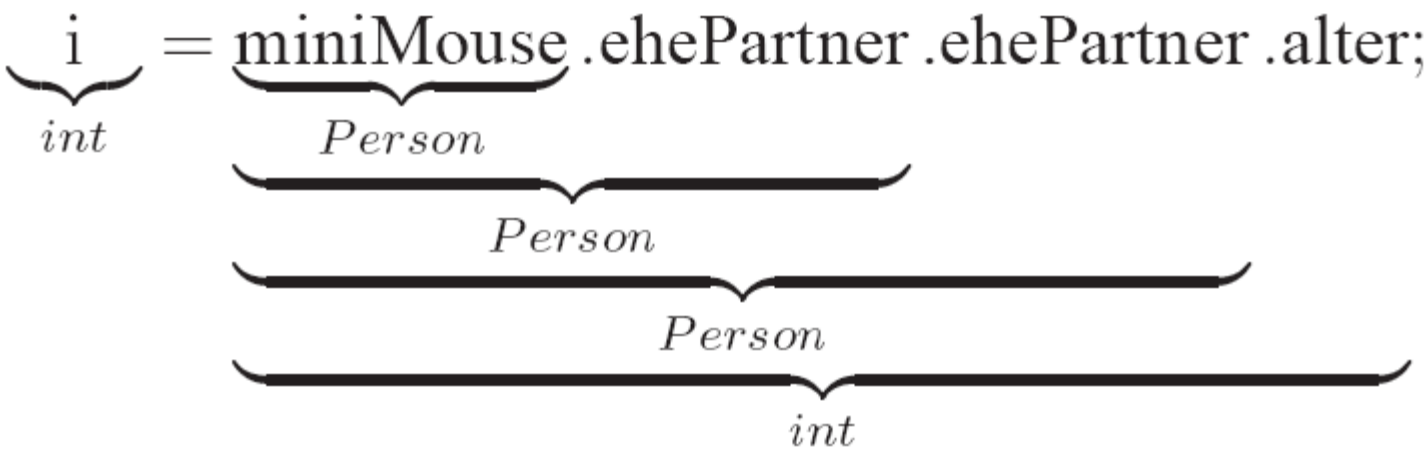
```
(1)einePerson.name;          // okay
(2)einePerson.gehalt;        // nicht erlaubt!
(3)einAngestellter.gehalt; // okay
```

Beispiele für die Typisierungsregeln

```
   Person miniMouse;
   Angestellter mickeyMouse;
   Angestellter chef;
   int i;
   ...
(1)mickeyMouse.ehePartner = miniMouse;          // okay
(2)miniMouse.ehePartner = mickeyMouse;          // okay
(3)mickeyMouse.boss = chef;                     // okay
(4)miniMouse.ehePartner.boss = chef;            // nicht erlaubt!
(5)i = mickeyMouse.boss.steuerNr;               // okay
(6)i = miniMouse.ehePartner.boss.steuerNr;      // nicht erlaubt!
(7)i = miniMouse.ehePartner.ehePartner.alter;   // okay
(8)i = mickeyMouse.ehePartner.boss.steuerNr;    // nicht erlaubt!
(9)mickeyMouse.boss.ehePartner.heiraten(chef);  // okay
```

$$\underbrace{\underbrace{\underbrace{\underbrace{\text{miniMouse}}_{Person}.\text{ehePartner}}_{Person}.\text{boss}}_{potential\ ERROR} = \underbrace{\text{chef}}_{Angestellter}\ ;$$

Weiteres Beispiel (7)



Abstrakte/Virtuelle Klassen & Schnittstellen/Interfaces

- Deklaration von Methoden ohne deren Implementierung schon durchzuführen
  - Interface:
    - nur Deklarationen
    - Eine Klasse kann mehrere Schnittstellen implementieren
  - Abstrakte Klasse:
    - Einige Methoden können schon implementiert werden
    - Andere nur deklariert
    - Virtuelle Klasse kann man nicht instanziieren

Abstrakte Klasse: GeoPrimitive

```
1  public abstract class GeoPrimitive {
2      public int geoID;
3      public Material mat;
4      public String farbe;
5
6      public double spezGewicht() {
7          return mat.spezGewicht;
8      }
9
10     public abstract String toString();
11
12     public abstract double volumen();
13
14     public double gewicht() {
15         return this.volumen() * this.spezGewicht();
16     }
17
18 } // end abstract class GeoPrimitive
```

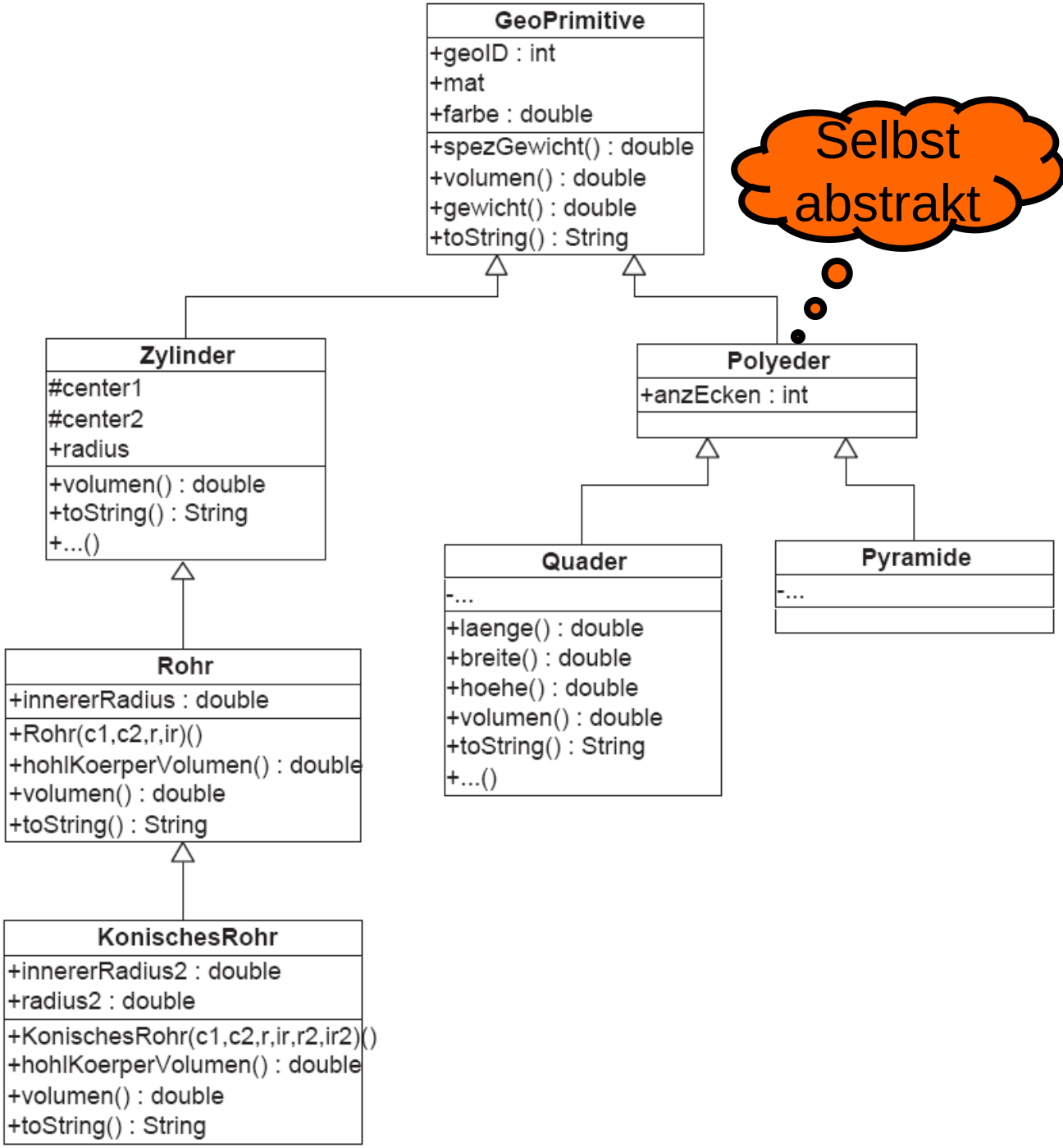
Erweiterung der abstrakten Klasse

```
class Zylinder extends GeoPrimitive {
    public double radius;
    protected Vertex center2;

    public Zylinder(Vertex c1, Vertex c2, double r) {
        center1 = c1;
        center2 = c2;
        radius = r;
    }

    public double laenge() {
        return center1.distanz(center2);
    }

    public double volumen() {
        return this.radius * this.radius * Math.PI * this.laenge();
    }

    public String toString() {
        return "Zylinder mit dem  Radius: " + this.radius +
               " Center1:" + this.center1 + " Center2: " +
               this.center2 + " Länge: " + this.laenge() +
               " Volumen: " + this.volumen();
    }
}
```

Nutzung der abstrakten Klasse

```
GeoPrimitive b;
double gesamtGewicht = 0.0;
double gesamtVolumen = 0.0;
for (int i = 0; i < basisTeile.length; i++) {
    b = basisTeile[i];
    gesamtGewicht = gesamtGewicht + b.gewicht();
    gesamtVolumen = gesamtVolumen + b.volumen();
}
```

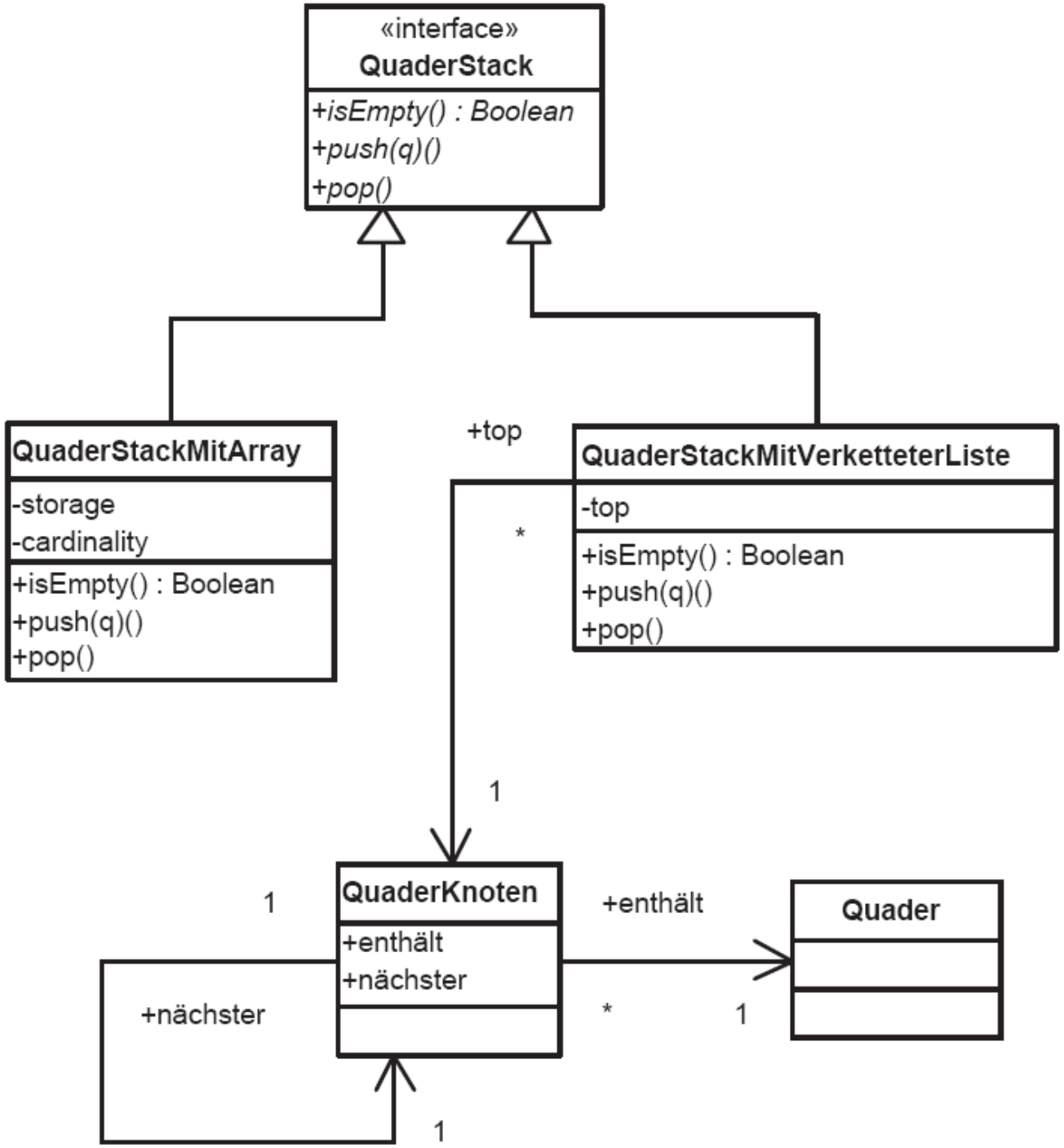
Hierarchie mit abstrakten Klassen



```
public class Quader extends Polyeder{
    private Vertex v1, v2, v3, v4, v5, v6, v7, v8;

    public Quader(Material m) {  // Konstruktor: Einheitswürfel
        this.v1 = new Vertex(0.0,0.0,0.0);
        this.v2 = new Vertex(1.0,0.0,0.0);
        this.v3 = new Vertex(1.0,1.0,0.0);
        this.v4 = new Vertex(0.0,1.0,0.0);
        this.v5 = new Vertex(0.0,0.0,1.0);
        this.v6 = new Vertex(1.0,0.0,1.0);
        this.v7 = new Vertex(1.0,1.0,1.0);
        this.v8 = new Vertex(0.0,1.0,1.0);
        this.mat = m; this.anzahlEcken = 8;
    }

    public double laenge() {
        return this.v1.distanz(this.v5);
    }
    public double breite() {
        return this.v1.distanz(this.v2);
    }
    public double hoehe() {
        return this.v1.distanz(this.v4);
    }
    public double volumen() {
        return this.laenge() * this.hoehe() * this.breite();
    }
    public double gewicht() {
        return this.volumen() * this.mat.spezGewicht;
    }

    public String toString() {
       return("Quader der Dimension " + this.laenge() + " X "
              + this.breite() + " X " + this.hoehe());
    }
```

Schnittstellen / Interfaces

```
interface QuaderStack {
    public boolean isEmpty();
    public void push(Quader c);
    public Quader pop();
}

class QuaderStackMitArray implements QuaderStack {
    private Quader[] storage;
    private int cardinality;
    ...
}

class QuaderStackMitVerketteteterListe implements QuaderStack {
    private QuaderKnoten top;  // class QuaderKnoten muss
    private int cardinality;   // als "Container" für einen Quader
        ...                    // und einem Attr. QuaderKnoten definiert sein
}
...
QuaderStack turmLinks, turmMitte, turmRechts;
turmLinks = new QuaderStackMitArray(...);
turmMitte = new QuaderStackMitVerketteteterListe(...);
turmRechts = new QuaderStackMitArray(...);
...
turmLinks.push(new Quader(...));
turmMitte.push(turmLinks.pop());
...
```


«interface»
QuaderStack
+isEmpty() : Boolean
+push(q)()
+pop()
QuaderStackMitArray
-storage
-cardinality
+isEmpty() : Boolean
+push(q)()
+pop()
+top
*
QuaderStackMitVerketteterListe
-top
+isEmpty() : Boolean
+push(q)()
+pop()
1
1
QuaderKnoten
+enthält
+nächster
+enthält
*
1
Quader
+nächster
1

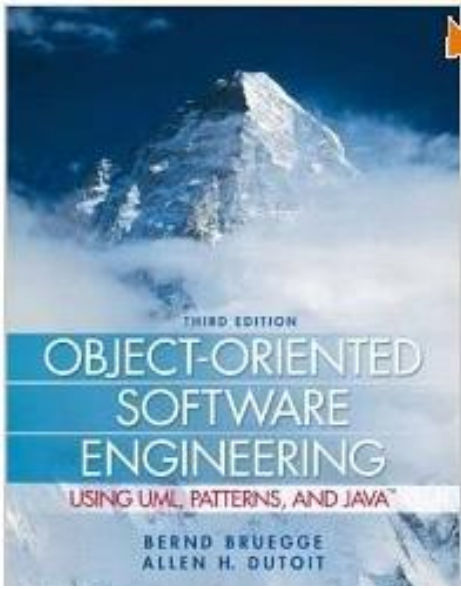
Typ-Anfragen und Type-Casting

```
Object preistraeger;    // sollte eine Person sein ... but who knows
Car jaguar = new Car("Jaguar",340,250);  // 340 PS, 250 km/h
double notenSchnittBonus = 0.9;   // default notenSchnitt-bonus
double gehaltBonus = 1.1;  // default gehalt-bonus
   // ...
        if (preistraeger instanceof Manager) {
            Manager m = (Manager)preistraeger;
            m.dienstWagen = jaguar;
            System.out.println("new car");
        }
        else if (preistraeger instanceof Angestellter) {
            Angestellter e = (Angestellter)preistraeger;
            e.gehalt = e.gehalt * gehaltBonus;
            System.out.println("Gehalt erhöht.");
        }
        else if (preistraeger instanceof Student) {
            Student s = (Student)preistraeger;
            s.notenSchnitt = s.notenSchnitt * notenSchnittBonus;
            System.out.println("NOTENSCHNITT upgrade.");
        }
        else if (preistraeger instanceof  Person)
            System.out.println("Großartig -- Danke!");
        else
            System.out.println("Deinen Typ kenne ich nicht!");
    }
```

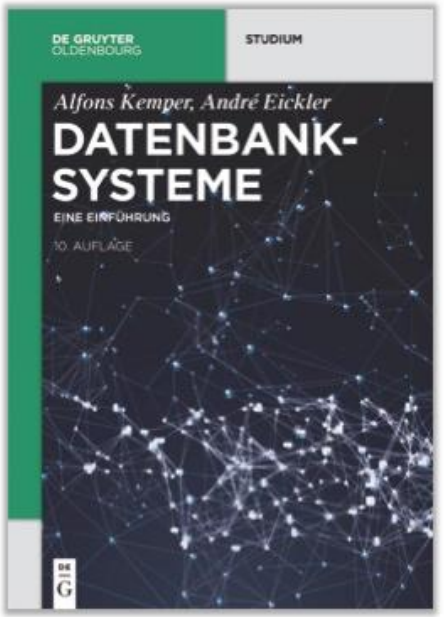
Einführung in die Informatik II für Ingenieurwissenschaften (MSE)

- Prof. Alfons Kemper, Ph.D.
- Christoph Anneser

- Teil 1:
  - Objektorientierte Modellierung (in UML) und
  - Programmierung in Java



- Teil 2:
  - Datenbanksysteme: Eine Einführung
  - Alfons Kemper und Andre Eickler
  - Oldenbourg Verlag, 10. Auflage, 2016



Kollektionen in Java
Aufzählungstypen, Generische Typen

- Wiederverwendbare Kollektionsklassen
  - Typparameter
  - Vordefinierte Kollektionen in der Java Collections Bibliothek
  - Insbesondere für die Modellierung von Assoziationen sinnvoll zu nutzen
  - Und für die Indexierung von Objekten
    - Schnelles Auffinden bei der Suche

Aufzählungstypen

- Notlösung:

```
1    public static final int MONTH_JAN =  1;
2    public static final int MONTH_FEB =  2;
3    ...
4    public static final int MONTH_NOV = 11;
5    public static final int MONTH_DEC = 12;
```

Aufzählungstypen

- besser:

```
1     public enum Month { JAN, FEB, MAR, APR, MAY, JUN,
2         JUL, AUG, SEP, OCT, NOV, DEC };
```

```
1  public class Enums {
2    public enum Month {
3      JAN  (31,  -2.2), FEB  (28,  -0.8), MAR  (31,   3.1),
4      APR  (30,   9.0), MAY  (31,  12.7), JUN  (30,  15.9),
5      JUL  (31,  20.1), AUG  (31,  17.1), SEP  (30,  15.4),
6      OCT  (31,   7.8), NOV  (30,   3.1), DEC  (31,  -0.8);
7
8      private final int     days;
9      private final double  avgTemperature;
10
11     Month(int days, double avgTemperature) {
12       this.days = days;
13       this.avgTemperature = avgTemperature;
14     }
15
16     public int    days()           { return days;  }
17     public double avgTemperature() { return avgTemperature; }
18
19     public double fractionOfYear() {
20       return days / 365.0;
21     }
22   };
23
24   public static void prettyPrint(Month m) {
25     System.out.println("Monat: " + m + ", Anzahl Tage: " + m.days() +
26                        ", Anteil am Jahr: " +  m.fractionOfYear());
27   }
28
29   public static void main(String[] args) {
30     prettyPrint(Month.JAN);
31
32     for (Month m : EnumSet.range(Month.FEB, Month.MAY))
33       System.out.println(m);
34   }
35 }
```

```
1     Month m = Month.JAN;
2
3     switch (m) {
4       case SEP:
5       case OCT:
6       case NOV:
7       case DEC:
8       case JAN:
9       case FEB:
10      case MAR:
11      case APR:
12        System.out.println("kalt");
13        break;
14      case MAY:
15      case JUN:
16      case JUL:
17      case AUG:
18        System.out.println("warm");
19        break;
20    }
```

Generische Klassen: Motivation

```
public class QuaderStack {
  private Quader[] elements;
  private int cardinality = 0;

  public QuaderStack(int capacity) {
    elements = new Quader[capacity];
  }
  public void push(Quader e) {
    elements[cardinality++] = e;
  }
  public Quader pop() {
    return elements[--cardinality];
  }
  public boolean isEmpty() {
    return (cardinality == 0);
  }
}
```

```
1  public class ZylinderStack {
2    private Zylinder[] elements;
3    private int cardinality = 0;
4
5    public ZylinderStack(int capacity) {
6      elements = new Zylinder[capacity];
7    }
8    public void push(Zylinder e) {
9      elements[cardinality++] = e;
10   }
11   public Zylinder pop() {
12     return elements[--cardinality];
13   }
14   public boolean isEmpty() {
15     return (cardinality == 0);
16   }
17 }
```

Generisch ... aber nicht typsicher

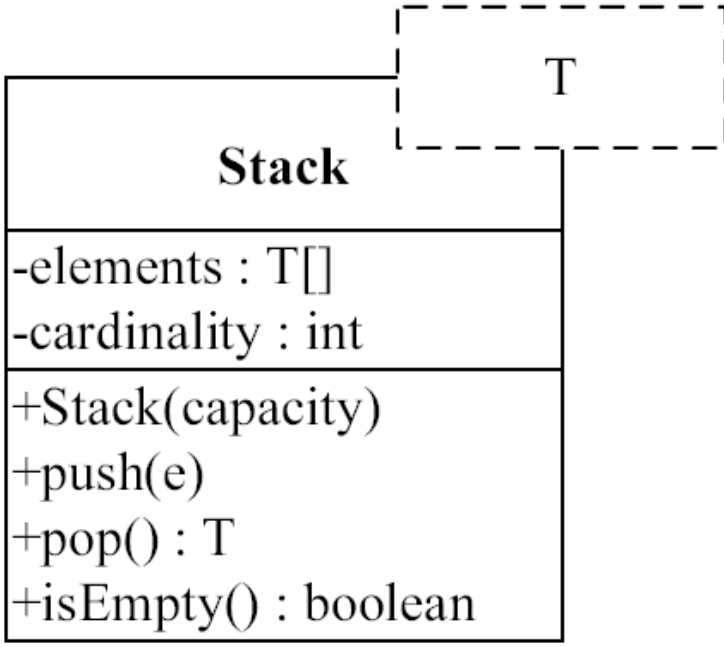
```
1  public class Stack {
2    private Object[] elements;
3    private int cardinality = 0;
4
5    public Stack(int capacity) {
6      elements = new Object[capacity];
7    }
8    public void push(Object e) {
9      elements[cardinality++] = e;
10   }
11   public Object pop() {
12     return elements[--cardinality];
13   }
14   public boolean isEmpty() {
15     return (cardinality == 0);
16   }
17 }
```

Nutzung ... durch type casting

```
1  // Quader q anlegen
2  // ...
3  QuaderStack myQuaderStack = new QuaderStack(5);
4  myQuaderStack.push(q);
5  Quader q2 = myQuaderStack.pop();
6
7  Stack myQuaderStack2 = new Stack(5);
8  myQuaderStack2.push("Dies␣führt␣später␣zu␣einem␣Fehler!")
9  myQuaderStack2.push(q);
10 Quader q3 = (Quader)myQuaderStack2.pop();
11 Quader q4 = (Quader)myQuaderStack2.pop();
```

Generische Typen in Java und UML

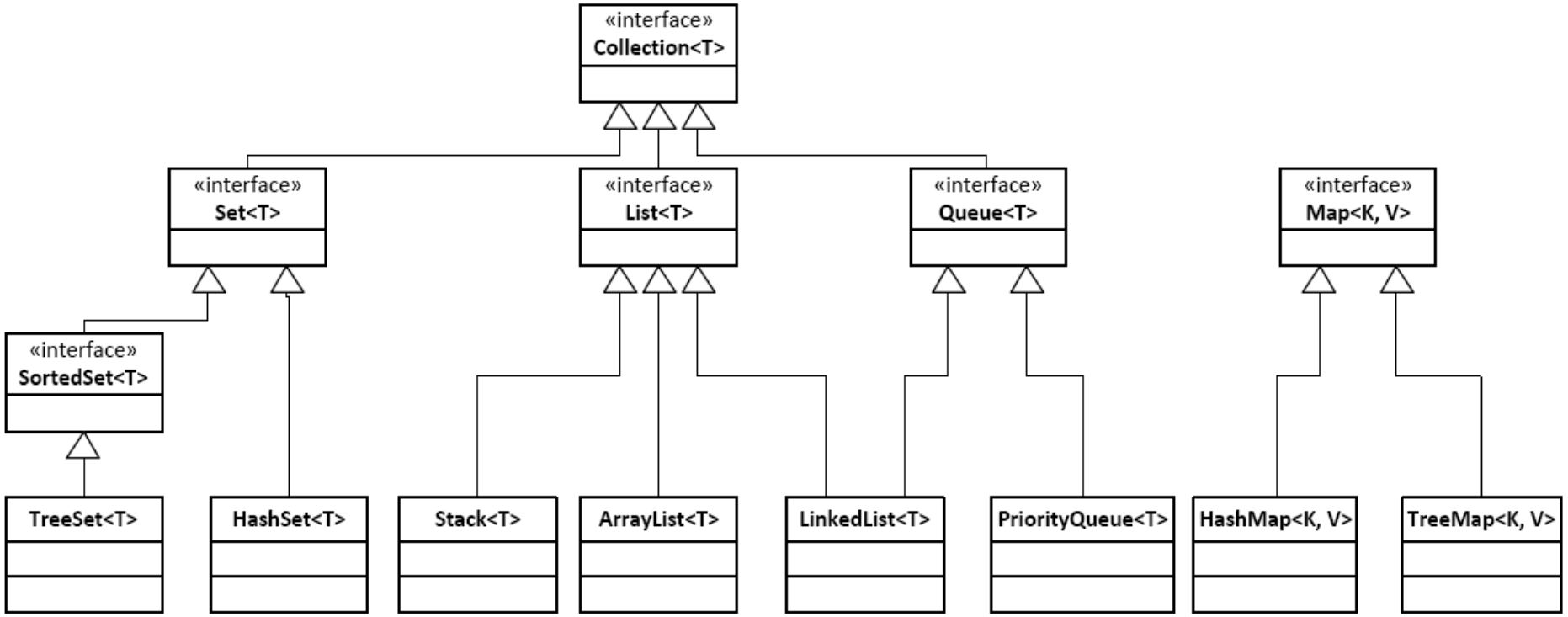
```
1  public class Stack<T> {
2    private T[] elements;
3    private int cardinality = 0;
4
5    public Stack(int capacity) {
6      elements = (T[])new Object[capacity];
7    }
8    public void push(T e) {
9      elements[cardinality++] = e;
10   }
11   public T pop() {
12     return elements[--cardinality];
13   }
14   public boolean isEmpty() {
15     return (cardinality == 0);
16   }
17 }
```



Nutzung

```
1  // Quader q anlegen
2  // ...
3  Stack<Quader> myQuaderStack = new Stack<Quader>(5);
4  myQuaderStack.push(q);
5  // myQuaderStack.push("Dies würde der Compiler beim
6  Quader q2 = myQuaderStack.pop();
```

Das Java Collection Framework



```
class PhoneBook {
        // Map for the standard lookup
        TreeMap<String, Integer> nameToNumber;
        // Map for the reverse lookup
        HashMap<Integer, String> numberToName;

        // Constructor
        public PhoneBook() {
                nameToNumber = new TreeMap<String, Integer>();
                numberToName = new HashMap<Integer, String>();
        }

        // Add an entry to the phone book
        void addEntry(String name, Integer phoneNumber) {
                nameToNumber.put(name, phoneNumber);    // O(log(n))
                numberToName.put(phoneNumber, name);    // O(1)
        }

        // Standard lookup: get the phone number for a name
        Integer lookup(String name) {
                return nameToNumber.get(name);          // O(log(n))
        }

        // Reverse lookup: get the name for a phone number
        String reverseLookup(Integer phoneNumber) {
                return numberToName.get(phoneNumber);   // O(1)
        }

        // Get all entries of the phone book whose names lie in the given range
        Set<Map.Entry<String, Integer>> rangeLookup(String from, String to) {
                return nameToNumber.subMap(from, to).entrySet();
        }
```

Nutzungsbeispiele

```
// Executable main method
public static void main(String[] args) {
        // Create the phone book
        PhoneBook phoneBook = new PhoneBook();
        phoneBook.addEntry("Maus, Micky", 4711);
        phoneBook.addEntry("Duck, Donald", 1234);
        phoneBook.addEntry("Maus, Minni", 1704);
        phoneBook.addEntry("Kolumbus, Christoph", 1492);
        // Lookup

        println("Donald's number: " + phoneBook.lookup("Duck, Donald"));
        println("1492 belongs to " + phoneBook.reverseLookup(1492));
        for (Map.Entry<String, Integer> entry
             : phoneBook.rangeLookup("Maier", "Meier")) {
                println(entry.getKey() + ": " + entry.getValue());
        }
   }
```

Nutzung für die Modellierung von Assoziationen

```
1  import java.util.Set;
2  import java.util.HashSet;
3
4  public class Student {
5    public String name;
6    public Set<Vorlesung> vorlesungen;
7    // ...
8    vorlesungen = new HashSet<Vorlesung>();
9    if (!vorlesungen.contains(grundzuege)) {
10     vorlesungen.add(grundzuege);
11   }
```

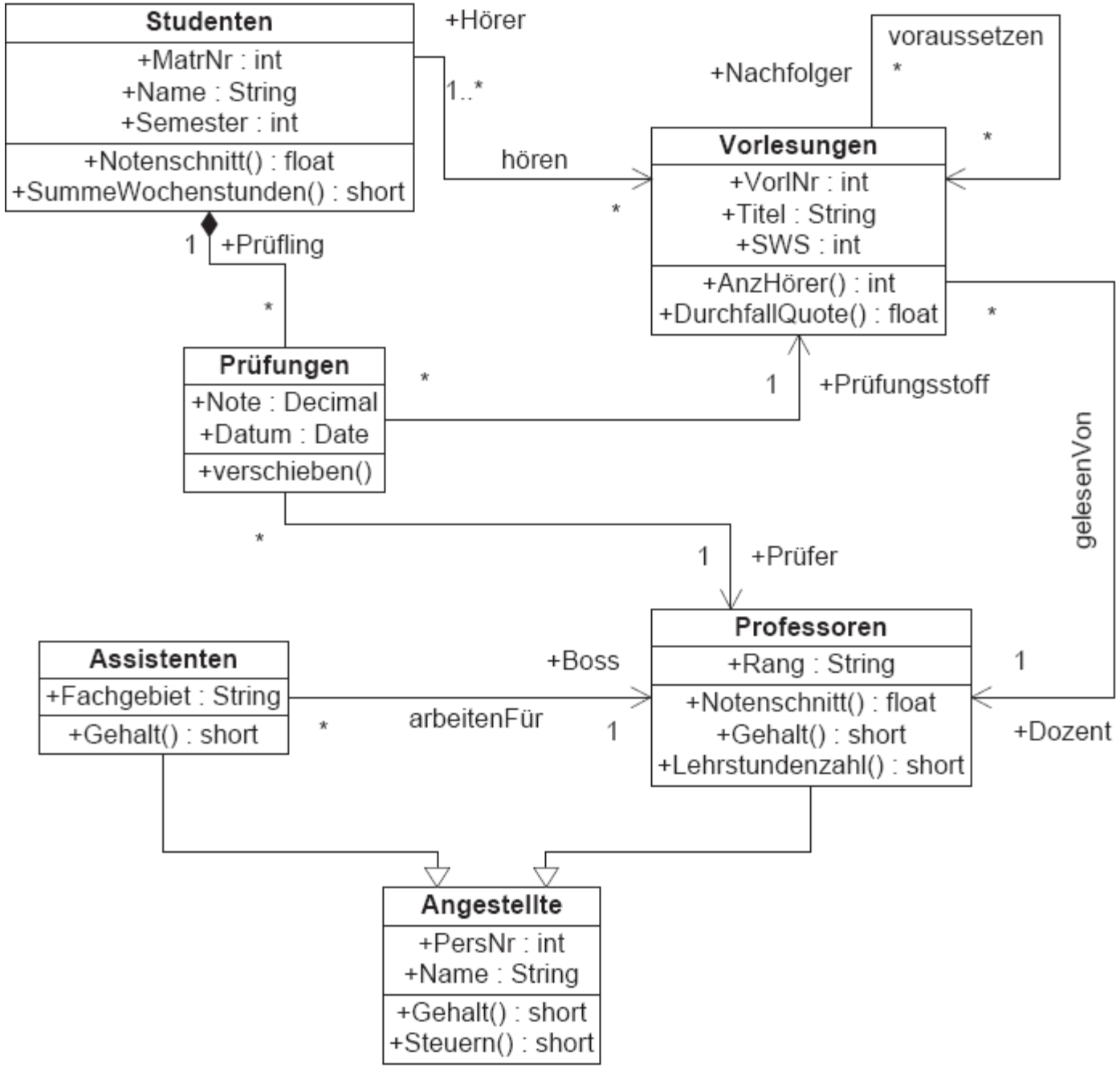
Wrapper für Sorten/Werte

```
1  Stack<Integer> myIntegerStack = new Stack<Integer>(10);
2  int fortysevenelleven = 4711;
3  myIntegerStack.push(new Integer(fortysevenelleven)); // Einp
4  Integer surprise = myIntegerStack.pop();             // Imme
5  int unwrapped = surprise.intValue();                 // Ausp
```

Auto-Boxing

```
Stack<Integer> myIntegerStack = new Stack<Integer>(10);
int fortyseveneleven = 4711;
myIntegerStack.push(fortyseveneleven); // Automatisches
int myInt = myIntegerStack.pop();      // Automatisches
```

Modellierungs-Beispiel: Universität



```
1  import java.util.Set;
2  import java.util.HashSet;
3
4  public class Student {
5    public int matrNr;
6    public String name;
7    public int semester;
8    public Set<Vorlesung> vorlesungen;
9    public Set<Pruefung> pruefungen;
10
11   public Student(int matrNr, String name, int semester) {
12     this.matrNr = matrNr;
13     this.name = name;
14     this.semester = semester;
15
16      vorlesungen = new HashSet<Vorlesung>();
17     pruefungen = new HashSet<Pruefung>();
18   }
19
20   public void belegeVorlesung(Vorlesung vorlesung) {
21     vorlesung.erhoeheAnzahlHoerer();
22     vorlesungen.add(vorlesung);
23   }
24
25   public void pruefen(Pruefung pruefung) {
26     pruefungen.add(pruefung);
```

```
20    public void belegeVorlesung(Vorlesung vorlesung) {
21      vorlesung.erhoeheAnzahlHoerer();
22      vorlesungen.add(vorlesung);
23    }
24
25    public void pruefen(Pruefung pruefung) {
26      pruefungen.add(pruefung);
27    }
28
29    public float notenschnitt() {
30      float durchschnittsNote = 0;
31      for (Pruefung p : pruefungen) {
32        durchschnittsNote += p.note/pruefungen.size();
33      }
34      return durchschnittsNote;
35    }
36
37    public short summeWochenstunden() {
38      short summeSWS = 0;
39      for (Vorlesung v : vorlesungen) {
40        summeSWS += v.sws;
41      }
42      return summeSWS;
43    }
44  }
```

```
1  import java.util.Calendar;
2
3  public class Pruefung {
4    public double note;
5    public Calendar datum;
6    public Student pruefling;
7    public Vorlesung pruefungsstoff;
8    public Professor pruefer;
9
10   private boolean bewertet;
11
12   public Pruefung(Student student, Vorlesung vorlesung, Professor professor,
13                   Calendar termin) {
14     this.pruefling = student;
15     this.pruefungsstoff = vorlesung;
16       this.pruefer = professor;
17       this.datum = termin;
18       bewertet = false;
19     }
20
21     public void bewerten(double note) {
22       if (!bewertet) {
23         bewertet = true;
24         this.note = note;
25         pruefling.pruefen(this);
26         pruefungsstoff.pruefen(this);
27         pruefer.pruefen(this);
28       }
29     }
```

```
1  import java.util.Set;
2  import java.util.HashSet;
3
4  public class Vorlesung {
5    public int vorlNr;
6    public String titel;
7    public int sws;
8    public Professor dozent;
9    public Set<Vorlesung> voraussetzungen;
10
11   private int anzahlHoerer;
12   private int anzahlPruefungen;
13   private int anzahlDurchgefallen;
14
15   public Vorlesung(int vorlNr, String titel, int sws, Professor dozent) {
16     this.vorlNr = vorlNr;
17     this.titel = titel;
18     this.sws = sws;
19     this.dozent = dozent;
20
21     voraussetzungen = new HashSet<Vorlesung>();
22
23     dozent.leseVorlesung(this);
24   }
25
```

```
26    public void pruefen(Pruefung pruefung) {
27      if (pruefung.note > 4.0) {
28        anzahlDurchgefallen++;
29      }
30      anzahlPruefungen++;
31    }
32
33    public void erhoeheAnzahlHoerer() {
34      anzahlHoerer++;
35    }
36
37    public int anzahlHoerer() {
38      return anzahlHoerer;
39    }
40
41    public float durchfallQuote() {
42      return (float)anzahlDurchgefallen/anzahlPruefungen;
43    }
44  }
```

```
16      this.pruefer = professor;
17      this.datum = termin;
18      bewertet = false;
19    }
20
21    public void bewerten(double note) {
22      if (!bewertet) {
23        bewertet = true;
24        this.note = note;
25        pruefling.pruefen(this);
26        pruefungsstoff.pruefen(this);
27        pruefer.pruefen(this);
28      }
29    }
30
31    public void verschieben(Calendar neuesDatum) {
32      if (datum.compareTo(Calendar.getInstance()) > 0) {
33        datum = neuesDatum;
34      }
35    }
36  }
```

```
1  public class Angestellter {
2    public int persNr;
3    public String name;
4
5    public Angestellter(int persNr, String name) {
6      this.persNr = persNr;
7      this.name = name;
8    }
9
10   public int gehalt() {
11     return 2000;
12   }
13
14   public int steuern() {
15     return gehalt()/2;
16   }
17 }
```

```
1  public class Professor extends Angestellter {
2    public enum Rang {
3      C1, C2, C3, C4
4    }
5
6    public Rang rang;
7
8    private short lehrstunden;
9    private int notenAnzahl;
10   private int notenSumme;
11
12   public Professor(int persNr, String name, Rang rang) {
13     super(persNr, name);
14     this.rang = rang;
15   }
16
17   public void leseVorlesung(Vorlesung vorlesung) {
18     lehrstunden += vorlesung.sws;
19   }
20
21   public short lehrstundenzahl() {
22     return lehrstunden;
23   }
24
25   public void pruefen(Pruefung pruefung) {
26     notenAnzahl++;
27     notenSumme += pruefung.note;
28   }
29
30   public float notenschnitt() {
31     return (float)notenSumme/notenAnzahl;
32   }
33
34   public int gehalt() {
```

```
35        int gehalt;
36        switch (rang) {
37          case C1: gehalt = 3000;
38          break;
39          case C2: gehalt = 3200;
40          break;
41          case C3: gehalt = 3400;
42          break;
43          case C4: gehalt = 3600;
44          break;
45          default: gehalt = 3000;
46          break;
47        }
48        return gehalt;
49      }
50    }
```

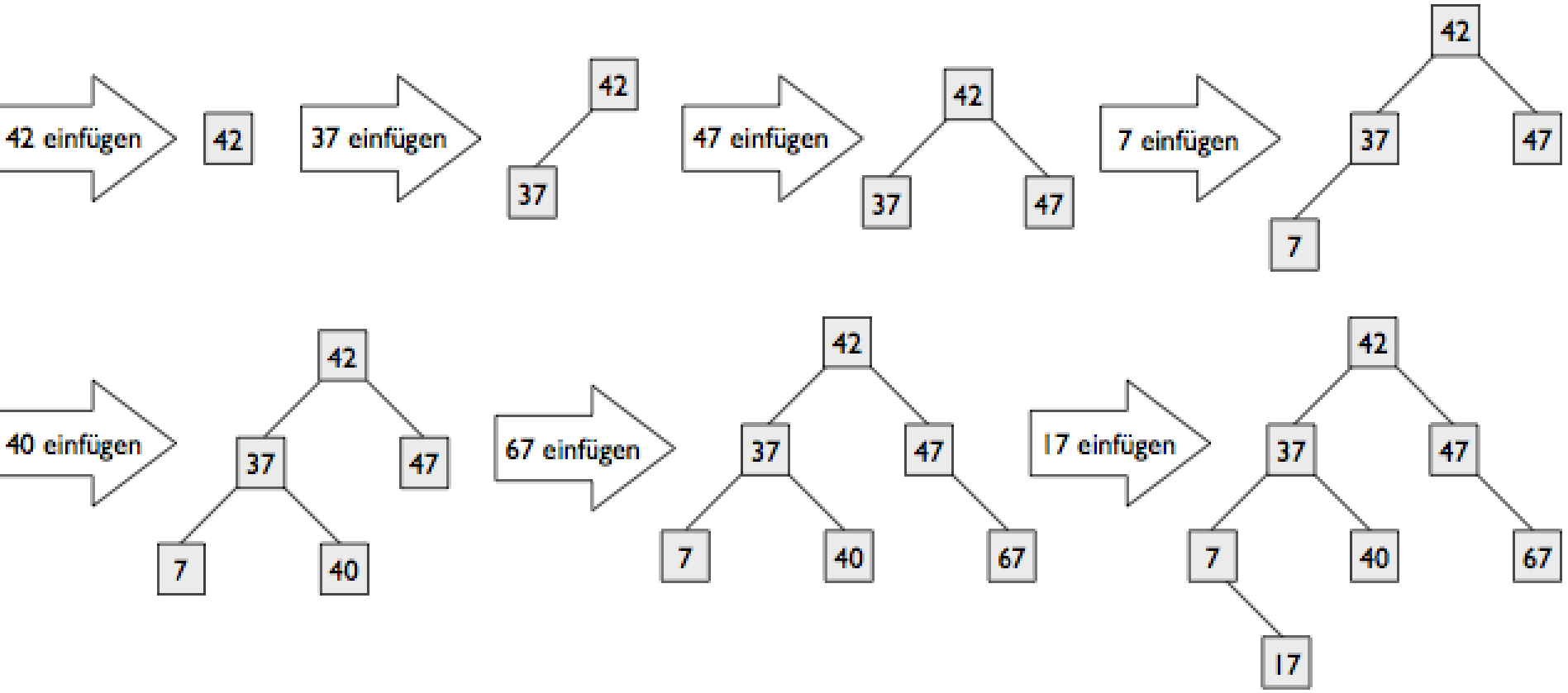
```
1  public class Assistent extends Angestellter {
2    public String fachgebiet;
3   public Professor boss;
4
5   public Assistent(int persNr, String name, String fachgebiet, Professor boss) {
6     super(persNr, name);
7     this.fachgebiet = fachgebiet;
8     this.boss = boss;
9   }
10
11  public int gehalt() {
12    return 2500;
13  }
14 }
```

Datenstrukturen für Kollektionen: Suchbäume und Hashing

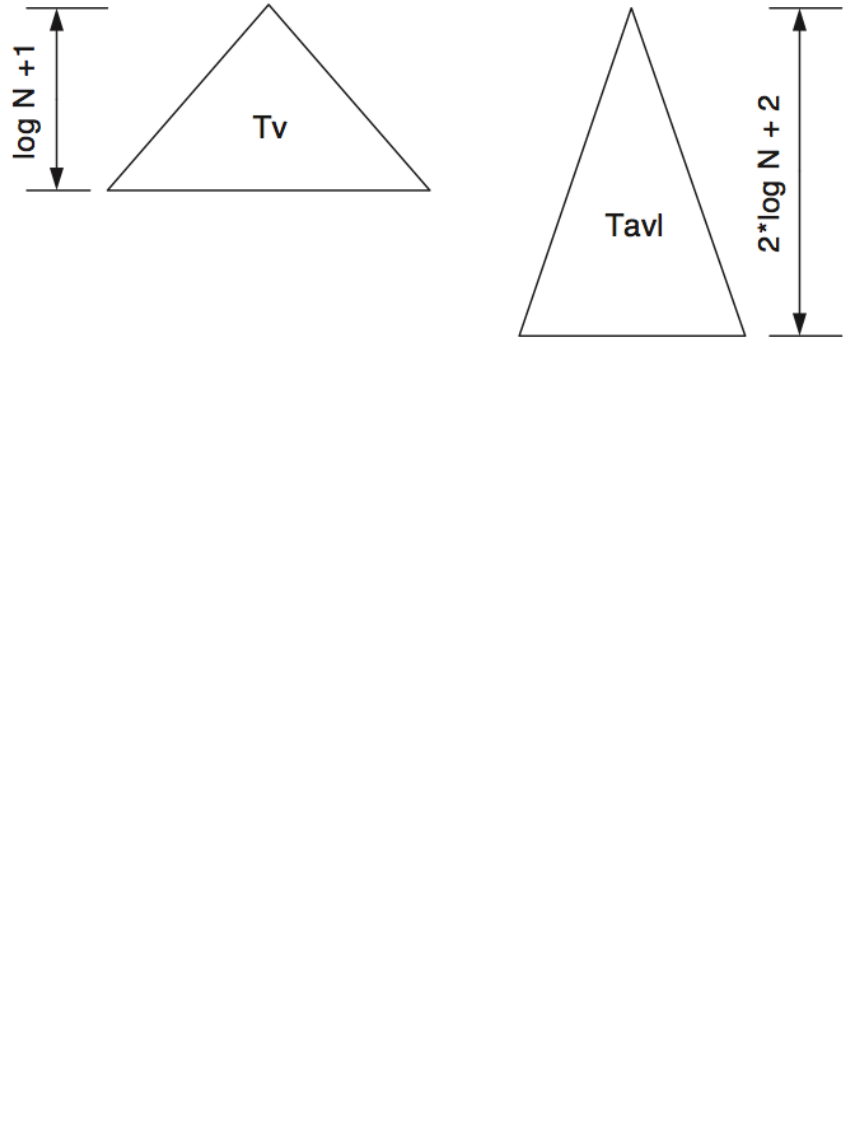
- Suchbäume haben logarithmische Höhe
  - Suche kostet dann O(log N)
    - N Elemente im Suchbaum
    - Bei 10.000.000.000 Einträge nicht zu vernachlässigen
  - Unterstützt auch Bereichsanfragen
  - TreeSet und TreeMap

- Hashing ist unabhängig von der Anzahl der Elemente
  - O(1) Suchkosten
  - Egal ob 200 oder 10.000.000.000 Einträge indexiert werden
  - Aber nur Punktanfragen (exact match)
  - HashSet und HashSet

Binäre Suchbäume



Problem: Degenerierter Suchbaum
Lösung: balancierter AVL-Baum



AVL-Baum: Balancierung während des einfügens

- Höhe des des linken Teilbaums unterscheidet sich von der Höhe des rechten Teilbaums um maximal 1

7.3 AVL-Bäume: Balancierte binäre Suchbäume

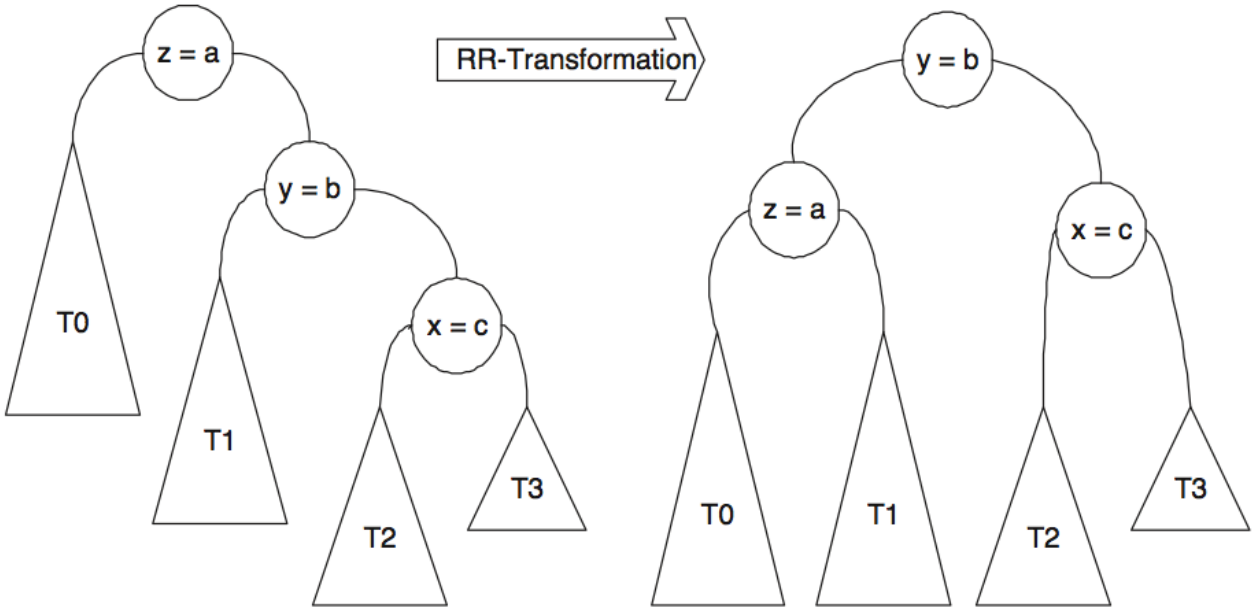
Die Höhe eines Baums mit der Wurzel V ist definiert als $h(v) = 1 + \max(h(T_l), h(T_r))$, wobei T_l und T_r das linke bzw. das rechte Kind (bzw. die Teilbäume) von v sind. Die Höhe eines leeren Teilbaums ist 0.

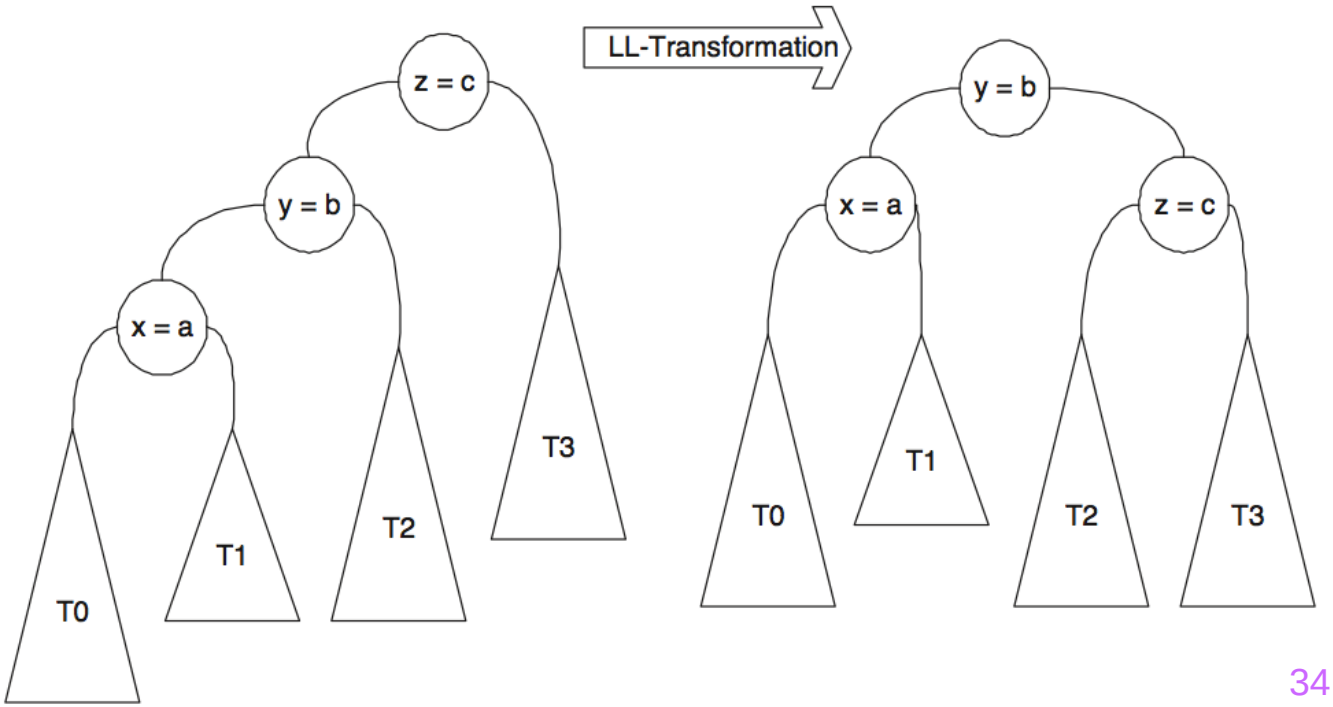
Ein binärer Suchbaum erfüllt die AVL-Eigenschaften, wenn für jeden Knten v des Baums gilt:

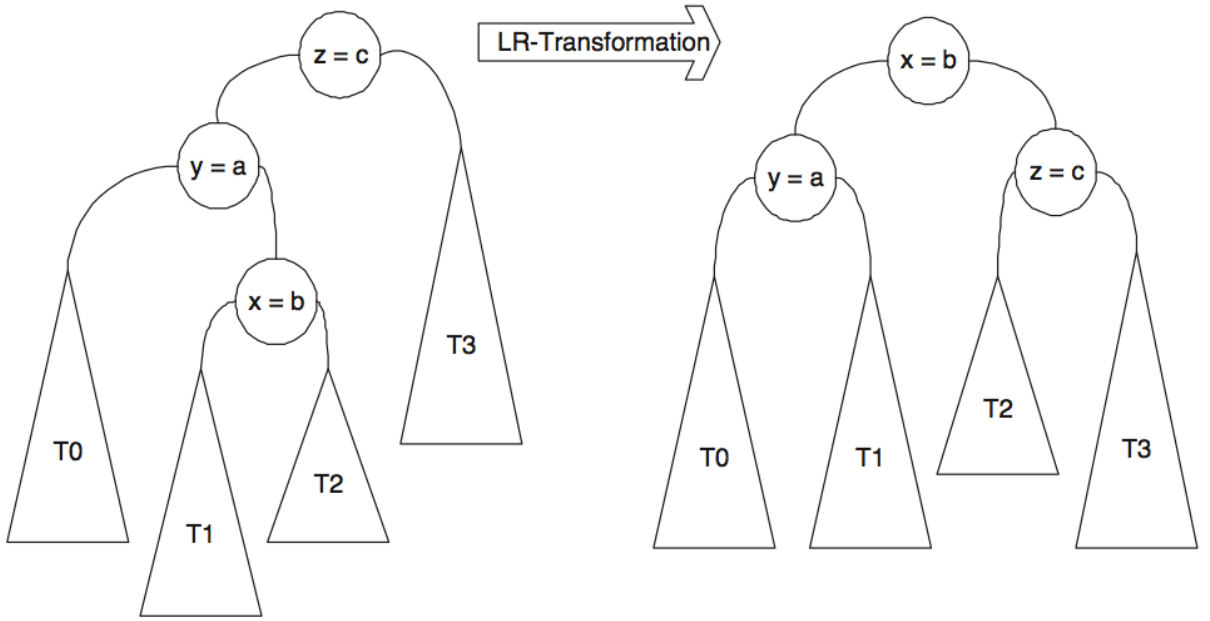
$$|h_l - h_r| \leq 1$$

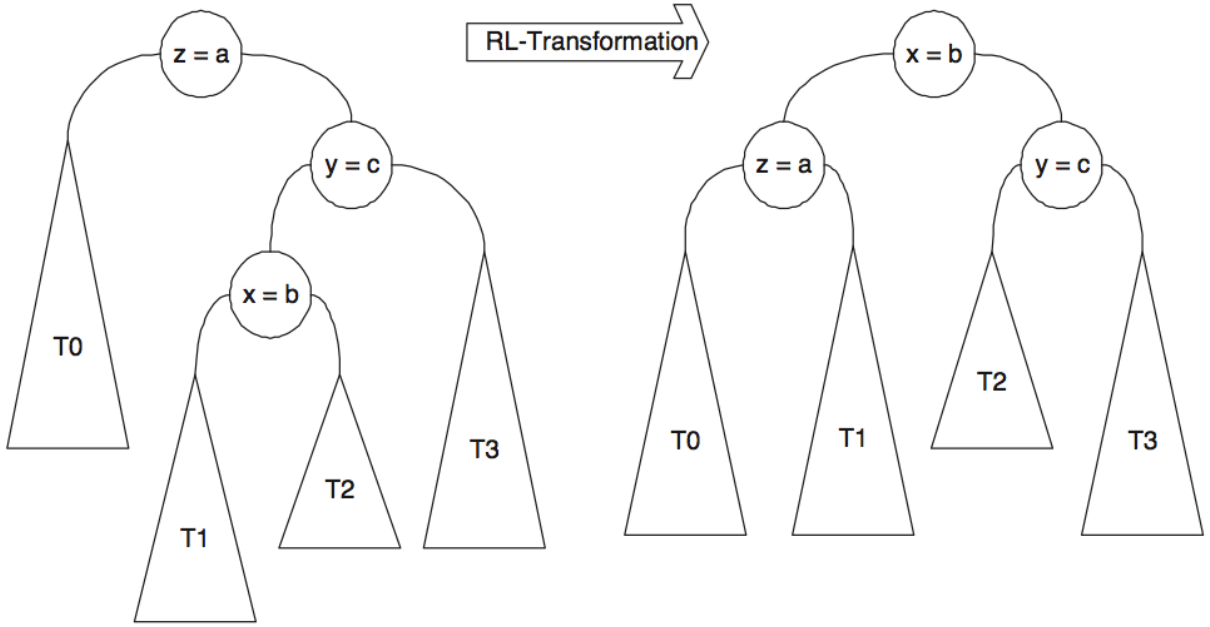
Der AVL-Baum ist benannt nach Adel'son-Vel'skii und Landis.

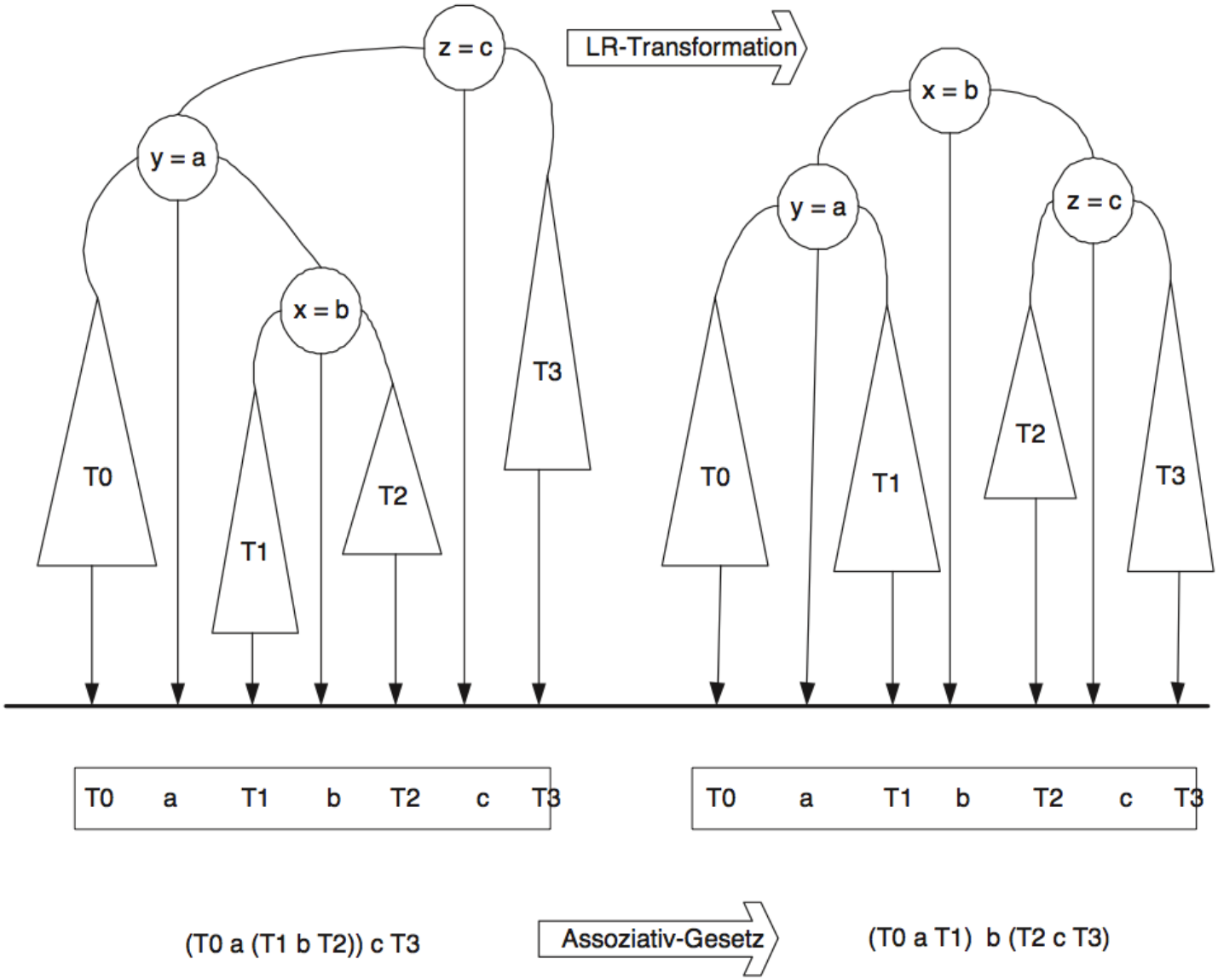
Den Wert $h_l - h_r$ nennt man den Balance-Faktor des Knotens. Gültige Werte für diesen Balance-Faktor sind $-1, 0, 1$. Wenn beim Einfügen oder Löschen eine "Unbalanciertheit" auftritt, muß diese durch entsprechende Transformationen revidiert werden.


z = a
y = b
x = c
T0
T1
T2
T3
RR-Transformation
y = b
z = a
x = c
T0
T1
T2
T3

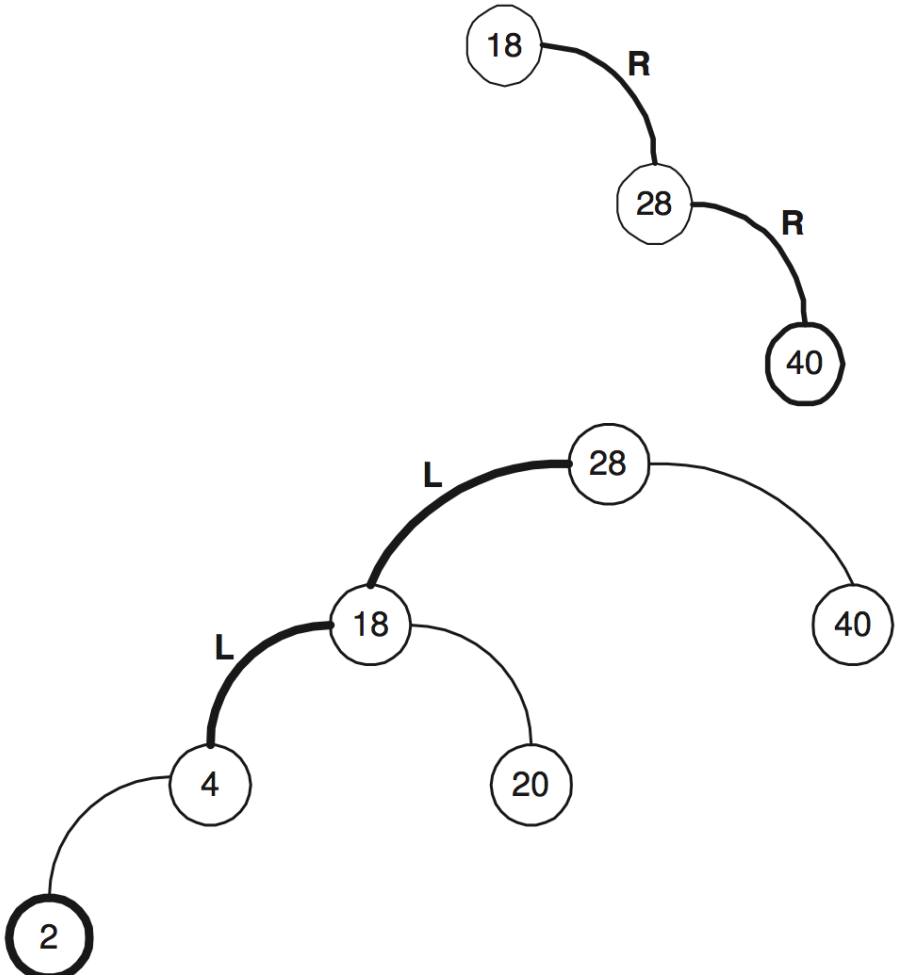

z = c
y = b
x = a
T0
T1
T2
T3
LL-Transformation
y = b
x = a
z = c
T0
T1
T2
T3

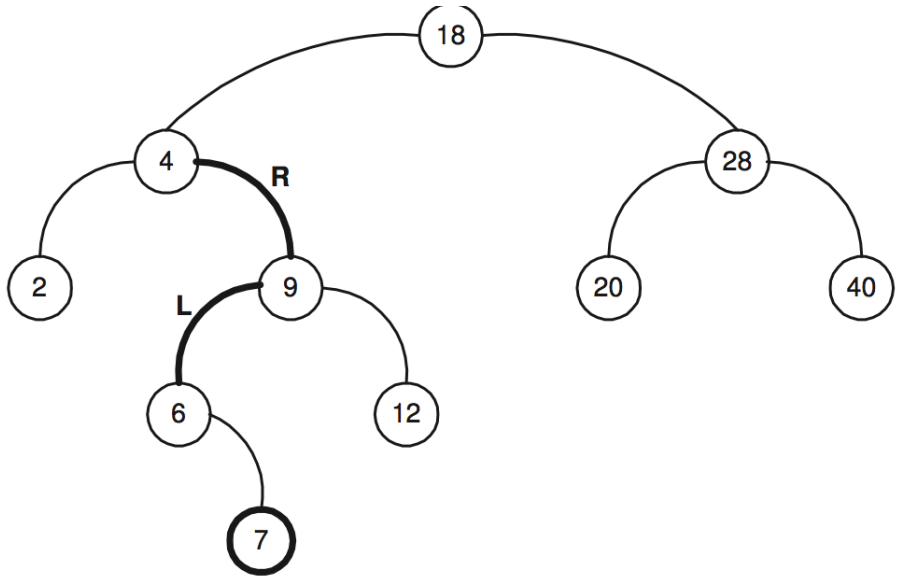

z = c
y = a
x = b
T0
T1
T2
T3
LR-Transformation
x = b
y = a
z = c
T0
T1
T2
T3

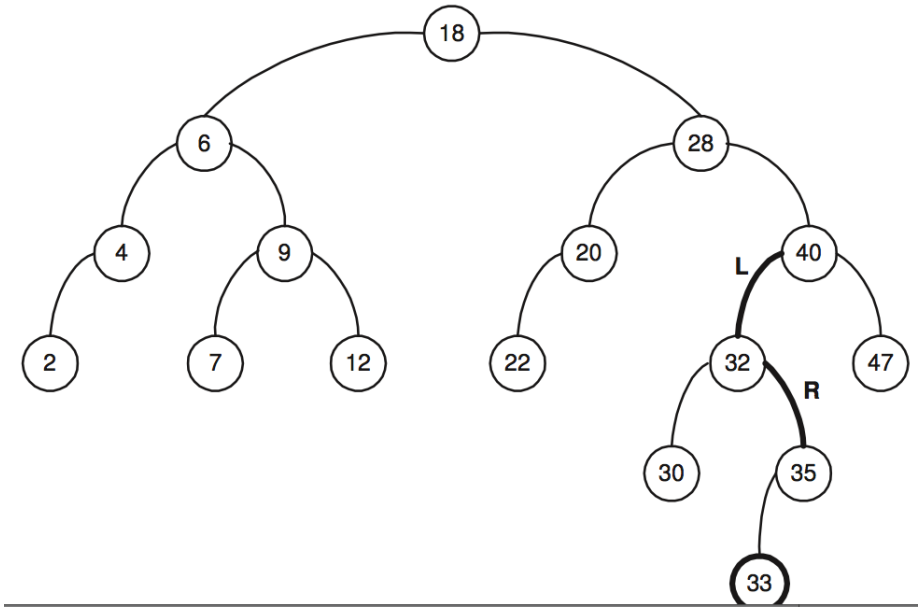

z = a
y = c
x = b
T0
T1
T2
T3
RL-Transformation
x = b
z = a
y = c
T0
T1
T2
T3

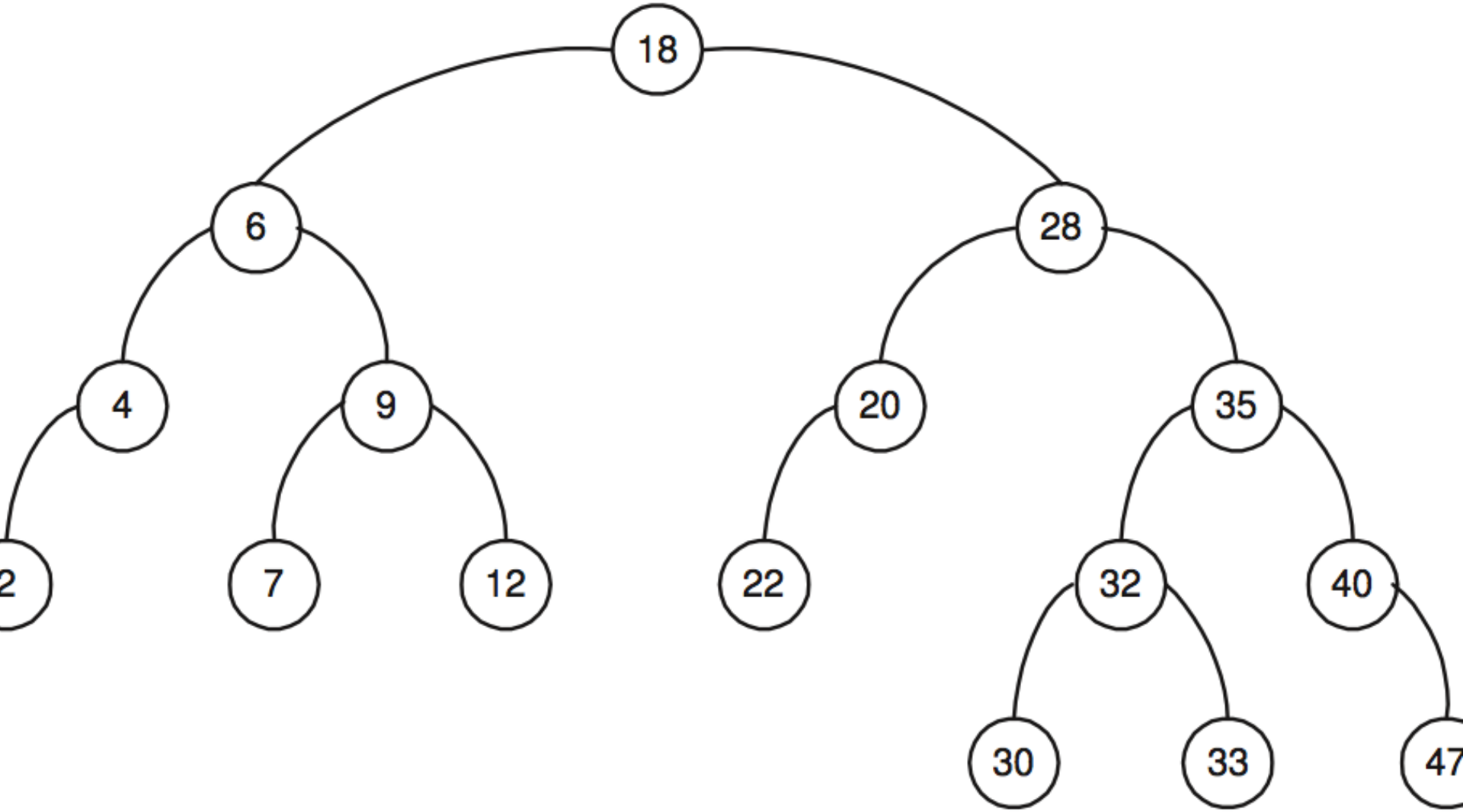

z = c
LR-Transformation
y = a
x = b
T0
T1
T2
T3
x = b
y = a
z = c
T0
T1
T2
T3
T0 a T1 b T2 c T3
T0 a T1 b T2 c T3
(T0 a (T1 b T2)) c T3
Assoziativ-Gesetz
(T0 a T1) b (T2 c T3)

Einfügen: 18, 28, 40, 20, 4, 2, 9, 6, 12, 7, 22, 32, 47, 30, 35, 33




18
4
R
28
2
9
20
40
L
6
12
7


18
6
28
4
9
20
40
L
2
7
12
22
32
47
R
30
35
33


18
6
28
4
9
20
35
2
7
12
22
32
40
30
33
47

Hashing

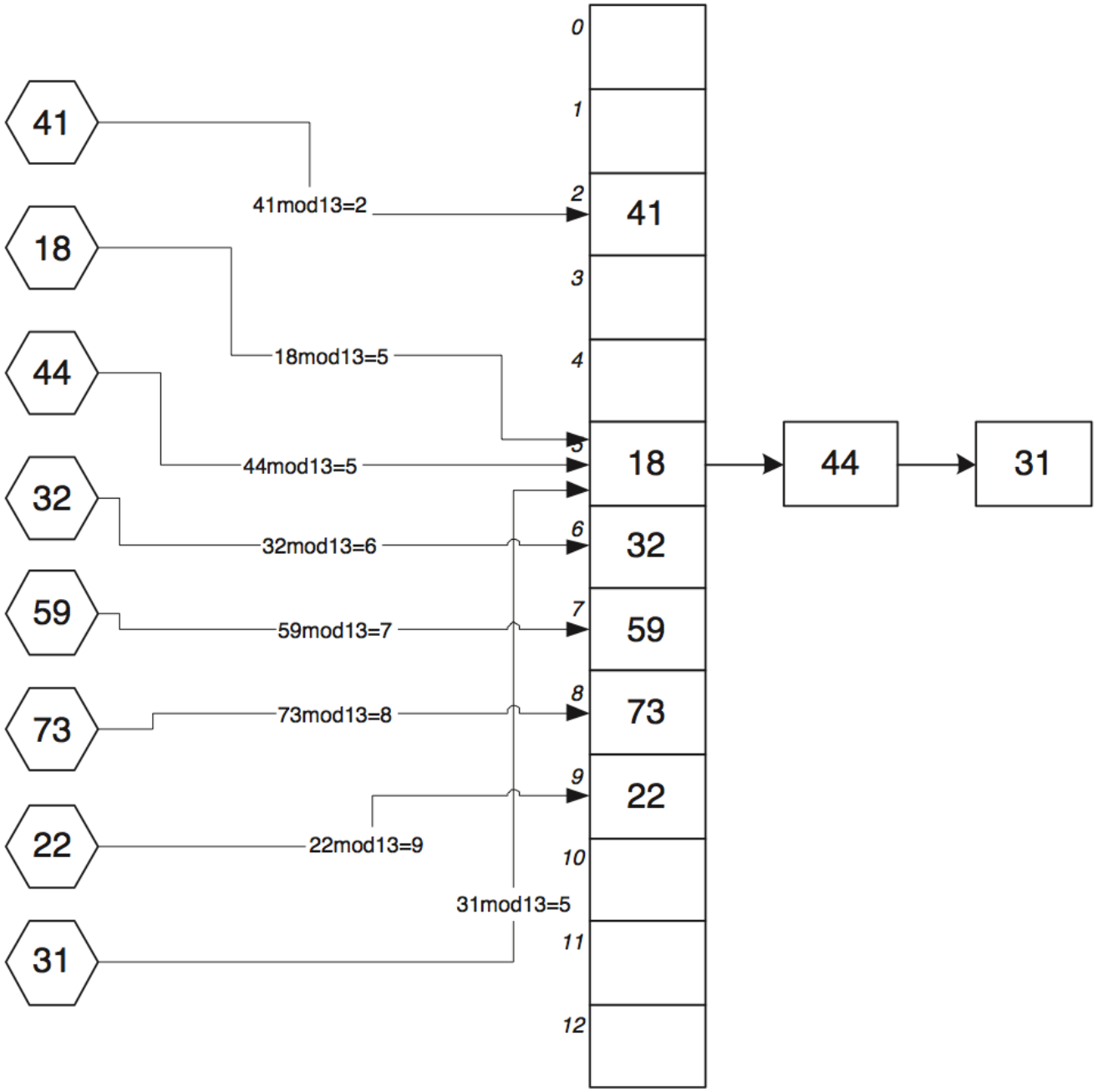


Abbildung 8.3: Hash-Tabelle mit Verkettung als Kollisionsbehandlung

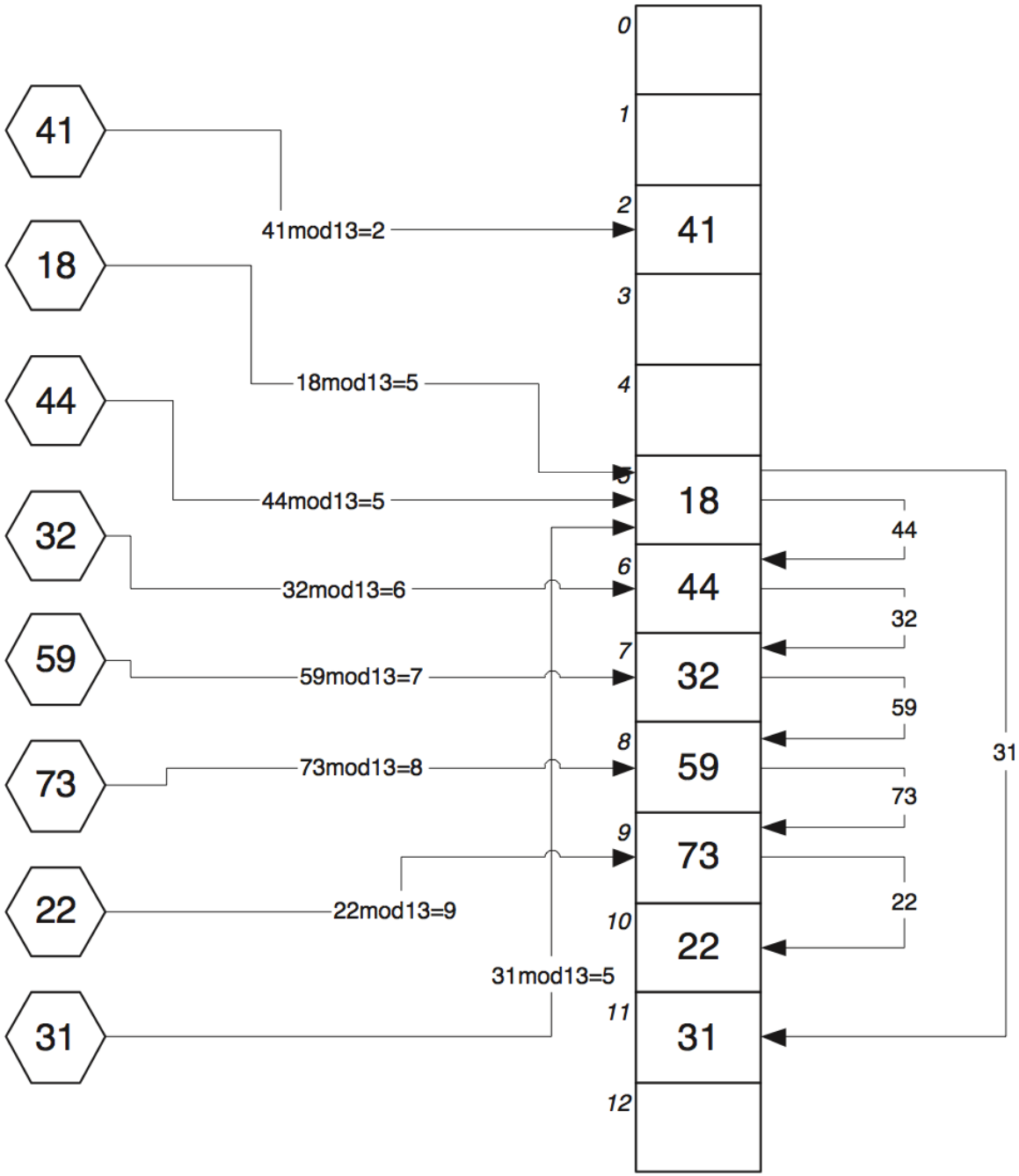


Abbildung 8.4: Hash-Tabelle mit *Linear Probing* als Kollisionsbehandlung

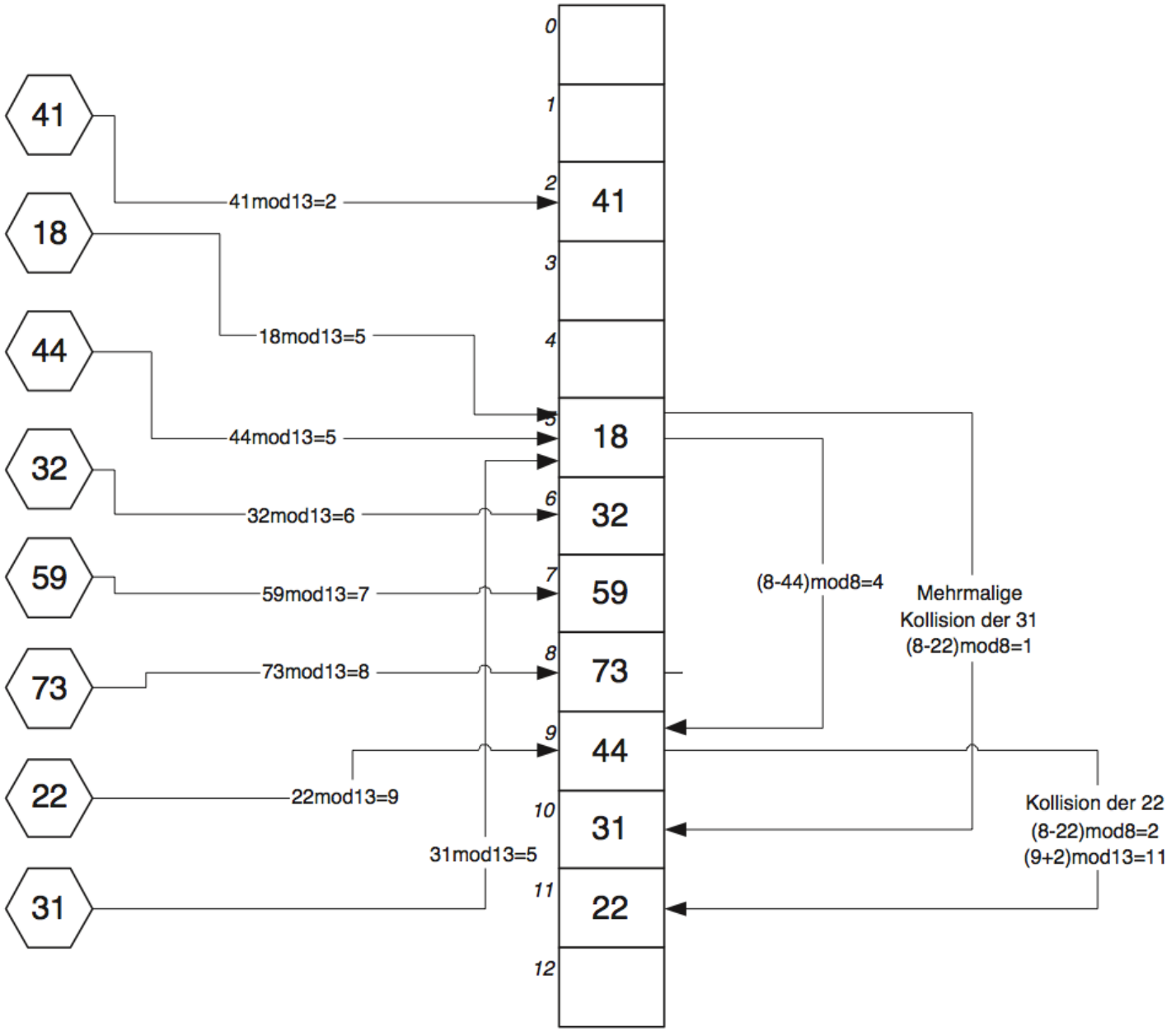


Abbildung 8.5: Hash-Tabelle mit *Double Hashing* als Kollisionsbehandlung: $h_1(K) = K \bmod 13$ und $h_2(K) = (8 - K) \bmod 8$

Datenbanksysteme
Eine Einführung

Alfons Kemper und Andre Eickler
Datenbanksysteme – Eine Einführung
Oldenbourg Verlag, München
(ca 40 Euro)

http://www-db.in.tum.de/research/publications/books/DBMSeinf

http://www-db.in.tum.de

Datenbanksysteme
Eine Einführung

Alfons Kemper und Andre Eickler

Datenbanksysteme – Eine Einführung

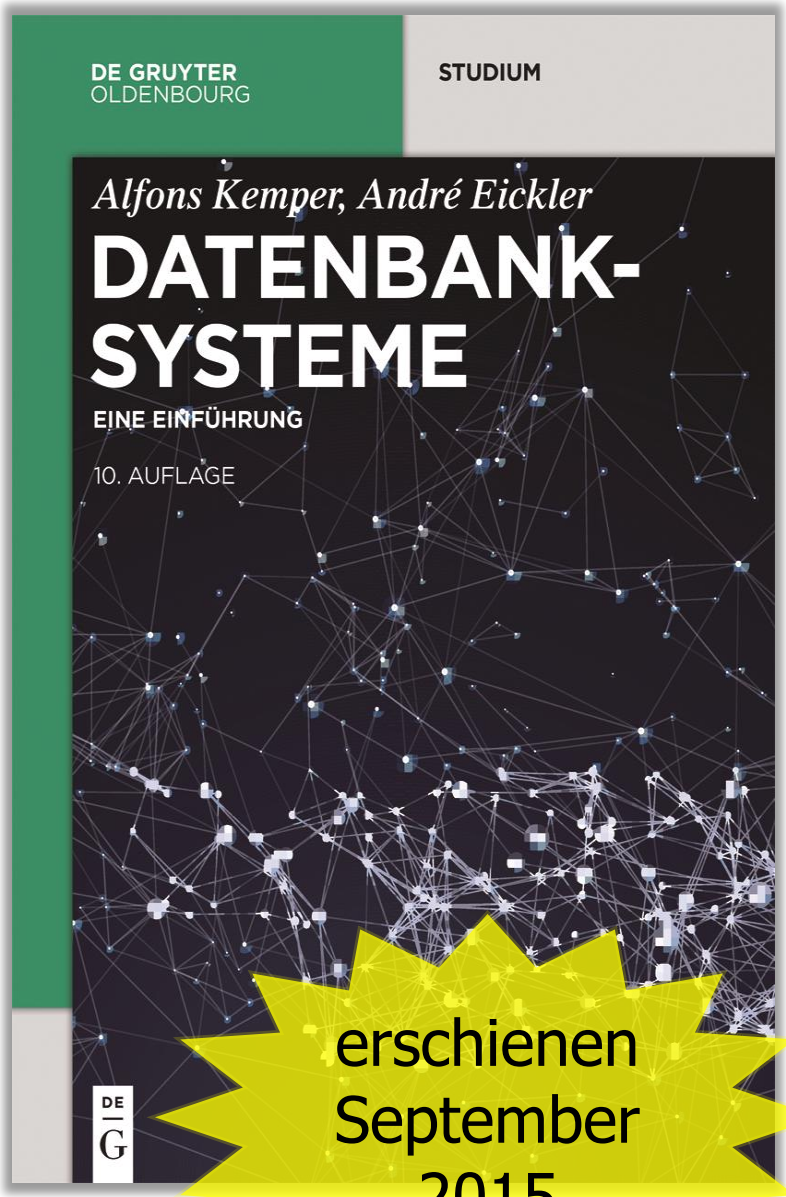
10. Auflage, 2015

Oldenbourg Verlag, München

(ca 50 Euro)

http://www-db.in.tum.de/research/publications/books/DBMSeinf

http://www-db.in.tum.de



erschienen September 2015

Aus dem Inhalt:

- Systematische und ausführliche Einführung in moderne Datenbanksysteme
- Fokus auf moderne Datenbanktechnologie
- Veranschaulichung durch Beispielanwendungen
- Aktualisierung neuer Entwicklungen: Hauptspeicher-Datenbanksysteme und BigData-Anwendungen

Ladenpreis: € 49.95 / US$ 70.00

Ca. 880 Seiten

Broschur isbn 978-3-11-044375-2

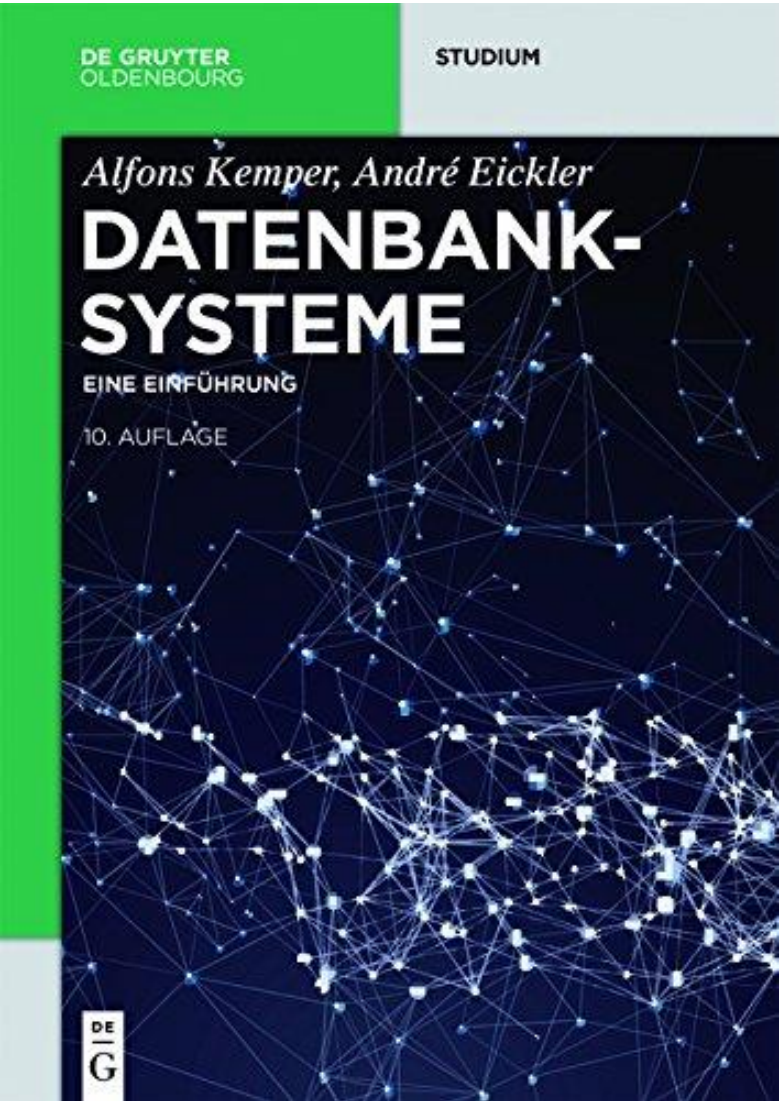
www.degruyter.com/books/978-3-11-044375-2

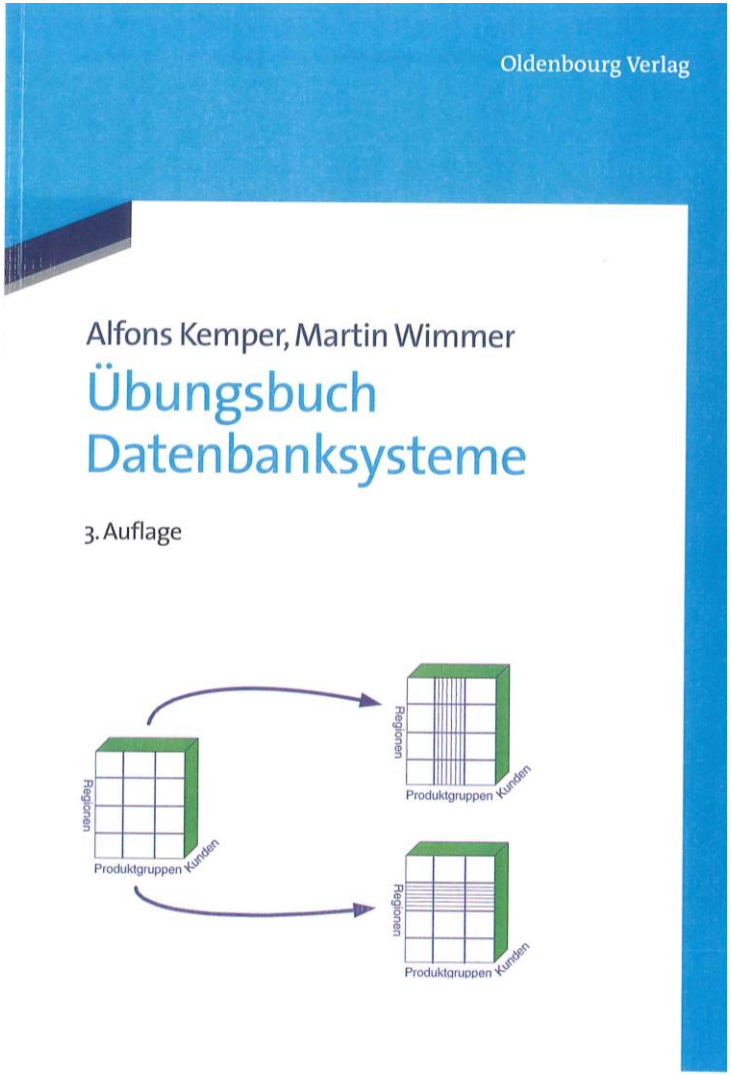
How Knowledge (or Exam-Material) expands ...

1-st Edition	2-nd Edition	3-rd Edition	4-th Edition	5-th Edition	6-th Edition	7-th Edition	8-th Edition	9-th Edition	10-th Edition
1996	1997	1999	2001	2004	2006	2009	2011	2013	2015
448 pages	504 pages	504 pages	608 pages	640 pages	672 pages	718 pages	792 pages	848 pages	**880** pages

Study fast ... the next (thicker) Edition is coming ☺

Komplementäres Übungsbuch



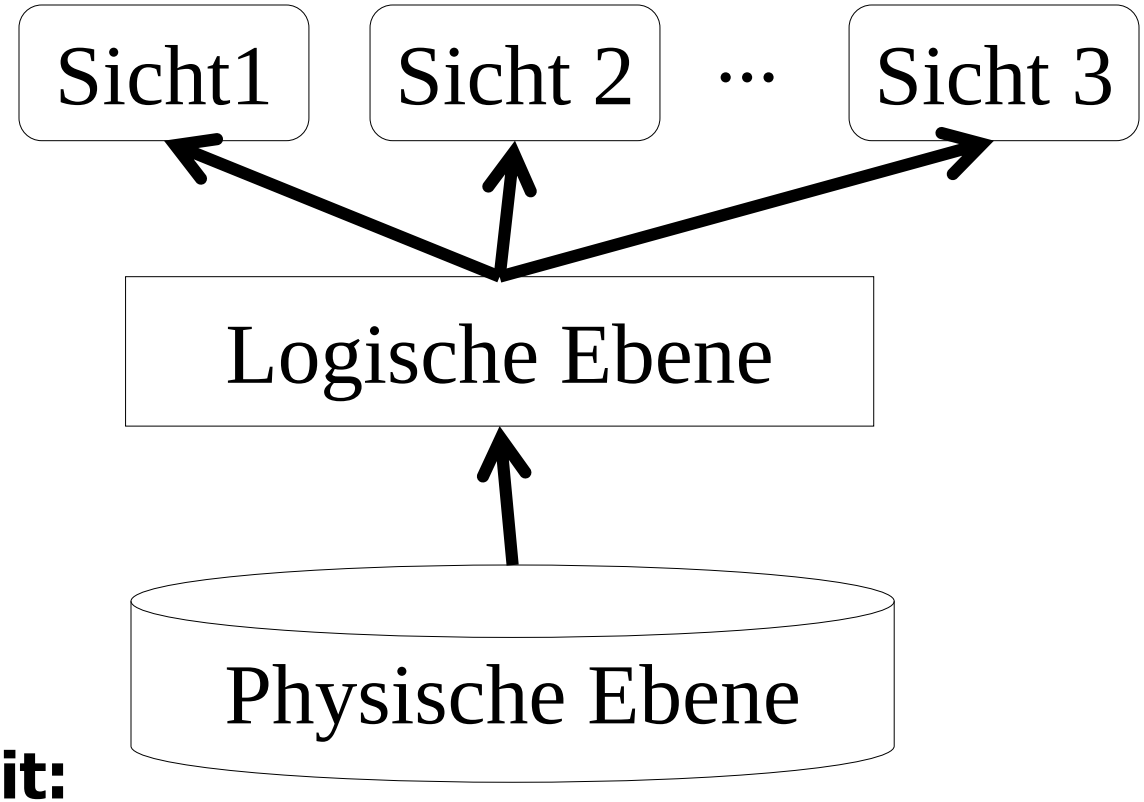


Motivation für den Einsatz eines Datenbank-Verwaltungssystems

Typische Probleme bei Informationsverarbeitung ohne DBMS

- Redundanz und Inkonsistenz
- Beschränkte Zugriffsmöglichkeiten
- Probleme beim Mehrbenutzerbetrieb
- Verlust von Daten
- Integritätsverletzung
- Sicherheitsprobleme
- hohe Entwicklungskosten für Anwendungsprogramme

Die Abstraktionsebenen eines Datenbanksystems



Datenunabhängigkeit:

- physische Unabhängigkeit
- logische Datenunabhängigkeit

Datenmodellierung

Manuelle/intellektuelle
Modellierung

Konzeptuelles Schema
(ER-Schema / **UML**)

Halbautomatische
Transformation

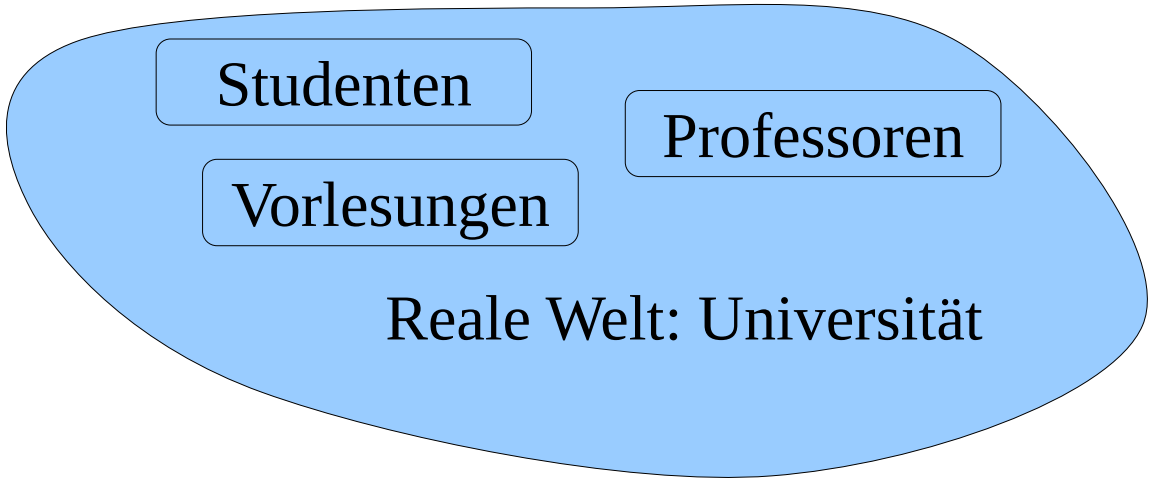
Relationales Schema

XML Schema

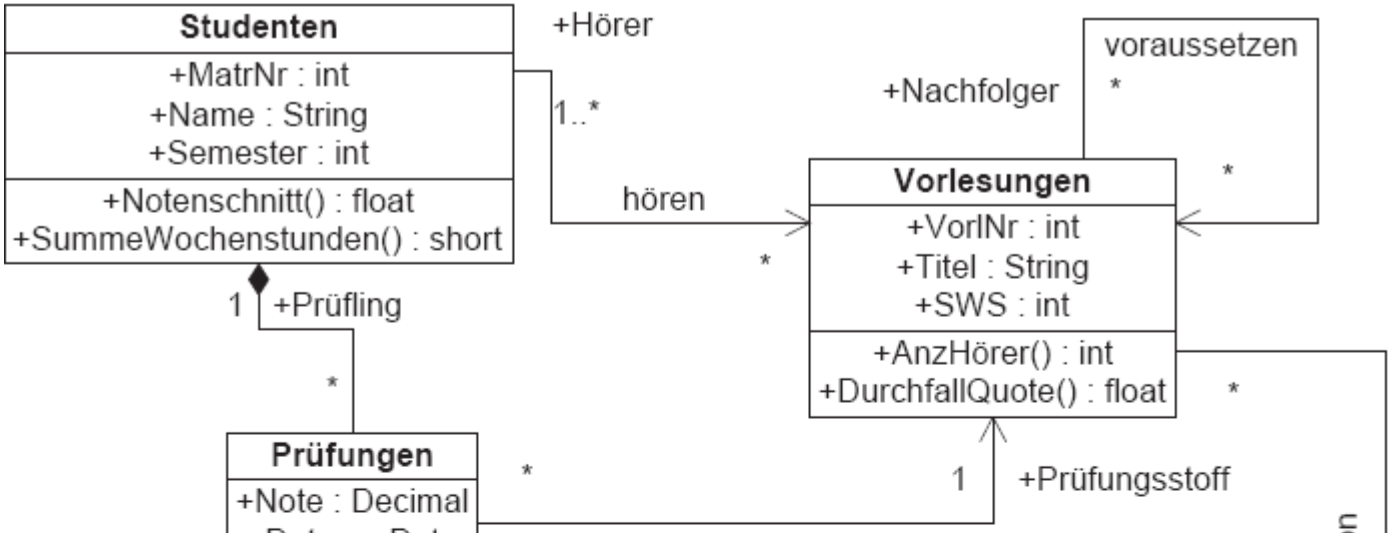
Netzwerk Schema

Objektorientiertes Modell (Java)

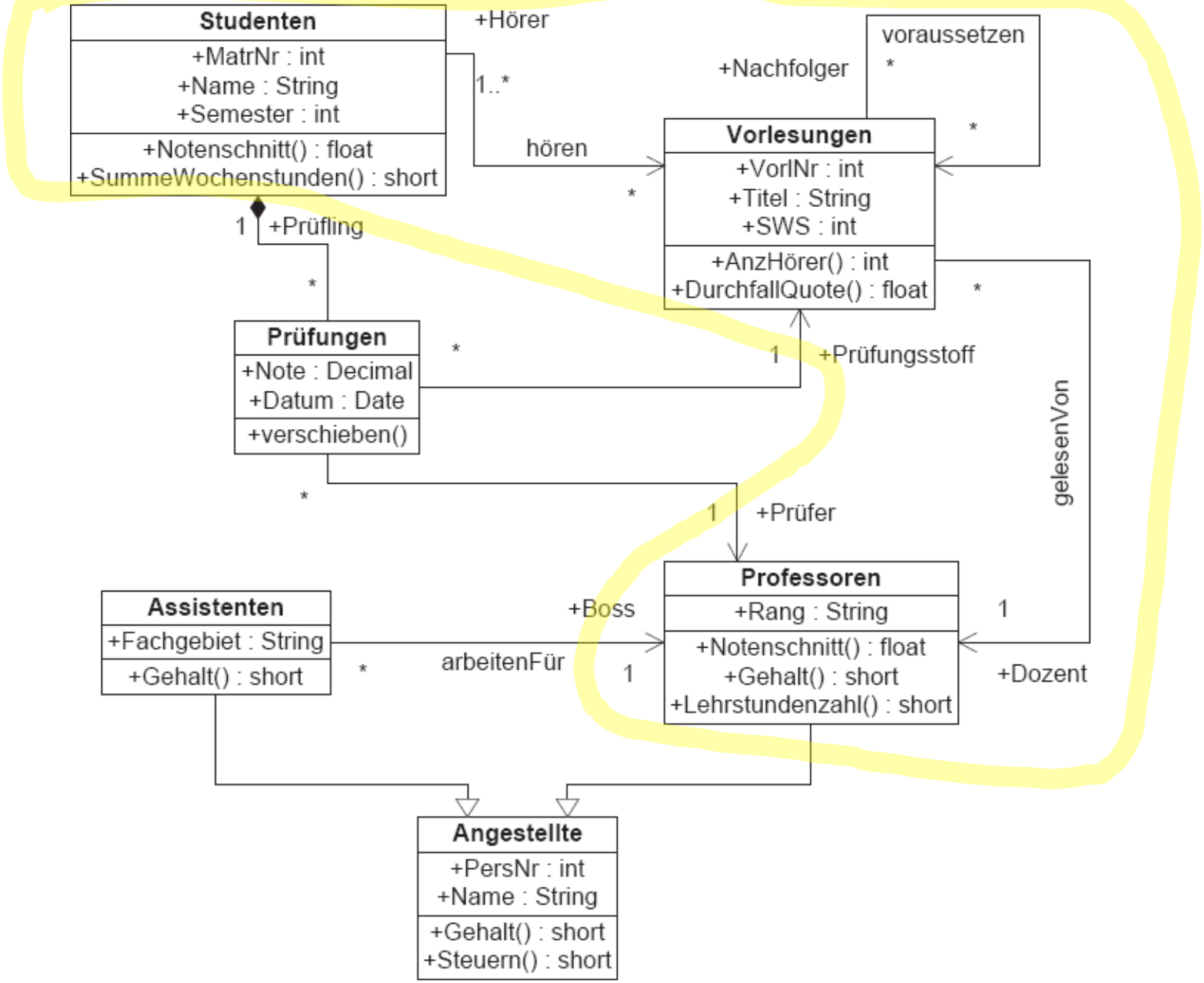
Modellierung einer kleinen Beispielanwendung



Konzeptuelle Modellierung



Relevanter Ausschnitt

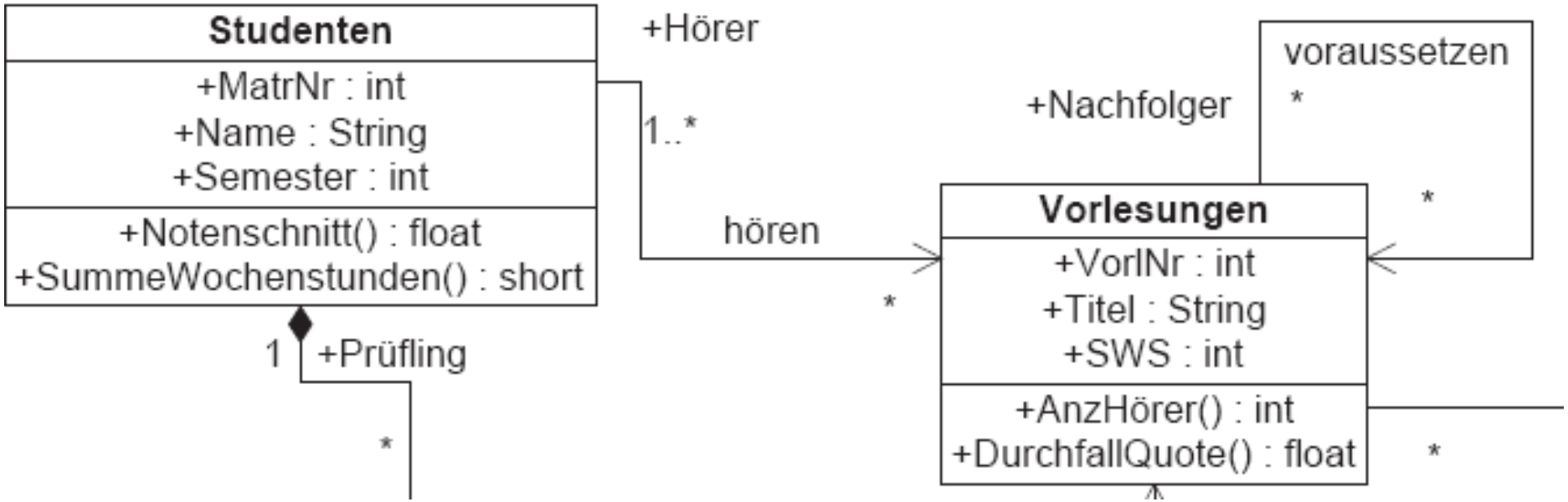


Logische Datenmodelle

- Netzwerkmodell
- Hierarchisches Datenmodell
- Relationales Datenmodell
- XML Schema
- Objektorientiertes Datenmodell
  - Objektrelationales Schema
- Deduktives Datenmodell

Wie werden Objektklassen und Assoziationen im relationealen Modell repräsentiert?

Studenten hören Vorlesungen



Das relationale Datenmodell

Studenten	
MatrNr	Name
26120	Fichte
25403	Jonas
...	...

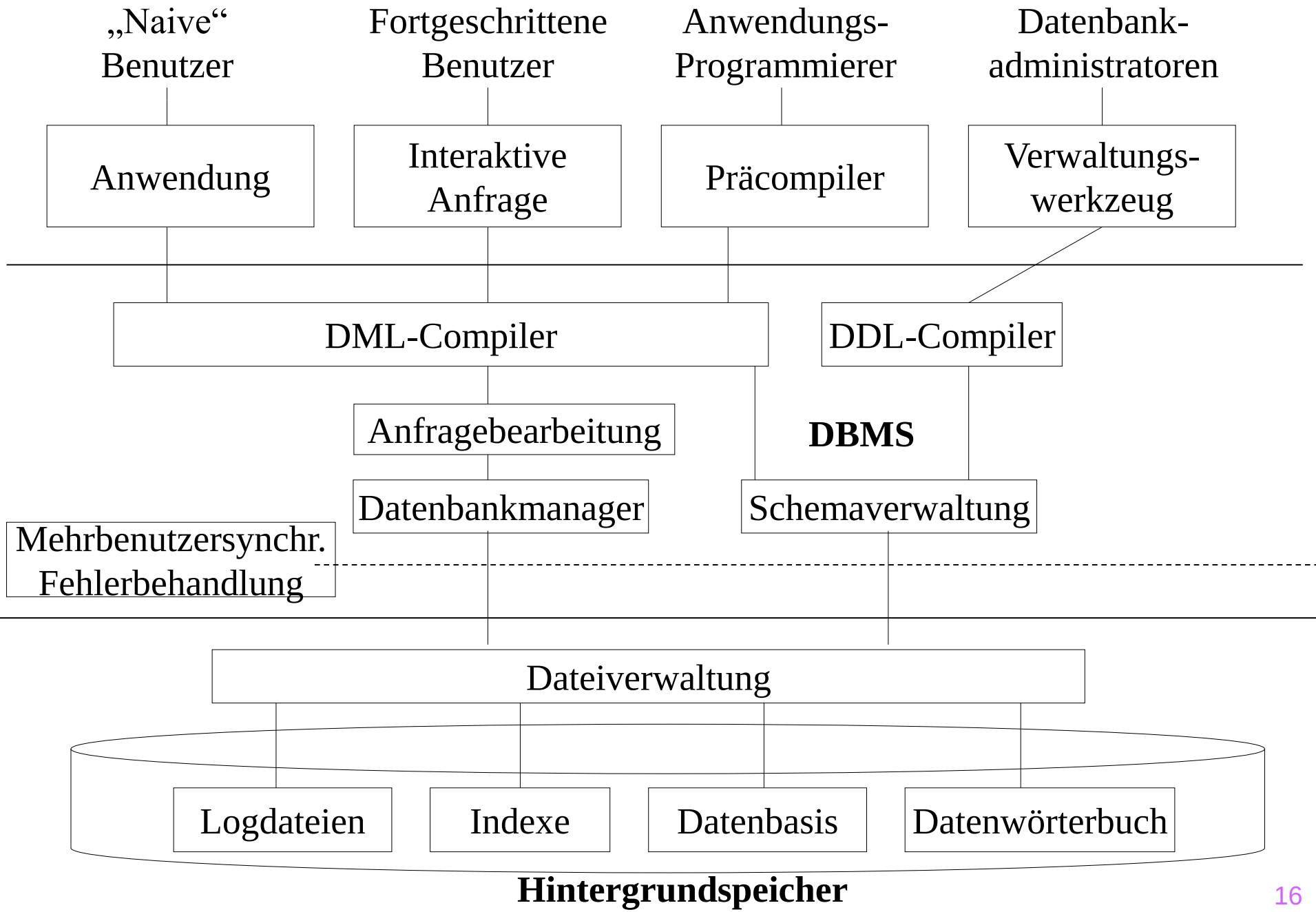
hören	
MatrNr	VorlNr
25403	5022
26120	5001
...	...

Vorlesungen	
VorlNr	Titel
5001	Grundzüge
5022	Glaube und Wissen
...	...

```
Select Name
From Studenten, hören, Vorlesungen
Where Studenten.MatrNr = hören.MatrNr  and
      hören.VorlNr = Vorlesungen.VorlNr  and
      Vorlesungen.Titel = `Grundzüge´;
```

```
update      Vorlesungen
   set      Titel = `Grundzüge der Logik´
 where      VorlNr = 5001;
```

Architekturübersicht eines DBMS



Historische Entwicklung relationaler DBMS

Grundlagen des relationalen Modells

Seien D_1, D_2, ..., D_n Domänen (~Wertebereiche)

- *Relation:* $R \subseteq D_1$ x ... x D_n

  Bsp.: Telefonbuch $\subseteq$ *string x string x integer*

- *Tupel:* $t \in R$

  Bsp.: t = („Mickey Mouse", „Main Street", 4711)

- *Schema:* legt die Struktur der gespeicherten Daten fest

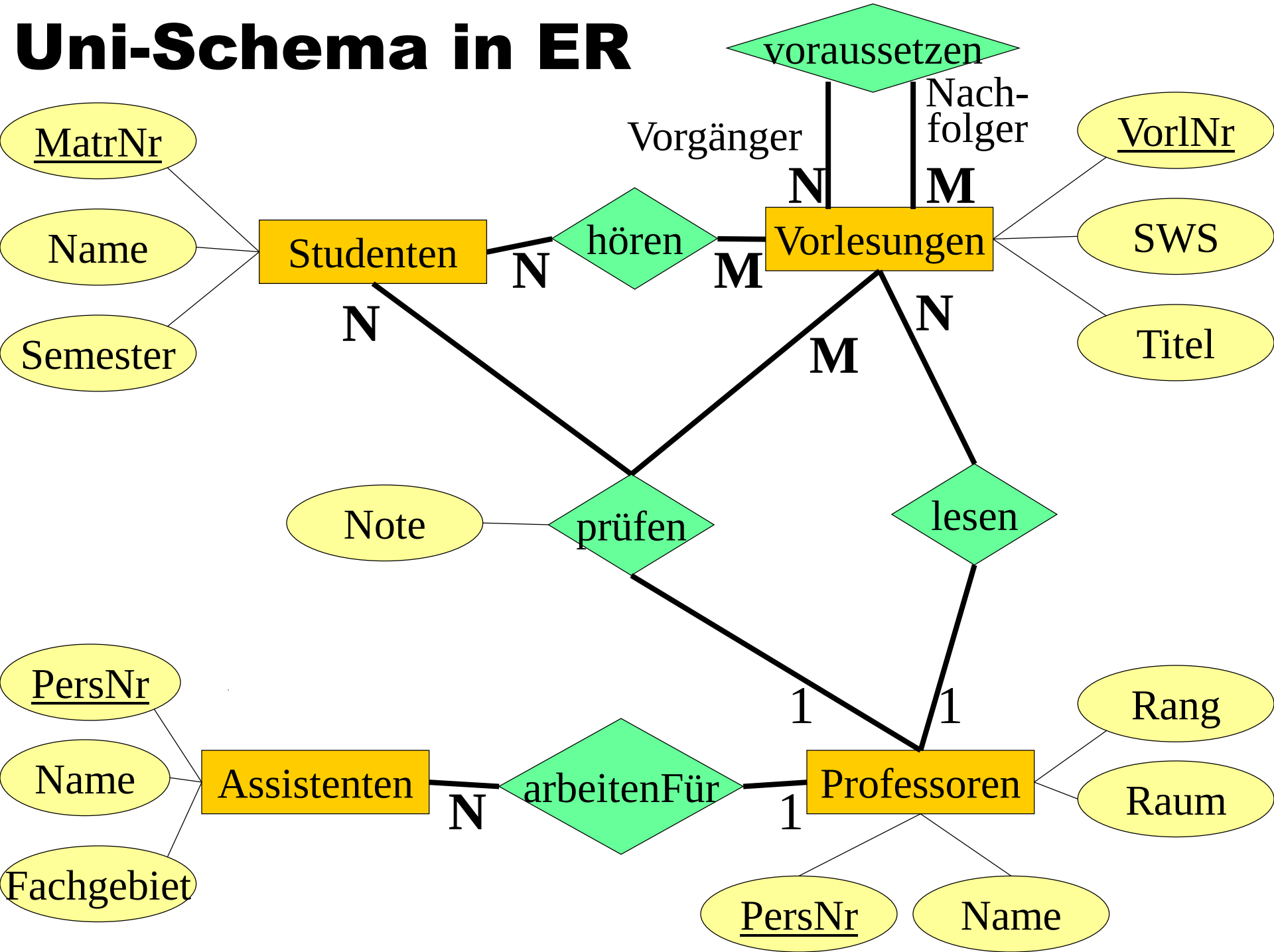
  Bsp.:

Telefonbuch: {[*Name: string, Adresse: string, Telefon#:integer*]}

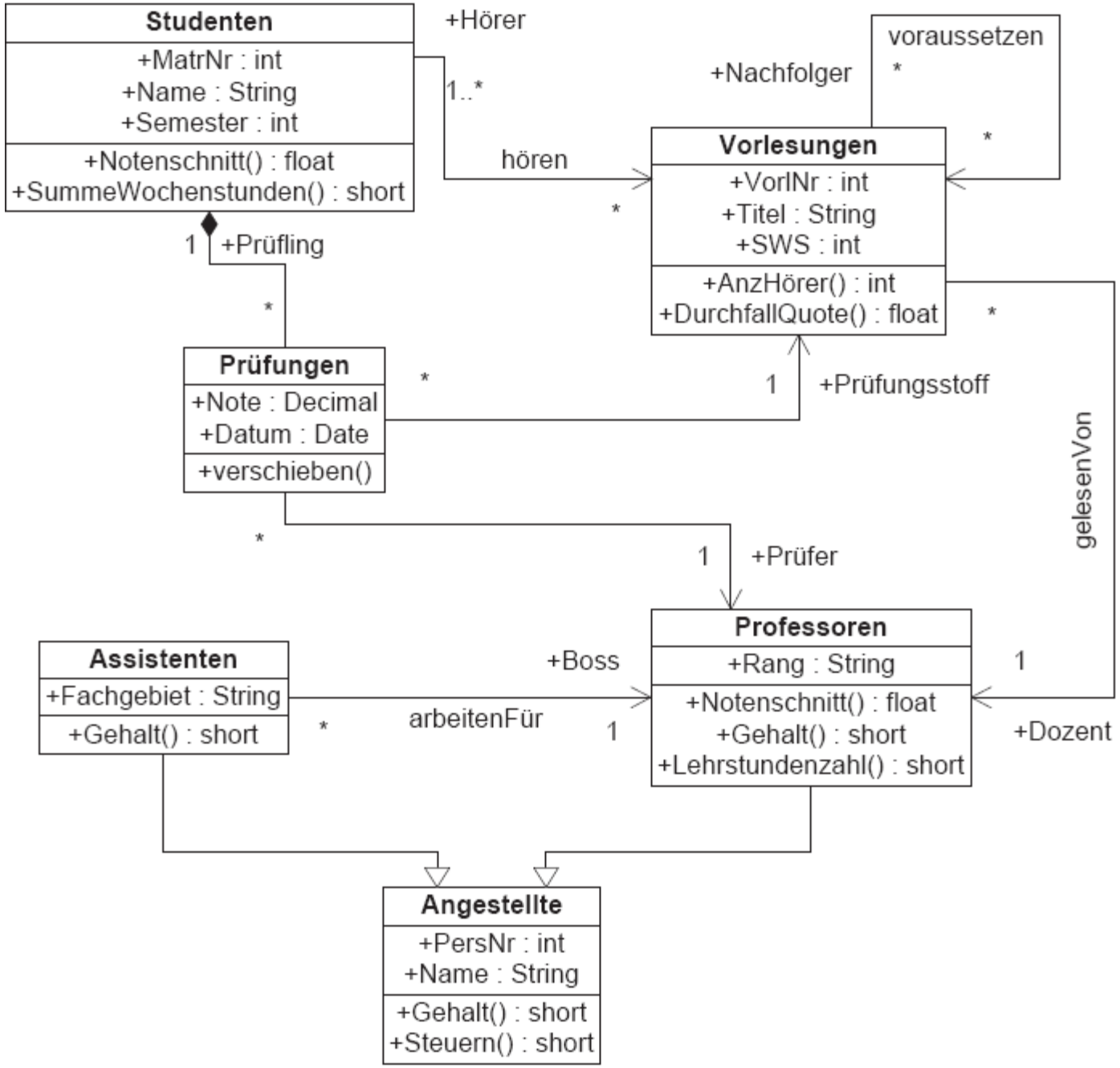
Telefonbuch		
Name	**Straße**	**Telefon#**
Mickey Mouse	Main Street	4711
Minnie Mouse	Broadway	94725
Donald Duck	Broadway	95672
...	...	...

- **Ausprägung:** der aktuelle Zustand der Datenbasis
- **Schlüssel:** minimale Menge von Attributen, deren Werte ein Tupel eindeutig identifizieren
- **Primärschlüssel:** wird unterstrichen
  - Einer der Schlüsselkandidaten wird als Primärschlüssel ausgewählt
  - Hat eine besondere Bedeutung bei der Referenzierung von Tupeln

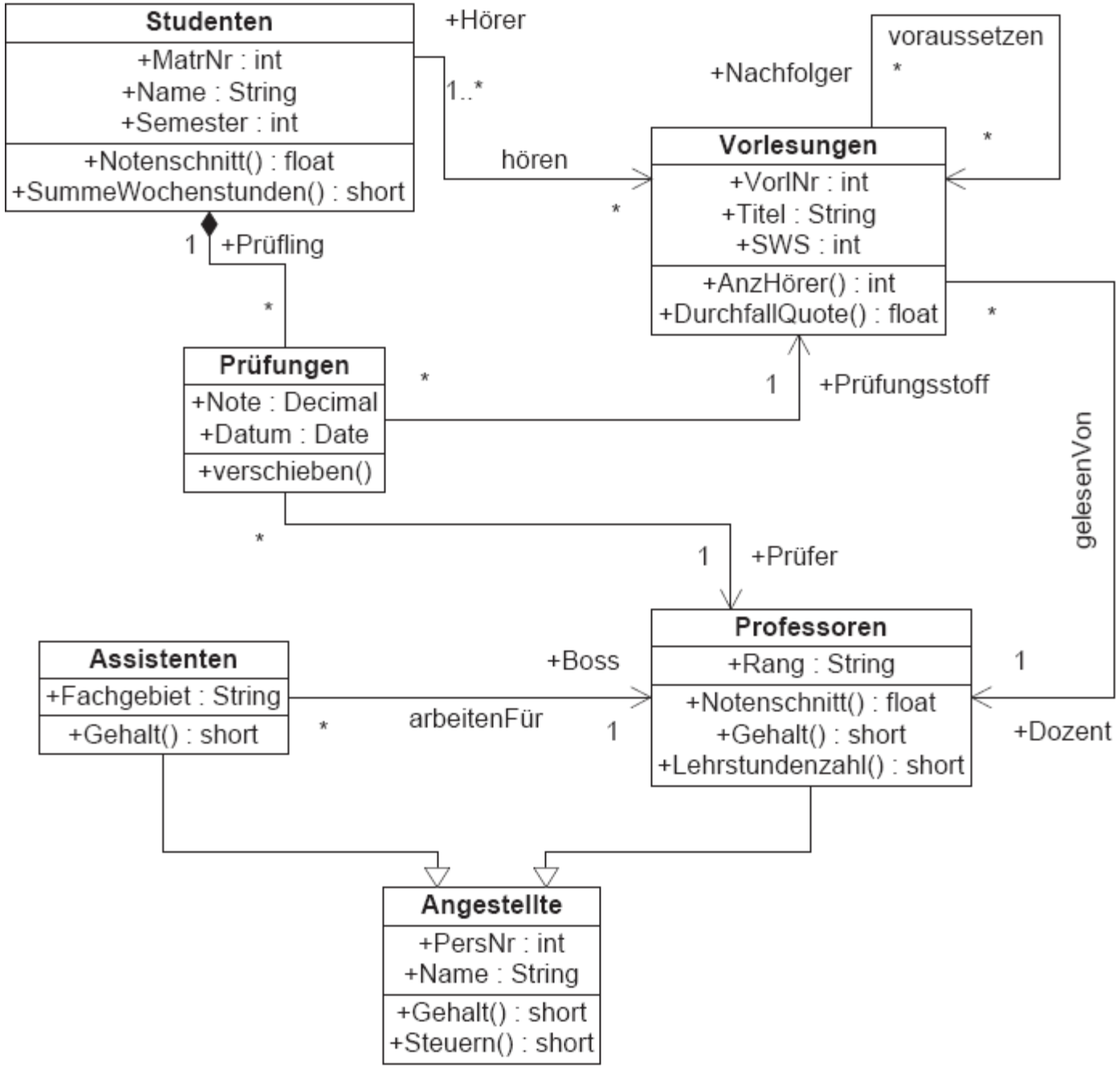
Uni-Schema in ER

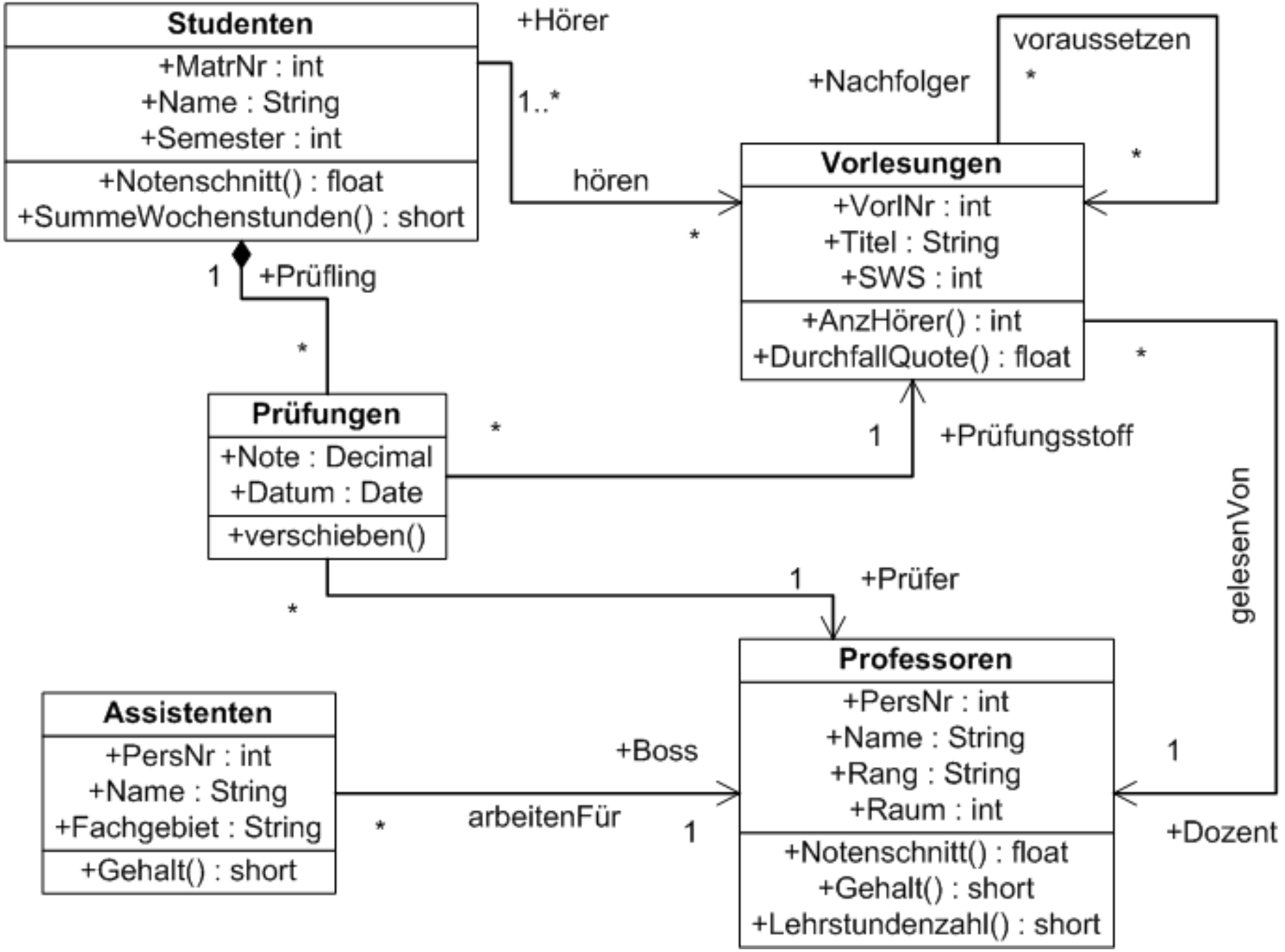


Uni-Schema in UML



Uni-Schema in UML -- Schlüssel




Studenten
+MatrNr : int
+Name : String
+Semester : int
+Notenschnitt() : float
+SummeWochenstunden() : short
+Hörer
1..*
hören
*
Vorlesungen
+VorlNr : int
+Titel : String
+SWS : int
+AnzHörer() : int
+DurchfallQuote() : float
+Nachfolger
voraussetzen
*
*
1
+Prüfling
*
Prüfungen
+Note : Decimal
+Datum : Date
+verschieben()
*
1
+Prüfungsstoff
*
gelesenVon
1
+Prüfer
*
Professoren
+PersNr : int
+Name : String
+Rang : String
+Raum : int
+Notenschnitt() : float
+Gehalt() : short
+Lehrstundenzahl() : short
1
+Dozent
Assistenten
+PersNr : int
+Name : String
+Fachgebiet : String
+Gehalt() : short
+Boss
*
arbeitenFür
1

Relationale Darstellung von Objekttypen

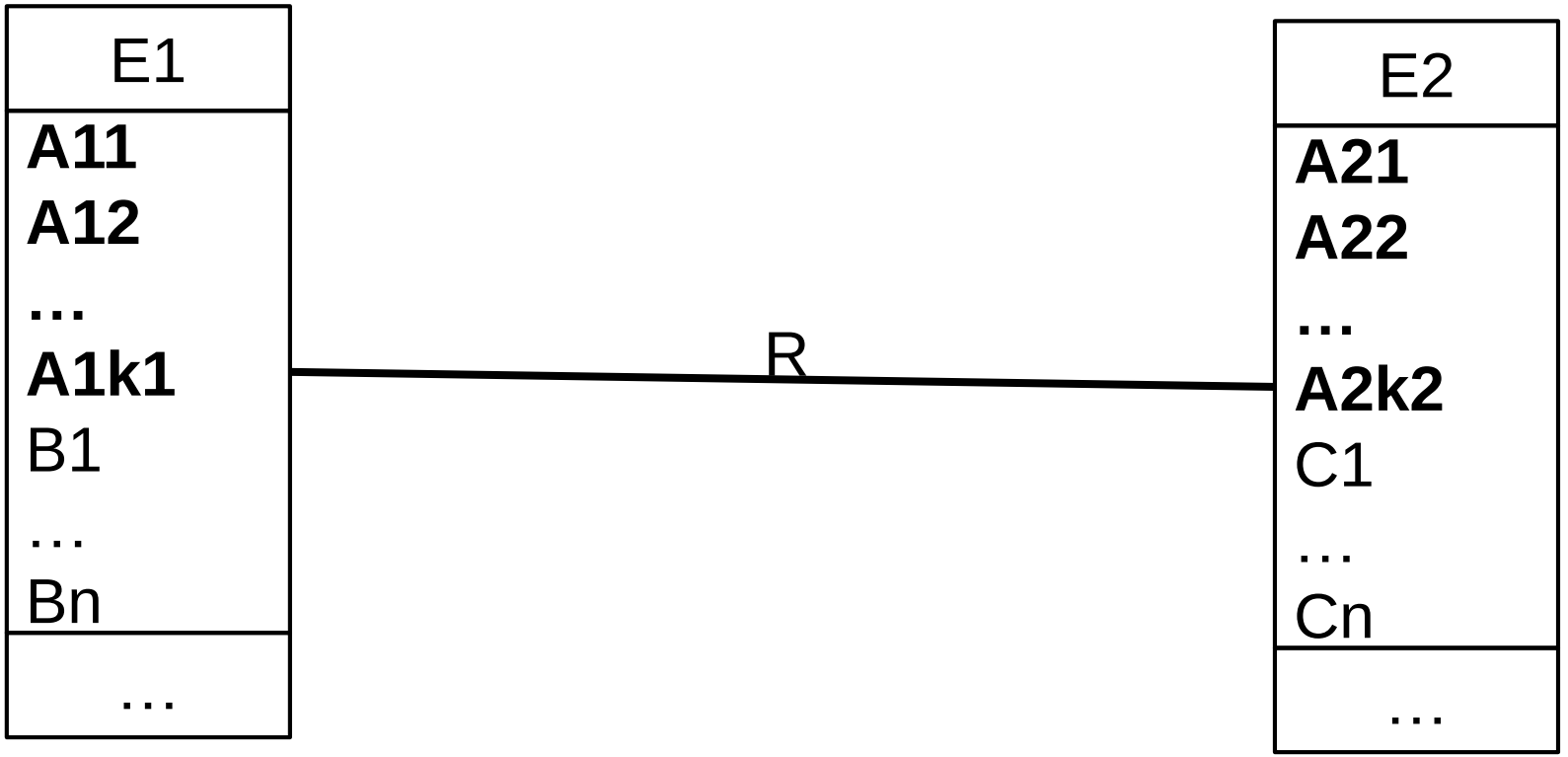
Studenten: {[MatrNr:integer, *Name*: *string, Semester*: *integer*]}

Vorlesungen: {[VorlNr:integer, *Titel*: *string, SWS*: *integer*]}

Professoren: {[PersNr:integer, *Name*: *string, Rang*: *string,*
Raum: *integer*]}

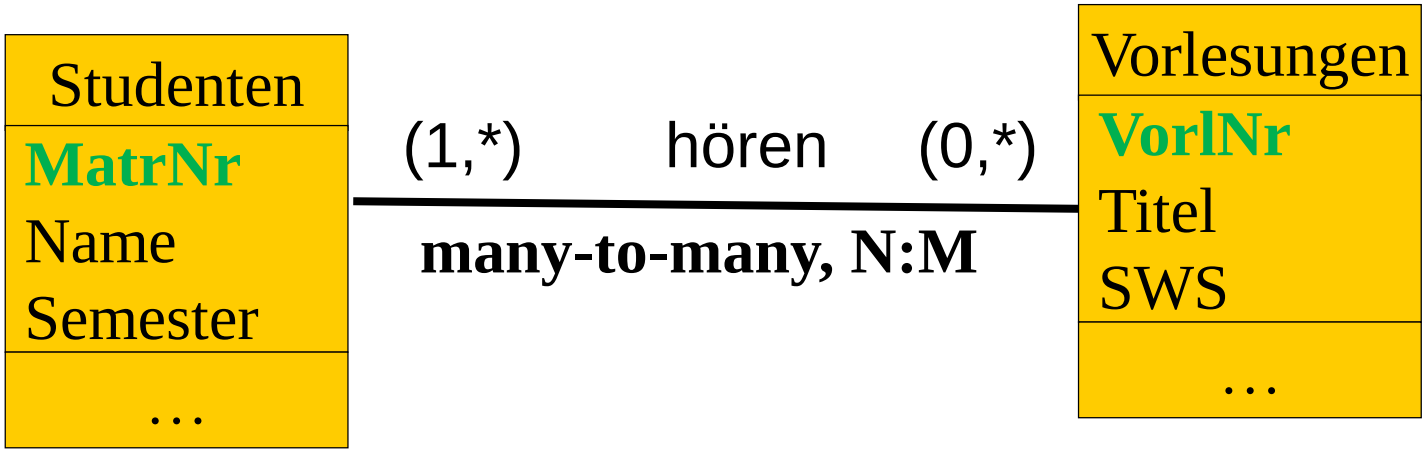
Assistenten: {[PersNr:integer, *Name*: *string, Fachgebiet*: *string*]}

Relationale Darstellung von Assoziationen



$$R:\{[\underbrace{A_{11},\ldots,A_{1k_1}}_{\text{Schlüssel von } E_1},\underbrace{A_{21},\ldots,A_{2k_2}}_{\text{Schlüssel von } E_2}]\}$$

Beziehungen unseres Beispiel-Schemas



hören : {[MatrNr: integer, VorlNr: integer]}

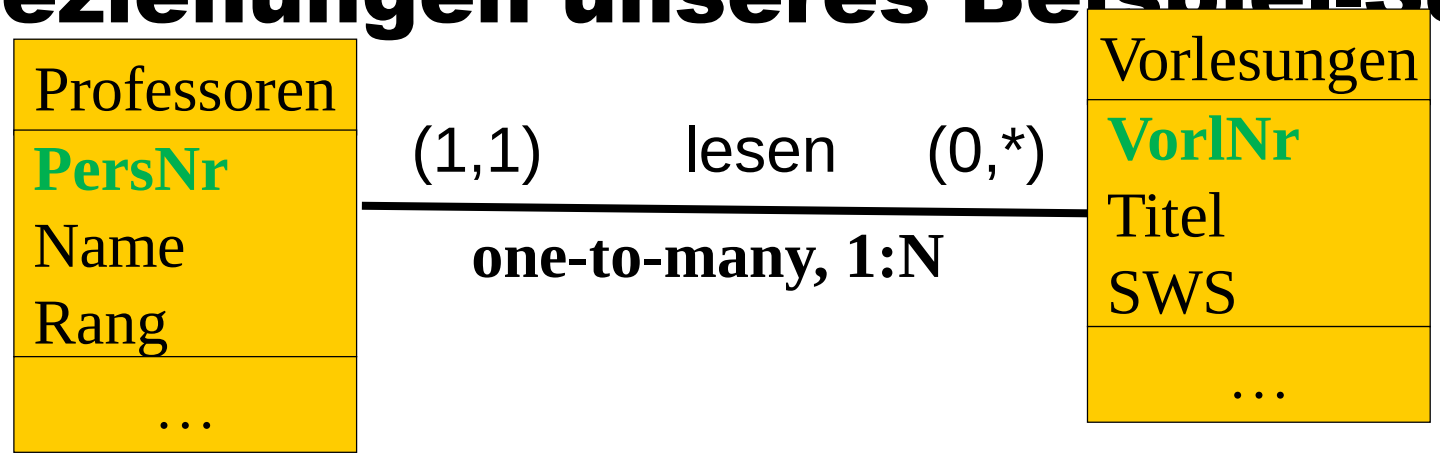
lesen : {[PersNr: integer, VorlNr: integer]}

arbeitenFür : {[AssistentenPersNr: integer, *ProfPersNr: integer*]}

voraussetzen : {[Vorgänger: integer, Nachfolger: integer]}

prüfen : {[MatrNr: integer, VorlNr: integer, PersNr: integer,
Note: decimal]}

Beziehungen unseres Beispiel-Schemas



hören : {[MatrNr: integer, VorlNr: integer]}

lesen : {[PersNr: integer, VorlNr: integer]}

arbeitenFür : {[AssistentenPersNr: integer, *ProfPersNr: integer*]}

voraussetzen : {[Vorgänger: integer, Nachfolger: integer]}

prüfen : {[MatrNr: integer, VorlNr: integer, PersNr: integer,
Note: decimal]}

Schlüssel der Relationen

hören : {[MatrNr: integer, VorlNr: integer]}

lesen : {[PersNr: integer, VorlNr: integer]}

arbeitenFür : {[AssistentenPersNr: integer, *ProfPersNr: integer*]}

voraussetzen : {[Vorgänger: integer, Nachfolger: integer]}

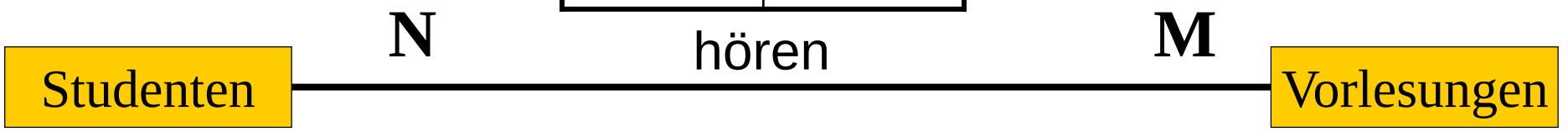
prüfen : {[MatrNr: integer, VorlNr: integer, PersNr: integer,
Note: decimal]}

Ausprägung der Beziehung *hören*

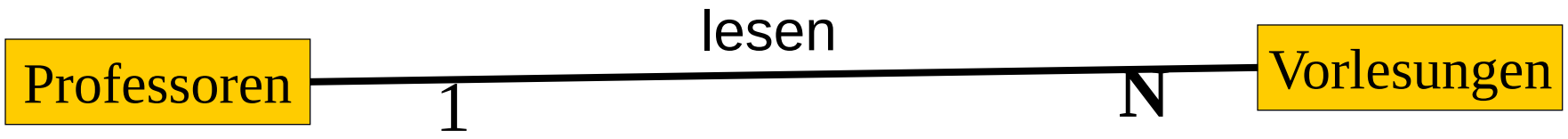
Studenten	
MatrNr	...
26120	...
27550	...
...	...

hören	
MatrNr	VorlNr
26120	5001
27550	5001
27550	4052
28106	5041
28106	5052
28106	5216
28106	5259
29120	5001
29120	5041
29120	5049
29555	5022
25403	5022
29555	5001

Vorlesungen	
VorlNr	...
5001	...
4052	...
...	...



Verfeinerung des relationalen Schemas



1:N-Beziehung

- Initial-Entwurf
  - ***Vorlesungen :*** {[*VorlNr, Titel, SWS*]}
  - ***Professoren :*** {[*PersNr, Name, Rang, Raum*]}
  - ***lesen:*** {[*VorlNr, PersNr*]}

Verfeinerung des relationalen Schemas

1:N-Beziehung

- Initial-Entwurf

  Vorlesungen : {[*VorlNr, Titel, SWS*]}
  Professoren : {[*PersNr, Name, Rang, Raum*]}
  lesen: {[*VorlNr, PersNr*]}

- Verfeinerung durch Zusammenfassung

  Vorlesungen : {[*VorlNr, Titel, SWS,* ***gelesenVon***]}
  Professoren : {[*PersNr, Name, Rang, Raum*]}

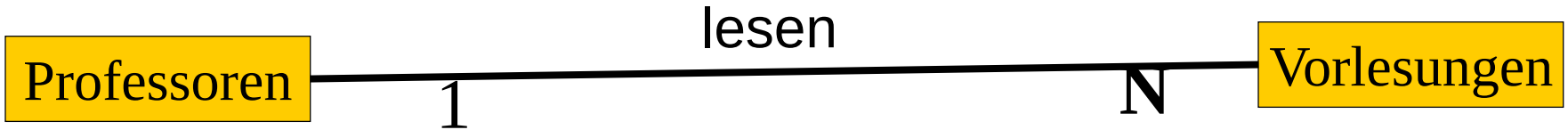
Regel

Relationen mit gleichem Schlüssel kann man zusammenfassen
aber nur diese und keine anderen!

Ausprägung von *Professoren* und *Vorlesung*

Professoren			
PersNr	Name	Rang	Raum
2125	Sokrates	C4	226
2126	Russel	C4	232
2127	Kopernikus	C3	310
2133	Popper	C3	52
2134	Augustinus	C3	309
2136	Curie	C4	36
2137	Kant	C4	7

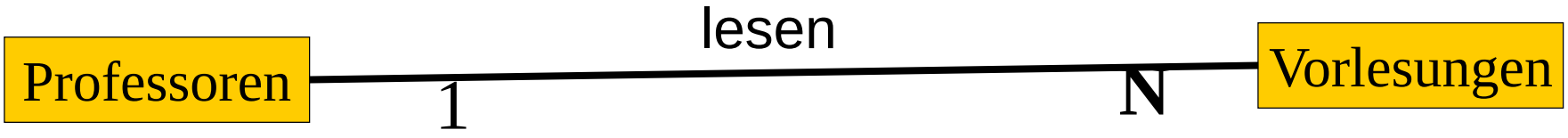
Vorlesungen			
VorlNr	Titel	SWS	Gelesen Von
5001	Grundzüge	4	2137
5041	Ethik	4	2125
5043	Erkenntnistheorie	3	2126
5049	Mäeutik	2	2125
4052	Logik	4	2125
5052	Wissenschaftstheorie	3	2126
5216	Bioethik	2	2126
5259	Der Wiener Kreis	2	2133
5022	Glaube und Wissen	2	2134
4630	Die 3 Kritiken	4	2137



Vorsicht: So geht es NICHT

Professoren				
PersNr	Name	Rang	Raum	liest
2125	Sokrates	C4	226	5041
2125	Sokrates	C4	226	5049
2125	Sokrates	C4	226	4052
...	...	...	...	...
2134	Augustinus	C3	309	5022
2136	Curie	C4	36	??

Vorlesungen		
VorlNr	Titel	SWS
5001	Grundzüge	4
5041	Ethik	4
5043	Erkenntnistheorie	3
5049	Mäeutik	2
4052	Logik	4
5052	Wissenschaftstheorie	3
5216	Bioethik	2
5259	Der Wiener Kreis	2
5022	Glaube und Wissen	2
4630	Die 3 Kritiken	4



Vorsicht: So geht es NICHT:

Folgen→Anomalien

Professoren				
PersNr	Name	Rang	Raum	liest
2125	Sokrates	C4	226	5041
2125	Sokrates	C4	226	5049
2125	Sokrates	C4	226	4052
...	...	...	...	...
2134	Augustinus	C3	309	5022
2136	Curie	C4	36	??

Vorlesungen		
VorlNr	Titel	SWS
5001	Grundzüge	4
5041	Ethik	4
5043	Erkenntnistheorie	3
5049	Mäeutik	2
4052	Logik	4
5052	Wissenschaftstheorie	3
5216	Bioethik	2
5259	Der Wiener Kreis	2
5022	Glaube und Wissen	2
4630	Die 3 Kritiken	4

- Update-Anomalie: Was passiert wenn Sokrates umzieht
- Lösch-Anomalie: Was passiert wenn „Glaube und Wissen“ wegfällt
- Einfügeanomalie: Curie ist neu und liest noch keine Vorlesungen

Die relationale Uni-DB

Professoren			
PersNr	Name	Rang	Raum
2125	Sokrates	C4	226
2126	Russel	C4	232
2127	Kopernikus	C3	310
2133	Popper	C3	52
2134	Augustinus	C3	309
2136	Curie	C4	36
2137	Kant	C4	7

Studenten		
MatrNr	Name	Semester
24002	Xenokrates	18
25403	Jonas	12
26120	Fichte	10
26830	Aristoxenos	8
27550	Schopenhauer	6
28106	Carnap	3
29120	Theophrastos	2
29555	Feuerbach	2

Vorlesungen			
VorlNr	Titel	SWS	gelesen von
5001	Grundzüge	4	2137
5041	Ethik	4	2125
5043	Erkenntnistheorie	3	2126
5049	Mäeutik	2	2125
4052	Logik	4	2125
5052	Wissenschaftstheorie	3	2126
5216	Bioethik	2	2126
5259	Der Wiener Kreis	2	2133
5022	Glaube und Wissen	2	2134
4630	Die 3 Kritiken	4	2137

voraussetzen	
Vorgänger	Nachfolger
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5259

hören	
MatrNr	VorlNr
26120	5001
27550	5001
27550	4052
28106	5041
28106	5052
28106	5216
28106	5259
29120	5001
29120	5041
29120	5049
29555	5022
25403	5022

Assistenten			
PerslNr	Name	Fachgebiet	Boss
3002	Platon	Ideenlehre	2125
3003	Aristoteles	Syllogistik	2125
3004	Wittgenstein	Sprachtheorie	2126
3005	Rhetikus	Planetenbewegung	2127
3006	Newton	Keplersche Gesetze	2127
3007	Spinoza	Gott und Natur	2126

prüfen			
MatrNr	VorlNr	PersNr	Note
28106	5001	2126	1
25403	5041	2125	2
27550	4630	2137	2

Professoren

PersNr	Name	Rang	Raum
2125	Sokrates	C4	226
2126	Russel	C4	232
2127	Kopernikus	C3	310
2133	Popper	C3	52
2134	Augustinus	C3	309
2136	Curie	C4	36
2137	Kant	C4	7

Studenten

MatrNr	Name	Semester
24002	Xenokrates	18
25403	Jonas	12
26120	Fichte	10
26830	Aristoxenos	8
27550	Schopenhauer	6
28106	Carnap	3
29120	Theophrastos	2
29555	Feuerbach	2

Vorlesungen

VorlNr	Titel	SWS	gelesen Von
5001	Grundzüge	4	2137
5041	Ethik	4	2125
5043	Erkenntnistheorie	3	2126
5049	Mäeutik	2	2125
4052	Logik	4	2125
5052	Wissenschaftstheorie	3	2126
5216	Bioethik	2	2126
5259	Der Wiener Kreis	2	2133
5022	Glaube und Wissen	2	2134
4630	Die 3 Kritiken	4	2137

voraussetzen

Vorgänger	Nachfolger
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5259

hören

MatrNr	VorlNr
26120	5001
27550	5001
27550	4052
28106	5041
28106	5052
28106	5216
28106	5259
29120	5001
29120	5041
29120	5049
29555	5022
25403	5022

prüfen

MatrNr	VorlNr	PersNr	Note
28106	5001	2126	1
25403	5041	2125	2
27550	4630	2137	2

Assistenten

PerslNr	Name	Fachgebiet	Boss
3002	Platon	Ideenlehre	2125
3003	Aristoteles	Syllogistik	2125
3004	Wittgenstein	Sprachtheorie	2126
3005	Rhetikus	Planetenbewegung	2127
3006	Newton	Keplersche Gesetze	2127
3007	Spinoza	Gott und Natur	2126

Die relationale Algebra

- σ Selektion
- π Pojektion
- x Kreuzprodukt
- $\bowtie$ Join (Verbund)
- ρ Umbenennung
- – Mengendifferenz
- $\div$ Division
- $\cup$ Vereinigung
- $\cap$ Mengendurchschnitt
- $\rtimes$ Semi-Join (linker)
- $\ltimes$ Semi-Join (rechter)
- $\bowtie$ linker äußerer Join
- $\bowtie$ rechter äußerer Join

Die relationalen Algebra-Operatoren

Selektion

$\sigma_{Semester > 10}$ (Studenten)

$\sigma_{Semester > 10}$ (Studenten)		
MatrNr	Name	Semester
24002	Xenokrates	18
25403	Jonas	12

Projektion

Π_{Rang}(Professoren)

Π_{Rang}(Professoren)
Rang
C4
C3

Die relationalen Algebra-Operatoren

Kartesisches Produkt

Professoren x hören

Professoren				hören	
PersNr	Name	Rang	Raum	MatrNr	VorlNr
2125	Sokrates	C4	226	26120	5001
...	...	...	...	...	...
2125	Sokrates	C4	226	29555	5001
...	...	...	...	...	...
2137	Kant	C4	7	29555	5001

- Problem: riesige Zwischenergebnisse
- Beispiel: (Professoren x hören)
- "bessere" Operation: Join (siehe unten)

Die relationalen Algebra-Operatoren

Umbenennung

- Umbenennung von Relationen
- Beispiel: Ermittlung indirekter Vorgänger 2. Stufe der Vorlesung 5216

$$\Pi_{\text{V1. Vorgänger}}(\sigma_{\text{V2. Nachfolger=5216} \wedge \text{V1.Nachfolger = V2.Vorgänger}} (\rho_{V1} \text{(voraussetzen)} \times \rho_{V2} \text{(voraussetzen)}))$$

- Umbennung von Attributen

$$\rho_{\text{Voraussetzung} \leftarrow \text{Vorgänger}} \text{(voraussetzen)}$$

Formale Definition der Algebra

Basisausdrücke

- Relation der Datenbank oder
- konstante Relationen

Operationen

- Selektion: $\sigma_p (E_1)$
- Projektion: $\Pi_S (E_1)$
- Kartesisches Produkt: $E_1 \times E_2$
- Umbenennung: $\rho_V (E_1)$, $\rho_{A \leftarrow B} (E_1)$
- Vereinigung: $E_1 \cup E_2$
- Differenz: $E_1 - E_2$

Der natürliche Verbund (Join)

Gegeben seien:

- $R(A_1,..., A_m, B_1,..., B_k)$
- $S(B_1,..., B_k, C_1,..., C_n)$

$$R \bowtie S = \Pi_{A1,..., Am, R.B1,..., R.Bk, C1,..., Cn}(\sigma_{R.B1=S.\ B1\ \wedge...\wedge\ R.Bk\ =\ S.Bk}(RxS))$$

$R \bowtie S$											
$R - S$				$R \cap S$				$S - R$			
A_1	A_2	...	A_m	B_1	B_2	...	B_k	C_1	C_2	...	C_n
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Drei-Wege-Join

(Studenten ⋈ hören) ⋈ Vorlesungen

(Studenten ⋈ hören) ⋈ Vorlesungen						
MatrNr	Name	Semester	VorlNr	Titel	SWS	gelesenVon
26120	Fichte	10	5001	Grundzüge	4	2137
27550	Jonas	12	5022	Glaube und Wissen	2	2134
28106	Carnap	3	4052	Wissenschftstheorie	3	2126
...	...	...	...	...	...	...

Allgemeiner Join (Theta-Join)

- Gegeben seien folgende Relationen(-Schemata)
  - R(A1, ..., An) und
  - S(B1, ..., Bm)

$$R \bowtie_\theta S = \sigma_\theta (R \times S)$$

$$R \bowtie_\theta S$$

$R \bowtie_\theta S$							
R				S			
A_1	A_2	...	A_n	B_1	B_2	...	B_m
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Andere Join-Arten

- natürlicher Join

L		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

$\bowtie$

R		
C	D	E
c_1	d_1	e_1
c_3	d_2	e_2

=

Resultat				
A	B	C	D	E
a_1	b_1	c_1	d_1	e_1

- linker äußerer Join

L		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

$\bowtie$

R		
C	D	E
c_1	d_1	e_1
c_3	d_2	e_2

=

Resultat				
A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_2	b_2	c_2	-	-

- rechter äußerer Join

L		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

$\bowtie$

R		
C	D	E
c_1	d_1	e_1
c_3	d_2	e_2

=

Resultat				
A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
-	-	c_3	d_2	e_2

Andere Join-Arten

- äußerer Join

L		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

$\bowtie$

R		
C	D	E
c_1	d_1	e_1
c_3	d_2	e_2

=

Resultat				
A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_2	b_2	c_2	-	-
-	-	c_3	d_2	e_2

- Semi-Join von L mit R

L		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

$\ltimes$

R		
C	D	E
c_1	d_1	e_1
c_3	d_2	e_2

=

Resultat		
A	B	C
a_1	b_1	c_1

Andere Join-Arten (Forts.)

- Semi-Join von R mit L

L		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

$\rtimes$

R		
C	D	E
c_1	d_1	e_1
c_3	d_2	e_2

=

Resultat		
C	D	E
c_1	d_1	e_1

Andere Join-Arten (Forts.)

- Anti-Semi-Join von L mit R

L		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

R		
C	D	E
c_1	d_1	e_1
c_3	d_2	e_2

=

Resultat		
A	B	C
a_2	b_2	c_2

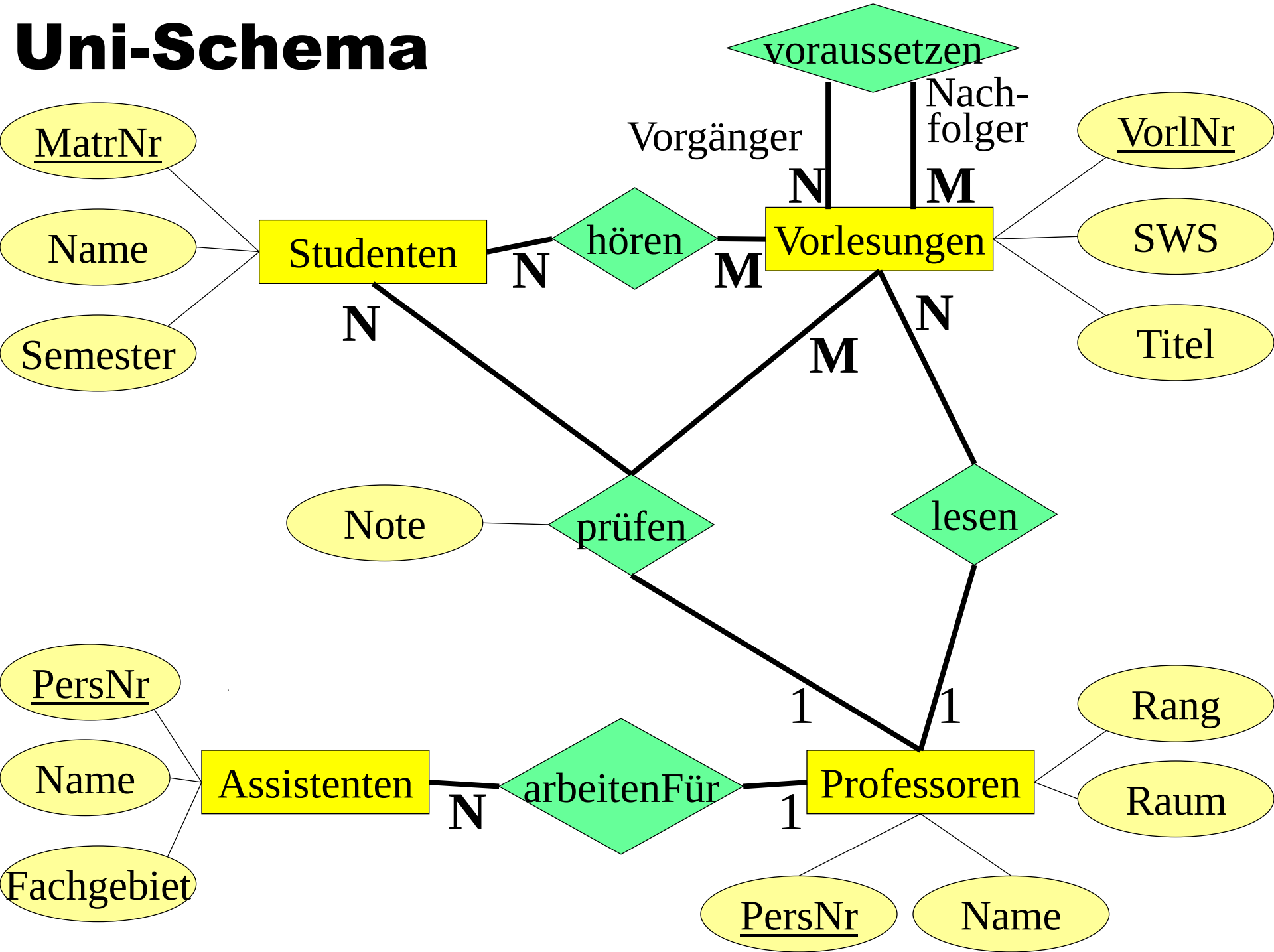
SQL

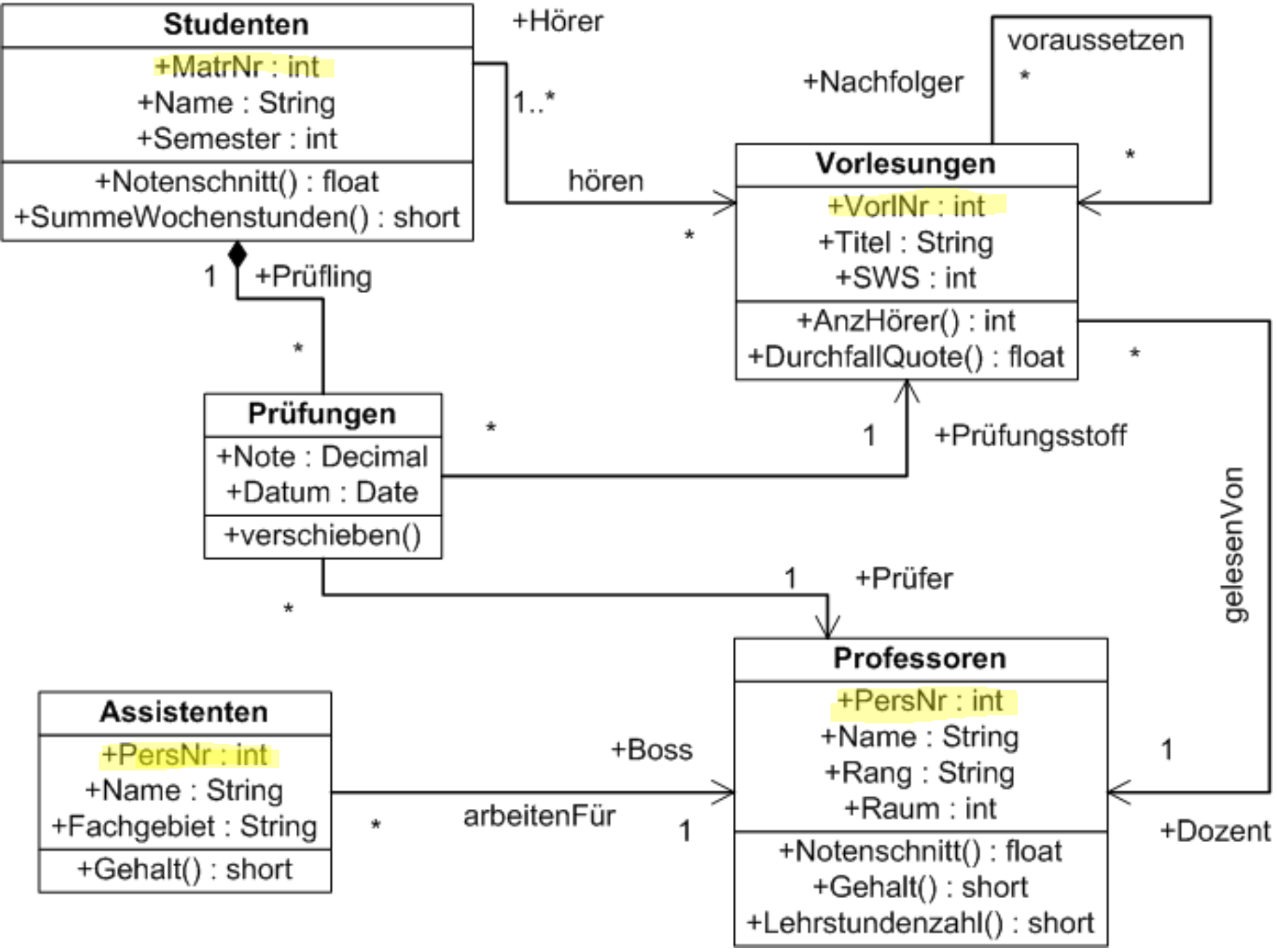
- standardisierte
  - Datendefinitions (DDL)-
  - Datenmanipulations (DML)-
  - Anfrage (Query)-Sprache
- derzeit aktueller Standard ist SQL 99 und SQL3 (2003)
  - objektrelationale Erweiterung
- Für praktische Übungen steht eine Web-Seite zur Verfügung: **http://www-db.in.tum.de/research/publications/books/DBMSeinf**
- Man kann eigene Relationen anlegen und/oder die Uni-DB verwenden
- DB2 von IBM „liegt dahinter"

http://www-db.in.tum.de/db2face/index.shtml

SQL Übungen

Uni-Schema


voraussetzen
Vorgänger
Nach-
folger
N
M
MatrNr
Name
Semester
Studenten
N
hören
M
Vorlesungen
VorlNr
SWS
Titel
N
M
N
Note
prüfen
lesen
1
1
PersNr
Name
Fachgebiet
Assistenten
N
arbeitenFür
1
Professoren
Rang
Raum
PersNr
Name


Studenten
+MatrNr : int
+Name : String
+Semester : int
+Notenschnitt() : float
+SummeWochenstunden() : short
+Hörer
1..*
hören
*
+Nachfolger
voraussetzen
*
*
Vorlesungen
+VorlNr : int
+Titel : String
+SWS : int
+AnzHörer() : int
+DurchfallQuote() : float
1
+Prüfling
*
Prüfungen
+Note : Decimal
+Datum : Date
+verschieben()
*
1
+Prüfungsstoff
*
gelesenVon
1
+Prüfer
*
Professoren
+PersNr : int
+Name : String
+Rang : String
+Raum : int
+Notenschnitt() : float
+Gehalt() : short
+Lehrstundenzahl() : short
1
+Dozent
Assistenten
+PersNr : int
+Name : String
+Fachgebiet : String
+Gehalt() : short
+Boss
*
arbeitenFür
1

Professoren

PersNr	Name	Rang	Raum
2125	Sokrates	C4	226
2126	Russel	C4	232
2127	Kopernikus	C3	310
2133	Popper	C3	52
2134	Augustinus	C3	309
2136	Curie	C4	36
2137	Kant	C4	7

Studenten

MatrNr	Name	Semester
24002	Xenokrates	18
25403	Jonas	12
26120	Fichte	10
26830	Aristoxenos	8
27550	Schopenhauer	6
28106	Carnap	3
29120	Theophrastos	2
29555	Feuerbach	2

Vorlesungen

VorlNr	Titel	SWS	gelesen Von
5001	Grundzüge	4	2137
5041	Ethik	4	2125
5043	Erkenntnistheorie	3	2126
5049	Mäeutik	2	2125
4052	Logik	4	2125
5052	Wissenschaftstheorie	3	2126
5216	Bioethik	2	2126
5259	Der Wiener Kreis	2	2133
5022	Glaube und Wissen	2	2134
4630	Die 3 Kritiken	4	2137

voraussetzen

Vorgänger	Nachfolger
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5259

hören

MatrNr	VorlNr
26120	5001
27550	5001
27550	4052
28106	5041
28106	5052
28106	5216
28106	5259
29120	5001
29120	5041
29120	5049
29555	5022
25403	5022

prüfen

MatrNr	VorlNr	PersNr	Note
28106	5001	2126	1
25403	5041	2125	2
27550	4630	2137	2

Assistenten

PerslNr	Name	Fachgebiet	Boss
3002	Platon	Ideenlehre	2125
3003	Aristoteles	Syllogistik	2125
3004	Wittgenstein	Sprachtheorie	2126
3005	Rhetikus	Planetenbewegung	2127
3006	Newton	Keplersche Gesetze	2127
3007	Spinoza	Gott und Natur	2126

(Einfache) Datendefinition in SQL

Datentypen

- **character** *(n)*, ***char*** *(n)*
- **character varying** *(n)*, **varchar** *(n)*
- **numeric** *(p,s)*, **integer**
- **blob** oder **raw** für sehr große binäre Daten
- **clob** für sehr große String-Attribute
- **date** für Datumsangaben
- **xml** für XML-Dokumente

Anlegen von Tabelle

- **create table** Professoren

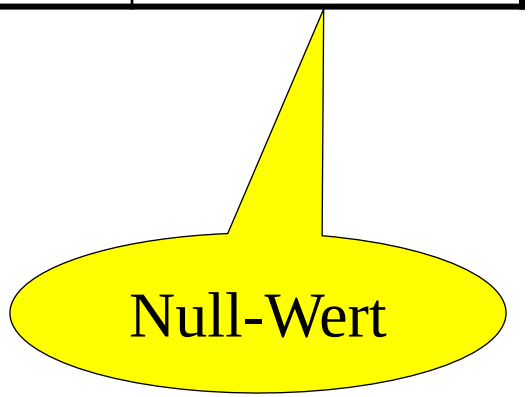
```
(PersNr     integer not null,
 Name       varchar (30) not null
 Rang       character (2) );
```

Veränderung am Datenbestand

Einfügen von Tupeln

```
insert into hören
        select MatrNr, VorlNr
        from Studenten, Vorlesungen
        where Titel= `Logik' ;


insert into Studenten (MatrNr, Name)
        values (28121,  `Archimedes');
```

Studenten		
MatrNr	Name	Semester
⋮	⋮	⋮
29120	Theophrastos	2
29555	Feuerbach	2
28121	Archimedes	-



Veränderungen am Datenbestand

Löschen von Tupeln

```
delete Studenten
where Semester > 13;
```

Verändern von Tupeln

```
update Studenten
    set Semester= Semester + 1;
```

Einfache SQL-Anfrage

```
select    PersNr, Name
from      Professoren
where     Rang= ´C4´;
```

PersNr	Name
2125	Sokrates
2126	Russel
2136	Curie
2137	Kant

Einfache SQL-Anfragen

Sortierung

select PersNr, Name, Rang

from Professoren

order by Rang **desc**, Name **asc**;

PersNr	Name	Rang
2136	Curie	C4
2137	Kant	C4
2126	Russel	C4
2125	Sokrates	C4
2134	Augustinus	C3
2127	Kopernikus	C3
2133	Popper	C3

Duplikateliminierung

select distinct Rang

from Professoren

Rang
C3
C4

Professoren

PersNr	Name	Rang	Raum
2125	Sokrates	C4	226
2126	Russel	C4	232
2127	Kopernikus	C3	310
2133	Popper	C3	52
2134	Augustinus	C3	309
2136	Curie	C4	36
2137	Kant	C4	7

Studenten

MatrNr	Name	Semester
24002	Xenokrates	18
25403	Jonas	12
26120	Fichte	10
26830	Aristoxenos	8
27550	Schopenhauer	6
28106	Carnap	3
29120	Theophrastos	2
29555	Feuerbach	2

Vorlesungen

VorlNr	Titel	SWS	gelesen Von
5001	Grundzüge	4	2137
5041	Ethik	4	2125
5043	Erkenntnistheorie	3	2126
5049	Mäeutik	2	2125
4052	Logik	4	2125
5052	Wissenschaftstheorie	3	2126
5216	Bioethik	2	2126
5259	Der Wiener Kreis	2	2133
5022	Glaube und Wissen	2	2134
4630	Die 3 Kritiken	4	2137

voraussetzen

Vorgänger	Nachfolger
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5259

hören

MatrNr	VorlNr
26120	5001
27550	5001
27550	4052
28106	5041
28106	5052
28106	5216
28106	5259
29120	5001
29120	5041
29120	5049
29555	5022
25403	5022

prüfen

MatrNr	VorlNr	PersNr	Note
28106	5001	2126	1
25403	5041	2125	2
27550	4630	2137	2

Assistenten

PerslNr	Name	Fachgebiet	Boss
3002	Platon	Ideenlehre	2125
3003	Aristoteles	Syllogistik	2125
3004	Wittgenstein	Sprachtheorie	2126
3005	Rhetikus	Planetenbewegung	2127
3006	Newton	Keplersche Gesetze	2127
3007	Spinoza	Gott und Natur	2126

Anfragen über mehrere Relationen

Welcher Professor liest "Mäeutik"?

select Name, Titel
from Professoren , Vorlesungen
where PersNr = gelesenVon **and** Titel = `Mäeutik‘ ;

$$\prod_{\text{Name, Titel}} (\sigma_{\text{PersNr} = \text{gelesenVon} \wedge \text{Titel} = \text{'Mäeutik'}} (\text{Professoren} \times \text{Vorlesungen}))$$

Anfragen über mehrere Relationen

Professoren			
PersNr	Name	Rang	Raum
2125	Sokrates	C4	226
2126	Russel	C4	232
⋮	⋮	⋮	⋮
2137	Kant	C4	7

Vorlesungen			
VorlNr	Titel	SWS	gelesen Von
5001	Grundzüge	4	2137
5041	Ethik	4	2125
⋮	⋮	⋮	⋮
5049	Mäeutik	2	2125
⋮	⋮	⋮	⋮
4630	Die 3 Kritiken	4	2137

Verknüpfung X

PersN	Name	Rang	Raum	VorlNr	Titel	SWS	gelesen Von
2125	Sokrates	C4	226	5001	Grundzüge	4	2137
1225	Sokrates	C4	226	5041	Ethik	4	2125
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2125	Sokrates	C4	226	5049	Mäeutik	2	2125
⋮	⋮		⋮	⋮	⋮	⋮	⋮
2126	Russel	C4	232	5001	Grundzüge	4	2137
2126	Russel	C4	232	5041	Ethik	4	2125
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2137	Kant	C4	7	4630	Die 3 Kritiken	4	2137

↓ Auswahl

PersNr	Name	Rang	Raum	VorlNr	Titel	SWS	gelesen Von
2125	Sokrates	C4	226	5049	Mäeutik	2	2125

↓ Projektion

Name	Titel
Sokrates	Mäeutik

Kanonische Übersetzung in die relationale Algebra

Allgemein hat eine (ungeschachtelte) SQL-Anfrage die Form:

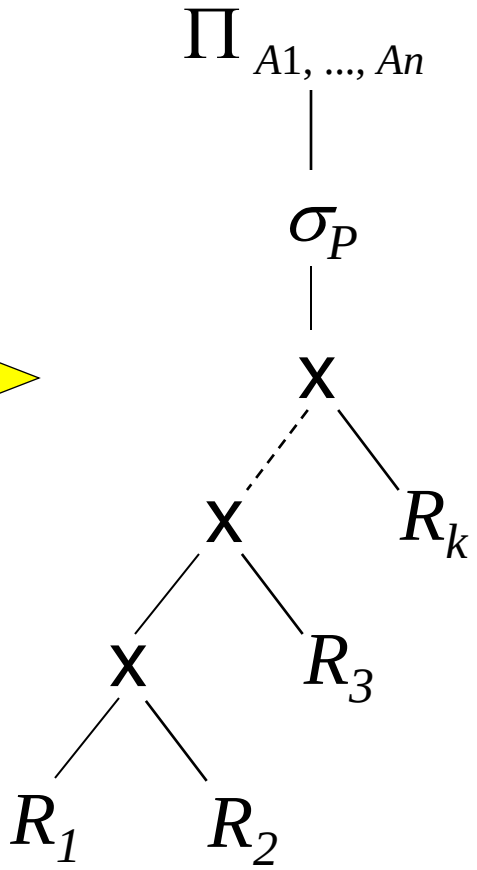
select $A_1, \ldots, A_n$

from $R_1, \ldots, R_k$

where P;

Übersetzung in die relationale Algebra:

$$\Pi_{A1, \ldots, An}(\sigma_P (R_1 \times \ldots \times R_k))$$



Anfragen über mehrere Relationen

Welche Studenten hören welche Vorlesungen?

```
select Name, Titel
from Studenten, hören, Vorlesungen
where Studenten.MatrNr = hören.MatrNr and
        hören.VorlNr = Vorlesungen.VorlNr;
```

Alternativ:

```
select s.Name, v.Titel
from Studenten s, hören h, Vorlesungen v
where s. MatrNr = h. MatrNr and
        h.VorlNr = v.VorlNr
```

```
Select s1.Name, s2.Name
From Studenten s1, hoeren h1, hoeren h2, Studenten s2
Where h1.VorlNr = h2.VorlNr and h1.MatrNr = s1.MatrNr and
    h2.MatrNr = s2.MatrNr
```

Professoren

PersNr	Name	Rang	Raum
2125	Sokrates	C4	226
2126	Russel	C4	232
2127	Kopernikus	C3	310
2133	Popper	C3	52
2134	Augustinus	C3	309
2136	Curie	C4	36
2137	Kant	C4	7

Studenten

MatrNr	Name	Semester
24002	Xenokrates	18
25403	Jonas	12
26120	Fichte	10
26830	Aristoxenos	8
27550	Schopenhauer	6
28106	Carnap	3
29120	Theophrastos	2
29555	Feuerbach	2

Vorlesungen

VorlNr	Titel	SWS	gelesen Von
5001	Grundzüge	4	2137
5041	Ethik	4	2125
5043	Erkenntnistheorie	3	2126
5049	Mäeutik	2	2125
4052	Logik	4	2125
5052	Wissenschaftstheorie	3	2126
5216	Bioethik	2	2126
5259	Der Wiener Kreis	2	2133
5022	Glaube und Wissen	2	2134
4630	Die 3 Kritiken	4	2137

voraussetzen

Vorgänger	Nachfolger
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5259

hören

MatrNr	VorlNr
26120	5001
27550	5001
27550	4052
28106	5041
28106	5052
28106	5216
28106	5259
29120	5001
29120	5041
29120	5049
29555	5022
25403	5022

prüfen

MatrNr	VorlNr	PersNr	Note
28106	5001	2126	1
25403	5041	2125	2
27550	4630	2137	2

Assistenten

PerslNr	Name	Fachgebiet	Boss
3002	Platon	Ideenlehre	2125
3003	Aristoteles	Syllogistik	2125
3004	Wittgenstein	Sprachtheorie	2126
3005	Rhetikus	Planetenbewegung	2127
3006	Newton	Keplersche Gesetze	2127
3007	Spinoza	Gott und Natur	2126

Mengenoperationen und geschachtelte Anfragen

Mengenoperationen **union, intersect, minus**

```
( select Name
  from Assistenten )
union
( select Name
  from Professoren);
```

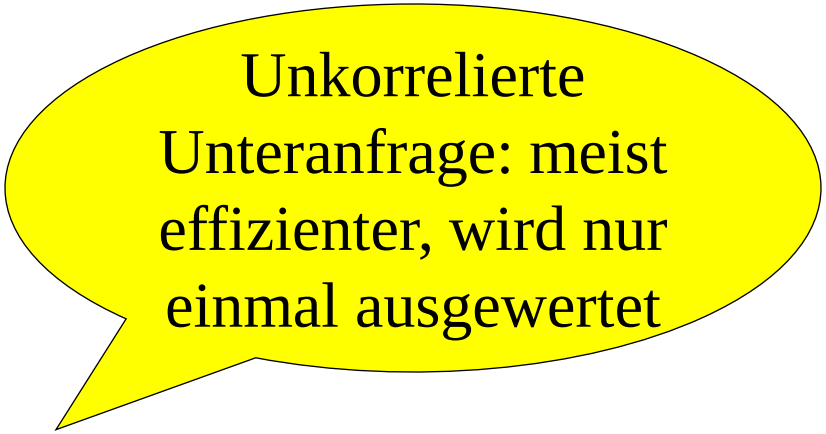
Existenzquantor **exists**

```
select p.Name
from Professoren p
where not exists ( select *
                   from Vorlesungen v
                   where v.gelesenVon = p.PersNr );
```

Existenzquantor **exists**


```
select p.Name
from Professoren p
where not exists ( select *
                   from Vorlesungen v
                   where v.gelesenVon = p.PersNr );
```

Mengenvergleich




```
select Name
from Professoren
where PersNr not in ( select gelesenVon
                      from Vorlesungen );
```

Der Vergleich mit "all"

Kein vollwertiger Allquantor!

```
select Name
from Studenten
where Semester >= all ( select Semester
                        from Studenten);
```

Aggregatfunktion und Gruppierung

Aggregatfunktionen **avg, max, min, count, sum**

```
select avg (Semester)
   from Studenten;

   select gelesenVon, sum (SWS)
   from Vorlesungen
   group by gelesenVon;

   select gelesenVon, Name, sum (SWS)
   from Vorlesungen, Professoren
   where gelesenVon = PersNr and Rang = ´C4´
   group by gelesenVon, Name
                having avg (SWS) >= 3;
```

Professoren

PersNr	Name	Rang	Raum
2125	Sokrates	C4	226
2126	Russel	C4	232
2127	Kopernikus	C3	310
2133	Popper	C3	52
2134	Augustinus	C3	309
2136	Curie	C4	36
2137	Kant	C4	7

Studenten

MatrNr	Name	Semester
24002	Xenokrates	18
25403	Jonas	12
26120	Fichte	10
26830	Aristoxenos	8
27550	Schopenhauer	6
28106	Carnap	3
29120	Theophrastos	2
29555	Feuerbach	2

Vorlesungen

VorlNr	Titel	SWS	gelesen Von
5001	Grundzüge	4	2137
5041	Ethik	4	2125
5043	Erkenntnistheorie	3	2126
5049	Mäeutik	2	2125
4052	Logik	4	2125
5052	Wissenschaftstheorie	3	2126
5216	Bioethik	2	2126
5259	Der Wiener Kreis	2	2133
5022	Glaube und Wissen	2	2134
4630	Die 3 Kritiken	4	2137

voraussetzen

Vorgänger	Nachfolger
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5259

hören

MatrNr	VorlNr
26120	5001
27550	5001
27550	4052
28106	5041
28106	5052
28106	5216
28106	5259
29120	5001
29120	5041
29120	5049
29555	5022
25403	5022

prüfen

MatrNr	VorlNr	PersNr	Note
28106	5001	2126	1
25403	5041	2125	2
27550	4630	2137	2

Assistenten

PerslNr	Name	Fachgebiet	Boss
3002	Platon	Ideenlehre	2125
3003	Aristoteles	Syllogistik	2125
3004	Wittgenstein	Sprachtheorie	2126
3005	Rhetikus	Planetenbewegung	2127
3006	Newton	Keplersche Gesetze	2127
3007	Spinoza	Gott und Natur	2126

Besonderheiten bei Aggregatoperationen

- SQL erzeugt pro Gruppe ein Ergebnistupel
- Deshalb müssen alle in der **select**-Klausel aufgeführten Attribute - außer den aggregierten – auch in der **group by**-Klausel aufgeführt werden
- Nur so kann SQL sicherstellen, dass sich das Attribut nicht innerhalb der Gruppe ändert

Ausführen einer Anfrage mit group by

Vorlesung x Professoren							
Vorl Nr	Titel	SWS	gelesen Von	PersNr	Name	Rang	Raum
5001	Grundzüge	4	2137	2125	Sokrates	C4	226
5041	Ethik	4	2125	2125	Sokrates	C4	226
...	...	...	...	...	...	...	...
4630	Die 3 Kritiken	4	2137	2137	Kant	C4	7

⇓ **where**-Bedingung

VorlNr	Titel	SWS	gelesen Von	PersNr	Name	Rang	Raum
5001	Grundzüge	4	2137	2137	Kant	C4	7
5041	Ethik	4	2125	2125	Sokrates	C4	226
5043	Erkenntnistheorie	3	2126	2126	Russel	C4	232
5049	Mäeutik	2	2125	2125	Sokrates	C4	226
4052	Logik	4	2125	2125	Sokrates	C4	226
5052	Wissenschaftstheorie	3	2126	2126	Russel	C4	232
5216	Bioethik	2	2126	2126	Russel	C4	232
4630	Die 3 Kritiken	4	2137	2137	Kant	C4	7

VorlNr	Titel	SWS	gelesenVon	PersNr	Name	Rang	Raum
5041	Ethik	4	2125	2125	Sokrates	C4	226
5049	Mäeutik	2	2125	2125	Sokrates	C4	226
4052	Logik	4	2125	2125	Sokrates	C4	226
5043	Erkenntnistheorie	3	2126	2126	Russel	C4	232
5052	Wissenschaftstheo.	3	2126	2126	Russel	C4	232
5216	Bioethik	2	2126	2126	Russel	C4	232
5001	Grundzüge	4	2137	2137	Kant	C4	7
4630	Die 3 Kritiken	4	2137	2137	Kant	C4	7

⇓ **having**-Bedingung

VorlN	Titel	SWS	gelesenVon	PersNr	Name	Rang	Raum
5041	Ethik	4	2125	2125	Sokrates	C4	226
5049	Mäeutik	2	2125	2125	Sokrates	C4	226
4052	Logik	4	2125	2125	Sokrates	C4	226
5001	Grundzüge	4	2137	2137	Kant	C4	7
4630	Die 3 Kritiken	4	2137	2137	Kant	C4	7

⇓ Aggregation (**sum**) und Projektion

gelesenVon	Name	**sum** (SWS)
2125	Sokrates	10
2137	Kant	8

Geschachtelte Anfrage (Forts.)

- Unteranfrage in der **where**-Klausel
- Welche Prüfungen sind besser als durchschnittlich verlaufen?

```
select *
 from prüfen
 where Note < ( select avg (Note)
                  from prüfen );
```

Geschachtelte Anfrage (Forts.)

- Unteranfrage in der **select**-Klausel
- Für jedes Ergebnistupel wird die Unteranfrage ausgeführt
- Man beachte, dass die Unteranfrage korreliert ist (greift auf Attribute der umschließenden Anfrage zu)

```
select PersNr, Name, ( select sum (SWS) as Lehrbelastung
                       from Vorlesungen
                       where gelesenVon=PersNr )
from Professoren;
```

Professoren

PersNr	Name	Rang	Raum
2125	Sokrates	C4	226
2126	Russel	C4	232
2127	Kopernikus	C3	310
2133	Popper	C3	52
2134	Augustinus	C3	309
2136	Curie	C4	36
2137	Kant	C4	7

Studenten

MatrNr	Name	Semester
24002	Xenokrates	18
25403	Jonas	12
26120	Fichte	10
26830	Aristoxenos	8
27550	Schopenhauer	6
28106	Carnap	3
29120	Theophrastos	2
29555	Feuerbach	2

Vorlesungen

VorlNr	Titel	SWS	gelesen Von
5001	Grundzüge	4	2137
5041	Ethik	4	2125
5043	Erkenntnistheorie	3	2126
5049	Mäeutik	2	2125
4052	Logik	4	2125
5052	Wissenschaftstheorie	3	2126
5216	Bioethik	2	2126
5259	Der Wiener Kreis	2	2133
5022	Glaube und Wissen	2	2134
4630	Die 3 Kritiken	4	2137

voraussetzen

Vorgänger	Nachfolger
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5259

hören

MatrNr	VorlNr
26120	5001
27550	5001
27550	4052
28106	5041
28106	5052
28106	5216
28106	5259
29120	5001
29120	5041
29120	5049
29555	5022
25403	5022

prüfen

MatrNr	VorlNr	PersNr	Note
28106	5001	2126	1
25403	5041	2125	2
27550	4630	2137	2

Assistenten

PerslNr	Name	Fachgebiet	Boss
3002	Platon	Ideenlehre	2125
3003	Aristoteles	Syllogistik	2125
3004	Wittgenstein	Sprachtheorie	2126
3005	Rhetikus	Planetenbewegung	2127
3006	Newton	Keplersche Gesetze	2127
3007	Spinoza	Gott und Natur	2126

Unkorrelierte versus korrelierte Unteranfragen

- korrelierte Formulierung

```
select s.*
from Studenten s
where exists
        (select p.*
        from Professoren
        where p.GebDatum > s.GebDatum);
```

- Äquivalente unkorrelierte Formulierung

```
select s.*
from Studenten s
where s.GebDatum <
        (select max (p.GebDatum)
         from Professoren p);
```

- Vorteil: Unteranfrageergebnis kann materialisiert werden
- Unteranfrage braucht nur einmal ausgewertet zu werden

Entschachtelung korrelierter Unteranfragen -- Forts.

```
select a.*
from Assistenten a
where exists
    ( select p.*
      from Professoren p
      where a.Boss = p.PersNr and p.GebDatum>a.GebDatum);
```

- Entschachtelung durch Join

```
select a.*
from Assistenten a, Professoren p
where a.Boss=p.PersNr and p.GebDatum > a.GebDatum;
```

Verwertung der Ergebnismenge einer Unteranfrage

```
select tmp.MatrNr, tmp.Name, tmp.VorlAnzahl
from (select s.MatrNr, s.Name, count(*) as VorlAnzahl
        from Studenten s, hören h
        where s.MatrNr=h.MatrNr
        group by s.MatrNr, s.Name)  tmp
where tmp.VorlAnzahl > 2;
```

MatrNr	Name	VorlAnzahl
28106	Carnap	4
29120	Theophrastos	3

Decision-Support-Anfrag mit geschachtelten Unteranfragen

```
select  h.VorlNr, h.AnzProVorl, g.GesamtAnz,
        h.AnzProVorl/g.GesamtAnz as Marktanteil
from    ( select VorlNr, count(*) as AnzProVorl
          from hören
          group by VorlNr ) h,
        ( select count (*) as GesamtAnz
          from Studenten) g;
```

Casting der Integer zu Decimal

```
select  h.VorlNr, h.AnzProVorl, g.GesamtAnz,
        cast(h.AnzProVorl as decimal(6,2)) / g.GesamtAnz
        as Marktanteil
from    ( select VorlNr, count(*) as AnzProVorl
          from hören
          group by VorlNr ) h,
        ( select count (*) as GesamtAnz
          from Studenten) g;
```

Das Ergebnis der Anfrage

VorlNr	AnzProVorl	GesamtAnz	Marktanteil
4052	1	8	.125
5001	4	8	.5
5022	2	8	.25
...	...	...	...

Modularisierung mit „with“

Professoren				
PersNr	**Name**	**Rang**	**Raum**	**FakName**
2125	Sokrates	C4	226	Philosophie
2126	Russel	C4	232	Philosophie
2127	Kopernikus	C3	310	Physik
2133	Popper	C3	52	Philosophie
2134	Augustinus	C3	309	Theologie
2136	Curie	C4	36	Physik
2137	Kant	C4	7	Philosophie

StudentenGF				
MatrNr	**Name**	**Semester**	**Geschlecht**	**FakName**
24002	Xenokrates	18	M	Philosophie
25403	Jonas	12	W	Theologie
26120	Fichte	10	W	Philosophie
26830	Aristoxenos	8	M	Philosophie
27550	Schopenhauer	6	M	Philosophie
28106	Carnap	3	W	Physik
29120	Theophrastos	2	M	Physik
29555	Feuerbach	2	W	Theologie

Frauenanteil pro Fakultät

```
select anz.FakName, anz.AnzStudenten, anzw.AnzWeiblich,
(cast(anzw.AnzWeiblich as decimal(5,2))/anz.AnzStudenten *
100) as ProzentWeiblich
from
        (select s.FakName, count(*) as AnzStudenten
         from StudentenGF s
         group by s.FakNAme) as anz,
        (select sw.FakName,count(*) as AnzWeiblich
         from StudentenGF sw
         where sw.Geschlecht ='W'
         group by sw.FakName) as anzw
where anz.FakName = anzw.FakName
```

Weitere Anfragen mit Unteranfragen

```
( select Name
  from Assistenten )
union
( select Name
  from Professoren );

  select Name
  from Professoren
  where PersNr not in ( select gelesenVon
                        from Vorlesungen );
```

```
select Name
from Studenten
where Semester > = all ( select Semester
                         from Studenten );
```

Quantifizierte Anfragen in SQL

- Existenzquantor: **exists**

```
select Name
from Professoren
where not exists ( select *
                   from Vorlesungen
                   where gelesen Von = PersNr );
```

www-db.in.tum.de/db2face/index.shtml

Allquantifizierung

- SQL-92 hat keinen Allquantor
- Allquantifizierung muß also durch eine äquivalente Anfrage mit Existenzquantifizierung ausgedrückt werden
- Kalkülformulierung der Anfrage: Wer hat **alle** vierstündigen Vorlesungen gehört?

$$\{s \mid s \in \text{Studenten} \wedge \forall v \in \text{Vorlesungen}\ (v.SWS=4 \Rightarrow \exists h \in \text{hören}\ (h.VorlNr=v.VorlNr \wedge h.MatrNr=s.MatrNr))\}$$

- Elimination von $\forall$ und $\Rightarrow$
- Dazu sind folgende Äquivalenzen anzuwenden

$$\forall t \in R\,(P(t)) = \neg(\exists t \in R(\neg P(t)))$$

$$R \Rightarrow T = \neg R \vee T$$

Umformung des Kalkül-Ausdrucks ...

- Elimination $\forall$

$$\{s \mid s \in \text{Studenten} \wedge \neg (\exists v \in \text{Vorlesungen} \neg (v.SWS=4 \Rightarrow \exists h \in \text{hören} (h.VorlNr=v.VorlNr \wedge h.MatrNr=s.MatrNr))\}$$

- Elimination $\Rightarrow$

$$\{s \mid s \in \text{Studenten} \wedge \neg (\exists v \in \text{Vorlesungen} \neg(\neg(v.SWS=4) \vee \exists h \in \text{hören} (h.VorlNr=v.VorlNr \wedge h.MatrNr=s.MatrNr))\}$$

- Anwendung von DeMorgan ergibt schließlich:

$$\{s \mid s \in \text{Studenten} \wedge \neg (\exists v \in \text{Vorlesungen} (v.SWS=4 \wedge \neg(\exists h \in \text{hören} (h.VorlNr=v.VorlNr \wedge h.MatrNr=s.MatrNr))))\}$$

- SQL-Umsetzung folgt direkt:

```
select s.*
from Studenten s
where not exists
   (select *
    from Vorlesungen v
    where v.SWS = 4 and not exists
        (select *
         from hören h
         where h.VorlNr = v.VorlNr and h.MatrNr=s.MatrNr ) );
```

Allquantifizierung durch count-Aggregation

- Allquantifizierung kann immer auch durch eine **count-**Aggregation ausgedrückt werden
- Wir betrachten dazu eine etwas einfachere Anfrage, in der wir die *(MatrNr* der) Studenten ermitteln wollen, die *alle* Vorlesungen hören:

```
select h.MatrNr
from hören h
group by h.MatrNr
        having count (*) = (select count (*) from Vorlesungen);
```

Herausforderung

- Wie formuliert man die komplexere Anfrage: Wer hat alle vierstündigen Vorlesungen gehört
- Grundidee besteht darin, vorher durch einen Join die Studenten/Vorlesungs-Paare einzuschränken und danach das Zählen durchzuführen

Nullwerte

- unbekannter Wert
- wird vielleicht später nachgereicht
- Nullwerte können auch im Zuge der Anfrageauswertung entstehen (Bsp. äußere Joins)
- manchmal sehr überraschende Anfrageergebnisse, wenn Nullwerte vorkommen

```
select count (*)
from Studenten
where Semester < 13 or Semester > =13
```

- Wenn es Studenten gibt, deren *Semester*-Attribut den Wert **null** hat, werden diese nicht mitgezählt
- Der Grund liegt in folgenden Regeln für den Umgang mit **null**-Werten begründet:

Auswertung bei Null-Werten

1. In arithmetischen Ausdrücken werden Nullwerte propagiert, d.h. sobald ein Operand **null** ist, wird auch das Ergebnis **null**. Dementsprechend wird z.B. **null** + 1 zu **null** ausgewertet- aber auch null * 0 wird zu **null** ausgewertet.

2. SQL hat eine dreiwertige Logik, die nicht nur **true** und **false** kennt, sondern auch einen dritten Wert **unknown**. Diesen Wert liefern Vergleichsoperationen zurück, wenn mindestens eines ihrer Argumente **null** ist. Beispielsweise wertet SQL das Prädikat *(PersNr=...)* immer zu **unknown** aus, wenn die *PersNr* des betreffenden Tupels den Wert **null** hat.

3. Logische Ausdrücke werden nach den folgenden Tabellen berechnet:

Diese Berechnungsvorschriften sind recht intuitiv. Unknown or true wird z.B. zu **true** - die Disjunktion ist mit dem **true**-Wert des rechten Arguments immer erfüllt, unabhängig von der Belegung des linken Arguments. Analog ist **unknown and false** automatisch **false** - keine Belegung des linken Arguments könnte die Konjunktion mehr erfüllen.

4. In einer **where**-Bedingung werden nur Tupel weitergereicht, für die die Bedingung **true** ist. Insbesondere werden Tupel, für die die Bedingung zu **unknown** auswertet, nicht ins Ergebnis aufgenommen.
5. Bei einer Gruppierung wird **null** als ein eigenständiger Wert aufgefaßt und in eine eigene Gruppe eingeordnet.

not	
true	false
unknown	unknown
false	true

and	true	unknown	false
true	true	unknown	false
unknown	unknown	unknown	false
false	false	false	false

or	true	unknown	false
true	true	true	true
unknown	true	unknown	unknown
false	true	unknown	false

Spezielle Sprachkonstrukte ("syntaktischer Zucker")

```
select *
from Studenten
where Semester > = 1 and Semester < = 4;

select *
from Studenten
where Semester between 1 and 4;

select *
from Studenten
where Semester in (1,2,3,4);
```

```
select *
from Studenten
where Name like `T%eophrastos‘;

select distinct s.Name
from Vorlesungen v, hören h, Studenten s
where s.MatrNr = h.MatrNr and h.VorlNr = v.VorlNr and
        v.Titel like `%thik%‘;
```

Das case-Konstrukt

```
select MatrNr, ( case when Note < 1.5 then ´sehr gut´
                      when Note < 2.5 then ´gut´
                      when Note < 3.5 then ´befriedigend´
                      when Note < 4.0 then ´ausreichend´
                      else ´nicht bestanden´end)
from prüfen;
```

- Die **erste** qualifizierende **when**-Klausel wird ausgeführt

Vergleiche mit like

Platzhalter "%" ; "_"

- "%" steht für beliebig viele (auch gar kein) Zeichen
- "_" steht für genau ein Zeichen

```
select *
from Studenten
where Name like ´T%eophrastos´;
```

```
select distinct Name
from Vorlesungen v, hören h, Studenten s
where s.MatrNr = h.MatrNr and h.VorlNr = v.VorlNr and
        v.Titel = ´%thik%´;
```

Joins in SQL-92

- **cross join:** Kreuzprodukt
- **natural join:** natürlicher Join
- Join oder inner join: Theta-Join
- left, right oder full outer join: äußerer Join
- union join: Vereinigungs-Join (wird hier nicht vorgestellt)

select *
from R_1, R_2
where $= \mathrm{R}_1.A = R_2.B$;

select *
from R_1 **join** R_2 **on** $\mathrm{R}_1.A = R_2.B$;

Äußere Joins

select p.PersNr, p.Name, f.PersNr, f.Note, f.MatrNr,
s.MatrNr, s.Name
from Professoren p **left outer join**
(prüfen f **left outer join** Studenten s **on** f.MatrNr= s.MatrNr)
on p.PersNr=f.PersNr;

PersNr	p.Name	f.PersNr	f.Note	f.MatrNr	s.MatrNr	s.Name
2126	Russel	2126	1	28106	28106	Carnap
2125	Sokrates	2125	2	25403	25403	Jonas
2137	Kant	2137	2	27550	27550	Schopen-hauer
2136	Curie	-	-	-	-	-
:	:	:	:	:	:	:

Äußere Joins

```
select p.PersNr, p.Name, f.PersNr, f.Note, f.MatrNr,
       s.MatrNr, s.Name
from Professoren p right outer join
                    (prüfen f right outer join Studenten s on
                     f.MatrNr= s.MatrNr)
     on p.PersNr=f.PersNr;
```

PersNr	p.Name	f.PersNr	f.Note	f.MatrNr	s.MatrNr	s.Name
2126	Russel	2126	1	28106	28106	Carnap
2125	Sokrates	2125	2	25403	25403	Jonas
2137	Kant	2137	2	27550	27550	Schopen-hauer
-	-	-	-	-	26120	Fichte
:	:	:	:	:	:	:

Äußere Joins

```
select p.PersNr, p.Name, f.PersNr, f.Note, f.MatrNr,
s.MatrNr, s.Name
from Professoren p full outer join
        (prüfen f full outer join Studenten s on
        f.MatrNr= s.MatrNr)
        on p.PersNr=f.PersNr;
```

p.PersNr	p.Name	f.PersNr	f.Note	f.MatrNr	s.MatrNr	s.Name
2126	Russel	2126	1	28106	28106	Carnap
2125	Sokrates	2125	2	25403	25403	Jonas
2137	Kant	2137	2	27550	27550	Schopen-hauer
-	-	-	-	-	26120	Fichte
⋮	⋮	⋮	⋮	⋮	⋮	⋮
2136	Curie	-	-	-	-	-
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Rekursion

select Vorgänger

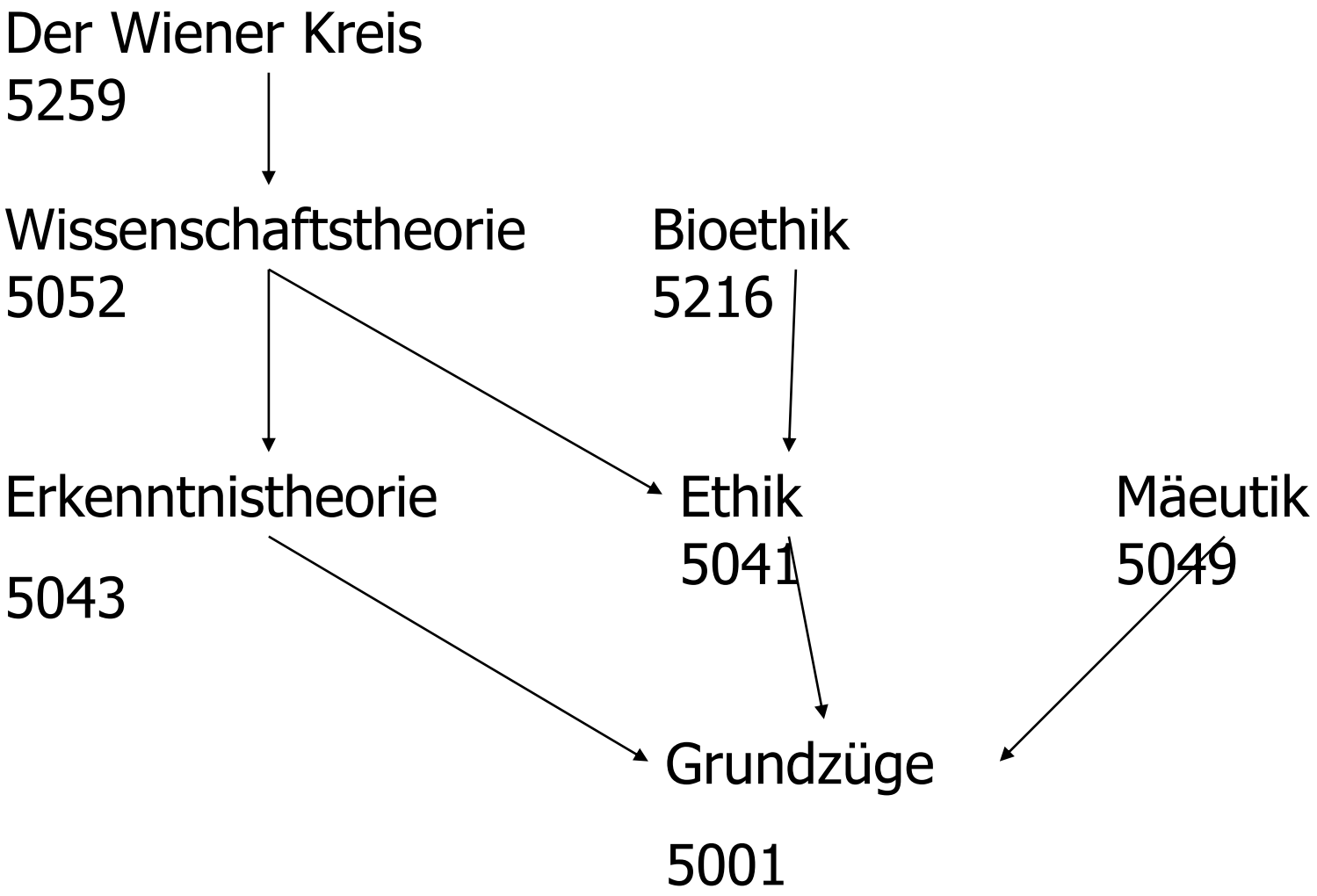
from voraussetzen, Vorlesungen

where Nachfolger= VorlNr and

Titel= `Der Wiener Kreis´

voraussetzen	
Vorgänger	**Nachfolger**
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5259

Vorlesungen			
VorlNr	**Titel**	**SWS**	**gelesen Von**
5001	Grundzüge	4	2137
5041	Ethik	4	2125
5043	Erkenntnistheorie	3	2126
5049	Mäeutik	2	2125
4052	Logik	4	2125
5052	Wissenschaftstheorie	3	2126
5216	Bioethik	2	2126
5259	Der Wiener Kreis	2	2133
5022	Glaube und Wissen	2	2134
4630	Die 3 Kritiken	4	2137


Der Wiener Kreis
5259
Wissenschaftstheorie
5052
Bioethik
5216
Erkenntnistheorie
5043
Ethik
5041
Mäeutik
5049
Grundzüge
5001

Rekursion

select v1.Vorgänger

from voraussetzen v1, voraussetzen v2, Vorlesungen v

where v1.Nachfolger= v2.Vorgänger **and**

v2.Nachfolger= v.VorlNr **and**

v.Titel=`Der Wiener Kreis´

Rekursion

select v1.Vorgänger

from voraussetzen v1, voraussetzen v2, Vorlesungen v

where v1.Nachfolger= v2.Vorgänger **and**

v2.Nachfolger= v.VorlNr **and**

v.Titel=`Der Wiener Kreis´

voraussetzen	
Vorgänger	**Nachfolger**
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5259

voraussetzen	
Vorgänger	**Nachfolger**
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5259

Vorlesungen			
VorlNr	**Titel**	**SWS**	**gelesen Von**
5001	Grundzüge	4	2137
5041	Ethik	4	2125
5043	Erkenntnistheorie	3	2126
5049	Mäeutik	2	2125
4052	Logik	4	2125
5052	Wissenschaftstheorie	3	2126
5216	Bioethik	2	2126
5259	Der Wiener Kreis	2	2133
5022	Glaube und Wissen	2	2134
4630	Die 3 Kritiken	4	2137

Rekursion

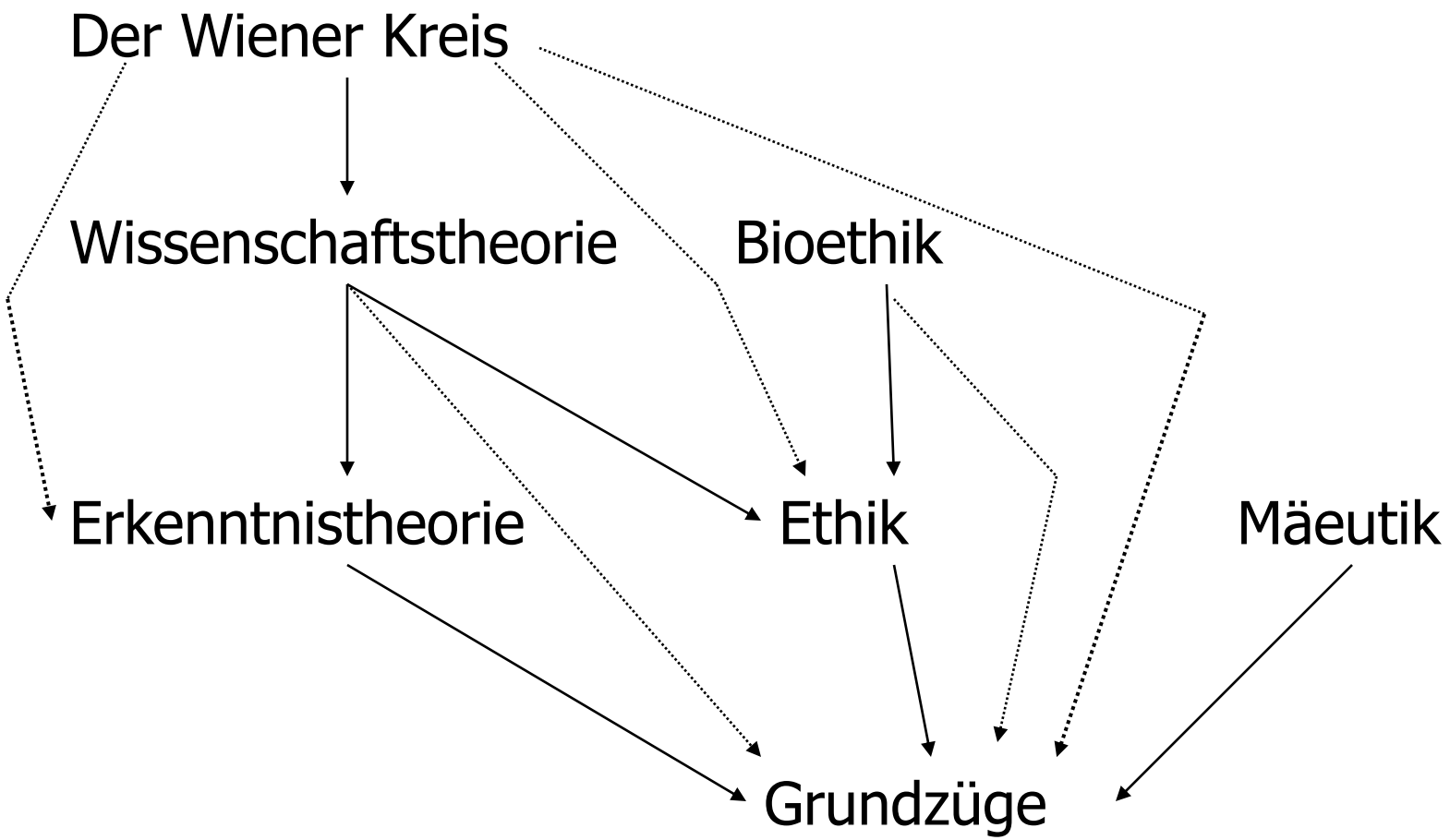
```
select v1.Vorgänger
from voraussetzen v1, voraussetzen v2, Vorlesungen v
where v1.Nachfolger= v2.Vorgänger and
        v2.Nachfolger= v.VorlNr and
        v.Titel=`Der Wiener Kreis´
```

Vorgänger des „Wiener Kreises“ der Tiefe n

```
select v1.Vorgänger
from voraussetzen v1
        ⋮
        voraussetzen vn_minus_1
        voraussetzen vn,

        Vorlesungen v

where v1.Nachfolger= v2.Vorgänger and
        ⋮

        vn_minus_1.Nachfolger= vn.Vorgänger and

        vn.Nachfolger = v.VorlNr and

        v.Titel= `Der Wiener Kreis´
```

Transitive Hülle

$$trans_{A,B}(R) = \{(a,b) \mid \exists k \in IN\ (\exists \Gamma 1, \ldots, \Gamma k \in R\ (\Gamma 1.A = \Gamma 2.B \wedge \vdots \Gamma k\text{-}1.A = \Gamma k.B \wedge \Gamma 1.A = a \wedge \Gamma k.B = b))\}$$

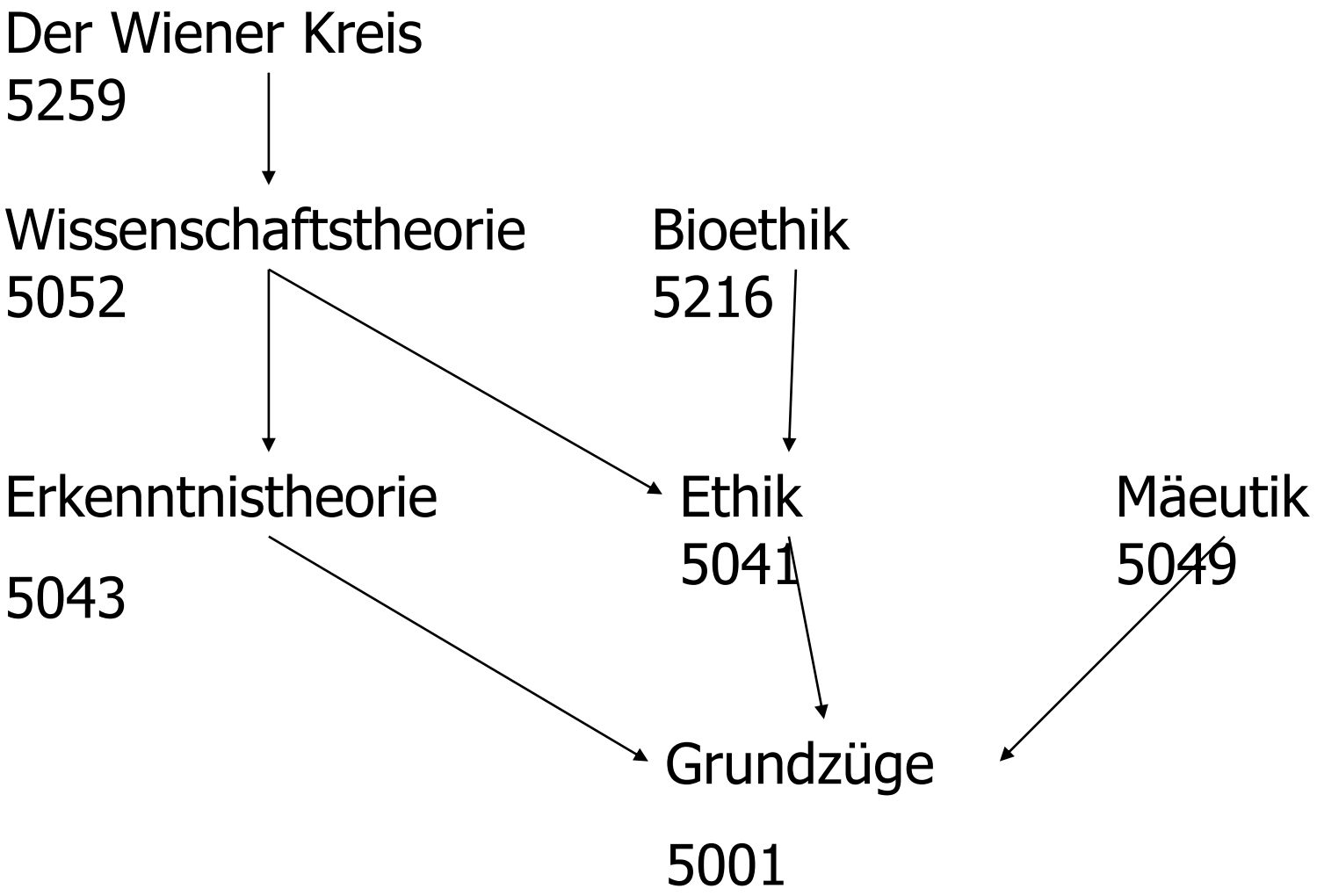

Der Wiener Kreis
Wissenschaftstheorie
Bioethik
Erkenntnistheorie
Ethik
Mäeutik
Grundzüge

Die connect by-Klausel

```
select Titel
from Vorlesungen
where VorlNr in (select Vorgänger
                 from voraussetzen
                 connect by Nachfolger= prior Vorgänger
                 start with Nachfolger= (select VorlNr
                                         from Vorlesungen
                                         where Titel= `Der
                                           Wiener Kreis´));
```

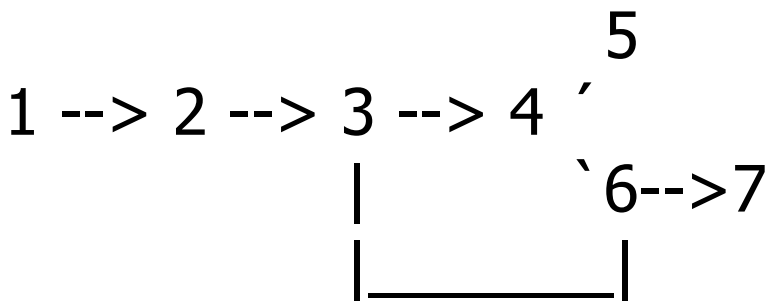
voraussetzen	
Vorgänger	**Nachfolger**
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5259

Vorlesungen			
VorlNr	**Titel**	**SWS**	**gelesen Von**
5001	Grundzüge	4	2137
5041	Ethik	4	2125
5043	Erkenntnistheorie	3	2126
5049	Mäeutik	2	2125
4052	Logik	4	2125
5052	Wissenschaftstheorie	3	2126
5216	Bioethik	2	2126
5259	Der Wiener Kreis	2	2133
5022	Glaube und Wissen	2	2134
4630	Die 3 Kritiken	4	2137


Der Wiener Kreis
5259
Wissenschaftstheorie
5052
Bioethik
5216
Erkenntnistheorie
5043
Ethik
5041
Mäeutik
5049
Grundzüge
5001

Grundzüge
Ethik
Erkenntnistheorie
Wissenschaftstheorie

Rekursion in Prolog/Datalog

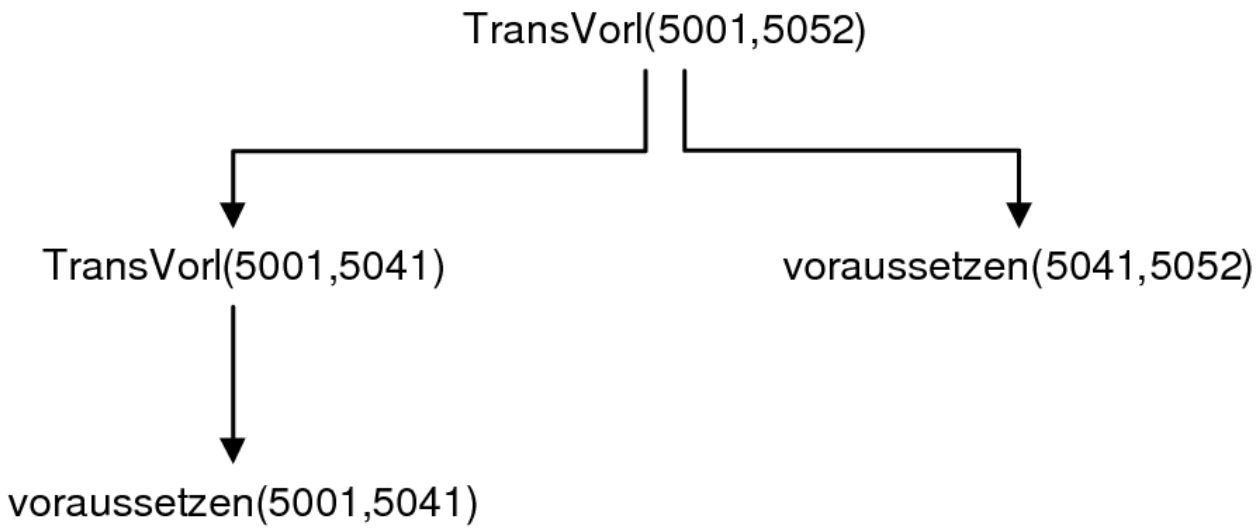


- kante(1,2).
- kante(2,3).
- kante(3,4).
- kante(4,5).
- kante(4,6).
- kante(6,7).
- kante(3,6).

- pfad(V,N) :- kante(V,N).
- pfad(V,N) :- kante(V,Z),pfad(Z,N).

Transitive Hülle der Relation voraussetzen

- TransVorl(V,N) :- voraussetzen(V,N).
- TransVorl(V,N) :- TransVorl(V,Z), voraussetzen(Z,N).



Rekursion in DB2/SQL99: gleiche Anfrage

```
with TransVorl (Vorg, Nachf)
as (select Vorgänger, Nachfolger from voraussetzen
        union all
    select t.Vorg, v.Nachfolger
    from TransVorl t, voraussetzen v
    where t.Nachf= v.Vorgänger)

select Titel from Vorlesungen where VorlNr in
    (select Vorg from TransVorl where Nachf in
        (select VorlNr from Vorlesungen
         where Titel= `Der Wiener Kreis´) )
```

- zuerst wird eine temporäre Sicht *TransVorl* mit der **with-**Klausel angelegt

- Diese Sicht *TransVorl* ist rekursiv definiert, da sie selbst in der Definition vorkommt

- Aus dieser Sicht werden dann die gewünschten Tupel extrahiert

- Ergebnis ist natürlich wie gehabt

Veränderung am Datenbestand

Einfügen von Tupeln

```
insert into hören
        select MatrNr, VorlNr
        from Studenten, Vorlesungen
        where Titel= `Logik´;

insert into Studenten (MatrNr, Name)
        values (28121,  `Archimedes´);
```

Studenten		
MatrNr	Name	Semester
⋮	⋮	⋮
29120	Theophrastos	2
29555	Feuerbach	2
28121	Archimedes	-

Veränderungen am Datenbestand

Löschen von Tupeln

delete Studenten

where Semester > 13;

Verändern von Tupeln

update Studenten

set Semester= Semester + 1;

Zweistufiges Vorgehen bei Änderungen

1. die Kandidaten für die Änderung werden ermittelt und "markiert"
2. die Änderung wird an den in Schritt 1. ermittelten Kandidaten durchgeführt

Anderenfalls könnte die Änderungsoperation von der Reihenfolge der Tupel abhängen, wie folgendes Beispiel zeigt:

```
delete from voraussetzen
    where Vorgänger in (select Nachfolger
                        from voraussetzen);
```

voraussetzen	
Vorgänger	Nachfolger
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5229

Ohne einen Markierungsschritt hängt das Ergebnis dieser Anfrage von der Reihenfolge der Tupel in der Relation ab. Eine Abarbeitung in der Reihenfolge der Beispielausprägung würde das letzte Tupel (5052, 5229) fälschlicherweise erhalten, da vorher bereits alle Tupel mit 5052 als *Nachfolger* entfernt wurden.

Sichten ...

für den Datenschutz

```
create view prüfenSicht as
    select MatrNr, VorlNr, PersNr
    from prüfen
```

Sichten ...

für den Datenschutz

```
create view prüfenSicht as
    select MatrNr, VorlNr, PersNr
    from prüfen
```

Statistische Sicht

```
create view PruefGuete(Name, GueteGrad) as
      (select prof.Name, avg(pruef.Note)
       from Professoren prof join pruefen pruef on
                        prof.PersNr = pruef.PersNr
       group by prof.Name, prof.PersNr
       having count(*) > 50)
```

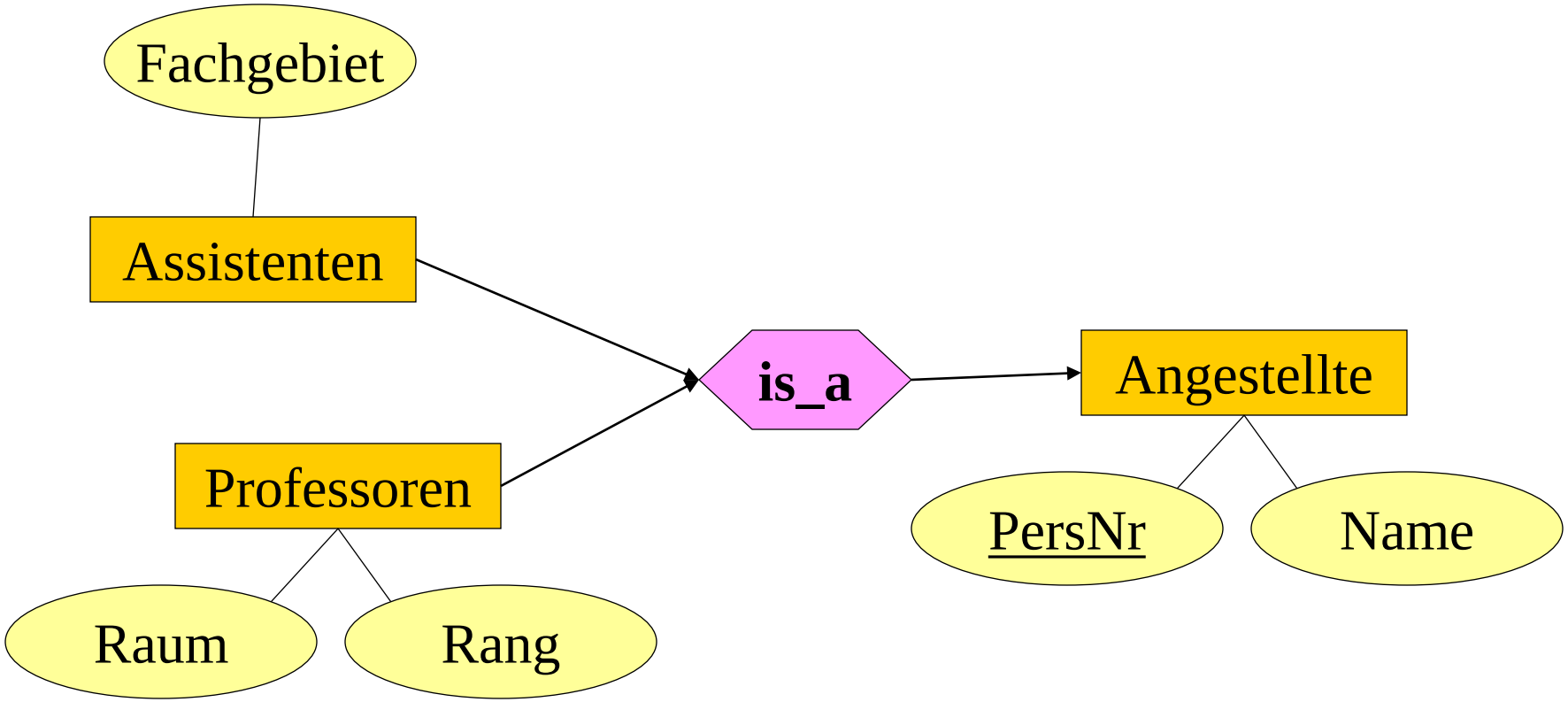
Sichten ...

für die Vereinfachung von Anfagen

```
create view StudProf (Sname, Semester, Titel, Pname) as
    select s.Name, s.Semester, v.Titel, p.Name
    from Studenten s, hören h, Vorlesungen v, Professoren p
    where s.Matr.Nr=h.MatrNr and h.VorlNr=v.VorlNr and
                v.gelesenVon = p.PersNr
```

```
select distinct Semester
from StudProf
where PName=`Sokrates‘;
```

Relationale Modellierung der Generalisierung



Angestellte: {[*PersNr, Name*]}

Professoren: {[*PersNr, Rang, Raum*]}

Assistenten: {[*PersNr, Fachgebiet*]}

Sichten zur Modellierung von Generalisierung

```
create table Angestellte
        (PersNr     integer not null,
        Name        varchar (30) not null);
create table ProfDaten
        (PersNr     integer not null,
        Rang        character(2),
        Raum        integer);
create table AssiDaten
        (PersNr          integer not null,
        Fachgebiet       varchar(30),
        Boss             integer);
```

```
create view Professoren as
        select *
        from Angestellte a, ProfDaten d
        where a.PersNr=d.PersNr;
create view Assistenten as
        select *
        from Angestellte a, AssiDaten d
        where a.PersNr=d.PersNr;
```

➔ Untertypen als Sicht

```
create table Professoren
        (PersNr         integer not null,
        Name            varchar (30) not null,
        Rang            character (2),
        Raum            integer);
create table Assistenten
        (PersNr         integer not null,
        Name            varchar (30) not null,
        Fachgebiet      varchar (30),
        Boss            integer);
create table AndereAngestellte
        (PersNr         integer not null,
        Name            varchar (30) not null);
```

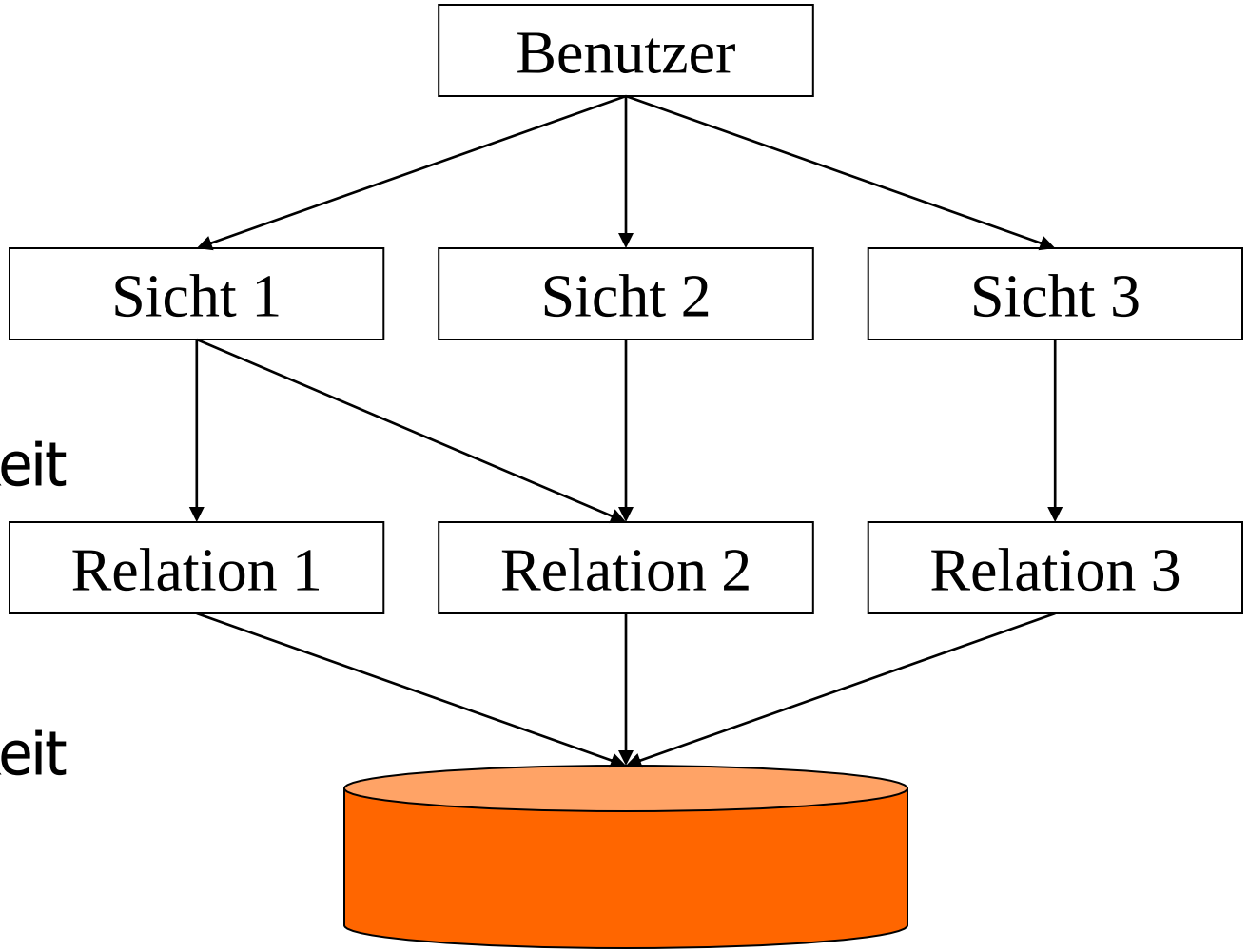
```
create view Angestellte as
        (select PersNr, Name
        from Professoren)
          union
        (select PersNr, Name
        from Assistenten)
         union
        (select*
        from AndereAngestellte);
```

➔ Obertypen als Sicht

Sichten zur Gewährleistung von Datenunabhängigkeit

logische Datenunabhängigkeit

physische Datenunabhängigkeit



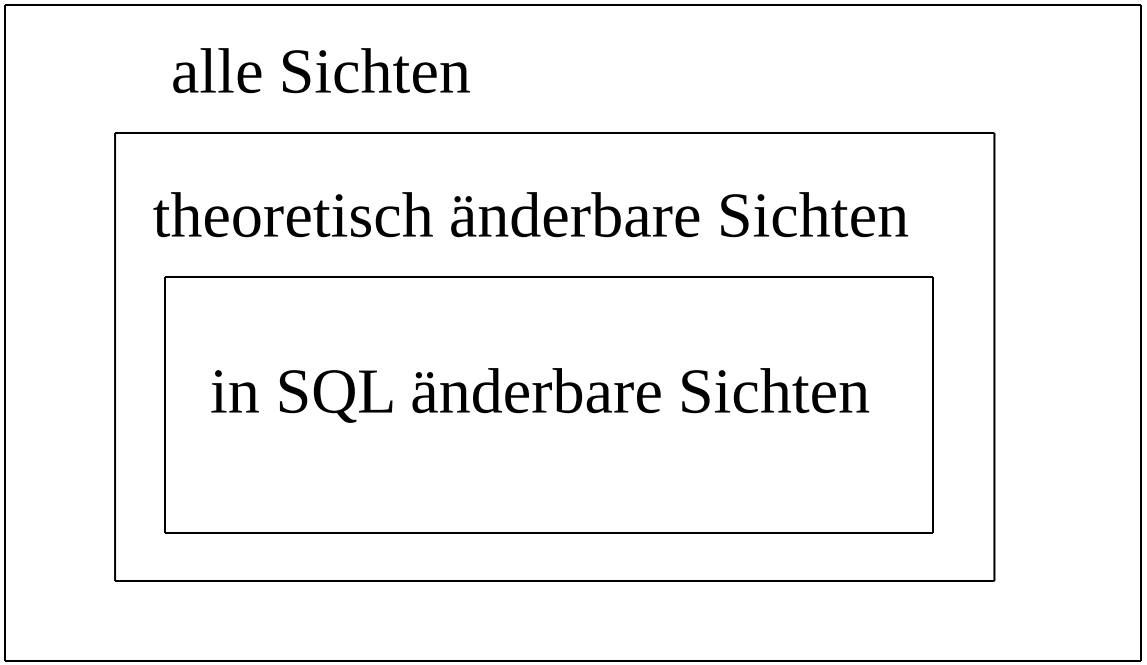
Änderbarkeit von Sichten

Beispiele für nicht änderbare Sichten

```
create view WieHartAlsPrüfer (PersNr, Durchschnittsnote) as
    select PersNr, avg(Note)
    from prüfen
    group by PersNr;
create view VorlesungenSicht as
    select Titel, SWS, Name
    from Vorlesungen, Professoren
    where gelesen Von=PersNr;
insert into VorlesungenSicht
    values (`Nihilismus‘, 2, `Nobody‘);
```

Änderbarkeit von Sichten

- in SQL
  - nur eine Basisrelation
  - Schlüssel muß vorhanden sein
  - keine Aggregatfunktionen, Gruppierung und Duplikateliminierung



Embedded SQL

```
#include <stdio.h>
/*Kommunikationsvariablen deklarieren */
exec sql begin declare section;
        varchar user_passwd[30];
        int exMatrNr;
exec sql end declare section;
exec sql include SQLCA;
main()
{
        printf("Name/Password:");
        scanf("%", user_passwd.arr);
```

```
user_passwd.len=strlen(user_passwd.arr);
exec sql whenever sqlerror goto error;
exec sql connect :user_passwd;
while (1) {
        printf("Matrikelnummer (0 zum beenden):");
        scanf("%d", &ecMatrNr);
        if (!exMatrNr) break;
        exec sql delete from Studenten
                where MatrNr= :exMatrNr;
}
exec sql commit work release;
exit(0);
```

```
error:
exec sql whenever sqlerror continue;
exec sql rollback work release;
printf("fehler aufgetreten!\n");
exit(-1);
}
```

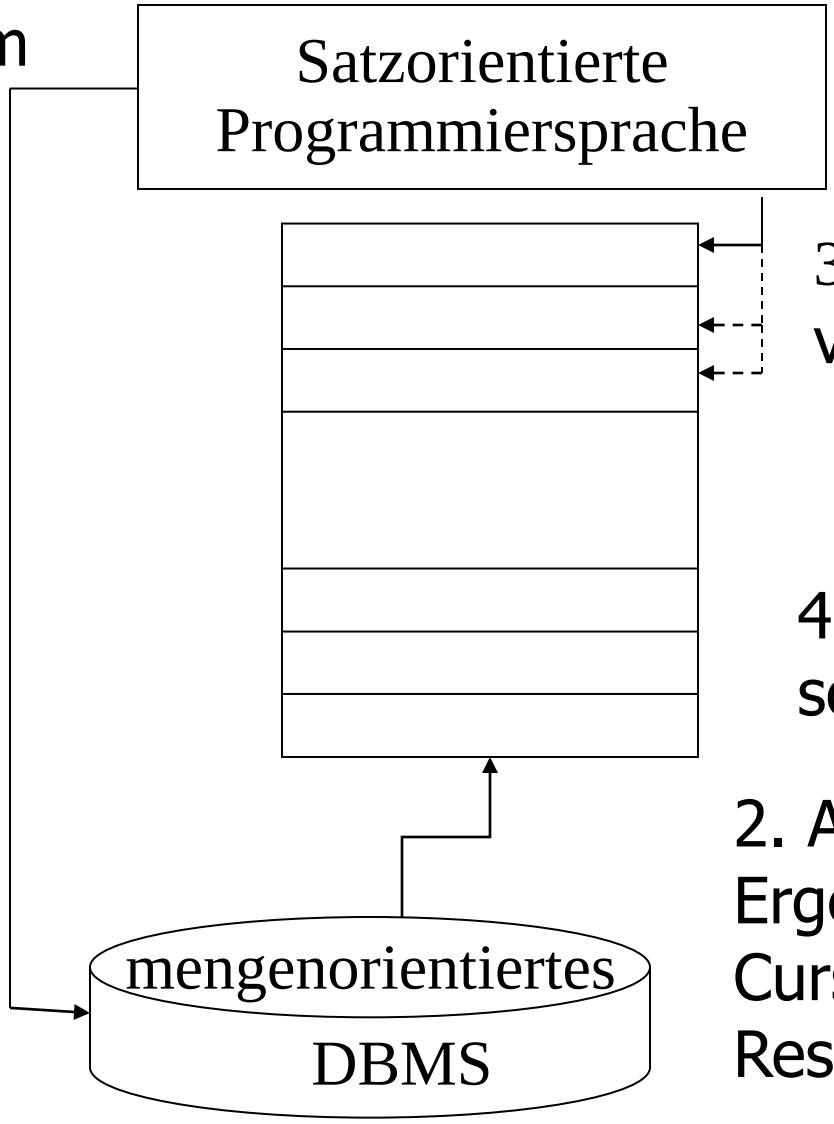
Anfragen in Anwendungsprogrammen

- genau ein Tupel im Ergebnis

```
exec sql select avg (Semester)
        into :avgsem
        from Studenten;
```

Anfragen in Anwendungsprogrammen

- mehrere Tupel im Ergebnis



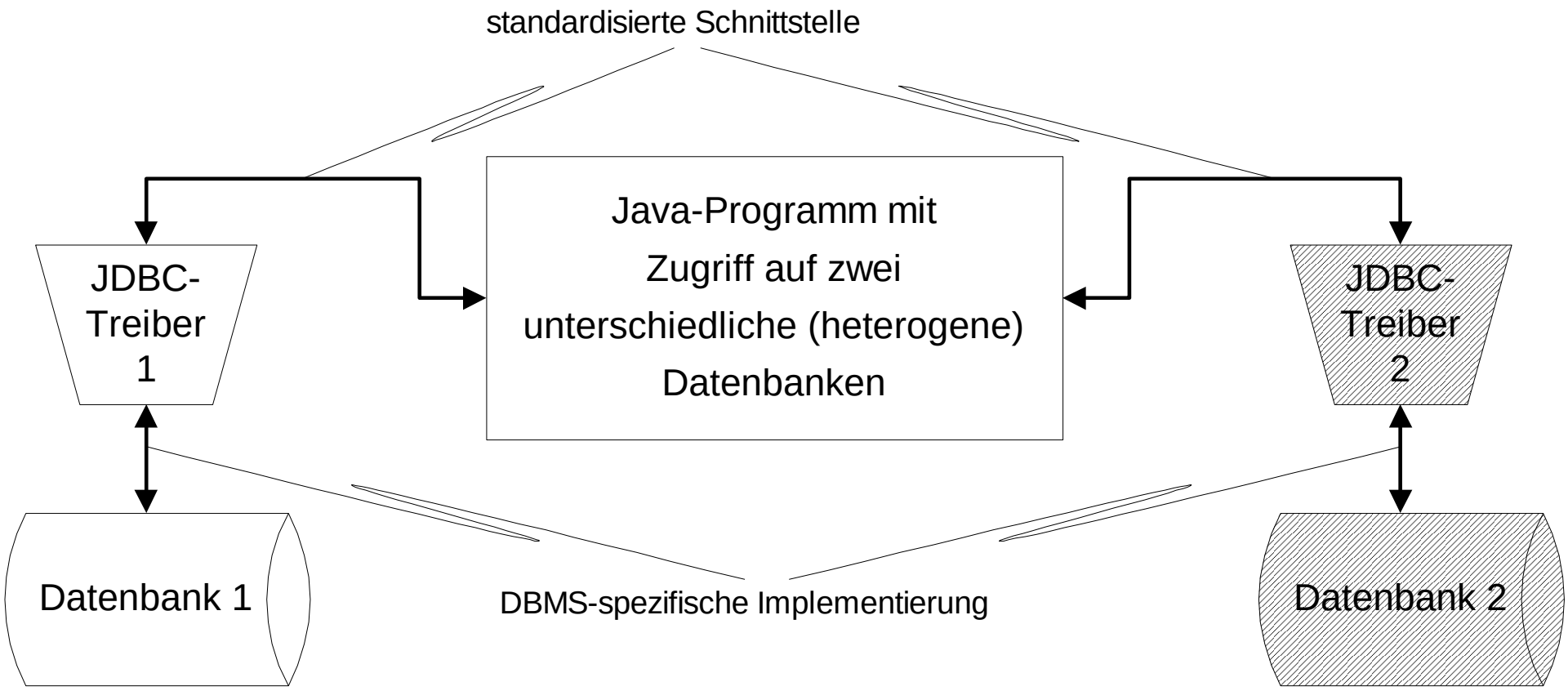
Cursor-Schnittstelle in SQL

1. **exec sql declare** c4profs **cursor for**
   select Name, Raum
   from Professoren
   where Rang='C4';
2. **exec sql open** c4profs;
3. **exec sql fetch** c4profs into :pname, :praum;
4. **exec sql close** c4profs;

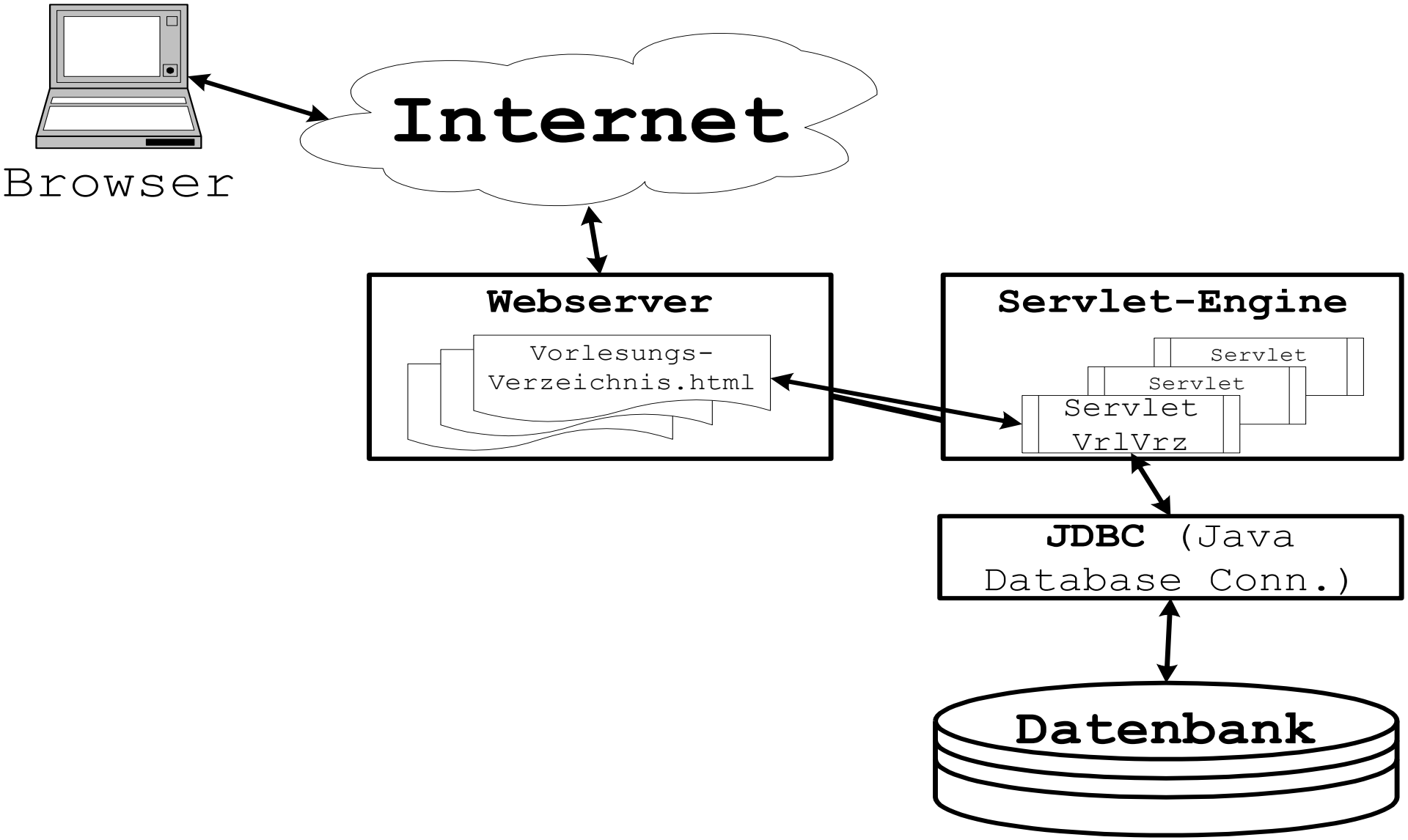
JDBC: Java Database Connectivity

- Standardisierte Schnittstelle zur Anbindung von relationalen Datenbanken an Java
- Wird heute fast immer für die Anbindung von Datenbanken an das Internet/Web verwendet
  - Java Servlets als dynamische Erweiterung von Webservern
  - Java Server Pages (JSP): HTML-Seiten mit eingebetteten Java Programmfragmenten

Zugriff auf Datenbanken via JDBC



Web-Anbindung von Datenbanken via Servlets/JDBC



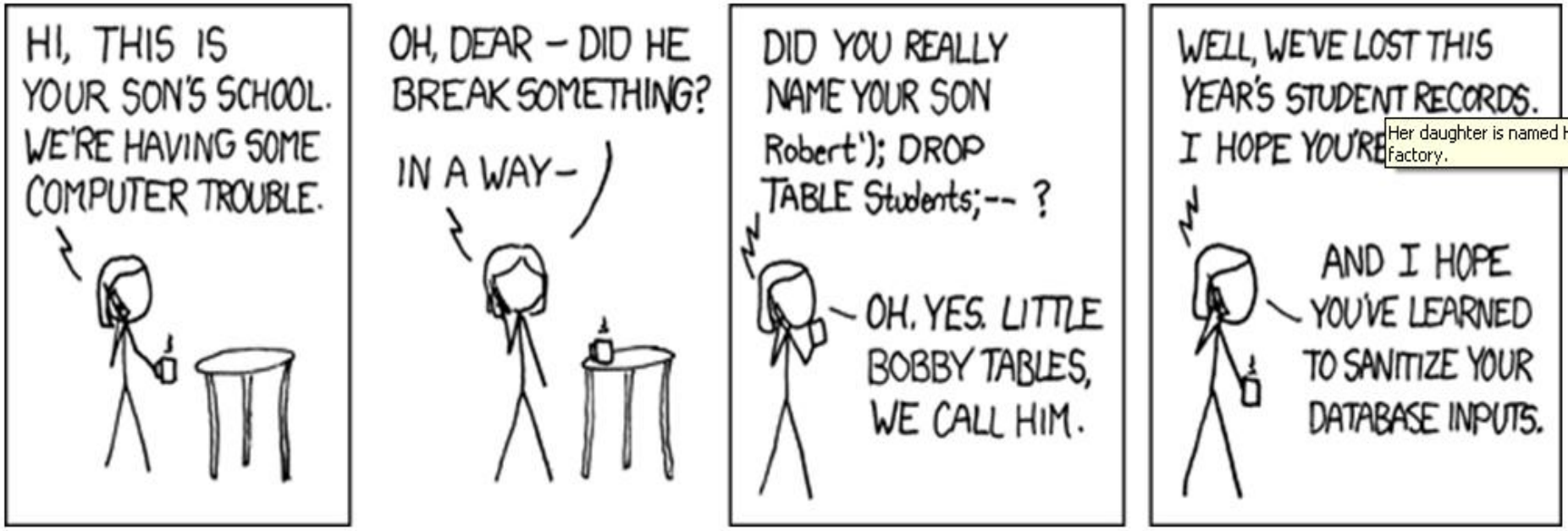
JDBC-Beispielprogramm

```
import java.sql.*;   import java.io.*;
public class ResultSetExample {
  public static void main(String[] argv) {
    Statement sql_stmt = null;
    Connection conn = null;
    try {
      Class.forName("oracle.jdbc.driver.OracleDriver");
      conn = DriverManager.getConnection
                ("jdbc:oracle:oci8:@lsintern-db", "nobody", "Passwort");
      sql_stmt = conn.createStatement();
    }
    catch (Exception e) {
      System.err.println("Folgender Fehler ist aufgetreten: " + e);
      System.exit(-1);      }
```

```
try {
    ResultSet rset = sql_stmt.executeQuery(
        "select avg(Semester) from Studenten");
    rset.next();   // eigentlich zu prüfen, ob Ergebnis leer
    System.out.println("Durchschnittsalter: " + rset.getDouble(1));
    rset.close();
  }
  catch(SQLException se) {
    System.out.println("Error: " + se);
  }
```

```
try {
      ResultSet rset = sql_stmt.executeQuery(
        "select Name, Raum from Professoren where Rang = 'C4'");
      System.out.println("C4-Professoren:");
      while(rset.next())  {
           System.out.println(rset.getString("Name") + " " +
                              rset.getInt("Raum"));
      }
      rset.close();
    }
    catch(SQLException se) {System.out.println("Error: " + se); }
    try {
      sql_stmt.close(); conn.close();
    }
    catch (SQLException e) {
      System.out.println("Fehler beim Schliessen der DB: " + e);
    }
  }
}
```

Sicherheitsproblem: SQL Injection

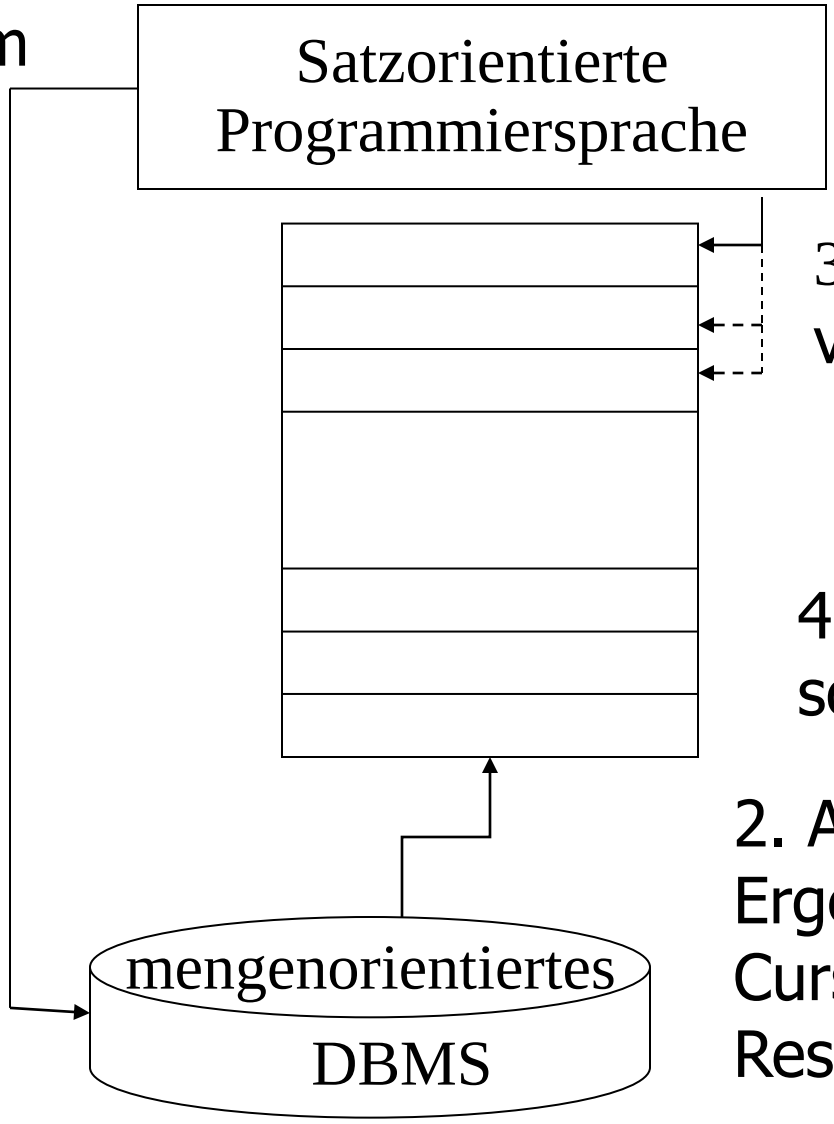


Vorübersetzung von SQL-Ausdrücken

```
PreparedStatement sql_exmatrikuliere =
    conn.prepareStatement
        ("delete from Studenten where MatrNr = ?");

int VomBenutzerEingeleseneMatrNr;
   // zu löschende MatrNr einlesen
sql_exmatrikuliere.setInt(1,VomBenutzerEingeleseneMatrNr);

int rows = sql_exmatrikuliere.executeUpdate();
if (rows == 1) System.out.println("StudentIn gelöscht.");
  else System.out.println("Kein/e StudentIn mit dieser MatrNr.");
```

Anfragen in Anwendungsprogrammen

- mehrere Tupel im Ergebnis



SQL/J-Beispielprogramm

```
import java.io.*; import java.sql.*;
import sqlj.runtime.*; import sqlj.runtime.ref.*;

#sql iterator StudentenItr (String Name, int Semester);

public class SQLJExmp {
  public static void main(String[] argv) {
    try {
      Class.forName("COM.ibm.db2.jdbc.app.DB2Driver");
      Connection con = DriverManager.getConnection
                                                ("jdbc:db2:uni");
      con.setAutoCommit(false);
      DefaultContext ctx = new DefaultContext(con);
      DefaultContext.setDefaultContext(ctx);
```

```
 StudentenItr Methusaleme;
  #sql Methusaleme = { select s.Name, s.Semester
                       from Studenten s
                       where s.Semester > 13 };
  while (Methusaleme.next()) {
     System.out.println(Methusaleme.Name() + ":" +
                        Methusaleme.Semester());
  }
  Methusaleme.close();
  #sql { delete from Studenten where Semester > 13 };
  #sql { commit };
}
catch (SQLException e) {
  System.out.println("Fehler mit der DB-Verbindung: " + e);
}
catch (Exception e) {
  System.err.println("Folgender Fehler ist aufgetreten: " + e);
  System.exit(-1);    }  }  }
```

Query by Example

Vorlesungen	VorlNr	Titel	SWS	gelesen Von
		p._t	> 3	

Analog

$$\{[t] \mid \exists v, s, r\ ([v,t,s,r] \in \text{Vorlesungen} \wedge s > 3)\}$$

Join in QBE

Vorlesungen	VorlNr	Titel	SWS	gelesen Von
		Mäeutik		_x

Professoren	PersNr	Name	Rang	Raum
	_x	**p.**_n		

Die Condition Box

Studenten	MatrNr	Name	Semester
		_s	_a

conditions
_a > _b

Studenten	MatrNr	Name	Semester
		_t	_b

Betreuen	potentieller Tutor	Betreuter
p.	_s	_t

Aggregatfunktion und Gruppierung

Vorlesungen	VorlNr	Titel	SWS	gelesen Von
			p.sum.all._x	**p.g.**

conditions
avg.all._x>2

Updates in QBE: Sokrates ist „von uns gegangen“

Professoren	PersNr	Name	Rang	Raum
d.	_x	Sokrates		

Vorlesungen	VorlNr	Titel	SWS	gelesen Von
d.	_y			_x

hören	VorlNr	MatrNr
d.	_y	

Uni.PunkteListe

NAME	AUFGABE	MAX	ERZIELT
Bond	1	10	4
Bond	2	10	10
Bond	3	11	4
Maier	1	10	4
Maier	2	10	2
Maier	3	11	3

With BonusSicht as (…)

select * from BonusSicht

Name	MaxMoeglich	Erzielt
Bond	31	25
Maier	31	17

Datenintegrität

- Integitätsbedingungen
  - Schlüssel
  - Beziehungskardinalitäten
  - Attributdomänen
  - Inklusion bei Generalisierung
- statische Integritätsbedingungen
  - Bedingungen an den Zustand der Datenbasis
- dynamische Integritätsbedingungen
  - Bedingungen an Zustandsübergänge

Referentielle Integrität

Fremdschlüssel

- verweisen auf Tupel einer Relation
- z.B. *gelesenVon* in *Vorlesungen* verweist auf Tupel in Professoren

referentielle Integrität

- Fremdschlüssel müssen auf existierende Tupel verweisen oder einen Nullwert enthalten

Referentielle Integrität in SQL

- Kandidatenschlüssel: **unique**
- Primärschlüssel: **primary key**
- Fremdschlüssel: **foreign key**

create table *R*
(α **integer primary key**,
...);

create table *S*
(...,
κ **integer references** *R*);

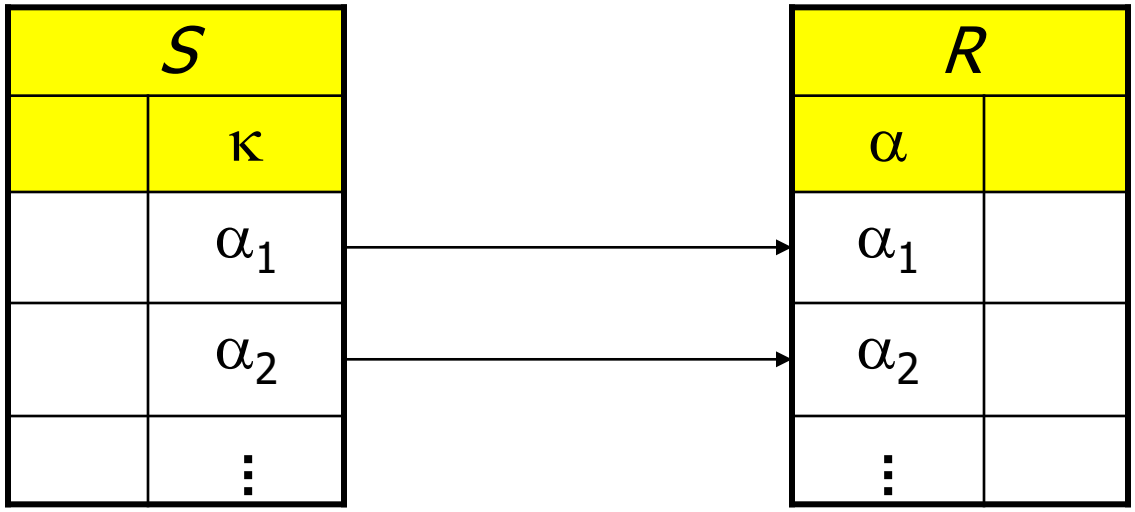
Einhaltung referentieller Integrität

Änderung von referenzierten Daten

1. Default: Zurückweisen der Änderungsoperation
2. Propagieren der Änderungen: **cascade**
3. Verweise auf Nullwert setzen: **set null**

Einhaltung referentieller Integrität

Originalzustand



Änderungsoperationen

update *R*

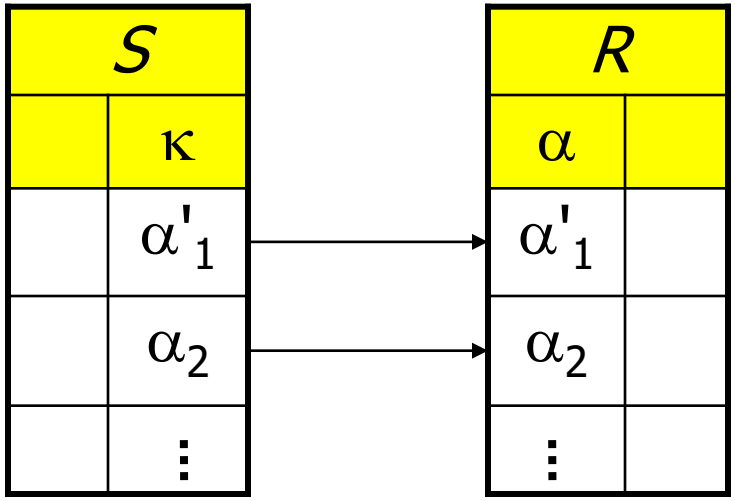
set $\alpha = \alpha'_1$

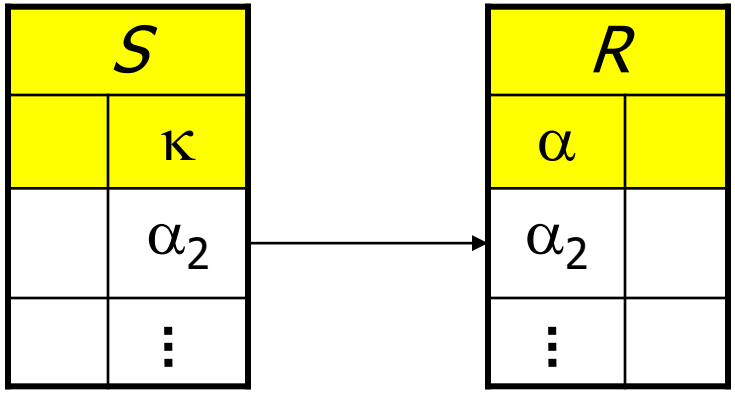
where $\alpha = \alpha_1$;

delete from *R*

where $\alpha = \alpha_1$;

Kaskadieren





create table *S*

(...,

κ **integer references** *R*

on update cascade);

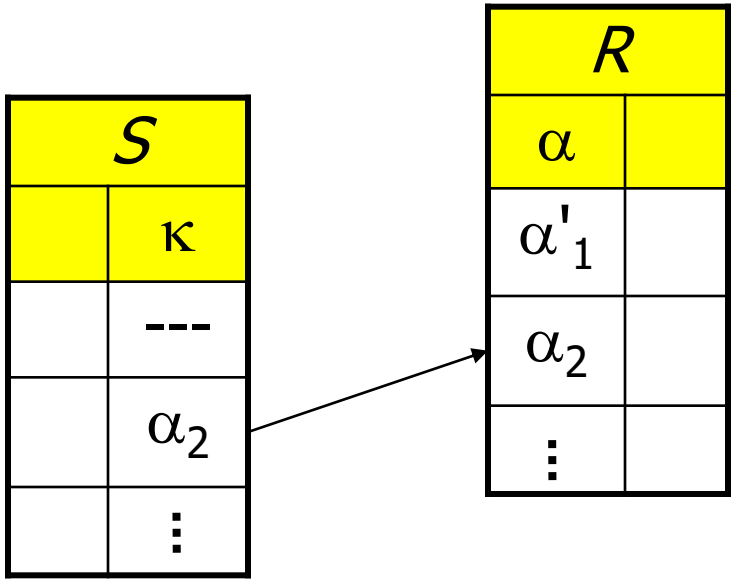
create table *S*

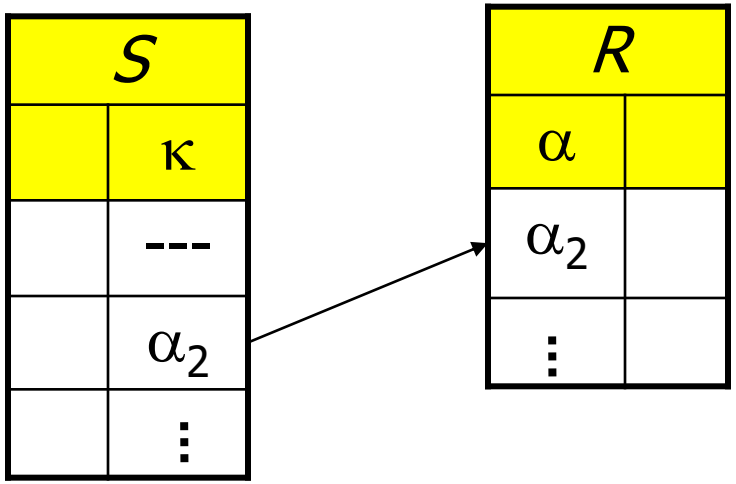
(...,

κ **integer references** *R*

on delete cascade);

Auf Null setzen





create table *S*

(...,

κ **integer references** *R*

on update set null);

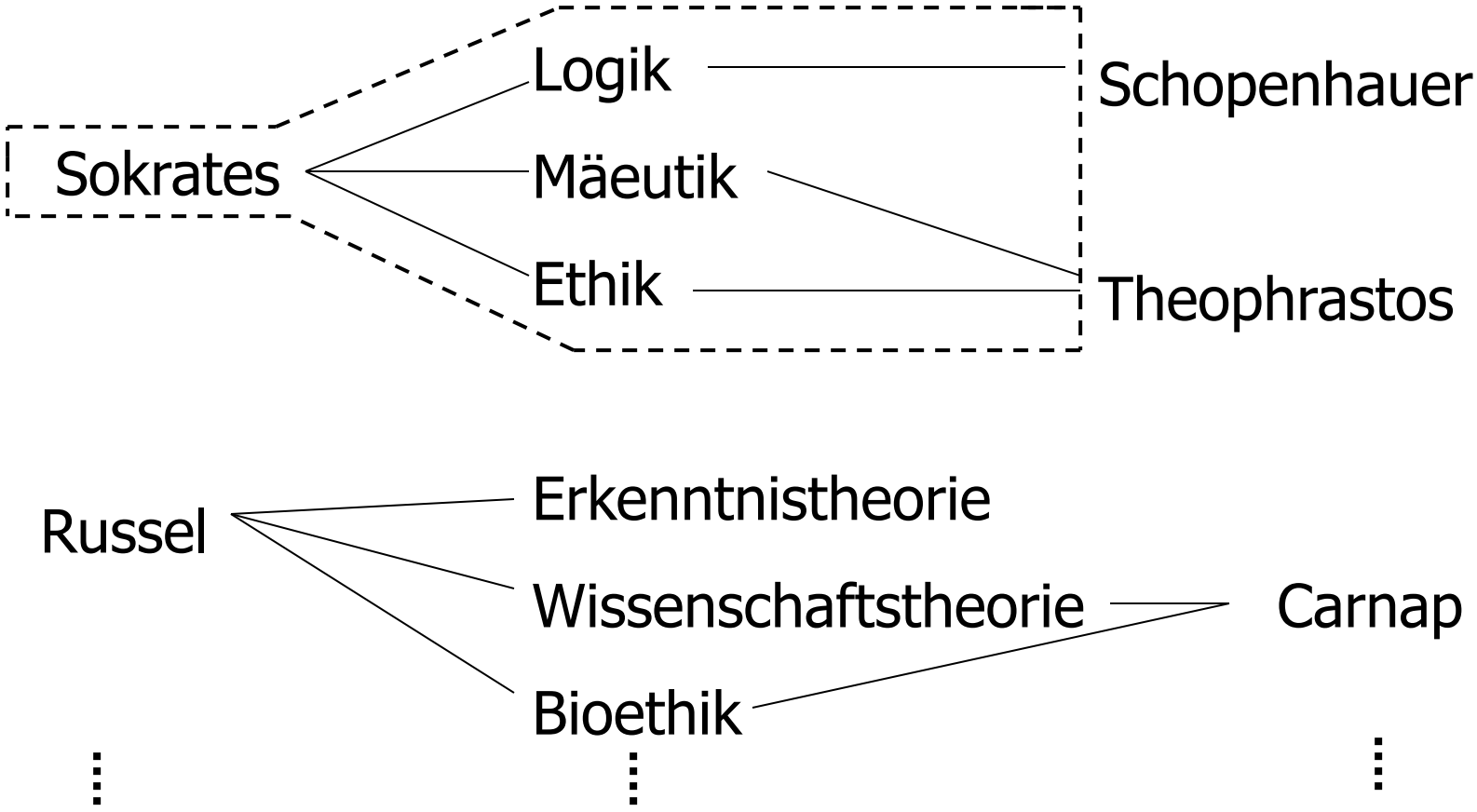
create table *S*

(...,

κ **integer references** *R*

on delete set null);

Kaskadierendes Löschen



```
create table Vorlesungen
    ( ...,
    gelesenVon integer
                    references Professoren
                    on delete cascade);

create table hören
        ( ...,
        VorlNr integer
                references Vorlesungen
                on delete cascade);
```

Einfache statische Integritätsbedingungen

- Wertebereichseinschränkungen
  … **check** Semester **between** 1 **and** 13
- Aufzählungstypen
  … **check** Rang **in** (`C2´, `C3´, `C4´) …

Das Universitätsschema mit Integritätsbedingungen

```
create table Studenten
    ( MatrNr     integer primary key,
    Name         varchar(30) not null,
    Semester     integer check Semester between 1 and 13),
create table Professoren
    ( PersNr     integer primary key,
    Name         varchar(30) not null,
    Rang         character(2) check (Rang in (`C2´,`C3´,`C4´)),
    Raum         integer unique );
```

```
create table Assistenten
    ( PersNr          integer primary key,
    Name              varchar(30) not null,
    Fachgebiet        varchar(30),
    Boss              integer,
    foreign key       (Boss) references Professoren on delete
                      set null);
create table Vorlesungen
    ( VorlNr          integer primary key,
    Titel             varchar(30),
    SWS               integer,
    gelesen Von       integer references Professoren on delete
                      set null);
```

```
create table hören
    ( MatrNr        integer references Studenten on delete
                    cascade,
    VorlNr          integer references Vorlesungen on delete
                    cascade,
    primary key     (MatrNr, VorlNr));
create table voraussetzen
    ( Vorgänger     integer references Vorlesungen on delete
                    cascade,
    Nachfolger      integer references Vorlesungen on
                    delete cascade,
    primary key     (Vorgänger, Nachfolger));
```

create table prüfen

(MatrNr **integer references** Studenten **on delete cascade**,

VorlNr **integer references** Vorlesungen,

PersNr **integer references** Professoren **on delete set null**,

Note **numeric** (2,1) **check** (Note **between** 0.7 **and** 5.0),

primary key (MatrNr, VorlNr));

Komplexere Konsistenzbedingungen: Leider selten / noch nicht unterstützt

```
create table prüfen
    ( MatrNr        integer references Studenten on delete cascade,
      VorlNr        integer references Vorlesungen,
      PersNr        integer references Professoren on delete set null,
      Note          numeric(2,1) check (Note between 0.7 and 5.0),
      primary key   (MatrNr, VorlNr)
      constraint VorherHören
            check (exists (select *
                           from hören h
                           where h.VorlNr = prüfen.VorlNr and
                                 h.MatrNr = prüfen.MatrNr))
    );
```

- Studenten können sich nur über Vorlesungen prüfen lassen, die sie vorher gehört haben
- Bei jeder Änderung und Einfügung wird die check-Klausel ausgewertet
- Operation wird nur durchgeführt, wenn der check true ergibt

Datenbank-Trigger

```
create trigger keine Degradierung
before update on Professoren
for each row
when (old.Rang is not null)
begin
      if :old.Rang = 'C3' and :new.Rang = 'C2' then
            :new.Rang := 'C3';
      end if;
      if :old.Rang = 'C4' then
            :new.Rang := 'C4'
      end if;
      if :new.Rang is null then
        :new.Rang := :old.Rang;
      end if;
end
```

Trigger-Erläuterungen: Oracle Konventionen

Dieser Trigger besteht aus vier Teilen:

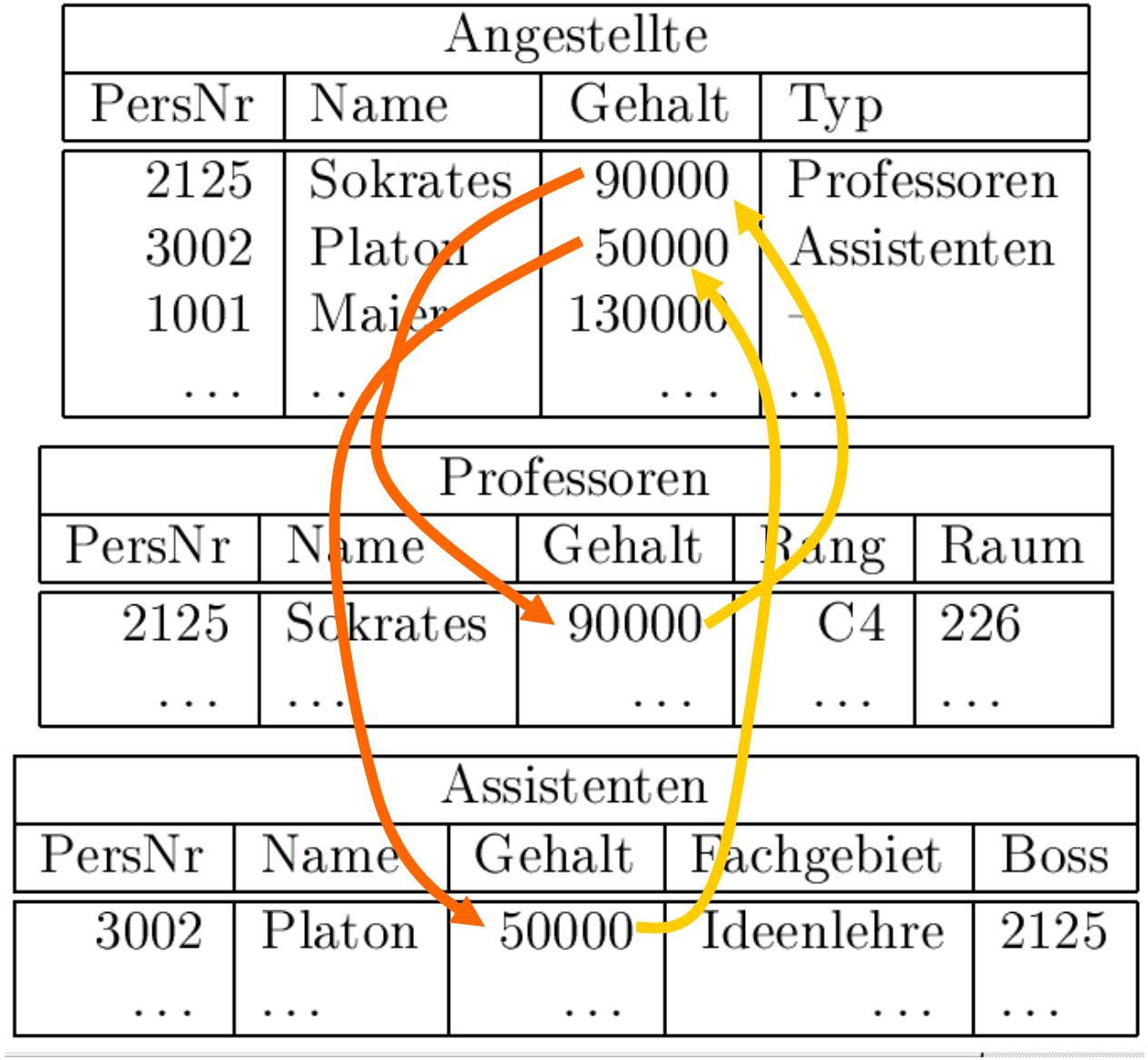
1. der **create trigger** Anweisung, gefolgt von einem Namen,
2. der Definition des Auslösers, in diesem Fall bevor eine Änderungsoperation (**before update on**) auf einer Zeile (**for each row**) der Tabelle *Professoren* ausgeführt werden kann,
3. einer einschränkenden Bedingung (**when**) und
4. einer Prozedurdefinition in der Oracle-proprietären Syntax

In der Prozedurdefinition bezieht sich *old* auf das noch unveränderte Tupel (den Originalzustand), *new* enthält bereits die Veränderungen durch die Operation.

Gleicher Trigger in DB2 / SQL:1999-Syntax

```
create trigger keineDegradierung
no cascade
before update of Rang on Professoren
referencing old as alterZustand
                  new as neuerZustand
for each row
mode DB2SQL
when (alterZustand.Rang is not null)
set neuerZustand.Rang = case
          when neuerZustand.Rang is null then alterZustand.Rang
          when neuerZustand.Rang < 'C2' then alterZustand.Rang
          when neuerZustand.Rang > 'C4' then alterZustand.Rang
          when neuerZustand.Rang < alterZustand.Rang then alterZustand.Rang
          else neuerZustand.Rang
          end;
```

Übung: Trigger zur Konsistenzhaltung redundanter Information bei Generalisierung



Angestellte

PersNr	Name	Gehalt	Typ
2125	Sokrates	90000	Professoren
3002	Platon	50000	Assistenten
1001	Maier	130000	–
...	...	...	...

Professoren

PersNr	Name	Gehalt	Rang	Raum
2125	Sokrates	90000	C4	226
...	...	...	...	...

Assistenten

PersNr	Name	Gehalt	Fachgebiet	Boss
3002	Platon	50000	Ideenlehre	2125
...	...	...	...	...

Kapitel 6
Relationale Entwurfstheorie

Funktionale Abhängigkeiten
Normalformen
Normalisierung durch Dekomposition

Ziele der relationalen Entwurfstheorie

- Bewertung der Qualität eines Relationenschemas
  - Redundanz
  - Einhaltung von Konsistenzbedingungen
    - Funktionaler Abhängigkeiten
- Normalformen als Gütekriterium
- Ggfls. Verbesserung eines Relationenschemas
  - Durch den Synthesealgorithmus
  - Durch Dekomposition

Funktionale Abhängigkeiten

- Schema
  - $\mathcal{R} = \{A, B, C, D\}$
- Ausprägung *R*

- Seien $\alpha \subseteq \mathcal{R}$, $\beta \subseteq \mathcal{R}$
- $\alpha \rightarrow \beta$ genau dann wenn $\forall r, s \in R$ mit $r.\alpha = s.\alpha \Rightarrow r.\beta = s.\beta$

R			
A	*B*	*C*	*D*
a4	b2	c4	d3
a1	b1	c1	d1
a1	b1	c1	d2
a2	b2	c3	d2
a3	b2	c4	d3

{A} → {B}

{C, D } → {B}

Nicht: {B} → {C}

Notationskonvention:

CD → B

Beispiel

Stammbaum				
Kind	Vater	Mutter	Opa	Oma
Sofie	Alfons	Sabine	Lothar	Linde
Sofie	Alfons	Sabine	Hubert	Lisa
Niklas	Alfons	Sabine	Lothar	Linde
Niklas	Alfons	Sabine	Hubert	Lisa
...	...	...	Lothar	Martha
...	...	...	...	...

Beispiel

Stammbaum				
Kind	Vater	Mutter	Opa	Oma
Sofie	Alfons	Sabine	Lothar	Linde
Sofie	Alfons	Sabine	Hubert	Lisa
Niklas	Alfons	Sabine	Lothar	Linde
Niklas	Alfons	Sabine	Hubert	Lisa
...	...	...	Lothar	Martha
...	...	...	...	...

- Kind → Vater,Mutter
- Kind,Opa → Oma
- Kind,Oma → Opa

Schlüssel

- $\alpha \subseteq \mathcal{R}$ ist ein Super-Schlüssel, falls folgendes gilt:
  - $\alpha \rightarrow \mathcal{R}$
- β ist voll funktional abhängig von α genau dann wenn gilt
  - $\alpha \rightarrow \beta$ und
  - α kann nicht mehr verkleinert werden, d.h.
    - $\forall A \in \alpha$ folgt, dass $(\alpha - \{A\}) \rightarrow \beta$ nicht gilt, oder kürzer
    - $\forall A \in \alpha: \neg((\alpha - \{A\}) \rightarrow \beta)$
- Notation für volle funktionale Abhängigkeit: $\alpha \rightarrow^{\cdot} \beta$
- $\alpha \subseteq \mathcal{R}$ ist ein Kandidaten-Schlüssel, falls folgendes gilt:
  - $\alpha \rightarrow^{\cdot} \mathcal{R}$

Schlüsselbestimmung

Städte			
Name	BLand	Vorwahl	EW
Frankfurt	Hessen	069	650000
Frankfurt	Brandenburg	0335	84000
München	Bayern	089	1200000
Passau	Bayern	0851	50000
...	...	...	...

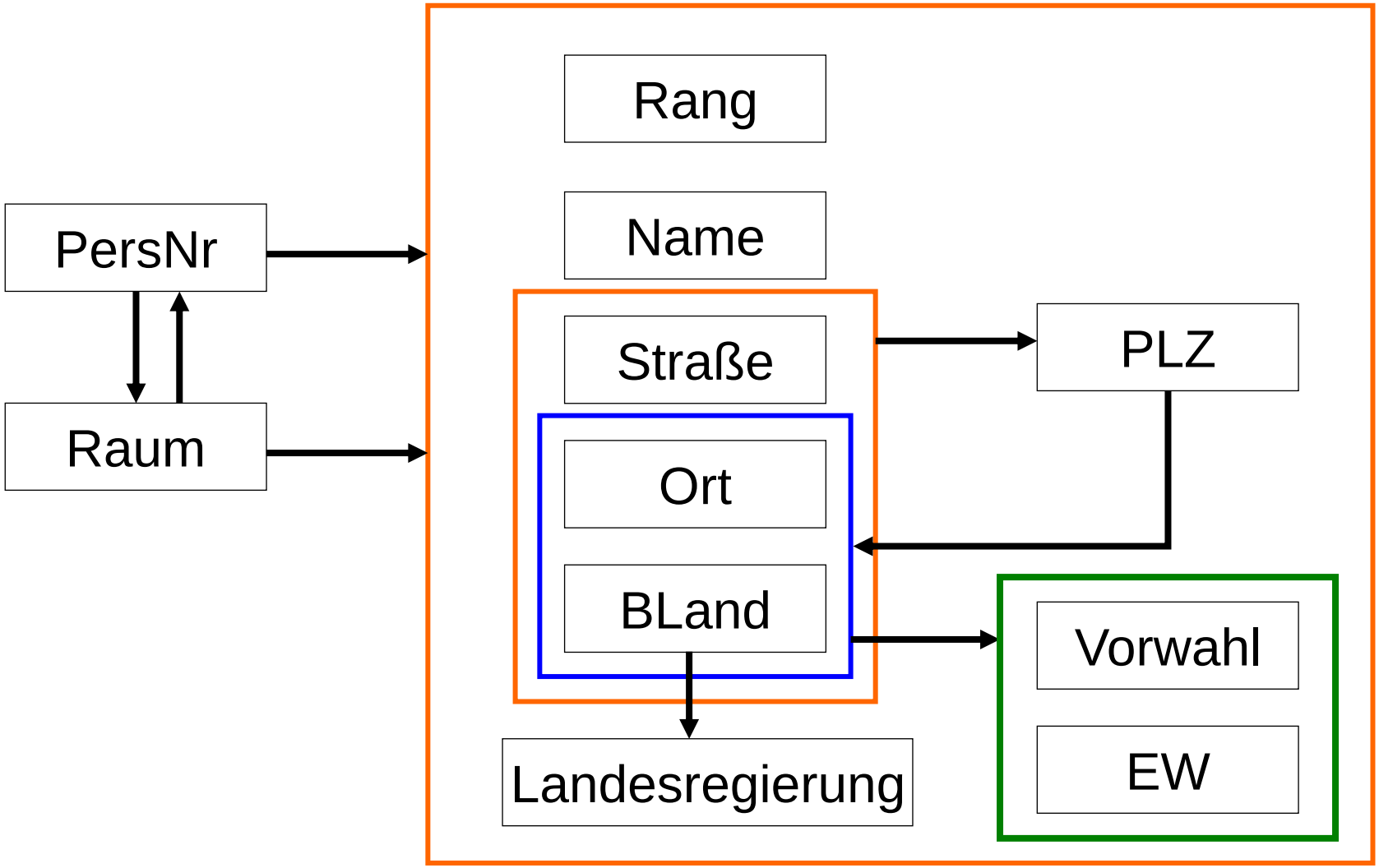
- Kandidaten-schlüssel von *Städte*:
  - {Name,BLand}
  - {Name,Vorwahl}

- Beachte, dass 2 kleinere Städte dieselbe Vorwahl haben können

Bestimmung funktionaler Abhängigkeiten

- Professoren: {[PersNr, Name, Rang, Raum, Ort, Straße, PLZ, Vorwahl, Bland, EW, Landesregierung]}
  - {PersNr} → {PersNr, Name, Rang, Raum, Ort, Straße, PLZ, Vorwahl, Bland, EW, Landesregierung}
  - {Ort,BLand} → {EW, Vorwahl}
  - {PLZ} → {Bland, Ort, EW}
  - {Bland, Ort, Straße} → {PLZ}
  - {Bland} → {Landesregierung}
  - {Raum} → {PersNr}
- Zusätzliche Abhängigkeiten, die aus obigen abgeleitet werden können:
  - {Raum} → {PersNr, Name, Rang, Raum, Ort, Straße, PLZ, Vorwahl, Bland, EW, Landesregierung}
  - {PLZ} → {Landesregierung}

Graphische Darstellung der funktionalen Abhängigkeiten



„Schlechte“ Relationenschemata

ProfVorl						
PersNr	Name	Rang	Raum	VorlNr	Titel	SWS
2125	Sokrates	C4	226	5041	Ethik	4
2125	Sokrates	C4	226	5049	Mäeutik	2
2125	Sokrates	C4	226	4052	Logik	4
...	...	...	...	...	...	...
2132	Popper	C3	52	5259	Der Wiener Kreis	2
2137	Kant	C4	7	4630	Die 3 Kritiken	4

- Update-Anomalien
  - Sokrates zieht um, von Raum 226 in R. 338. Was passiert?
- Einfüge-Anomalien
  - Neue/r Prof ohne Vorlesungen?
- Löschanomalien
  - Letzte Vorlesung einer/s Profs wird gelöscht? Was passiert?

Zerlegung (Dekomposition) von Relationen

- Es gibt zwei Korrektheitskriterien für die Zerlegung von Relationenschemata:

1. Verlustlosigkeit
   - Die in der ursprünglichen Relationenausprägung R des Schemas $\mathcal{R}$ enthaltenen Informationen müssen aus den Ausprägungen $R1$, ..., Rn der neuen Relationenschemata $\mathcal{R}1$, .., $\mathcal{R}n$ rekonstruierbar sein.

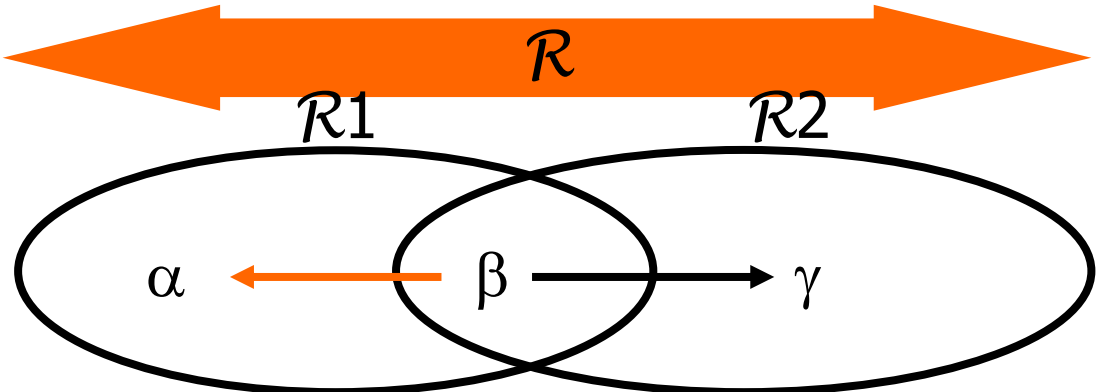
2. Abhängigkeitserhaltung
   - Die für $\mathcal{R}$ geltenden funktionalen Anhängigkeiten müssen auf die Schemata $\mathcal{R}1$, ..., $\mathcal{R}n$ übertragbar sein.

Kriterien für die Verlustlosigkeit einer Zerlegung

- $\mathcal{R} = \mathcal{R}1 \cup \mathcal{R}2$
  - R1 := $\Pi_{\mathcal{R}1}$ (R)
  - R2 := $\Pi_{\mathcal{R}2}$ (R)

- Die Zerlegung von $\mathcal{R}$ in $\mathcal{R}1$ und $\mathcal{R}2$ ist verlustlos, falls für jede mögliche (gültige) Ausprägung R von $\mathcal{R}$ gilt:
  - R = R1 A R2

- Hinreichende Bedingung für die Verlustlosigkeit einer Zerlegung
  - $(\mathcal{R}1 \cap \mathcal{R}2) \rightarrow \mathcal{R}1$ oder
  - $(\mathcal{R}1 \cap \mathcal{R}2) \rightarrow \mathcal{R}2$



Biertrinker-Beispiel

Biertrinker		
Kneipe	*Gast*	*Bier*
Kowalski	Kemper	Pils
Kowalski	Eickler	Hefeweizen
Innsteg	Kemper	Hefeweizen

„Verlustige“ Zerlegung

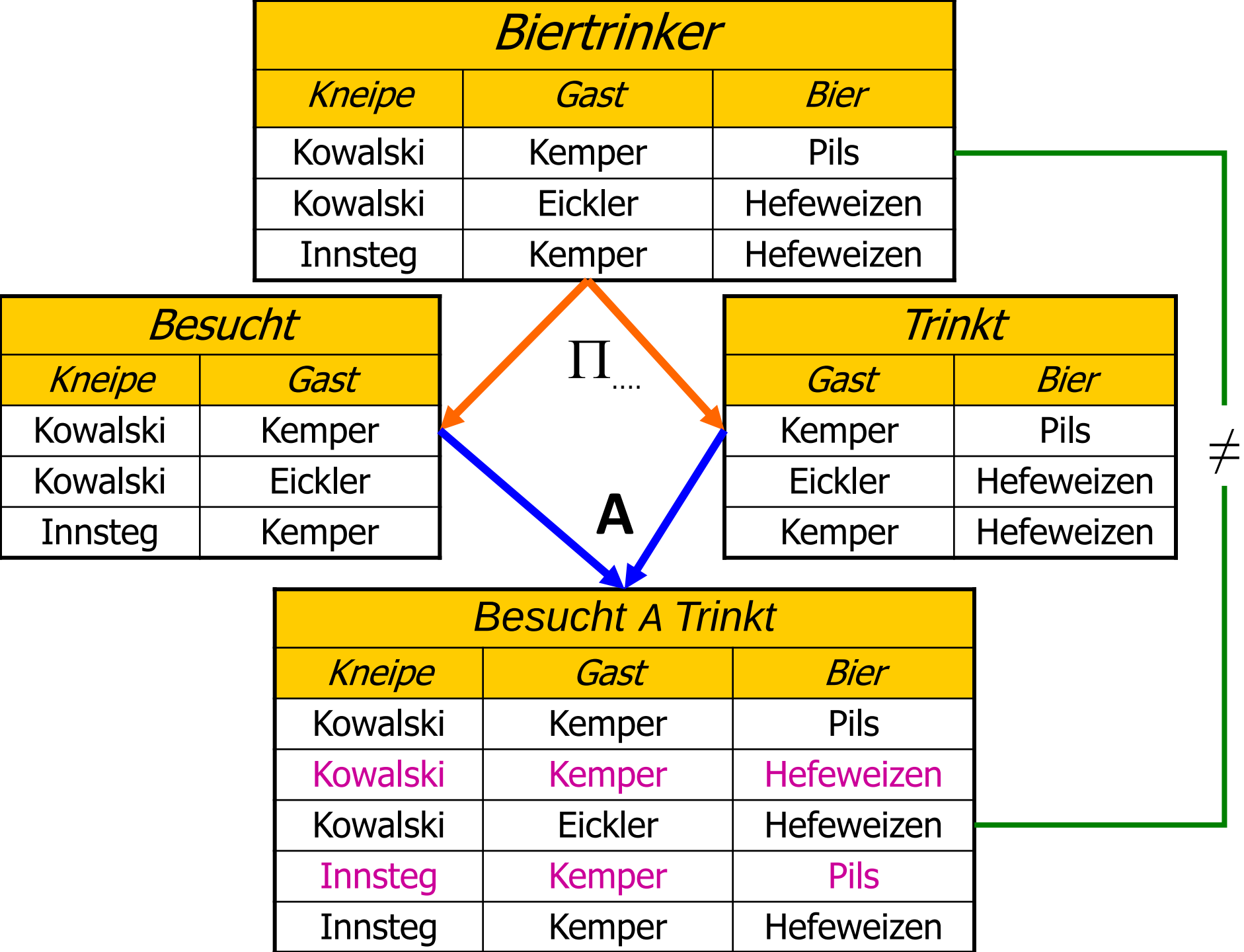
Biertrinker		
Kneipe	*Gast*	*Bier*
Kowalski	Kemper	Pils
Kowalski	Eickler	Hefeweizen
Innsteg	Kemper	Hefeweizen

$\Pi_{\text{Kneipe, Gast}}$

$\Pi_{\text{Gast, Bier}}$

Besucht	
Kneipe	*Gast*
Kowalski	Kemper
Kowalski	Eickler
Innsteg	Kemper

Trinkt	
Gast	*Bier*
Kemper	Pils
Eickler	Hefeweizen
Kemper	Hefeweizen


Biertrinker
Kneipe Gast Bier
Kowalski Kemper Pils
Kowalski Eickler Hefeweizen
Innsteg Kemper Hefeweizen
Besucht
Kneipe Gast
Kowalski Kemper
Kowalski Eickler
Innsteg Kemper
Π....
A
Trinkt
Gast Bier
Kemper Pils
Eickler Hefeweizen
Kemper Hefeweizen
≠
Besucht A Trinkt
Kneipe Gast Bier
Kowalski Kemper Pils
Kowalski Kemper Hefeweizen
Kowalski Eickler Hefeweizen
Innsteg Kemper Pils
Innsteg Kemper Hefeweizen

Erläuterung des Biertrinker-Beispiels

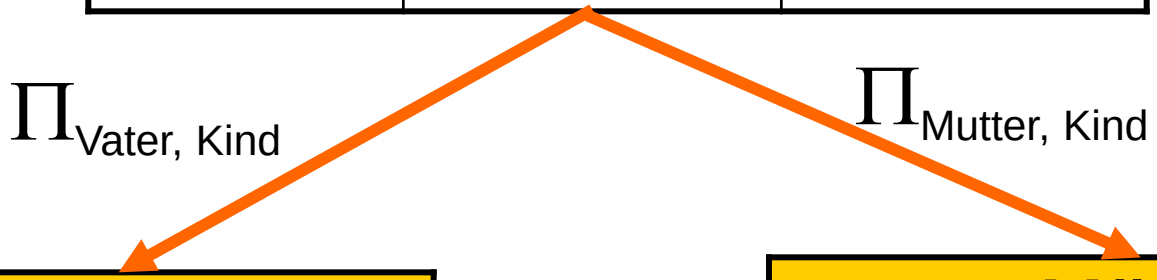
- Unser Biertrinker-Beispiel war eine „verlustige“ Zerlegung und dementsprechend war die hinreichende Bedingung verletzt. Es gilt nämlich nur die eine nicht-triviale funktionale Abhängigkeit
  - {Kneipe,Gast}→{Bier}

- Wohingegen keine der zwei möglichen, die Verlustlosigkeit garantierenden FDs gelten
  - {Gast}→{Bier}
  - {Gast}→{Kneipe}

- Das liegt daran, dass die Leute (insbes. Kemper) in unterschiedlichen Kneipen unterschiedliches Bier trinken. In derselben Kneipe aber immer das gleiche Bier
    - (damit sich die KellnerInnen darauf einstellen können?)

Verlustfreie Zerlegung

Eltern		
Vater	*Mutter*	*Kind*
Johann	Martha	Else
Johann	Maria	Theo
Heinz	Martha	Cleo



Väter	
Vater	*Kind*
Johann	Else
Johann	Theo
Heinz	Cleo

Mütter	
Mutter	*Kind*
Martha	Else
Maria	Theo
Martha	Cleo

Erläuterung der verlustfreien Zerlegung der Eltern-Relation

- Eltern: {[Vater, Mutter, Kind]}
- Väter: {[Vater, Kind]}
- Mütter: {[Mutter, Kind]}

- Verlustlosigkeit ist garantiert
- Es gilt nicht nur eine der hinreichenden FDs, sondern gleich beide
  - {Kind}→{Mutter}
  - {Kind}→{Vater}

- Also ist {Kind} natürlich auch der Schlüssel der Relation Eltern

- Die Zerlegung von Eltern ist zwar verlustlos, aber auch ziemlich unnötig, da die Relation in sehr gutem Zustand (~Normalform) ist

Boyce-Codd-Normalform

- Die Boyce-Codd-Normalform (BCNF) ist nochmals eine Verschärfung der 3 NF.
- Ein Relationenschema $\mathcal{R}$ mit FDs F ist in BCNF, wenn für jede für $\mathcal{R}$ geltende funktionale Abhängigkeit der Form $\alpha \rightarrow \beta \in F$ und mindestens eine von zwei Bedingungen gilt:
  - $\beta \subseteq \alpha$, d.h., die Abhängigkeit ist trivial oder
  - α ist Superschlüssel von $\mathcal{R}$

- Man kann jede Relation verlustlos in BCNF-Relationen zerlegen
- Manchmal lässt sich dabei die Abhängigkeiterhaltung aber nicht erzielen

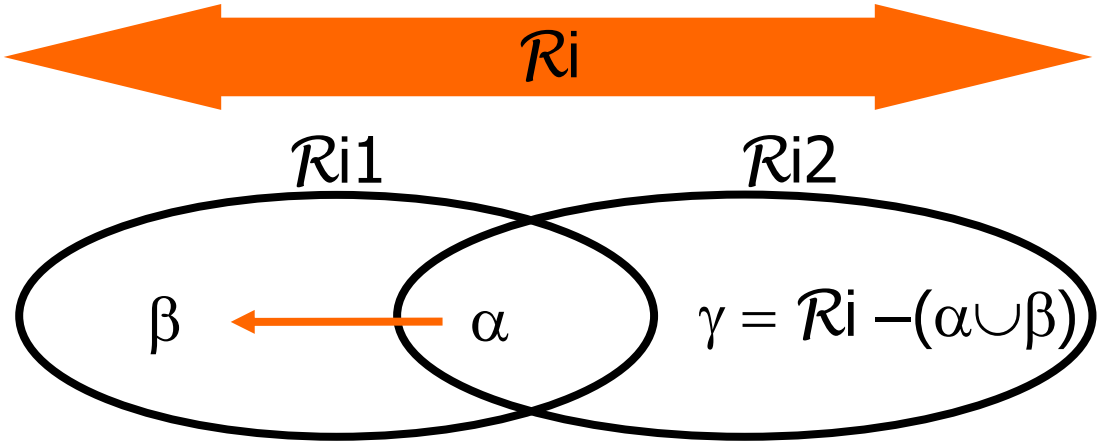
Dekomposition

- Man kann grundsätzlich jedes Relationenschema $\mathcal{R}$ mit funktionalen Anhängigkeiten F so in $\mathcal{R}1$, ..., $\mathcal{R}n$ zerlegen, dass gilt:
  - $\mathcal{R}1$, ..., $\mathcal{R}n$ ist eine verlustlose Zerlegung von $\mathcal{R}$.
  - Alle $\mathcal{R}1$, ..., $\mathcal{R}n$ sind in BCNF.
  - Es kann leider nicht immer erreicht werden, dass die Zerlegung $\mathcal{R}1$, ..., $\mathcal{R}n$ abhängigkeitserhaltend ist.

Dekompositions-Algorithmus

- Starte mit Z = $\{\mathcal{R}\}$
- Solange es noch ein Relationenschema $\mathcal{R}$i in Z gibt, das nicht in BCNF ist, mache folgendes:
  - Es gibt also eine für $\mathcal{R}$i geltende nicht-triviale funktionale Abhängigkeit ($\alpha \rightarrow \beta$) mit
    - $\alpha \cap \beta = \varnothing$
    - $\neg(\alpha \rightarrow \mathcal{R}i)$
  - Finde eine solche FD
    - Man sollte sie so wählen, dass β alle von α funktional abhängigen Attribute B $\in$ ($\mathcal{R}$i - α) enthält, damit der Dekompositionsalgorithmus möglichst schnell terminiert.
  - Zerlege $\mathcal{R}$i in $\mathcal{R}$i1 := $\alpha \cup \beta$ und $\mathcal{R}$i2 := $\mathcal{R}$i - β
  - Entferne $\mathcal{R}$i aus Z und füge $\mathcal{R}$i1 und $\mathcal{R}$i2 ein, also
    - Z := (Z – $\{\mathcal{R}i\}$) $\cup$ $\{\mathcal{R}i1\}$ $\cup$ $\{\mathcal{R}i2\}$

Veranschaulichung der Dekomposition



Dekomposition der Relation Städte in BCNF-Relationen

- Städte: {[Ort, BLand, Ministerpräsident/in, EW]}
- Geltende FDs:
  - {BLand} → {Ministerpräsident/in}
  - {Ort, BLand} → {EW}
  - {Ministerpräsident/in} → {BLand}

- $\mathcal{R}$i1:
  - Regierungen: {[BLand, Ministerpräsident/in]}

- $\mathcal{R}$i2:
  - Städte: {[Ort, BLand, EW]}

- Zerlegung ist verlustlos und auch abhängigkeitserhaltend