Kapitel 16:
Objektorientierte Datenbanken

Weiterentwicklung relationaler Systeme

der evolutionäre Ansatz:

- komplexe Typen
- geschachteltes relationales Modell

der revolutionäre Ansatz:

- strukturelle Information
- verhaltensmäßige Information

Schwächen relationaler Systeme

Buch: {[ISBN, Verlag, Titel, Autor, Version, Stichwort]}

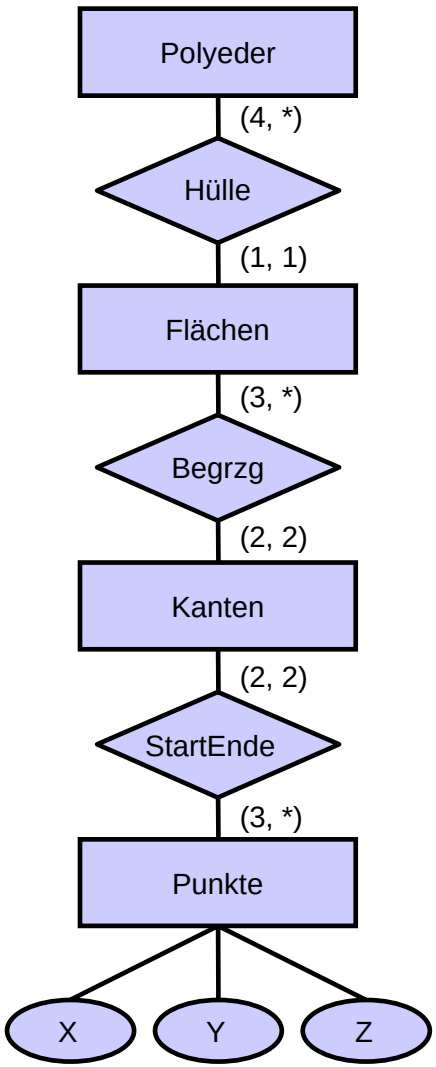
2 Autoren, 5 Versionen, 6 Stichworte

$\Rightarrow 2 \times 5 \times 6 = 60$ Einträge

Buch : {[ISBN, Titel, Verlag]}
Autor : {[ISBN, Name, Vorname]}
Version : {[ISBN, Auflage, Jahr]}
Stichwort : {[ISBN, Stichwort]}

Problem:
"Liste Bücher mit den Autoren Meier & Schmidt"

Modellierung von Polyedern



Polyeder

PolyID	Gewicht	Material	. . .
cubo#5	25.765	Eisen	. . .
tetra#7	37.985	Glas	. . .
. . .	. . .	. . .	. . .

Flächen

FlächenID	PolyID	Oberflächen
f1	cubo#5	. . .
f2	cubo#5	. . .
. . .	. . .	. . .
f6	cubo#5	. . .
f7	tetra#7	. . .

Kanten

KantenID	F1	F2	P1	P2
k1	f1	f4	p1	p4
k2	f1	f2	p2	p3
. . .	. . .	. . .	. . .	. . .

Punkte

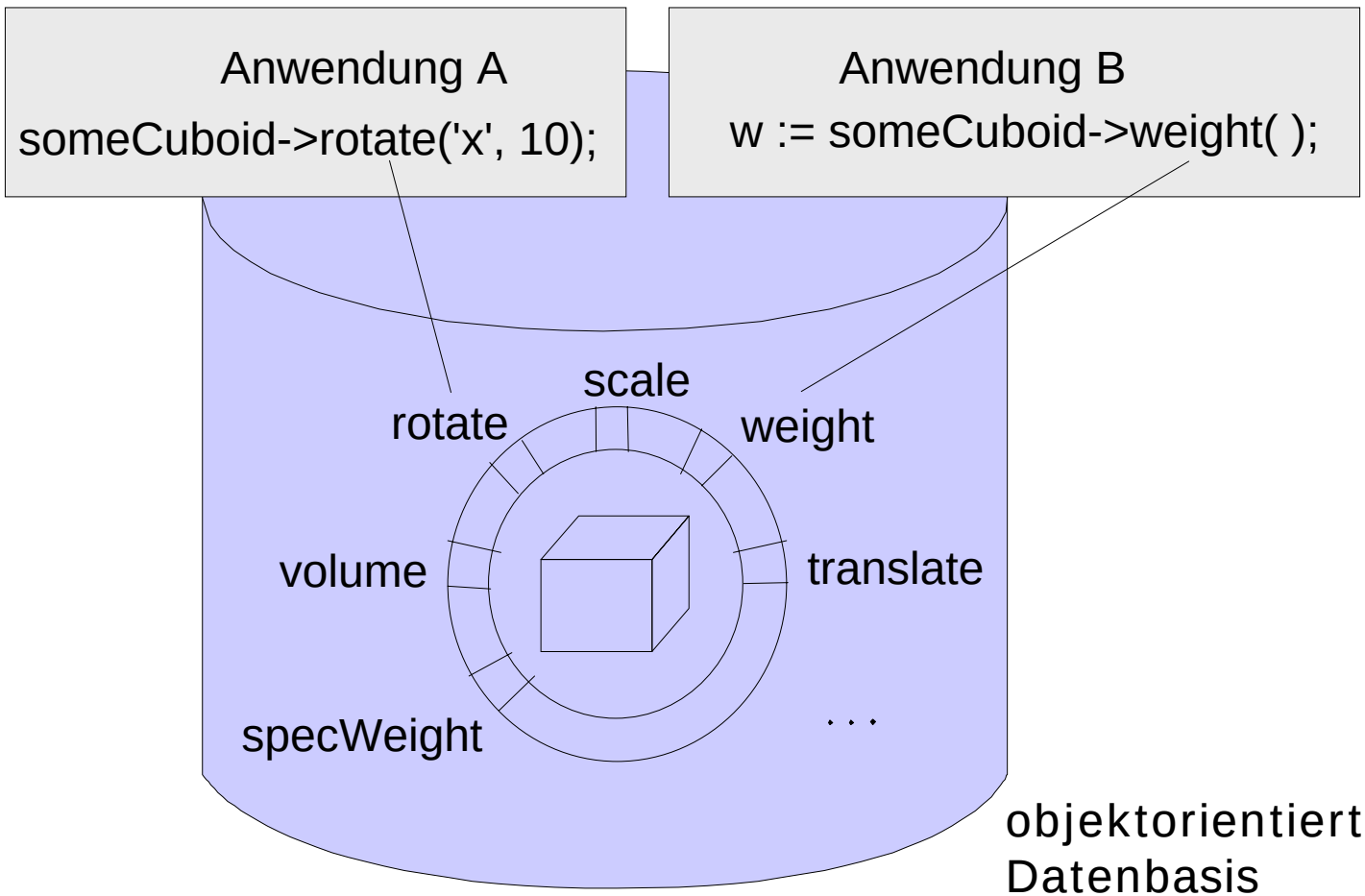
PunktID	X	Y	Z
p1	0.0	0.0	0.0
p2	1.0	0.0	0.0
. . .	. . .	. . .	. . .
p8	0.0	1.0	1.0
. . .	. . .	. . .	. . .

Schwachpunkte

- Segmentierung
- Künstliche Schlüsselattribute
- Fehlendes Verhalten
- Externe Programmierschnittstelle

Vorteile der objektorientierten Datenmodellierung

Struktur + Verhalten in einem Objekt-Typ integriert



ODMG

Object Database Management Group
http://www.odmg.org

entwickelt Standards für

- objektorientiertes Datenmodell
- objektorientierte Abfragesprache OQL
- Schnittstellen zu C++, Smalltalk, Java

Eigenschaften von Objekten

Ein Objekt

- gehört zu einer Klasse
  entsteht durch Instanziierung
  ist Teil der Extension = alle Objekte

- hat Identität
  systemweit eindeutig, unveränderbar

- Wert = Zustand = Ausprägung

Identität

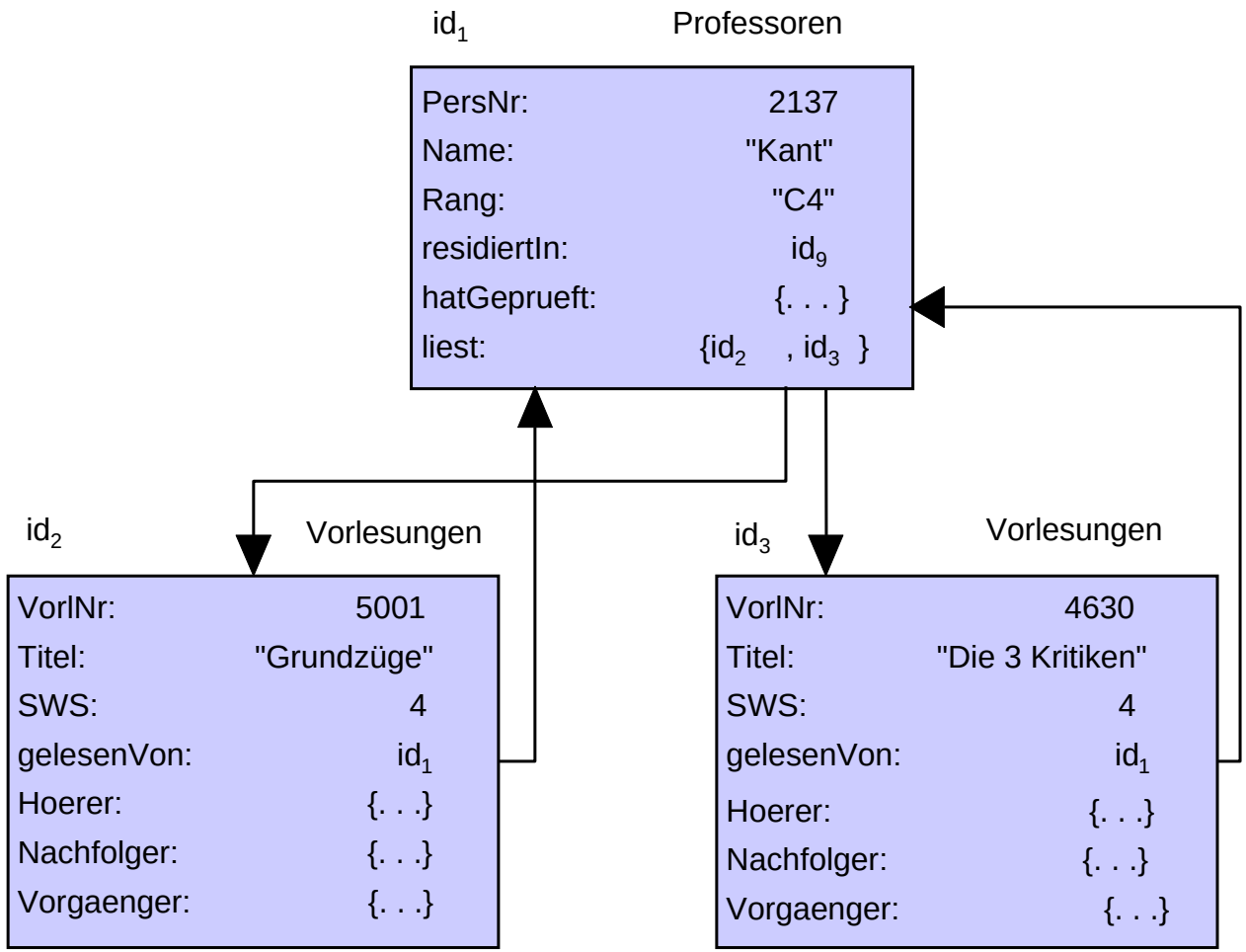
relational:

- Schlüssel
- vom Anwender erdacht

objektorientiert:

- Objektidentifikator OID
- vom System generiert
- unabhängig vom Zustand
- unabhängig vom Speicherort
- transient + persistent
- momentaner Speicherort = Tabelle[OID]

Objekte aus der Universitätswelt



Objekttypen-Definition

- die Typeigenschaften
  mit Generalisierungs- und
  Spezialisierungangaben

- die Strukturbeschreibung
  mit Attributen und Beziehungen

- die Verhaltensbeschreibung
  mit Operationen

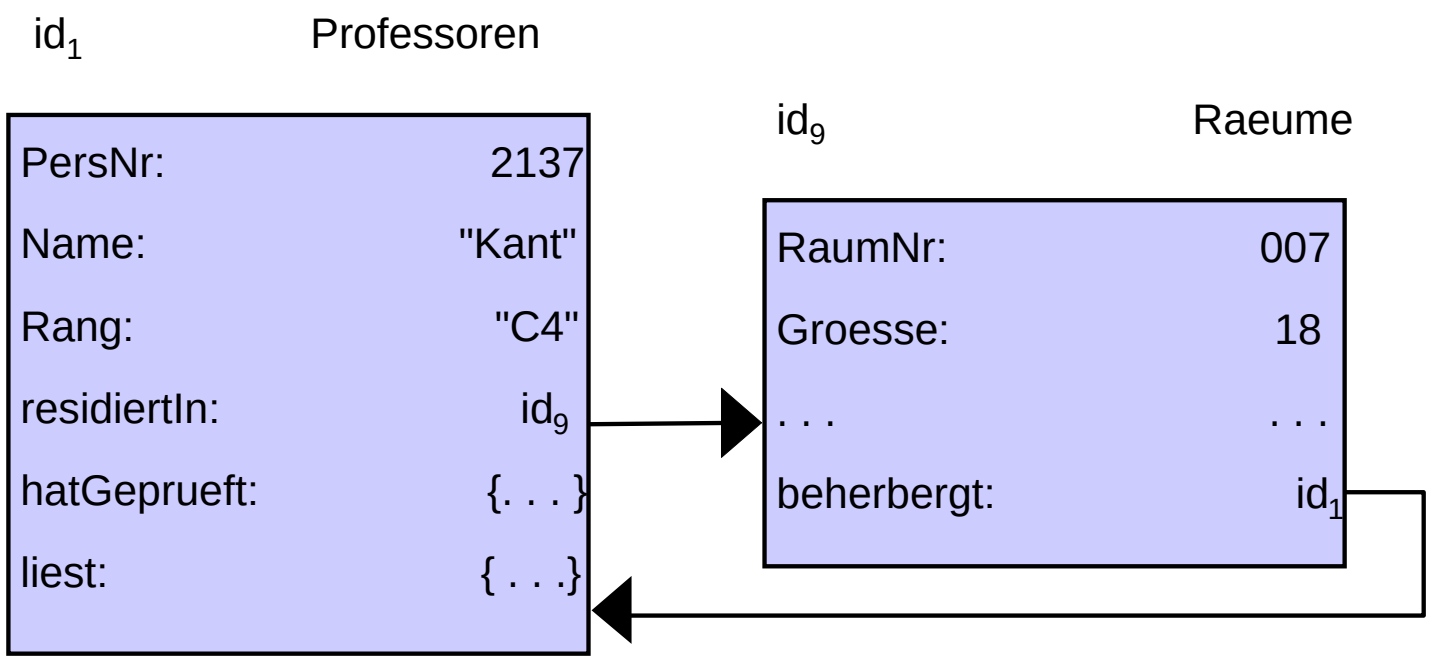
Attribute

```
class Professoren {
  attribute long   PersNr;
  attribute string Name;
  attribute string Rang;
};
```

strukturierte Attribute

```
class Person {
  attribute string Name;
  attribute struct Datum {
    short Tag;
    short Monat;
    short Jahr;
  } GebDatum;
};
```

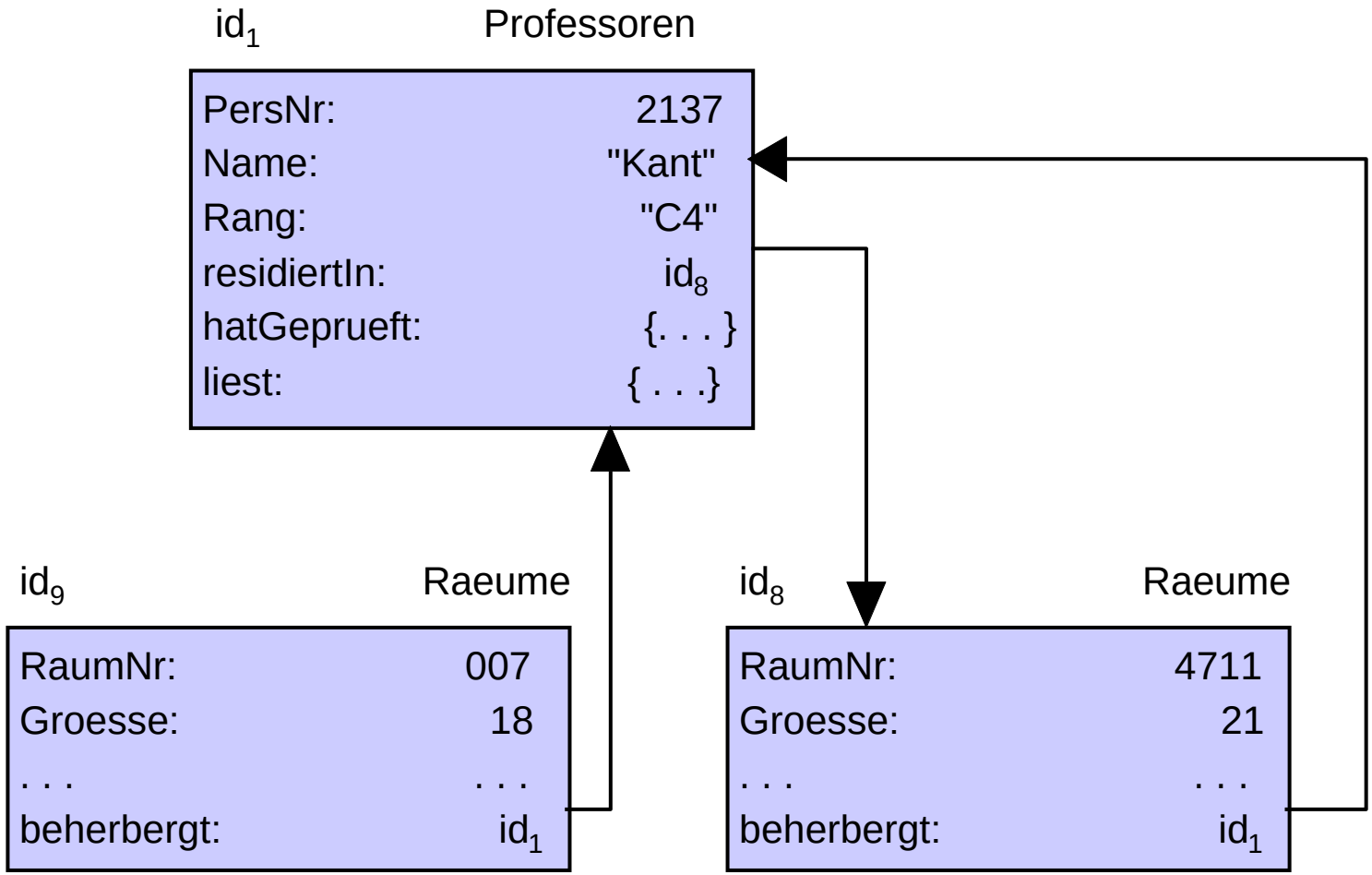
1:1-Beziehung



1:1-Beziehung

```
class Professoren {
  attribute long  PersNr;
  ...
  relationship Raeume residiertIn;
};

class Raeume {
  attribute long  RaumNr;
  attribute short Groesse;
  ...
  relationship Professoren beherbergt;
};
```

Inkonsistenter Zustand einer Beziehung



Referentielle Integrität

```
class Professoren {
  attribute long  PersNr;
  ...
  relationship Raeume residiertIn
                       inverse Raeume::beherbergt;
};

class Raeume {
  attribute long  RaumNr;
  attribute short Groesse;
  ...
  relationship Professoren beherbergt
                    inverse Professoren::residiertIn;
};
```

1:N-Beziehung

```
class Professoren {
  ...
  relationship set <Vorlesungen> liest
             inverse Vorlesungen::gelesenVon;
};

class Vorlesungen {
  ...
  relationship Professoren gelesenVon
              inverse Professoren::liest;
};
```

Binäre N:M-Beziehung

```
class Studenten {
   ...
   relationship set <Vorlesungen> hoert
            inverse Vorlesungen::Hoerer;
};

class Vorlesungen {
   ...
   relationship set <Studenten> Hoerer
                inverse Studenten::hoert;
};
```

Rekursive N:M-Beziehung

```
class Vorlesungen {
   ...
   relationship set <Vorlesungen> Vorgaenger
            inverse Vorlesungen::Nachfolger;

   relationship set <Vorlesungen> Nachfolger
            inverse Vorlesungen::Vorgaenger;
};
```

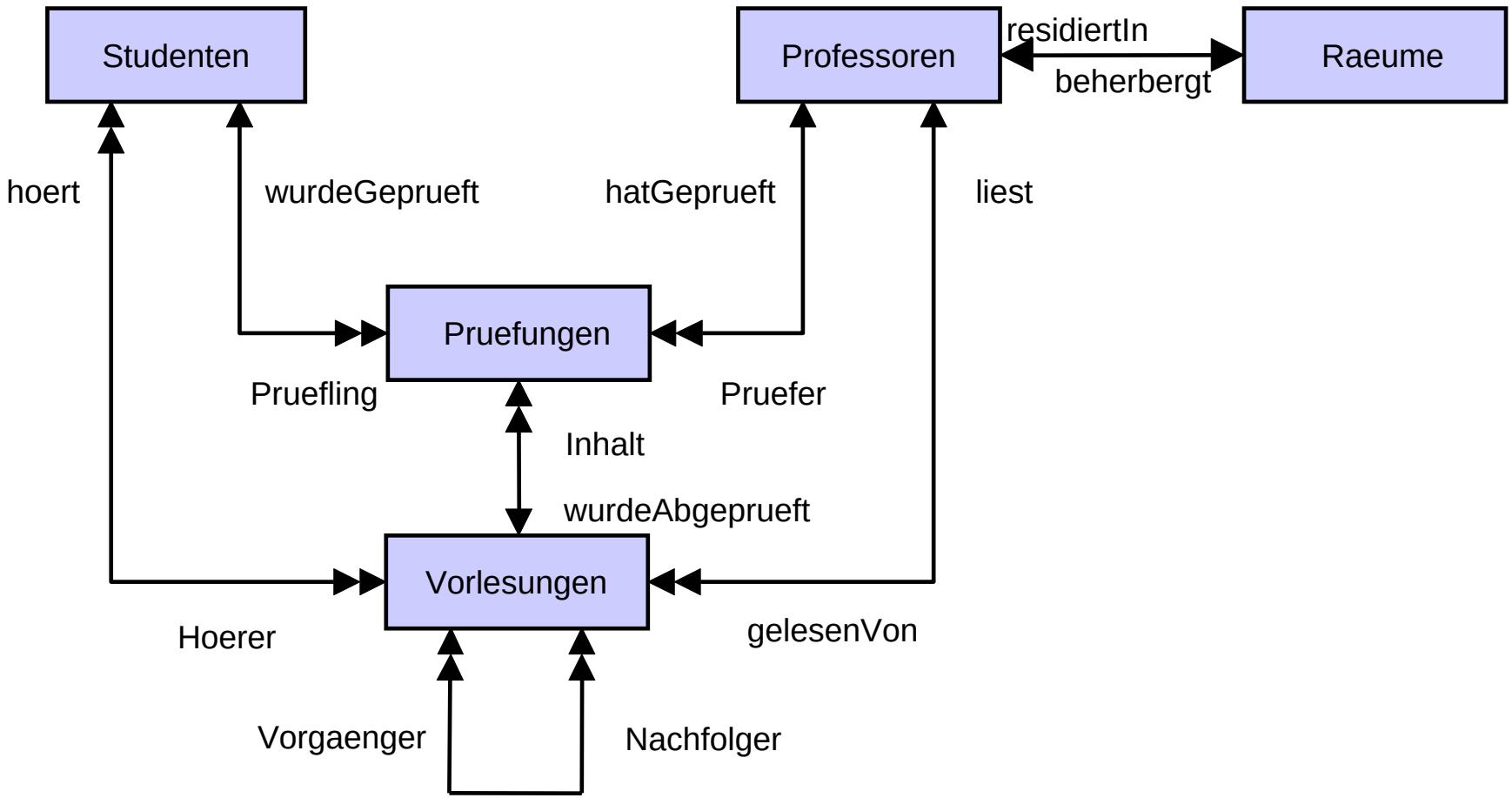
ternäre Beziehungen

Ternäre (oder *n* > *3* stellige) Beziehungen benötigen einen eigenständigen Objekttyp

pruefen : {[MatrNr, VorlNr, PersNr, Note]}

```
class Prüfungen {
  attribute    float       Note;
  relationship Professoren Prüfer
               inverse Professoren::hatGeprueft;
  relationship Studenten   Pruefling
               inverse Studenten::wurdeGeprueft;
  relationship Vorlesungen Inhalt
               inverse Vorlesungen::wurdeAbgeprueft;
};
```

Universität



Universität

```
class Pruefungen {
  attribute    float       Note;
  relationship Professoren Pruefer   inverse Professoren::hatgeprueft;
  relationship Studenten   Pruefling inverse Studenten::wurdegeprueft;
  relationship Vorlesungen Inhalt    inverse Vorlesungen::wurdeAbgeprueft;
};
class Professoren {
  attribute    long            PersNr;
  attribute    string          Name;
  attribute    string          Rang;
  relationship Raeume          residiertIn inverse Raeume::beherbergt;
  relationship set<Vorlesungen> liest       inverse Vorlesungen::gelesenVon;
  relationship set<Pruefungen>  hatgeprueft inverse Pruefungen::Pruefer;
};
class Vorlesungen {
  attribute    long            VorlNr;
  attribute    string          Titel;
  attribute    short           SWS;
  relationship Professoren     gelesenVon inverse Professoren::liest;
  relationship set<Studenten>  Hoerer     inverse Studenten::hoert;
  relationship set<Vorlesungen> Nachfolger inverse Vorlesungen::Vorgaenger;
  relationship set<Vorlesungen> Vorgaenger inverse Vorlesungen::Nachfolger;
  relationship set<Pruefungen>  wurdeAbgeprueft inverse Pruefungen::Inhalt;
};
class Studenten {
  attribute    long            MatrNr;
  attribute    string          Name;
  attribute    short           Semester;
  relationship set<Pruefungen>  wurdeGeprueft inverse Pruefungen::Pruefling;
  relationship set<Vorlesungen> hoert         inverse Vorlesungen::Hoerer;
};
```

Extensionen und Schlüssel

```
class Studenten (extent AlleStudenten key MatrNr) {
  attribute    long              MatrNr;
  attribute    string            Name;
  attribute    short             Semester;

  relationship set<Vorlesungen> hoert
                  inverse Vorlesungen::Hoerer;

  relationship set<Pruefungen>  wurdeGeprueft
                  inverse Pruefungen::Pruefling;
};
```

Operationen

- Objekt erzeugen mit Konstruktor
- Zustand erfragen mit Observer
- Zustand verändern mit Mutator
- Objekt zerstören mit Destruktor

Modellierung des Verhaltens

```
class Professoren {
  exception hatNochNichtGeprueft { };
  exception schonHoechsteStufe   { };
  ...
  float wieHartAlsPruefer() raises (hatNochNichtgeprueft);
  void befoerdert() raises (schonHoechsteStufe);
};
```

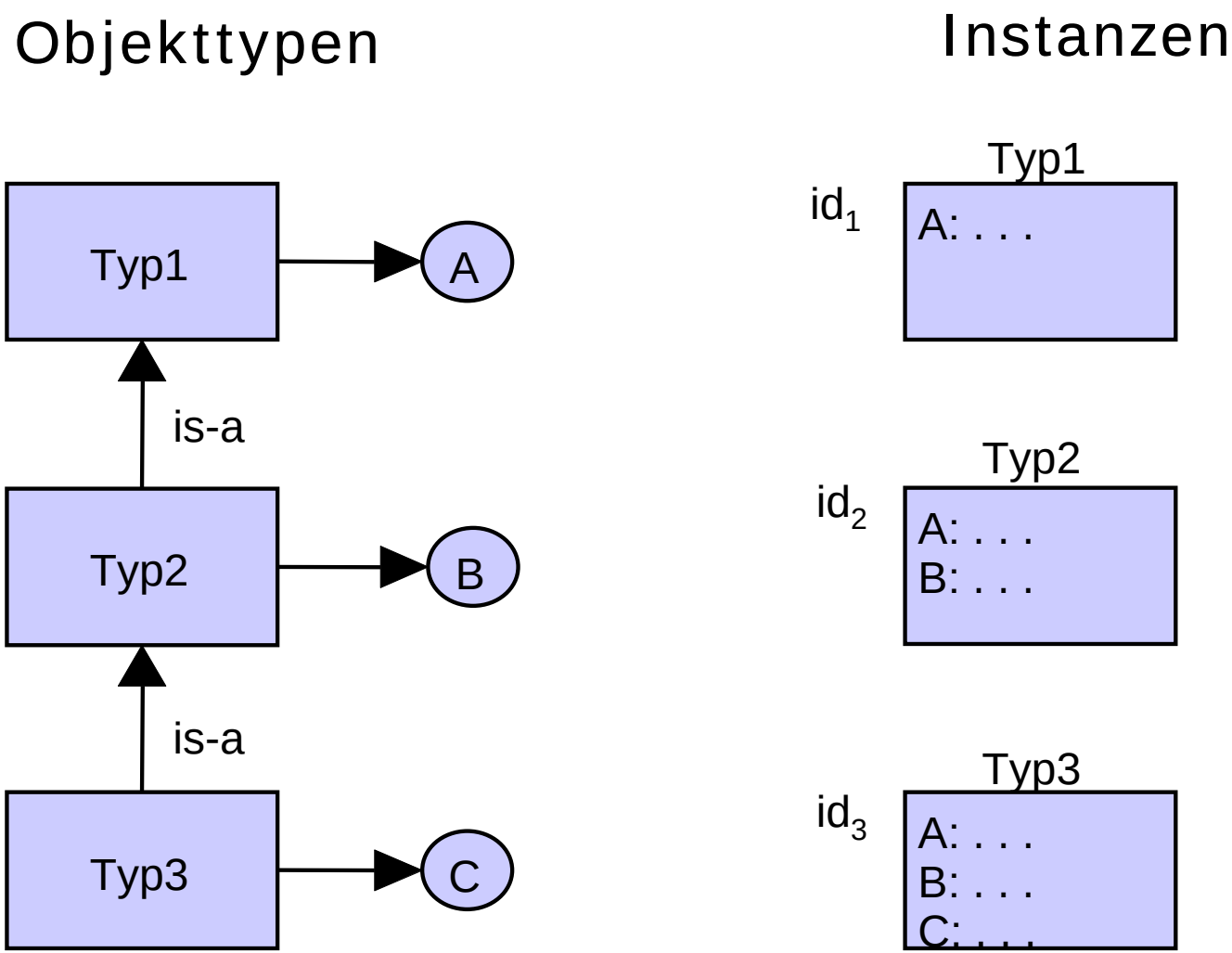
in C++:

```
meinLieblingsProf->befoerdert();
```

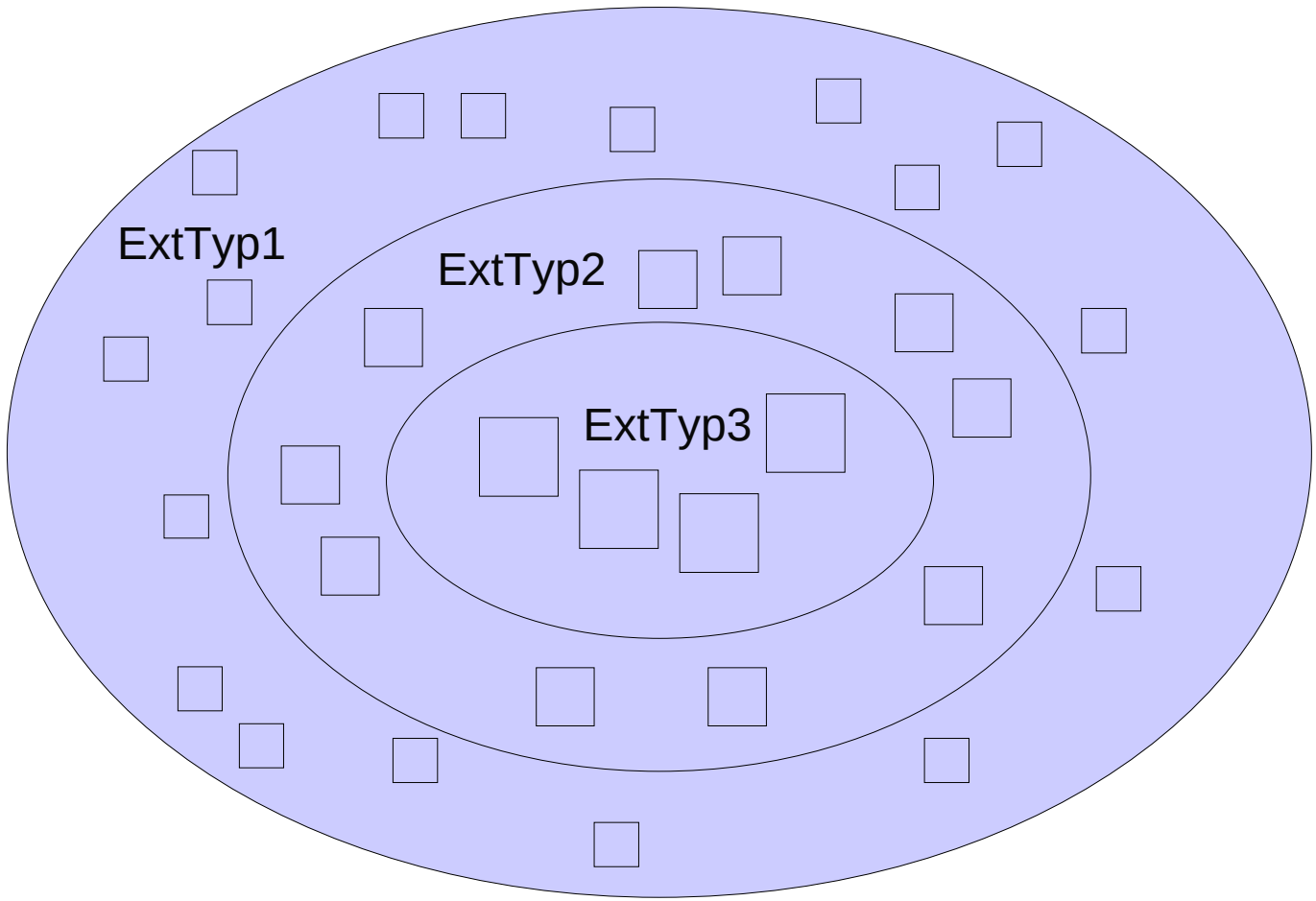
in OQL:

```
select p.wieHartAlsPruefer()
from p in AlleProfessoren
where p.name = "Kant";
```

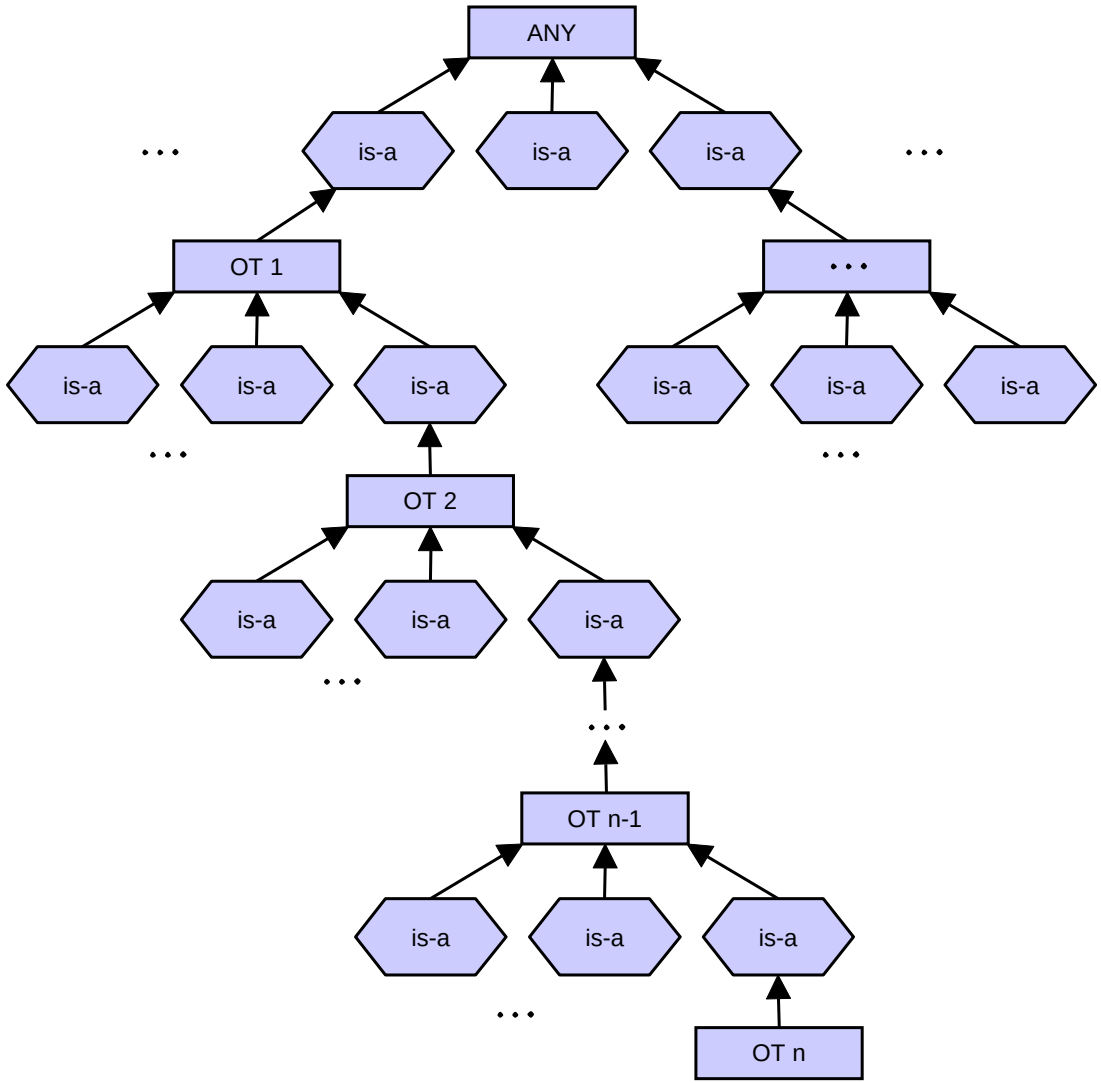
Vererbung


Objekttypen
Instanzen
Typ1
A
is-a
Typ2
B
is-a
Typ3
C
Typ1
id1
A: . . .
Typ2
id2
A: . . .
B: . . .
Typ3
id3
A: . . .
B: . . .
C: . . .

Subtypisierung



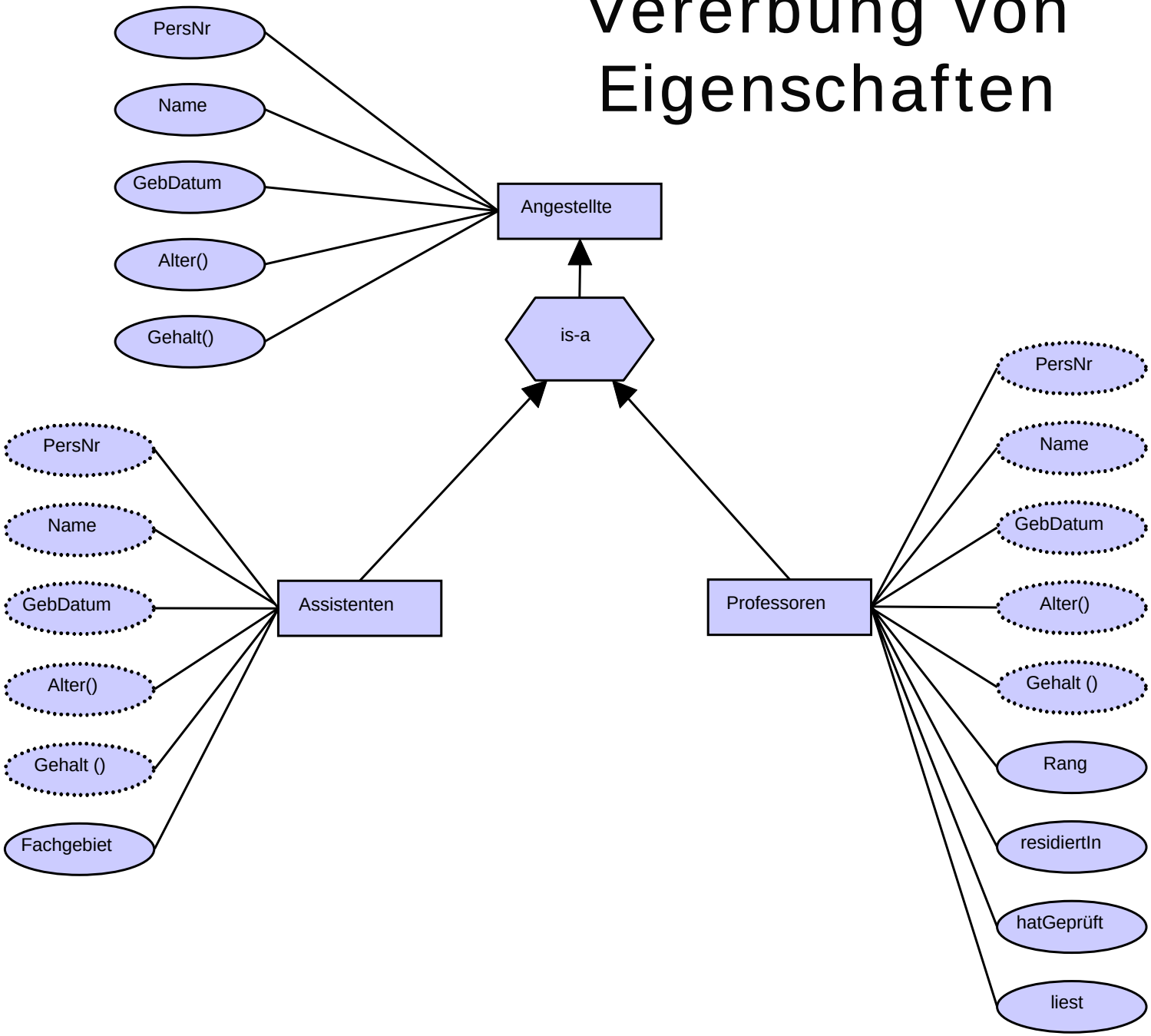
Typhierarchie bei einfacher Vererbung



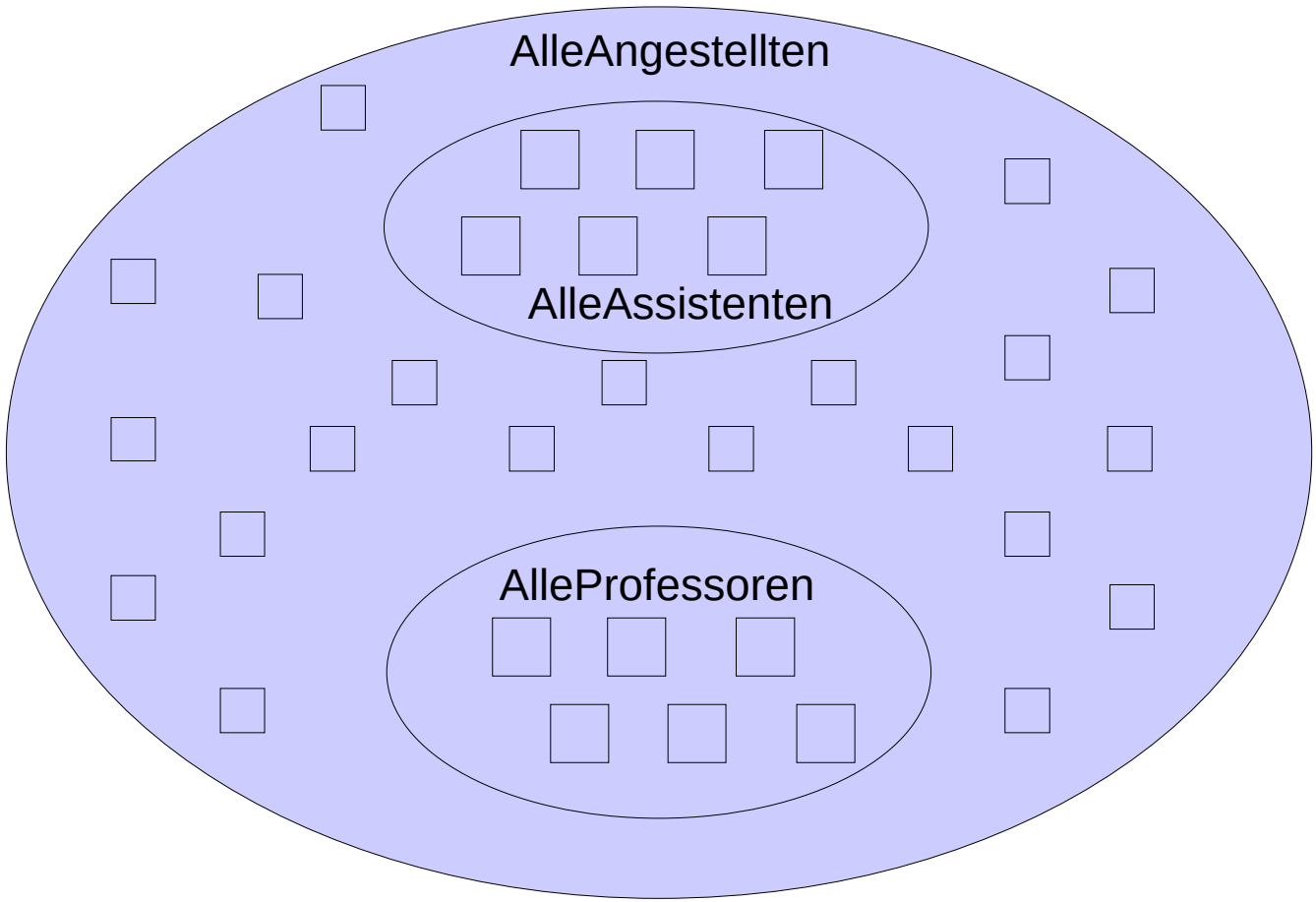
Vererbung

```
class Angestellte (extent AlleAngestellte) {
  attribute long PersNr;
  attribute string Name;
  attribute date GebDatum;
  short Alter();
  long Gehalt();
};

class Assistenten extends Angestellte (extent AlleAssistenten) {
  attribute string Fachgebiet;
};

class Professoren extends Angestellte (extent AlleProfessoren) {
  attribute string Rang;
  relationship Raeume residiertIn inverse Raeume::beherbergt;
  relationship set(Vorlesungen) liest inverse Vorlesungen::gelesenVon;
  relationship set(Pruefungen) hatgeprueft inverse Pruefungen::Pruefer;
};
```

Vererbung von Eigenschaften



Visualisierung der Extension



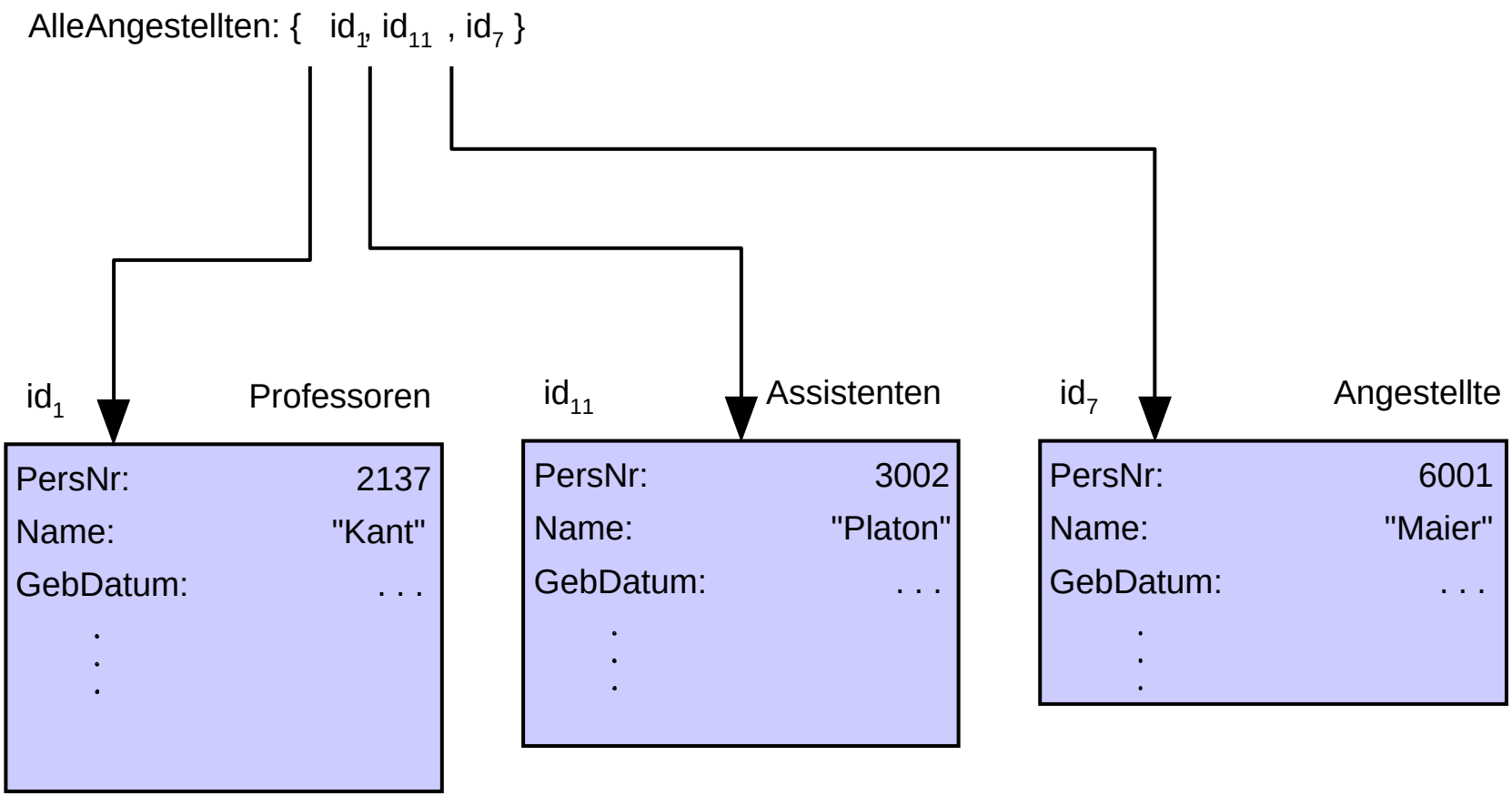
Verfeinerung bzw. *Spezialisierung*

Operationen können bei Vererbung spezialisiert werden:

z.B. **`Gehalt():`**

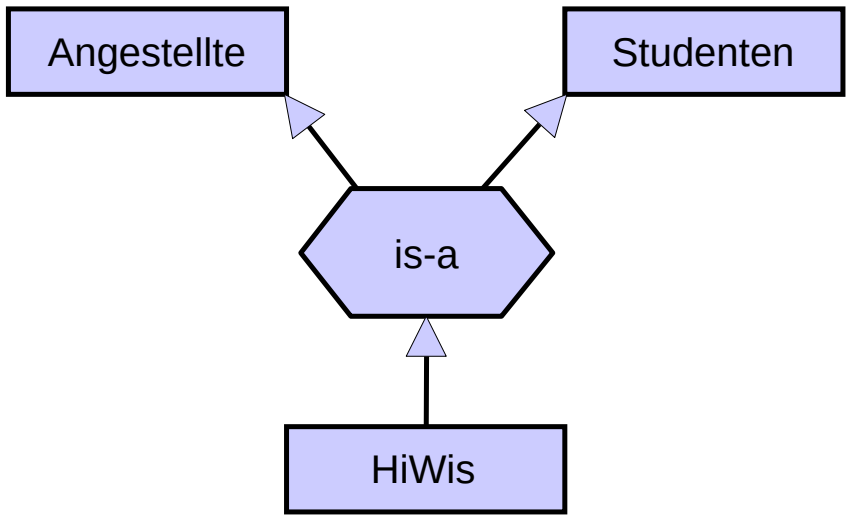
Angestellte erhalten 2000 + (Alter() - 21) * 100 €,
Assistenten erhalten 2500 + (Alter() - 21) * 125 €,
Professoren erhalten 3000 + (Alter() - 21) * 150 €.

Extension AlleAngestellten




```
select sum(a.Gehalt()) from a in AlleAngestellten
```

Mehrfachvererbung




```
class HiWis extends Studenten, Angestellte (extent AlleHiwis)
{
  attribute short Arbeitsstunden;
  ...
}
```

Problem:

Attribut Name wird von Studenten und Angestellte geerbt !

OQL: select

Liste alle C4-Professoren (als Objekte):

```
select p
from p in AlleProfessoren
where p.Rang = "C4";
```

Liste Name und Rang aller C4-Professoren (als Werte):

```
select p.Name, p.Rang
from p in AlleProfessoren
where p.Rang = "C4";
```

Liste Name und Rang aller C4-Professoren (als Tupel):

```
select struct (n: p.Name, r: p.Rang)
from p in AlleProfessoren
where p.Rang = "C4";
```

OQL: struct

Liste alle Angestellte mit einem Gehalt über 50.000 €:

```
select struct (
        n: a.Name,
        g: a.Gehalt()
      )
from a in AlleAngestellte
where a.Gehalt() > 50.000;
```

Liste Name und Lehrbelastung aller Professoren:

```
select struct (
        n: p.Name,
        a: sum(select v.SWS from v in p.liest)
      )
from p in AlleProfessoren;
```

OQL: Gruppierung

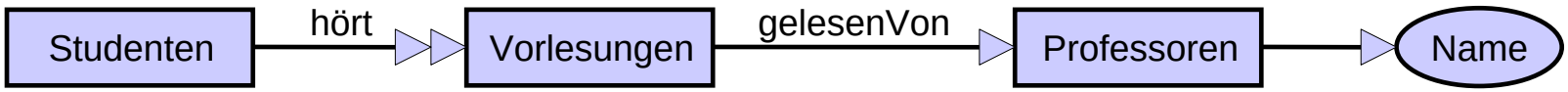
Gruppiere alle Vorlesungen nach der Semesterstundenzahl:

```
select *
from v in AlleVorlesungen
group by kurz:   v.SWS  < 3,
         mittel: v.SWS  = 3,
         lang:   v.SWS  > 3;
```

Das Ergebnis sind drei Tupel vom Typ

```
struct (
      kurz:          boolean,
      mittel:        boolean,
      lang:          boolean,
      partition :    bag<struct(v: Vorlesungen)>
)
```

OQL: Pfadausdrücke

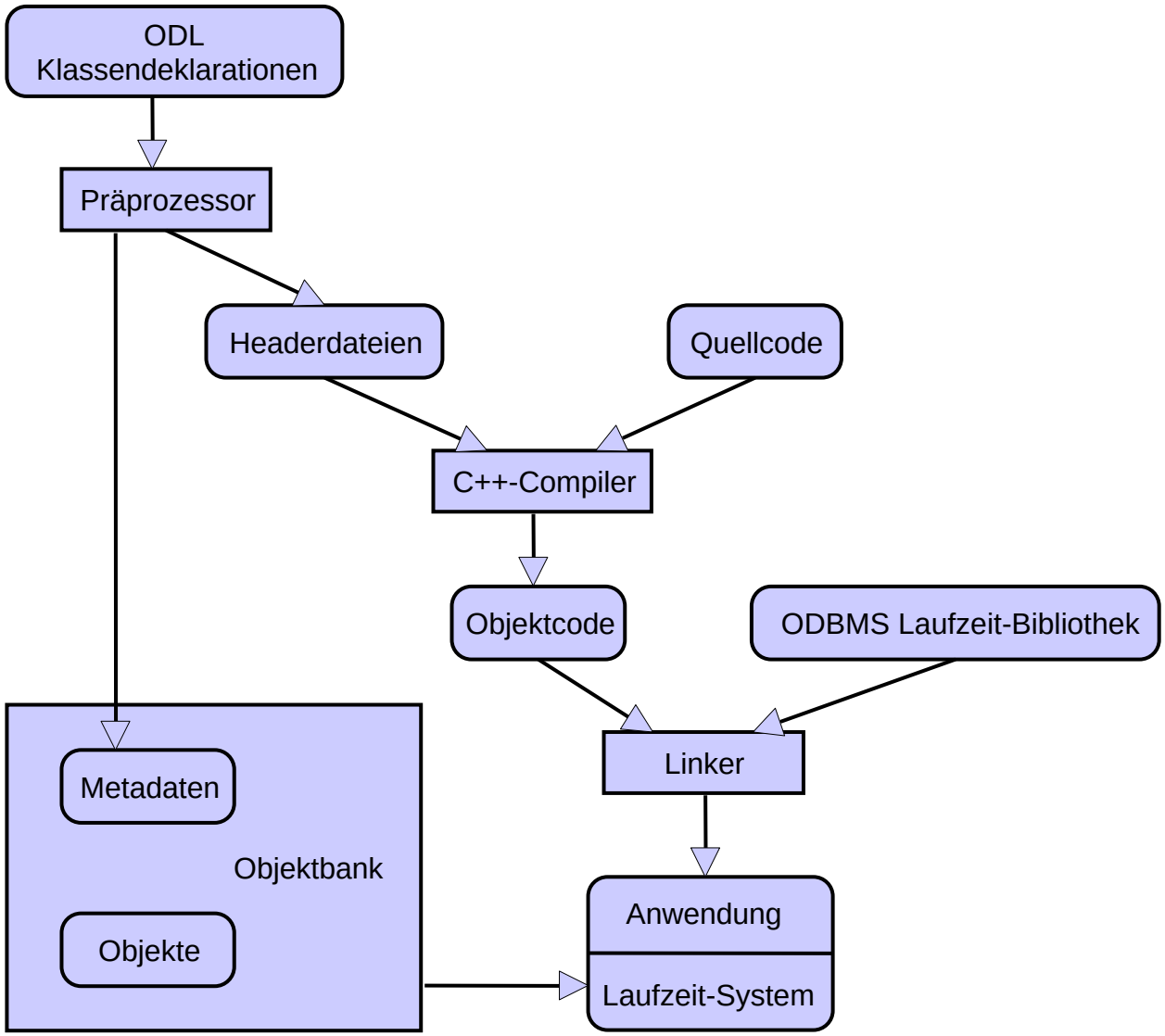



```
select s.Name
from s in AlleStudenten, v in s.hoert
where v.gelesenVon.Name = 'Sokrates'
```

Erzeugung von Objekten

```
Vorlesungen (
     vorlNr:       4711,
     Titel:        "Selber Atmen",
     SWS:          4,
     gelesenVon : (
          select p
          from p in AlleProfessoren
          where p.Name = "Sokrates"
     )
);
```

C++ Einbettung



C++ Quelltext

```
const char _liest[]      = "liest";
const char _gelesenVon[] = "gelesenVon";

class Vorlesungen : public d_Object {
  d_String Titel;
  d_Short  SWS;
  ...
  d_Rel_ref <Professoren, _liest> gelesenVon;
}

class Professoren : public Angestellte {
  d_Long PersNr;
  ...
  d_Rel_Set <Vorlesungen, _gelesenVon> liest;
}
```

C++ Instantiierung

```
d_Ref <Professoren> Russel;

Russel = new(UniDB,"Professoren") Professoren
                    (2126,"Russel","C4", ...);
```

C++ Abfrage

```
d_Bag <Studenten> Schüler;
char * profname = "Sokrates";
d_OQL_Query anfrage ("select s
                      from s in v.Hoerer,
                      v in p.liest,
                      p in AlleProfessoren
                      where p.Name = $1");
anfrage << profname;
d_oql_execute(anfrage, Schüler);
```