Datenintegrität

- Einschränkung der möglichen Datenbankzustände und -übergänge auf die in der Realität möglichen
- Formulierung von Integritätsbedingungen ist die wichtigste Aufgabe des DB-Administrators!

Statische vs. dynamische Integritätsbedingungen

- Statische Integritätsbedingungen
  - Bedingungen an (jeden) einzelnen Zustand der Datenbasis
- dynamische Integritätsbedingungen
  - Bedingungen an Folgen von Datenbankzuständen
  - Transitionale Bedingungen: Übergang zwischen zwei Zuständen
  - Temporale Bedingungen: Zustandsfolgen beliebiger Länge

Statische vs. dynamische Integritätsbedingungen

- Statische Integritätsbedingungen
  - *Die in der Relation hören referenzierten Studenten und Vorlesungen müssen als Tupel in den entsprechenden Relationen existieren*
  - *Vorlesungen müssen mindestens drei Hörer haben*
- dynamische Integritätsbedingungen
  1. Transitionale Bedingungen
     - *Die Semesterzahl eines Studenten darf nur steigen*
     - *Professoren dürfen nicht degradiert werden*
  2. Temporale Bedingungen
     - *Studenten müssen mindestens alle zwei Semester eine Prüfung ablegen*
  - Temporale Bedingungen können auf transitionale zurückgeführt werden - daher betrachten wir nur transitionale Bedingungen.
    - *Prüfe zu Beginn jedes Semesters (globale Relation lfdSemester), wann jeder Student seine letzte Prüfung abgelegt hat*

Statische Integritätsbedingungen

- Attributdomänen
- Schlüssel
  - Kandidatenschlüssel
  - Primärschlüssel
  - Fremdschlüssel
- Beziehungskardinalitäten
- Inklusion bei Generalisierung

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

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

PersNr	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

Referentielle Integrität in SQL

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

- Beispiel:

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

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

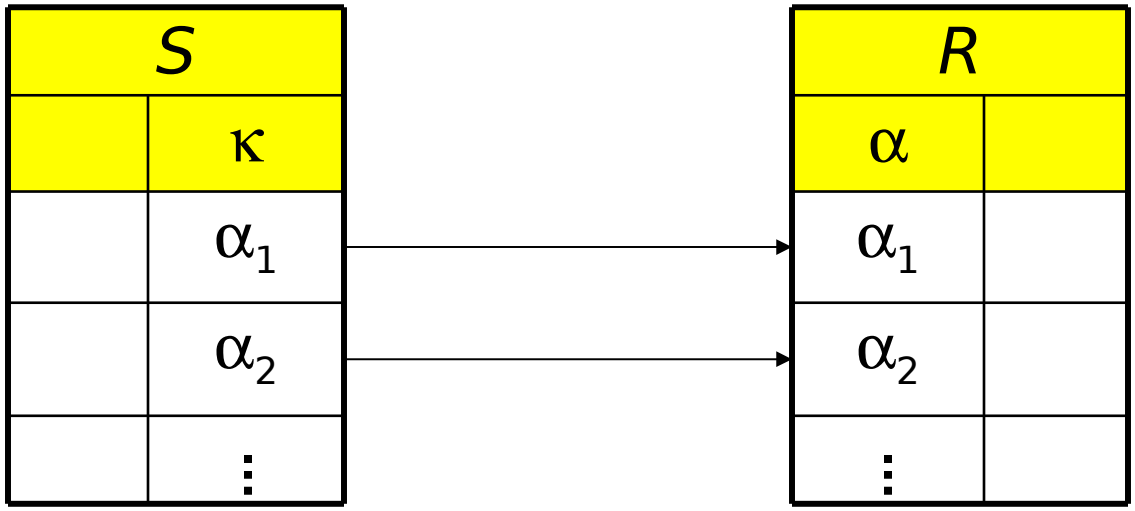
Einhaltung referentieller Integrität

Änderung von referenzierten Daten

- Default: Zurückweisen der Änderungsoperation
- Propagieren der Änderungen: **cascade**
- Verweise auf Nullwert setzen: **set null**

Einhaltung referentieller Integrität

Originalzustand



Änderungsoperationen

update *R*

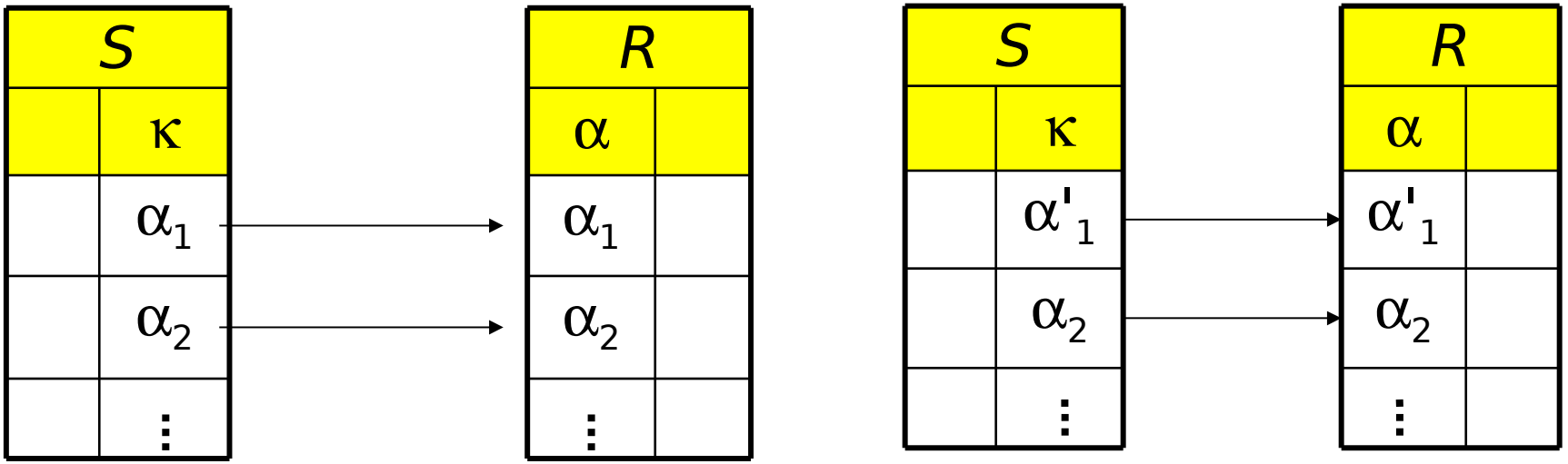
set $\alpha = \alpha'_1$

where $\alpha = \alpha_1$;

delete from *R*

where $\alpha = \alpha_1$;

Kaskadieren



create table *S*

(...,

κ **integer references** *R*

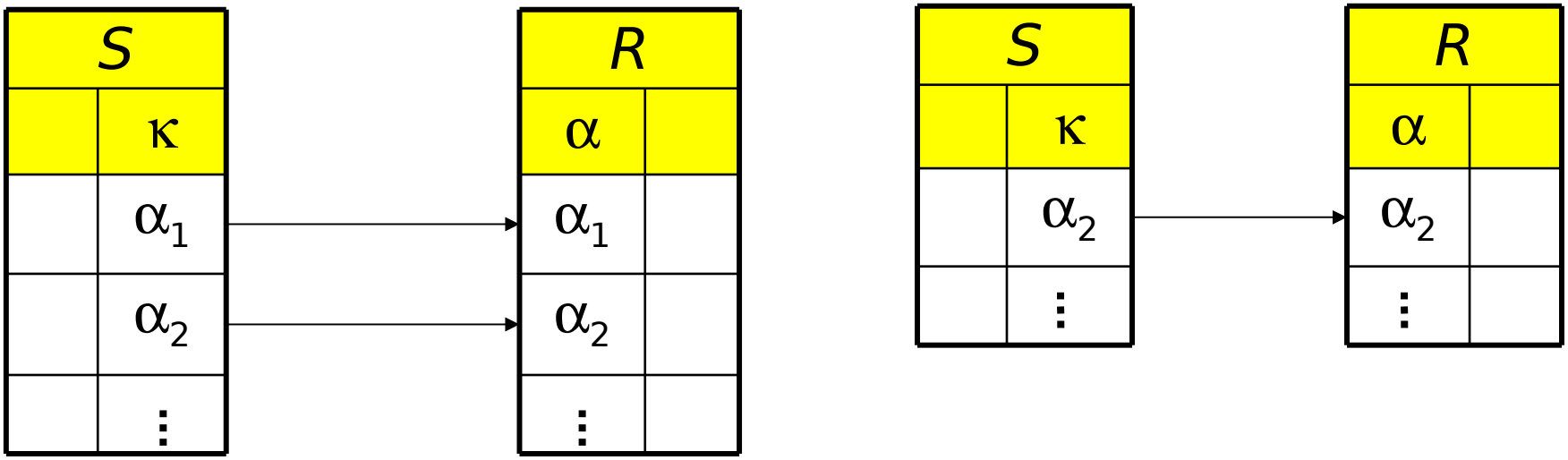
on update cascade);

update *R*

set $\alpha = \alpha'_1$

where $\alpha = \alpha_1$;

Kaskadieren



create table *S*

(...,

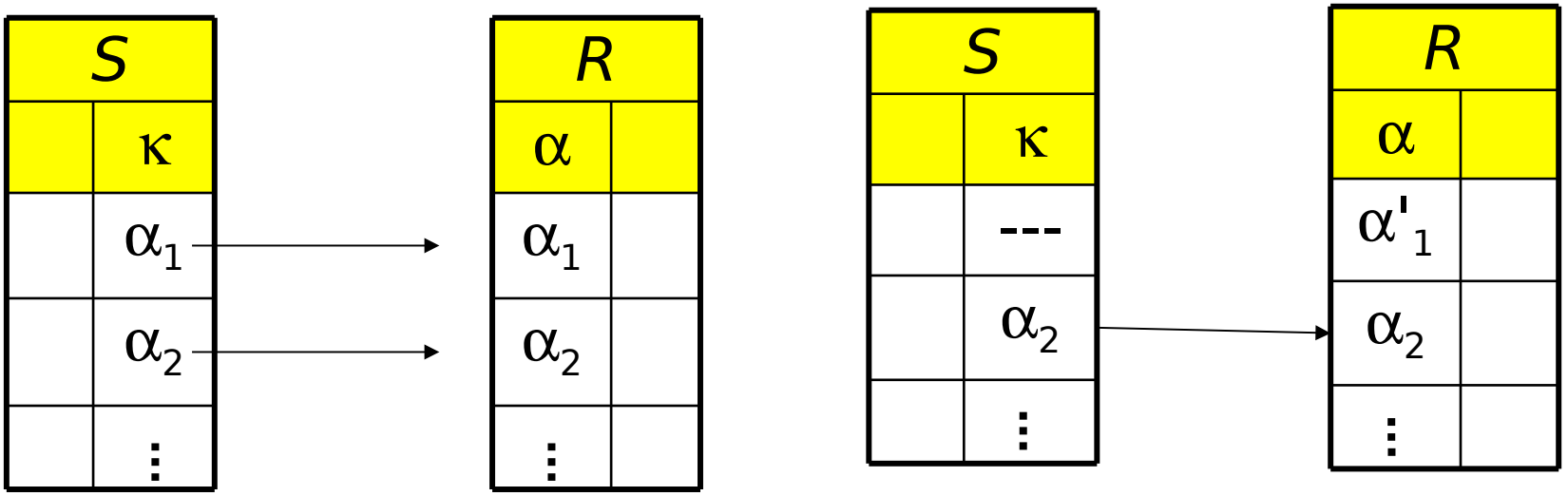
κ **integer references** *R*

on delete cascade);

delete from *R*

where $\alpha = \alpha_1$;

Auf Null setzen



create table *S*

(...,

κ **integer references** *R*

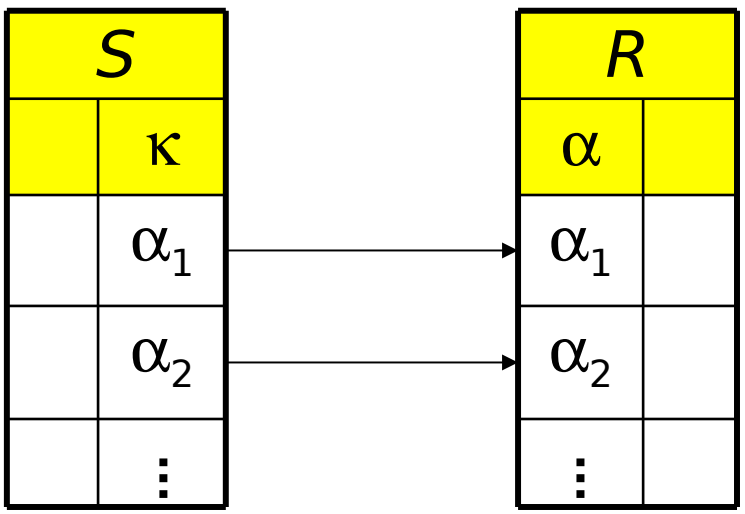
on update set null);

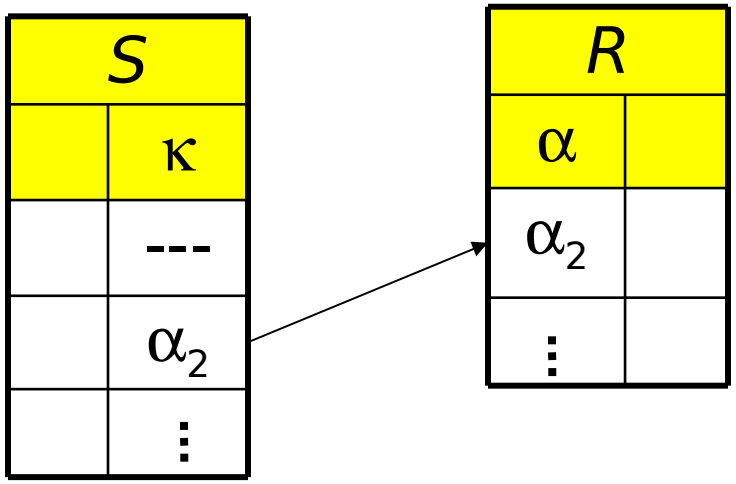
update *R*

set $\alpha = \alpha'_1$

where $\alpha = \alpha_1$;

Auf Null setzen





create table *S*

(...,

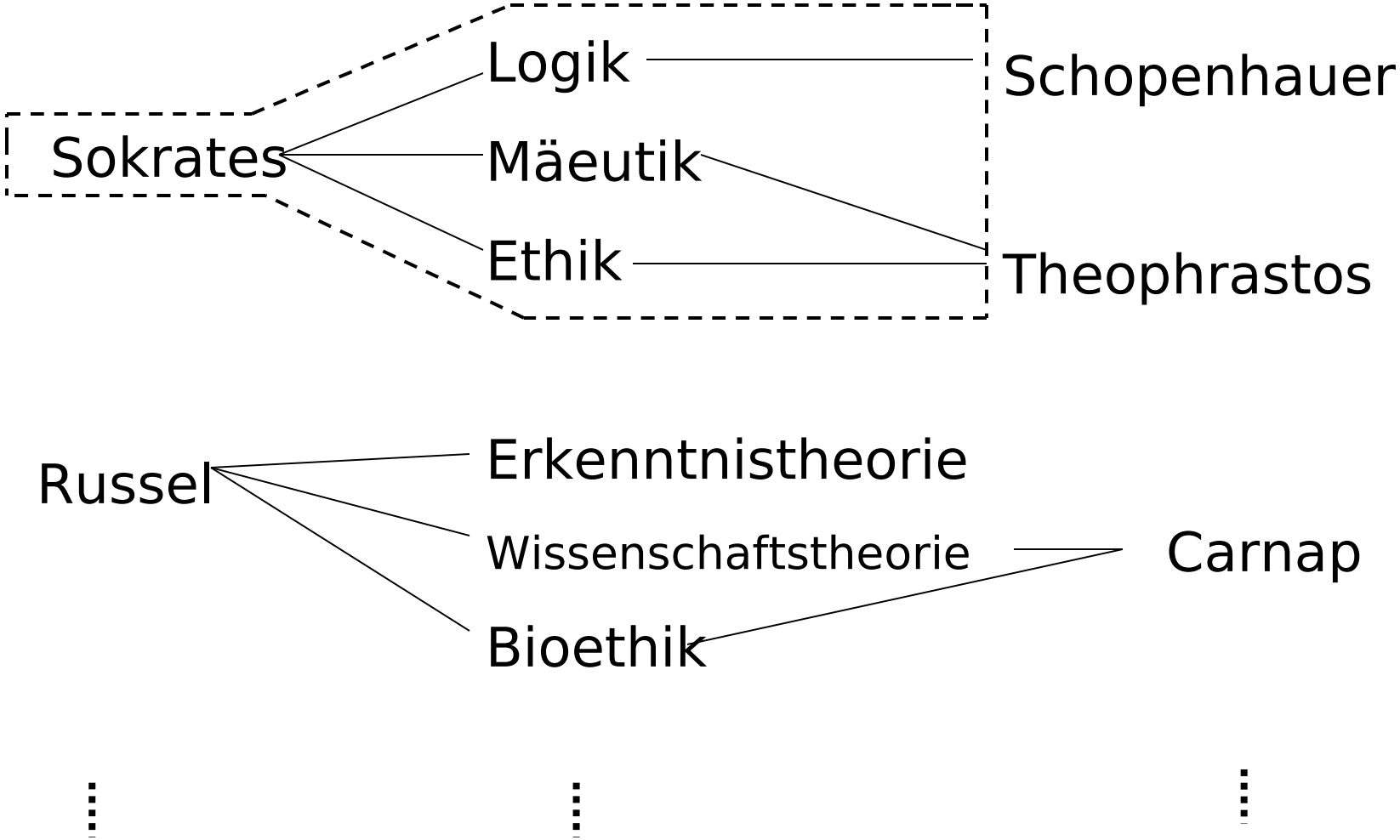
κ **integer references** *R*

on delete set null);

delete from *R*

where $\alpha = \alpha_1$;

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:

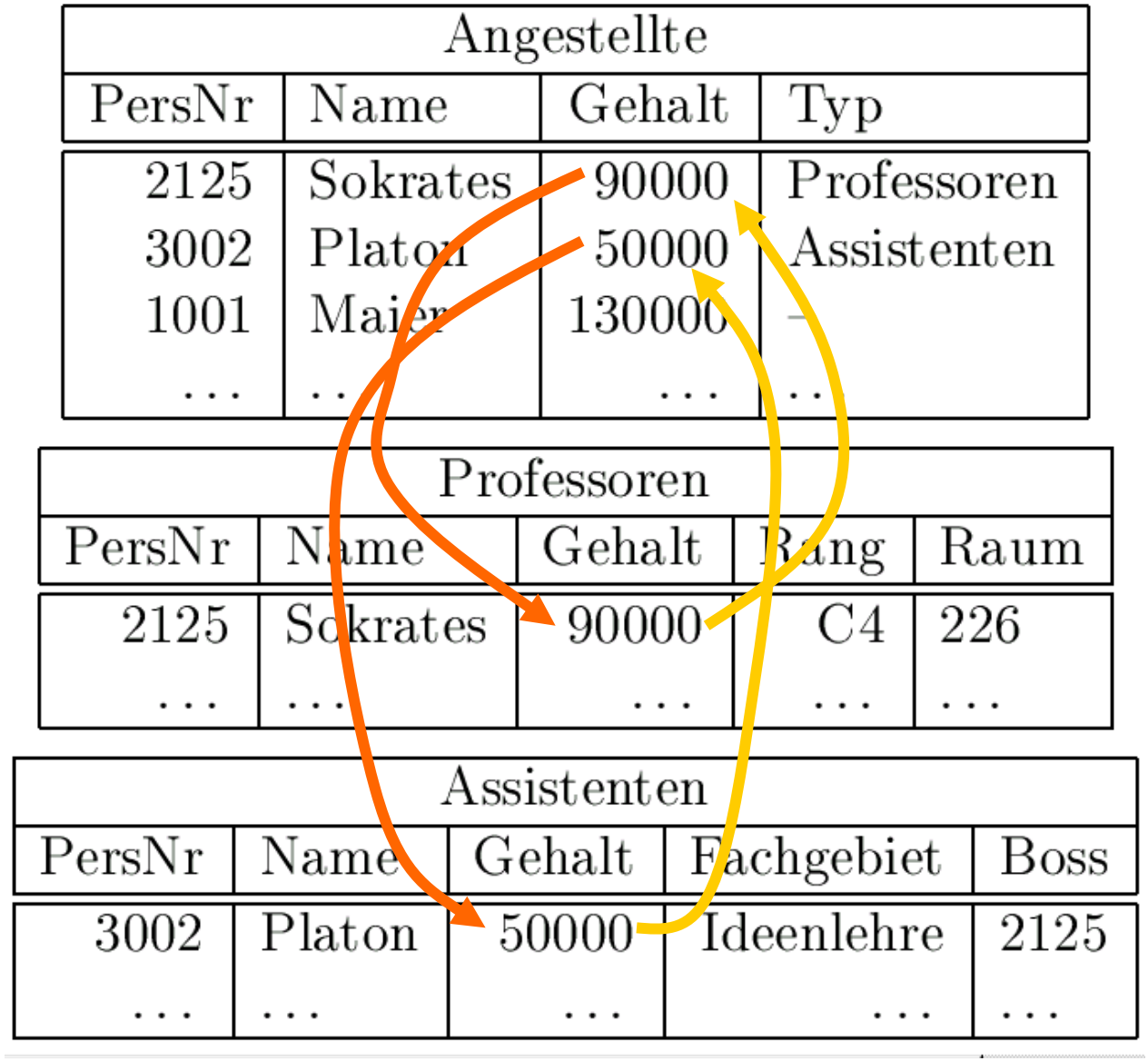
1. der **create trigger** Anweisung, gefolgt von einem Namen,
2. der Definition des Auslösers, in diesem Fall bevor eine Änderungsoperation (**before update on**) auf einer Zeile (**for each row**) der Tabelle *Professoren* ausgeführt werden kann,
3. einer einschränkenden Bedingung (**when**) und
4.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