Datenintegrität

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

Kapitel 5

Bisherige Integritätsbedingungen

- Schlüssel: es dürfen keine zwei Tupel existieren, die auf allen Schlüsselattributen gleich sind.
- Kardinalitäten der Beziehungen
- Generalisierungsbeziehung: jede Entität eines Typs muss auch in allen Obertypen enthalten sein
- Festgelegte Domänen für jedes Attribut

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

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

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

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 );
```

- Syntaktische Varianten zwischen den verschiedenen Herstellern werden hier beschrieben:
  - http://www.1keydata.com/sql/sql-primary-key.html
  - http://www.1keydata.com/sql/sql-foreign-key.html

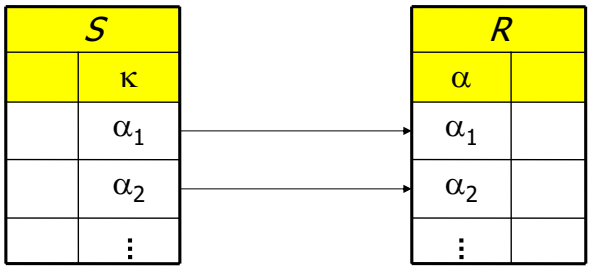
Einhaltung referentieller Integrität

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*

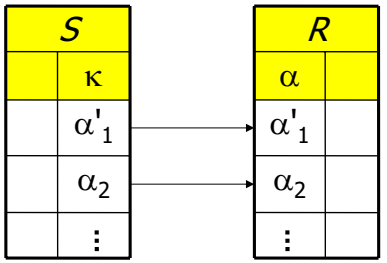
set $\alpha = \alpha'_1$

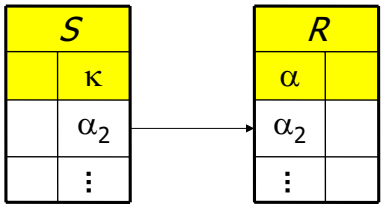
where $\alpha = \alpha_1$;

delete from *R*

where $\alpha = \alpha_1$;

Kaskadieren





create table *S*

(...,

κ **integer references** *R*

on update cascade);

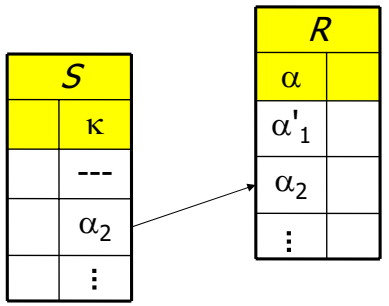
create table *S*

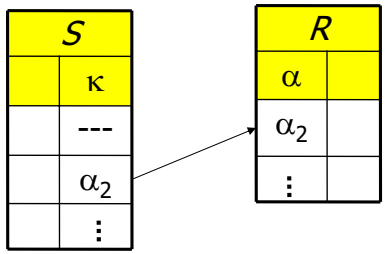
(...,

κ **integer references** *R*

on delete cascade);

Auf Null setzen





create table *S*

(...,

κ **integer references** *R*

on update set null);

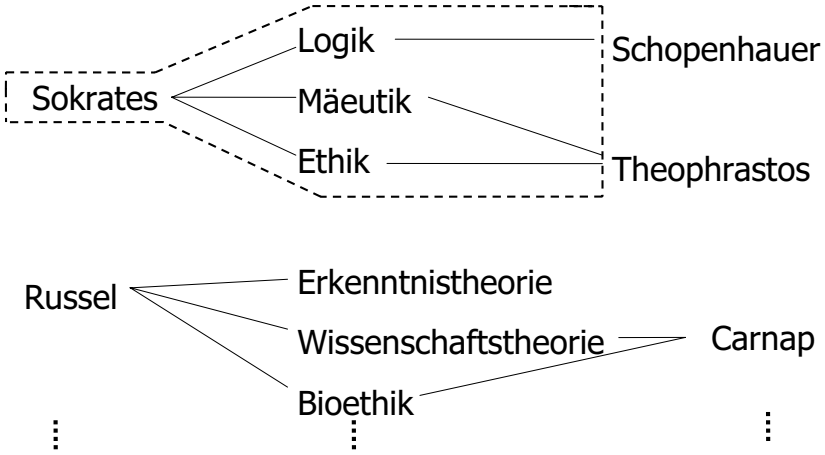
create table *S*

(...,

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,
        gelesenVon      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