

# Datenintegrität



## Integritätsbedingungen

- Schlüssel
- Beziehungskardinalitäten
- Attributdomänen
- Inklusion bei Generalisierung

## statische Integritätsbedingungen

- Bedingungen an den Zustand der Datenbasis

## dynamische Integritätsbedingungen

- Bedingungen an Zustandsübergänge

## Temporale Daten in SQL

- System-versioniert
- Anwendungs-versioniert

# 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  
    (  $\alpha$  integer primary key,  
      ... );
```

```
create table S  
    ( ...,  
       $\kappa$  integer references R );
```

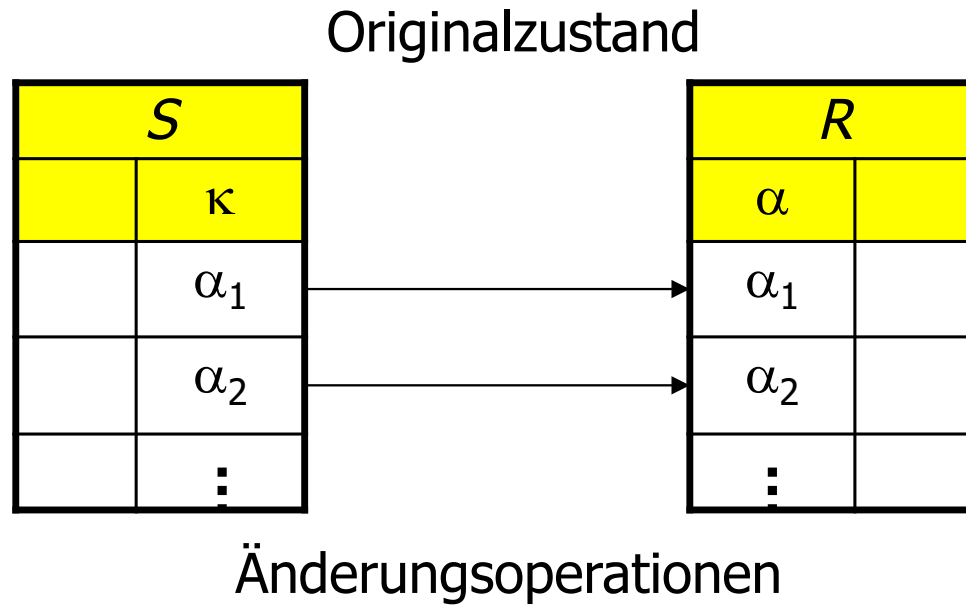
# Einhaltung referentieller Integrität



Änderung von referenzierten Daten

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

# Einhaltung referentieller Integrität



**update** *R*

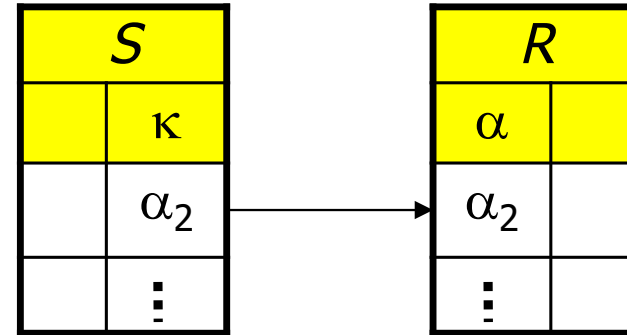
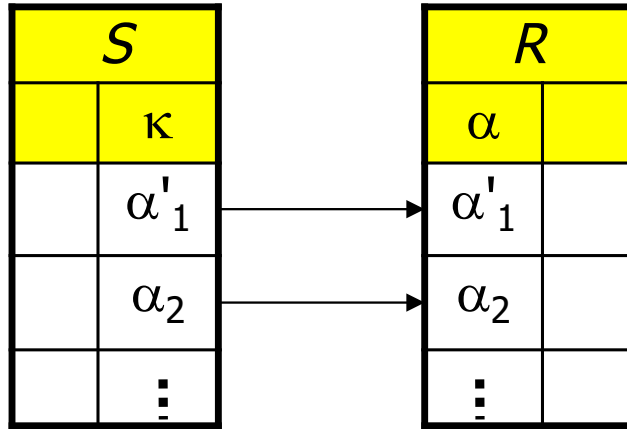
**set**  $\alpha = \alpha'_1$

**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** α = α'<sub>1</sub>  
**where** α = α<sub>1</sub>;

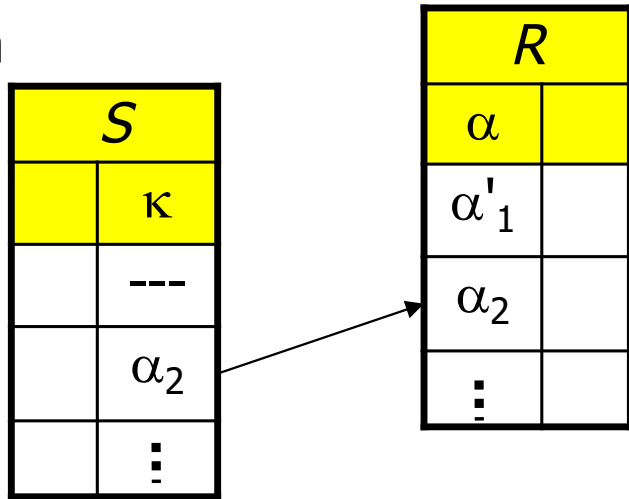
**create table S**

( ...,  
κ **integer references R**  
**on delete cascade** );

**delete from R**

**where** α = α<sub>1</sub>;

Auf Null setzen

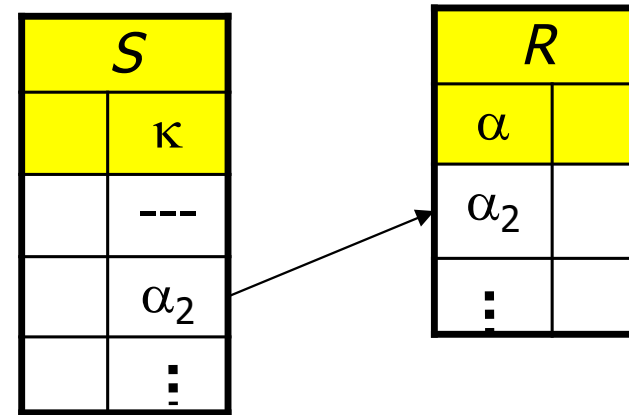


**create table S**

( ...,  
 κ integer references R  
 on update set null );

**update R**

**set** α = α'₁  
**where** α = α₁;



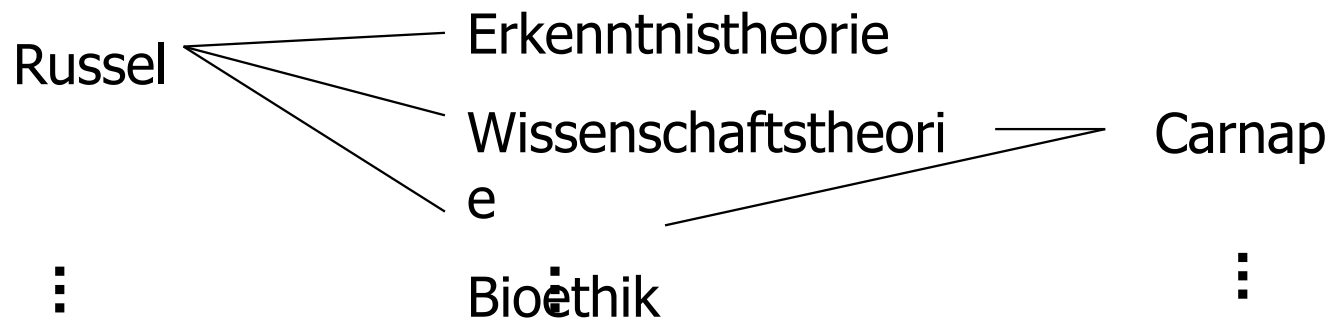
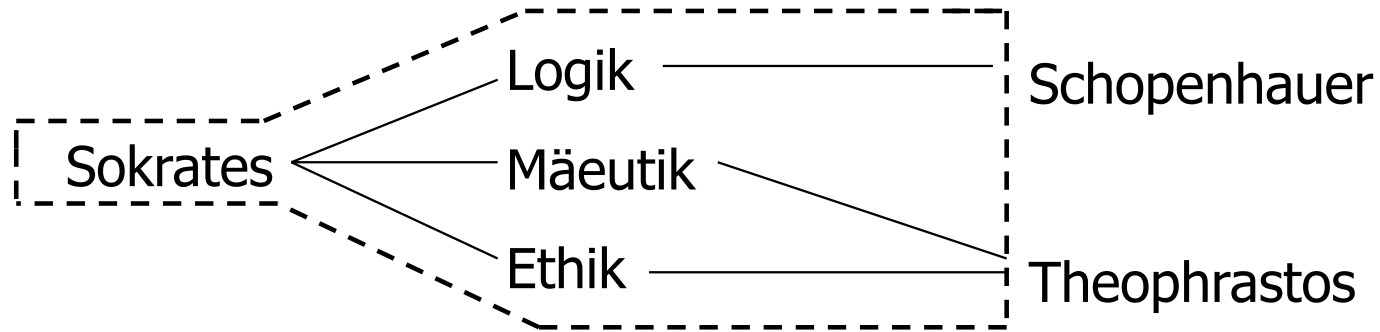
**create table S**

( ...,  
 κ integer references R  
 on delete set null );

**delete from R**

**where** α = α₁;

# 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:

```
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

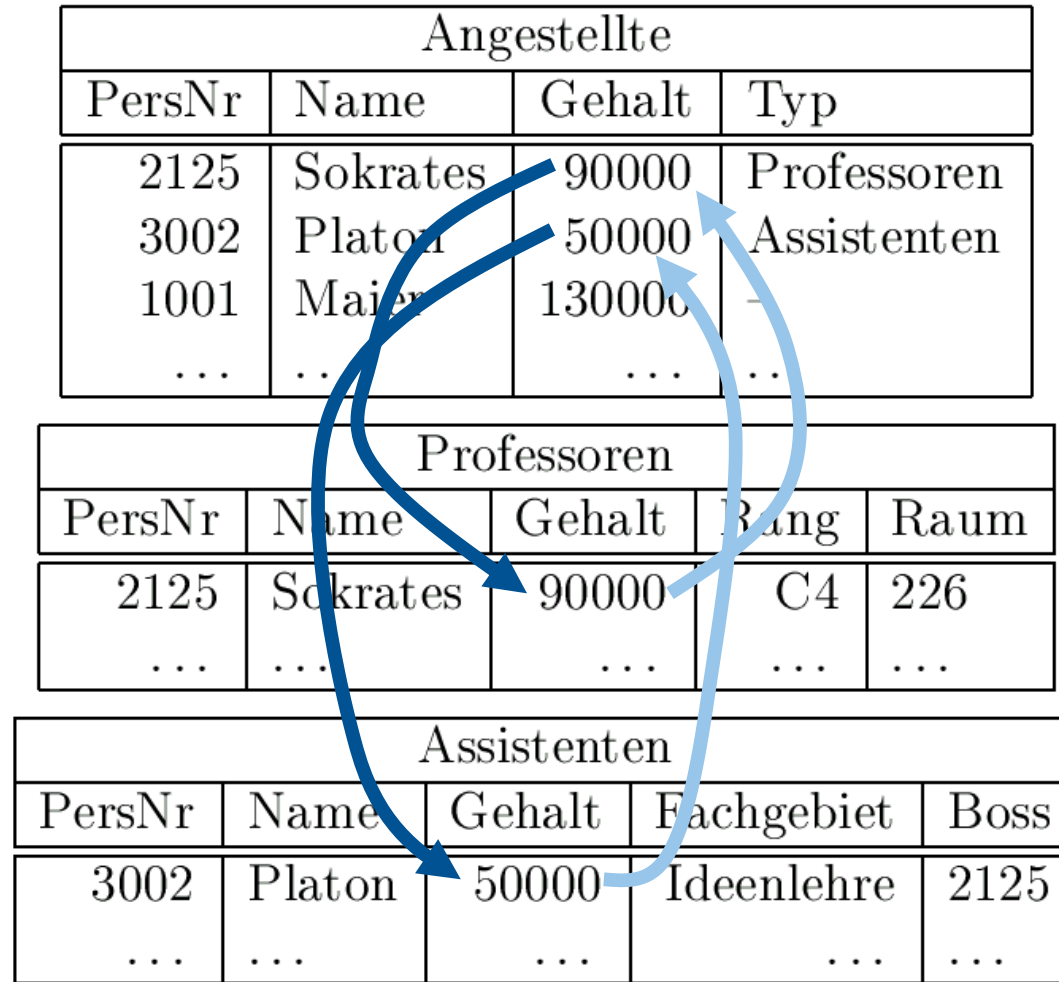
In der Prozedurdefinition bezieht sich *old* auf das noch unveränderte Tupel (den Originalzustand), *new* enthält bereits die Veränderungen durch die Operation.

# 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	Maijer	130000	-
...	...	...	...

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

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

System-versionierte Daten – transaction time Versionierung

```
create table Studiengebühren  
  ( Bundesland varchar(30) not null,  
    Beitrag integer not null,  
    Beginn date not null generated always as row start,  
    Ende date not null generated always as row end,  
    period for system_time (Beginn, Ende),  
    primary key(Bundesland)  
  ) with system versioning;
```

# Zustand der Relation nach Abschaffung der Studien-Gebühren in Bayern



Studiengebühren			
Bundesland	Beitrag	Beginn	Ende
Thüringen	0	1990.10.03	9999.12.31
Bayern	0	1990.10.03	2007.04.01
Bayern	500	2007.04.01	2013.10.01
Bayern	0	2013.10.01	9999.12.31
...	...	...	...

# Abfragen gegen temporale Daten



„Normale“ SQL-Anfragen beziehen sich immer nur auf die derzeit *gültigen* Tupel der Relation, so dass die Anfrage

```
select Beitrag  
from Studiengebühren  
where Bundesland = 'Bayern'
```

den Wert 0 ergibt. Bayerische Studenten könnten ihre im Sommersemester 2011 entrichteten Beiträge aus dem Ergebnis der folgenden Anfrage ermitteln:

```
select Beitrag  
from Studiengebühren  
where Bundesland = 'Bayern' and system_time as of date('2011.04.01')
```

Es ist auch über die **system\_time between ... and ...**-Klausel möglich, auf das bzw. die während einer bestimmten Zeitperiode gültig gewesene/n Tupel zuzugreifen.

```
create table TutorFürVorlesung  
  ( AssiPersNr integer not null references Assistenten,  
    BetreuteVorlNr integer not null references Vorlesungen,  
    Von date not null,  
    Bis date not null,  
    period for Zeitraum(Von,Bis),  
    primary key(AssiPersNr,Zeitraum without overlaps)  
  );
```

# Explizit kontrollierter Zustand der Datenbasis



TutorFürVorlesung			
AssiPersNr	BetreuteVorlNr	Von	Bis
3002	5049	2012.10.01	2013.04.01
3002	4052	2013.04.01	2013.10.01
3003	5049	2012.04.01	2013.10.01
...	...	...	...



# Automatische Erzeugung der Zeitintervalle



```
update TutorFürVorlesung for portion of Zeitraum  
  from date('2012.10.01') to date('2013.04.01')  
  set VorlNr = 5041  
where AssiPersNr = 3003
```

TutorFürVorlesung			
AssiPersNr	BetreuteVorlNr	Von	Bis
3002	5049	2012.10.01	2013.04.01
3002	4052	2013.04.01	2013.10.01
3003	5049	2012.04.01	2012.10.01
3003	5049	2012.10.01	2013.04.01
3003	5049	2013.04.01	2013.10.01
...	...	...	...

# Erweiterte SQL Syntax für Anfragen gegen Zeitintervalle



- Contains
- Precedes
- Succeeds
- Immediately precedes
- Immediately succeeds
- overlaps