Datenbanken

SQL (Teil 2)

Dr. Özgür Özçep

Universität zu Lübeck

Institut für Informationssysteme

Elementtest

Beispiel für einen Elementtest

```
select Name
from Professoren
where PersNr in (select gelesenVon
                 from Vorlesungen)
```

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

Elementtest mit geschachtelter Anfrage häufig ersetzbar durch nichtgeschachtelte Anfrage mit Join

Quantifizierung (eingeschränkte Form)

Universelle Quantifizierung:

- $\{ x \in R | \forall y \in S : x > y \}$
- Hier: Tabelle aller Projekte *x*, die ein höheres Budget als *alle* externen Projekte *y* haben

```
select *
from Projekte x
where x.Budget > all
        (select y.Budget
         from ExterneProjekte y);
```

Existentielle Quantifizierung:

- $\{ x \in R | \exists y \in S : x > y \}$
- Hier: Tabelle aller Projekte *x*, die ein höheres Budget als *mindestens ein* externes Projekt *y* haben
- Hier: Tabelle aller Projekte *x*, die mindestens an *einer* Projektdurchführung *y* beteiligt sind
- **= any** synonym zu **in**.

```
select *
from Projekte x
where x.Budget > any
        (select y.Budget
         from ExterneProjekte y);
```

```
select *
from Projekte as x
where x.Nr = any
        (select y.Nr
         from Projektdurchfuehrungen y);
```

Quantifizierung mit **exists**

Beispiel: Liefere alle Professoren, die eine Vorlesung anbieten

Negierter Existenzquantor

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

Realisierung als Mengenvergleich

Unkorrelierte Unteranfrage: meist effizienter, wird nur einmal ausgewertet

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

Was ist das Besondere im Unterschied zu vorheriger Anfrage?

Allquantifizierung

SQL-92 hat keinen (generellen) Allquantor

Allquantifizierung muss also durch eine äquivalente Anfrage mit Existenzquantifizierung ausgedrückt werden

Logische Formulierung der Anfrage: Wer hat **alle** vierstündigen Vorlesungen gehört?

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

Elimination von $\forall$ und $\longrightarrow$

Dazu sind folgende Äquivalenzen anzuwenden

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

$$R \longrightarrow T \equiv \neg R \vee T$$

Umformung des Kalkül-Ausdrucks ...

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

Wir erhalten

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

Anwendung der DeMorgan-Regel ergibt schließlich:

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

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 ) );
```

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

Allquantifizierung durch count-Aggregation

Allquantifizierung kann 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);
```

Weiterverwendung von Anfragen

Definition einer Sicht (View) am Beispiel

```
create view SWUnterabteilungen as
select Name, Kurz
from Abteilungen where Oberabt = 'LTSW';
```

- Nicht das Ergebnis, sondern die Anfrage wird benannt.
- Bei jeder Verwendung wird die Basisanfrage über dem aktuellen Datenbestand ausgewertet

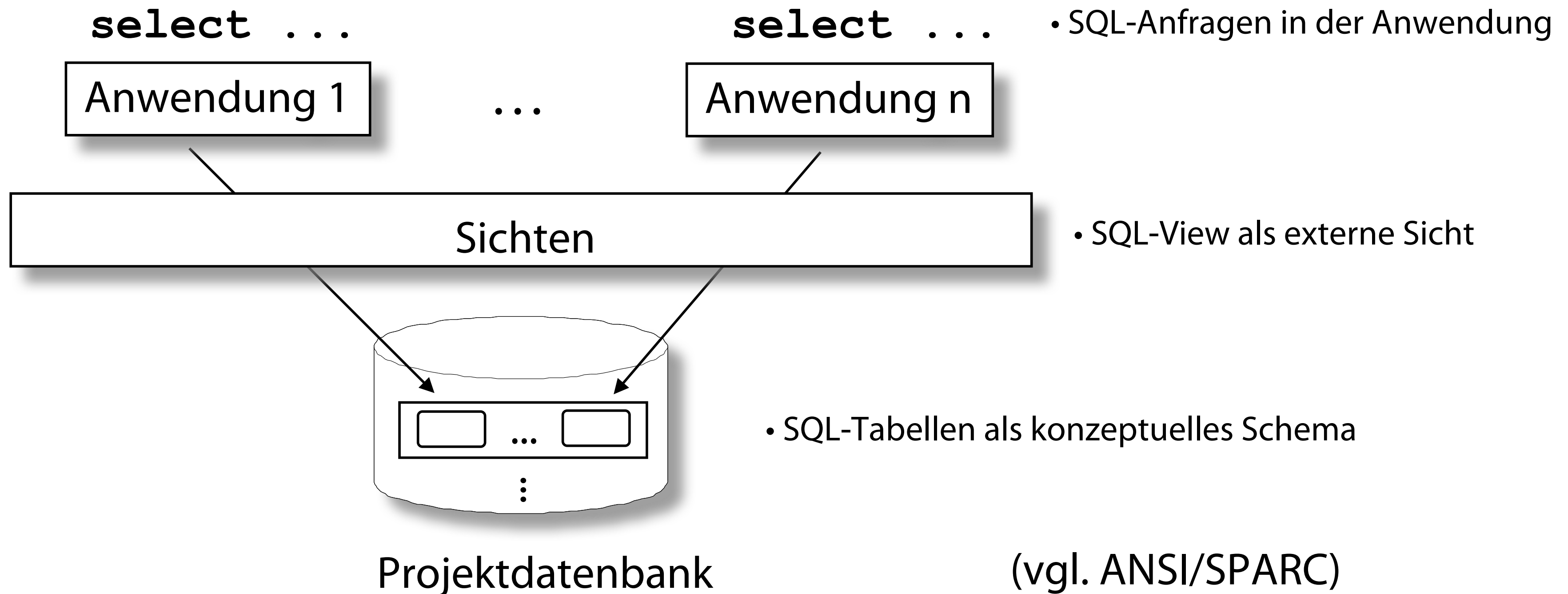
```
select u.name, p.nr
from SWUnterabteilungen u,
     Projektdurchfuehrungen p
where u.kurz = p.kurz;
```

SWUnterabteilungen wird wie eine gewöhnliche Basistabelle verwendet.

- Direkte Verwendung eines Anfrageergebnisses als Bereichsrelation einer komplexen Anfrage

```
select u.Name, p.Nr
from (select Name, Kurz
       from Abteilungen
       where Oberabt = 'LTSW‘) u,
     Projektdurchfuehrungen p
where u.Kurz = p.Kurz;
```

Sichten (1)



Ziel:

- Kapselung der Anwendung
- Entkopplung ... (Schemaevolution)
  - Anwendung: Externe Sicht
  - DB: Konzeptuelle Sicht

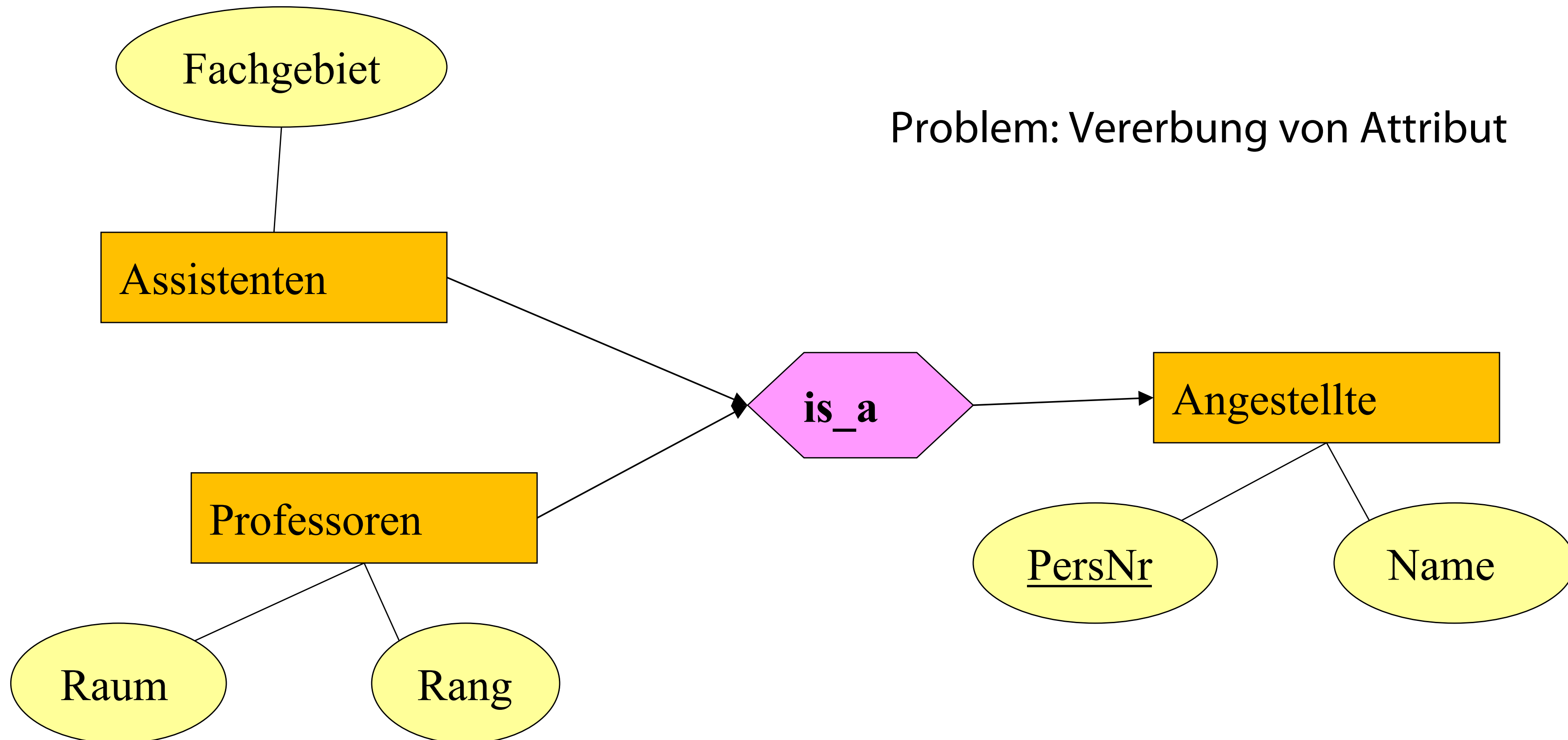
```
create view ReicheProjekte
as select *
from Projekte
where Budget > 200000;
```

Sichten für Vereinfachung

```
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



Problem: Vererbung von Attribut

Angestellte: {[*PersNr*, *Name*]}

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

Assistenten: {[*PersNr*, *Fachgebiet*]}

Untertyp als Sicht

```
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);
```

Untertyp als Sicht (Forts.)

```
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;
```

Obertyp 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);
```

Obertyp als Sicht (Forts.)

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

Frage: Welche Modellierung ist zu präferieren?

Antwort: Hängt von Anwendung ab.

- Erste Modellierung bevorzugt Zugriff auf Obertyp (Angestellte), da für die Untertypen Join nötig.
- 2. Modellierung benötigt zwar keinen Join ("nur" Union), hat aber dafür Redundanz bzgl. der gemeinsamen Attribute.

Sichten für den Datenschutz

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

Note nicht herausgegeben

```
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)
```

k-Anonymität
(k hier 50)

Vorsicht mit Sichten

Doch Vorsicht ist geboten:
Manchmal sind verschiedene Sichten zusammen genommen ausreichend, um eine Anfrage zu modellieren.

Sichtenbasiertes Umschreiben einer Anfrage (View-Based Query Reformulation)

Kann man Anfrage Q umschreiben unter Nutzung von Sichten $V_1, ..., V_n$?!

Sprich: Ist Q äquivalent (relativ zu einer Theorie, die nur aus den Sichtdefinitionen besteht) zu einer Anfrage, die nur die Symbole $V_1, ..., V_n$ enthält?
Beantwortung hängt auch davon ab, wie (in welcher Sprache) man die V_i kombinieren darf.

Vorsicht mit Sichten

- **DB** Straßennetzwerk mit **2-steliger Tabelle** : Straße(x,y)!

- **Sichten**

  $V_2(x,y)$ = "Es gibt einen Pfad der Länge 2 von x nach y" =

  = $\exists$ u Straße(x,u) $\wedge$ Straße(u,y)!

  $V_3(x,y)$ = "Es gibt einen Pfad der Länge 3 von x nach y" =

  = $\exists$ u,v Straße(x,u) $\wedge$ Straße(u,v) $\wedge$ Straße(v,y)!

  ...

- **Beobachtung**: V_4 can durch V_2 ausgedrückt werden. (Wake-Up-Frage)
- **Problem** (Afrati'07): Kann V_5 (in Logik erster Stufe) mittels V_3 ausgedrückt werden?

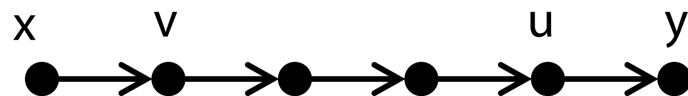
Lösung

- $V_5(x,y) \Leftrightarrow \exists u \,(V_4(x,u) \wedge \forall v \,(V_3(v,u) \rightarrow V_4(v,y)))$

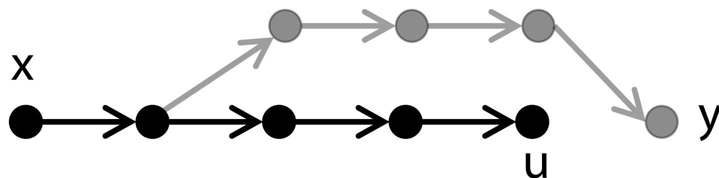
Lösung

- $V_5(x,y) \Leftrightarrow \exists u \, (V_4(x,u) \wedge \forall v \, (V_3(v,u) \rightarrow V_4(v,y)))$

- Beweis
  - Richtung von links nach rechts:



  - Richtung von rechts nach links



- V_3,V_4,V_5 definierbar über CQs (SPJ-Fragment von SQL) über Straßentabelle
- Aber V_5 nicht als CQ von V_4 und V_5 definierbar (Allquantor)

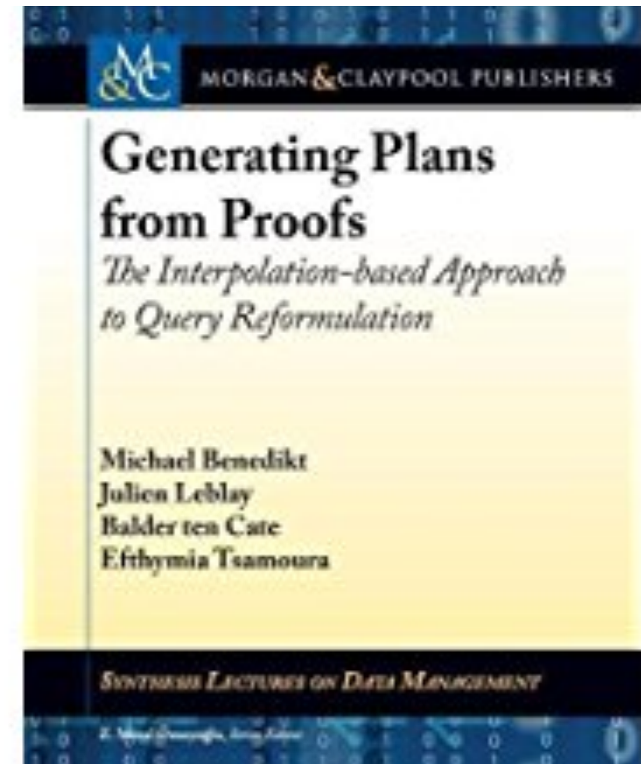
Umschreiben von Sichten

- Stark erforschtes Thema
- **Relevant auch für Datenintegration**

- Jüngst: Erweiterung auf allgemeine Zugriffsmethoden (Relational Access Restrictions)

- Nutzt Craig‘s Interpolationstheorem für Logik erster Stufe:

  Wenn Formel G aus Formel F folgt,
  dann gibt es interpolierende Formel H
  d.h.:
  - H folgt aus F
  - G folgt aus H
  - H enthält nur gemeinsame Symbole von F und G



Änderbarkeit von Sichten

```
create view VorlesungenSicht as
select Titel, SWS, Name
from Vorlesungen, Professoren
where gelesen_von=PersNr;
```

```
insert into VorlesungenSicht
values ('Nihilismus', 2, 'Nobody');
```

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	gelesenVon
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

Änderbarkeit von Sichten

```
create view VorlesungenSicht as
select Titel, SWS, Name
from Vorlesungen, Professoren
where gelesen_von=PersNr;
```

```
insert into VorlesungenSicht
values ('Nihilismus', 2, 'Nobody');
```

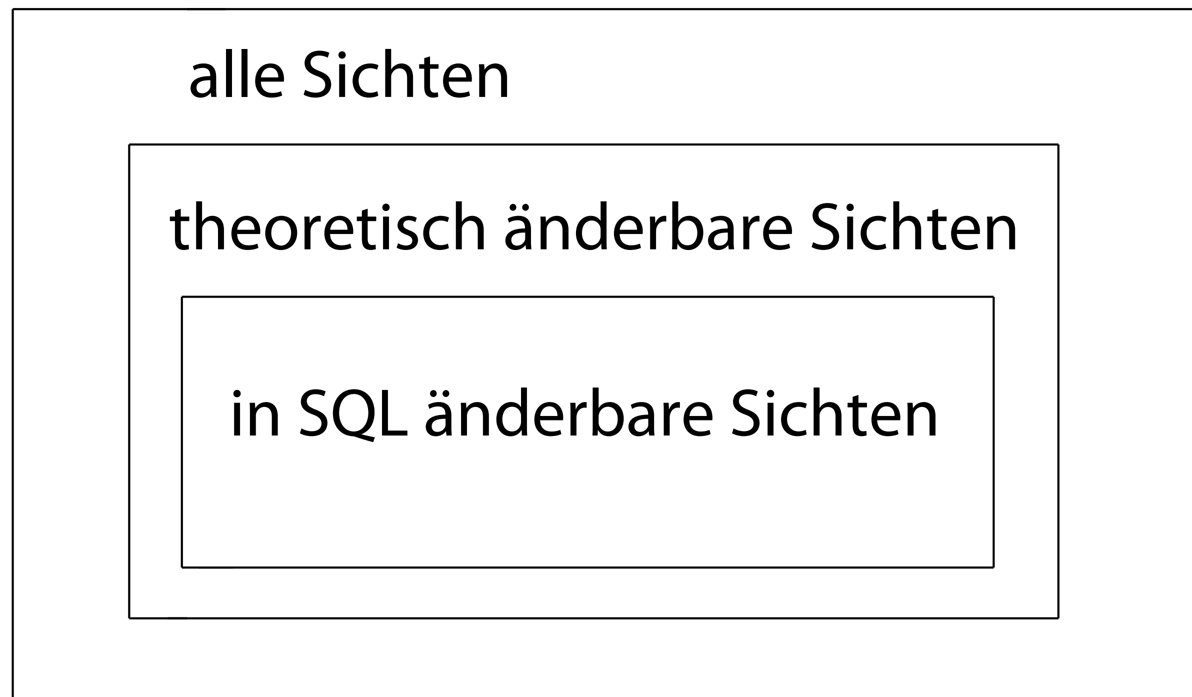
Zentrales Problem beim Datenaustausch und bei der Datenintegration

```
create view WieHartAlsPrüfer (PersNr, Durchschnittsnote) as
select PersNr, avg(Note)
from prüfen
group by PersNr;
```

Änderbarkeit von Sichten

in SQL

- nur eine Basisrelation
- Schlüssel muss vorhanden sein
- keine Aggregatfunktionen, Gruppierung und Duplikateliminierung



Aufwach-Frage: Warum ist die Menge der in SQL änderbaren Sichten
eine (echte) Teilmenge der theoretisch änderbaren Sichten?

Antwort: Es kann Anfragen geben, die das syntaktische Kriterium nicht erfüllen, aber äquivalent sind zu einer Anfrage, die den syntaktichen Kriterien genügt. Leider ist die Äquivalenz für volles SQL nicht entscheidbar.

Integritätssicherung in SQL (1)

SQL-inhärente Integritätsbedingungen (⟶ *statische Typisierung*):

- Typisierung der Spalten: nur typkompatible Werte
- Tupel haben identische Spaltenstruktur

Applikationsspezifische Integritätsbedingungen

- Selbstdefinierte SQL-Domänen im aktuellen Schema

```
create domain Schulnote integer
  constraint NoteDefiniert check(value is not null)
  constraint NoteZwischen1und6 check(value in(1,2,3,4,5,6));
```

- Zusicherungen für Tabellen und Schemata

Integritätssicherung in SQL (2)

– Tabellenzusicherungen (Constraints)

```
create table Tabellenname (...
  constraint Zusicherungsname
    check (Prädikat)) ;

alter table add
 constraint Zusicherungsname
    check (Prädikat);
```

– Schemazusicherungen (Assertions, tabellenübergreifend)

```
create assertion Zusicherungsname
  check(Prädikat);
```

Ein **Datenbankzustand** heißt **konsistent**, wenn alle im Schema deklarierten Zusicherungen erfüllt sind (Tabellen- und Schemazusicherungen konjunktiv verknüpft)

Spaltenwertintegrität

Tabellenzusicherung, bezogen auf Spaltennamen: Spaltenintegrität

In folgenden Modellierungssituationen eingesetzt:

- Vermeidung von Nullwerten
- Definition von Unterbereichstypen
- Definition von Formatinformationen durch Stringvergleiche
- Definition von Aufzählungstypen

```
check(Alter is not null)
```

```
check(Alter >=0 and Alter <=150)
```

```
check(Postleitzahl like 'D-_____')
```

```
check(Note in (1,2,3,4,5,6))
```

Reihenintegrität

Tabellenzusicherung bezogen auf Spaltennamen:
Zeilenenintegritätsbeziehung (von jeder Zeile einer Tabelle zu erfüllen)

```
check(Ausgaben <= Einnahmen)

check((HatVordiplom, HatDiplom) in values(
   ('nein', 'nein')
   ('ja',   'nein')
   ('ja',   'ja')))
```

Tabellenintegrität (1)

Überprüfung durch komplette mengenorientierte Anfrage:

```
check((select sum(Budget) from Projekte) >= 0)

check(exists(select * from Abteilung
             where Oberabt = 'LTSW'))
```

Beschleunigung durch *Indexstrukturen* (z.B. B-Bäume, Hash-Tabelle)
⟶ *Effizienzgewinn* bei Anfragen und Änderungsoperationen

Tabellenintegrität (2)

Spezielle Konstrukte für häufig auftretende Muster von Zusicherungen:

Eindeutigkeit von Spaltenwertkombinationen in einer Tabelle
(⟶ *Schlüsselkandidat*).

```
create table Projekte(...
   unique(Name))
```

```
create table Projekte(...
   check(all x, all y: ...
          (
            (x.Name <> y.Name or x = y)
          ) ) )
```

(x.Name = y.Name) ⟶ (x = y)

Mehrere Schlüsselkandidaten ⟶ separate unique-Klauseln

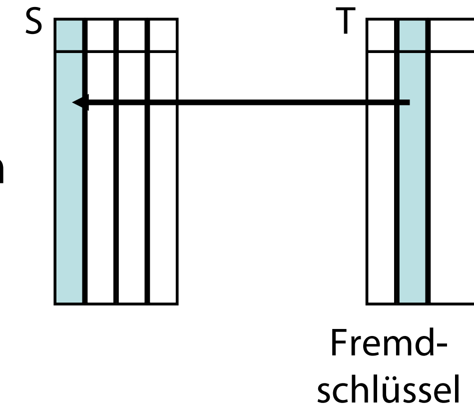
Primärschlüssel: keine Nullwerte

```
create table Projekte (...
   primary key(Nr))
```

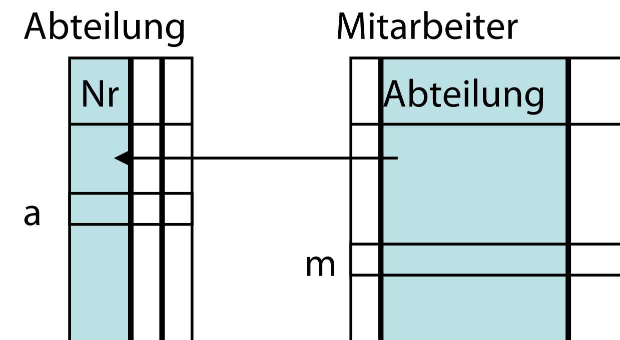
```
create table Projekte(...
   unique Nr
   check(Nr is not null))
```

Referentielle Integrität (1)

Zu jeder Reihe in Tabelle *T* existiert zugehörige Reihe in Tabelle *S*, die Fremdschlüsselwert von *T* als Wert ihres Schlüsselkandidaten besitzt.




```
create table Mitarbeiter (...
 constraint MitarbeiterHatAbteilung
   foreign key(Abteilung)
     references Abteilung(Nr) ...)
```




```
create assertion MitarbeiterHatAbteilung
  check(not exists(select * from Mitarbeiter m where
         not exists(select * from Abteilung a where m.Abteilung = a.Nr)))
```

$\forall$ m $\in$ Mitarbeiter : $\exists$ a $\in$ Abteilung: m.Abteilung = a.Nr

Aufwache-Frage: Ist die zweite Formulierung (assertion ...) äquivalent zu der Fremdschlüssel-Formulierung?

Antwort: Nein, die Bedingung für Fremdschlüssel fordert, dass das referenzierte Attribut ein Schlüssel in der referenzierten Tabelle ist.

Referentielle Integrität (2)

Im allgemeinen besteht Fremdschlüssel einer Tabelle T aus Liste von Spalten, der eine typkompatible Liste von Spalten in S entspricht:

```
create table T
  (...
   constraint Name
     foreign key(A1, A2, ..., An) references (S(B1, B2, ..., Bn))
```

Sind B_1, B_2, ...,B_n die Primärschlüsselspalten von S, kann ihre Angabe entfallen.

Beachte: Rekursive Beziehungen (z.B. Abteilung : Oberabteilung) führen zu reflexiven Fremdschlüsseldeklarationen (S = T).

Behandlung von Integritätsverletzungen (1)

- Annahme: Fremdschlüsselreferenz von T nach S
- Fremdschlüsselintegrität durch vier Operationen verletzbar:
  - **insert into** T
  - **update** T **set** ...
  - **delete from** S
  - **update** S **set** ...

Behandlung von Integritätsverletzungen (2)

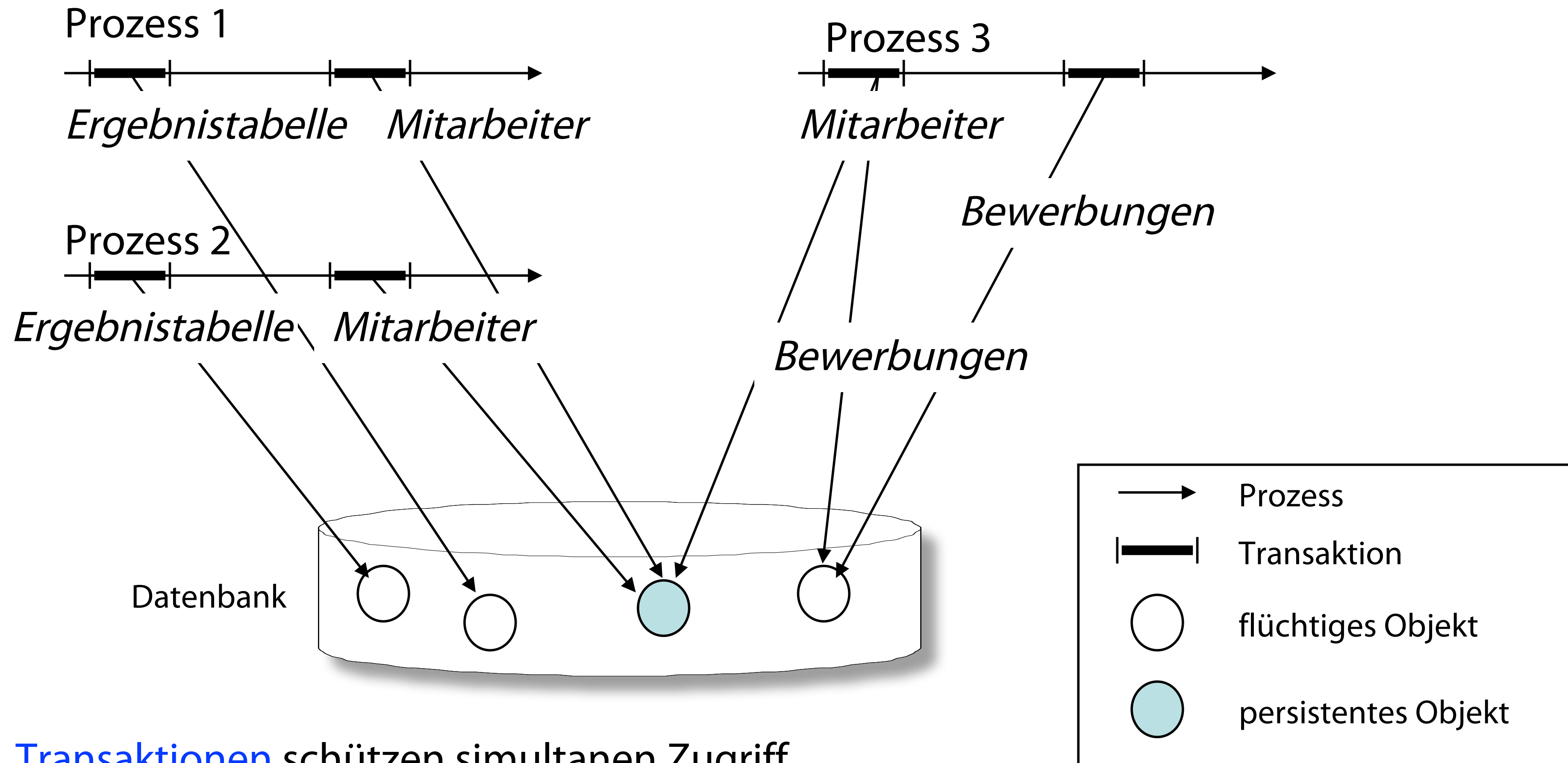
- Fall 1 und 2:
  - Fremdschlüsselreferenz in S evtl. nicht definiert (→ Fehler)
- Fall 3 oder 4:
  - Tupel in S gelöscht, auf das Fremdschlüsselreferenz zeigt (→ Fehler)
  - Fehlerbehandlung kann angegeben werden
    - **set null**: Der Fremdschlüsselwert aller betroffener Reihen in T durch **null** ersetzt
    - **set default**: Der Fremdschlüsselwert aller betroffener Reihen in T durch Standardwert der Fremdschlüsselspalte ersetzt
    - **cascade**:
      - Im Fall 3 (**delete**) betroffene Reihen in T gelöscht
      - Im Falle 4 (**update**) Fremdschlüsselwerte aller betroffenen Reihen in T durch die **neuen** Schlüsselwerte der korrespondierenden Reihen ersetzt
    - **no action**: Anweisung zur Änderung von S wird ignoriert

Zeitpunkt der Integritätsprüfung

- Transaktionsende (deferrable)
- Nach jeder SQL-Anweisung (not deferrable)

Lebensdauer, Sichtbarkeit, gemeinsame Nutzung (1)

Die gleiche Datenbank kann von verschiedenen informationsverarbeitenden Prozessen simultan oder sequentiell nacheinander benutzt werden.



Transaktionen schützen simultanen Zugriff

SQL-Standard – Kurzer Historischer Überblick

- SO/IEC 9075 Datenbanksprache SQL
  - SQL-86 – Transaktionen, Create, Read, Update, Delete
  - SQL-89 – Referentielle Integrität
  - SQL-92 – ISO-Standardisierung
  - SQL:1999 – Benutzerdefinierte Typen, Trigger, Rekursion
  - SQL:2003 – XML, Fensteroperator, Sequenzen
  - SQL:2008 – Erweiterungen und Korrekturen
  - SQL:2011 – Temporale Konstrukte
  - SQL:2016 – JSON, PTFs, RPR
  - SQL:2019 – mehrdimensionale Felder (MDA)
- 30+ Jahre an Support und Entwicklung des Standards

ISO-Dokumente zum SQL-2016

Der Standard besteht insgesamt aus 9 einzelnen Publikationen:[3]

- ISO/IEC 9075-1:2016 Part 1: Framework (SQL/Framework)
- ISO/IEC 9075-2:2016 Part 2: Foundation (SQL/Foundation)
- ISO/IEC 9075-3:2016 Part 3: Call-Level Interface (SQL/CLI)
- ISO/IEC 9075-4:2016 Part 4: Persistent stored modules (SQL/PSM)
- ISO/IEC 9075-9:2016 Part 9: Management of External Data (SQL/MED)
- ISO/IEC 9075-10:2016 Part 10: Object language bindings (SQL/OLB)
- ISO/IEC 9075-11:2016 Part 11: Information and definition schemas (SQL/Schemata)
- ISO/IEC 9075-13:2016 Part 13: SQL Routines and types using the Java TM programming language (SQL/JRT)
- ISO/IEC 9075-14:2016 Part 14: XML-Related Specifications (SQL/XML)

und wird durch 6 bzw. 7 ebenfalls standardisierte *SQL multimedia and application packages* ergänzt:

- ISO/IEC 13249-1:2016 Part 1: Framework
- ISO/IEC 13249-2:2003 Part 2: Full-Text
- ISO/IEC 13249-3:2016 Part 3: Spatial
- ISO/IEC 13249-5:2003 Part 5: Still image
- ISO/IEC 13249-6:2006 Part 6: Data mining
- ISO/IEC 13249-7:2013 Part 7: History
- ISO/IEC 13249-8:xxxx Part 8: Metadata registries (MDR) (noch nicht verabschiedet)

Der offizielle Standard ist nicht frei verfügbar, jedoch existiert ein Zip-Archiv mit einer Arbeitsversion von 2008.[4]

[Wikipedia]

SQL:2016 Erneuerungen

- Support für Java Script Object Notation (JSON)
  - Speichern, anfragen und abrufen
- Polymorphic table functions (PTF)
  - Parameter and Funktionsausgaben können Tabellen sein, deren Schema zur Anfragezeit nicht bekannt ist.
- Row pattern recognition (RPR)
  - Reguläre Ausdrücke über Folgen von Tabellenzeilen
- Zusätzliche Funktionen (z.B., für statistische Analysen)
  - Trigonometrische und logarithmische Funktionen
  - Konkatenation über Gruppen von Zeilen
- Default-Werte und –Namen für Argumente von SQL-Funktionen

Was noch kommt

- SQL:2019: Mehrdimensionale Felder/Multi Dimensional Arrays – SQL/MDA
  - Anwendungen in Naturwissenschaften (heat maps), Geoanwendungen (z.B. Bildverarbeitung)
  - 1D-Felder schon in SQL:1999 (allerdings sehr eingeschränkte Operatoren zum Anfragen und Ändern der Felder)
  - Mehrdimensionale Felder schon seit > 20Jahren in der Diskussion
  - Prominentes Beispiel sind die Feld-Datenbanken:
    - Rasdaman (Peter Baumann)
    - SciDB (Michael Stonebraker)
- SQL:2020+: Eigenschaftsgraphen
  - Auch seit > 20J ahren diskutiert
    - Neo4j graph database (Cypher)
    - RDF (SPARQL)
- In der Mache: Stromverarbeitung mit SQL
- Probabilistische Modelle in SQL:2030?