Informationssysteme

Sommersemester 2016

Prof. Dr.-Ing. Sebastian Michel
TU Kaiserslautern

smichel@cs.uni-kl.de

Unteranfragen in where-Klausel

```
select *
from pruefen
where Note < (select avg(Note)
              from pruefen);
```

Vergleich von Wert mit Wert

Unteranfragen in select-Klausel

- **Für jedes Ergebnistupel wird die Unteranfrage ausgeführt**
- Unteranfrage ist korreliert (greift auf Attribute der umschließenden Anfrage zu)

```
select PersNr, Name, (select sum(SWS)
                      from Vorlesungen
                      where gelesenVon = PersNr) as Lehrbelastung
from Professoren;
```

Pro äußerer Zeile wird ein Wert berechnet

Unkorrelierte vs. korrelierte Unteranfragen

Korrelierte Formulierung

```
select s.*
from Studenten s
where exists
        (select p.*
        from Professoren p
        where p.GebDatum > s.GebDatum);
```

Unkorrelierte vs. korrelierte Unteranfragen

Äquivalente unkorrelierte Formulierung

```
select s.*
from Studenten s
where s.GebDatum <
        (select max(p.GebDatum)
        from Professoren p);
```

- **Vorteil: Ergebnis der Unteranfrage kann materialisiert werden**
- **Unteranfrage braucht nur einmal ausgewertet zu werden**

Entschachtelung korrelierter Unteranfragen

```
select a.*
from Assistenten a
where exists
            (select p.*
            from Professoren p
            where a.Boss = p.PersNr and
                  p.GebDatum > a.GebDatum);
```

Entschachtelung durch Join

```
select a.*
from Assistenten a, Professoren p
where a.Boss=p.PersNr and p.GebDatum > a.GebDatum);
```

WITH statements

```
with AnzahlStudentenJeVL as (
        select VorlNr, count(*) as AnzProVorl
        from hoeren
        group by VorlNr
),
GesamtanzahlStudenten as (
        select count (*) as GesamtAnz
        from Studenten
)
select h.VorlNr, h.AnzProVorl, g.GesamtAnz,
h.AnzProVorl/cast(g.GesamtAnz as float) as Marktanteil
from AnzahlStudentenJeVL h,
GesamtanzahlStudenten g;
```

Erzeugt temporäre Tabelle innerhalb der Anfrage

Nullwerte

- **null entspricht "unbekannter Wert", "nicht definiert"**
- **Nullwerte können auch im Zuge der Anfrageauswertung entstehen (z.B. äußere Joins)**
- Manchmal sehr überraschende Anfrageergebnisse, wenn Nullwerte vorkommen. **Beispiel:**

  select count (*)
  from Studenten **where** Semester $<$ 13 **or** Semester $>=$ 13;
- Wenn es Studenten gibt, deren Semester-Attribut den Wert null hat, werden diese nicht mitgezählt.
- Der Grund liegt in folgenden Regeln für den Umgang mit Nullwerten begründet.

COUNT und Nullwerte

Bei count(spaltenname) werden Nullwerte nicht mitgezählt, ebenso wie bei count(distinct spaltenname).

Bei count(*) werden hingegen auch Tupel, die nur aus Nullwerten bestehen mitgezählt.

“select count(*) from (select null as x) as y” liefert also 1 und
“select count(x) from (select null as x) as y” liefert 0

Auswertung bei Nullwerten

- In arithmetischen Ausdrücken werden Nullwerte propagiert, d.h. sobald ein Operand null ist, wird auch das Ergebnis null. Dementsprechend wird z.B. null + 1 zu null ausgewertet-aber auch null * 0 wird zu null ausgewertet.
- **SQL hat eine dreiwertige Logik**, die nicht nur **true** und **false** kennt, sondern auch einen dritten Wert **unknown**. Diesen Wert liefern Vergleichsoperationen zurück, wenn mindestens eines ihrer Argumente null ist. Beispielsweise wertet SQL das Prädikat (PersNr=...) immer zu unknown aus, wenn die PersNr des betreffenden Tupels den Wert null hat.
- Logische Ausdrücke werden nach den folgenden Tabellen berechnet:

Auswertung bei Nullwerten

not	
true	false
unknown	unknown
false	true

and	true	unknown	false
true	true	unknown	false
unknown	unknown	unknown	false
false	false	false	false

or	true	unknown	false
true	true	true	true
unknown	true	unknown	unknown
false	true	unknown	false

Das case-Konstrukt

```
select MatrNr, ( case when Note < 1.5 then 'sehr gut'
                      when Note < 2.5 then 'gut'
                      when Note < 3.5 then 'befriedigend'
                      when Note <= 4.0 then 'ausreichend'
                      else 'nicht bestanden' end)
from pruefen;
```

Die erste qualifizierende when-Klausel wird ausgeführt.

Joins

- **cross join**: Kreuzprodukt
- **natural join:** natürlicher Join
- **join oder inner join:** Theta-Join
- **left, right** oder **full outer join**: äußerer Join

select *
from R_1, R_2
where $R_1.A = R_2.B$;

kann wie folgt geschrieben werden:

select *
from R_1 **join** R_2 **on** $R_1.A = R_2.B$;

Right Outer Join

select *
from pruefen f **right outer join** Studenten s **on**
f.MatrNr=s.MatrNr;

	matrnr integer	vorlnr integer	persnr integer	note numeric(2,1)	matrnr integer	name character varying(30)	semester integer
1					24002	Xenokrates	18
2	25403	5041	2125	2.0	25403	Jonas	12
3					26120	Fichte	10
4					26830	Aristoxenos	8
5	27550	4630	2137	2.0	27550	Schopenhauer	6
6	28106	5001	2126	1.0	28106	Carnap	3
7					29120	Theophrastos	2
8					29555	Feuerbach	2
9					42	mueller	

Dreiwege Right Outer Join

```
select p.PersNr, p.Name, f.PersNr, f.Note, f.MatrNr,
s.MatrNr, s.Name
from Professoren p right outer join
                    (pruefen f right outer join Studenten s on
                    f.MatrNr=s.MatrNr)
                    on p.PersNr=f.PersNr;
```

	persnr integer	name character varying(30)	persnr integer	note numeric(2,1)	matrnr integer	matrnr integer	name character varying(30)
1						24002	Xenokrates
2	2125	Sokrates	2125	2.0	25403	25403	Jonas
3						26120	Fichte
4						26830	Aristoxenos
5	2137	Kant	2137	2.0	27550	27550	Schopenhauer
6	2126	Russel	2126	1.0	28106	28106	Carnap
7						29120	Theophrastos
8						29555	Feuerbach
9						42	mueller

Dreiwege Left Outer Join

```
select p.PersNr, p.Name, f.PersNr, f.Note, f.MatrNr,
s.MatrNr, s.Name
from Professoren p left outer join
     (pruefen f left outer join Studenten s on f.MatrNr=
     s.MatrNr)
     on p.PersNr=f.PersNr;
```

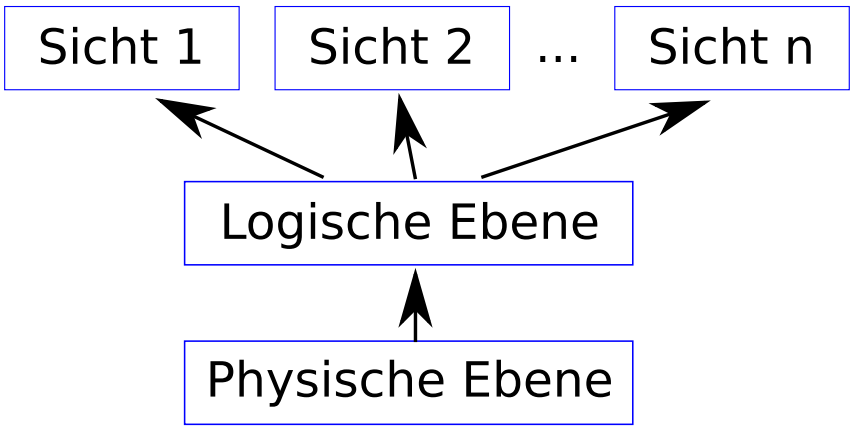
	persnr integer	name character varying(30)	persnr integer	note numeric(2,1)	matrnr integer	matrnr integer	name character varying(30)
1	2125	Sokrates	2125	2.0	25403	25403	Jonas
2	2126	Russel	2126	1.0	28106	28106	Carnap
3	2127	Kopernikus					
4	2133	Popper					
5	2134	Augustinus					
6	2136	Curie					
7	2137	Kant	2137	2.0	27550	27550	Schopenhauer

Full Outer Join

select p.PersNr, p.Name, f.PersNr, f.Note, f.MatrNr, s.MatrNr, s.Name
from Professoren p **full outer join**
(pruefen f **full outer join** Studenten s **on**
f.MatrNr= s.MatrNr)
on p.PersNr=f.PersNr;

	persnr integer	name character varying(30)	persnr integer	note numeric(2,1)	matrnr integer	matrnr integer	name character varying(30)
1	2125	Sokrates	2125	2.0	25403	25403	Jonas
2	2126	Russel	2126	1.0	28106	28106	Carnap
3	2127	Kopernikus					
4	2133	Popper					
5	2134	Augustinus					
6	2136	Curie					
7	2137	Kant	2137	2.0	27550	27550	Schopenhauer
8						26830	Aristoxenos
9						29120	Theophrastos
10						42	mueller
11						29555	Feuerbach
12						24002	Xenokrates
13						26120	Fichte

Abstraktionsebenen eines Datenbanksystems



Teilmenge des Datenbankschemas, z.B., verschiedene Rollen

Datenbankschema, z.B. Menge von Tabellen

Geräte, Dateien, Dateiformate, z.B., Dateien, Festplatten

Datenunabhängigkeit:

- physische Unabhängigkeit
- logische Datenunabhängigkeit $\rightarrow$ **Sichten (Views)**

Beispiele für Sichten

```
create view ProfsUndIhreVorlesungen as
    select v.titel, p.name
    from Professoren p, Vorlesungen v
    where p.persnr=v.gelesenvon;

select * from  ProfsUndIhreVorlesungen;

create view AnzahlSWSProProf as
    select p.name, sum(v.sws)
    from professoren p, vorlesungen v
    where p.persnr=v.gelesenvon
    group by p.name;

select sum(sum) from AnzahlSWSProProf;
```

Beispiele für Sichten

```
create or replace view
        ProfsUndIhreVorlesungen as
    select v.titel, p.name
    from Professoren p, Vorlesungen v
    where p.persnr=v.gelesenvon;
```

Achtung:

replace view erwartet dieselben Spalten in derselben Reihenfolge mit denselben Typen.

```
drop view AnzahlSWSProProf;
create view AnzahlSWSProProf as
    select p.name, p.persnr, sum(v.sws)
    from professoren p, vorlesungen v
    where p.persnr=v.gelesenvon
    group by p.name, p.persnr;
```

Alternative:

erst löschen, dann neu erzeugen. oder: Postgres SQL-Erweiterung **alter view**

Beispiele für Sichten

Relation **pruefen**: ([MatrNr, VorlNr, PersNr, Note])

```
create view pruefenSicht as
    select MatrNr, VorlNr, PersNr
    from pruefen;
```

Was ist der Unterschied zu **pruefen**?

```
create view pruefenSicht2 (M,V,P) as
    select MatrNr, VorlNr, PersNr
    from pruefen;
```

Was ist der Unterschied zu **pruefenSicht**?

```
create view pruefenSicht3 (M,V) as
    select MatrNr, VorlNr, PersNr
    from pruefen;
```

Was ist der Unterschied zu **pruefenSicht2**?

Sichten vs. Materialisierte Sichten

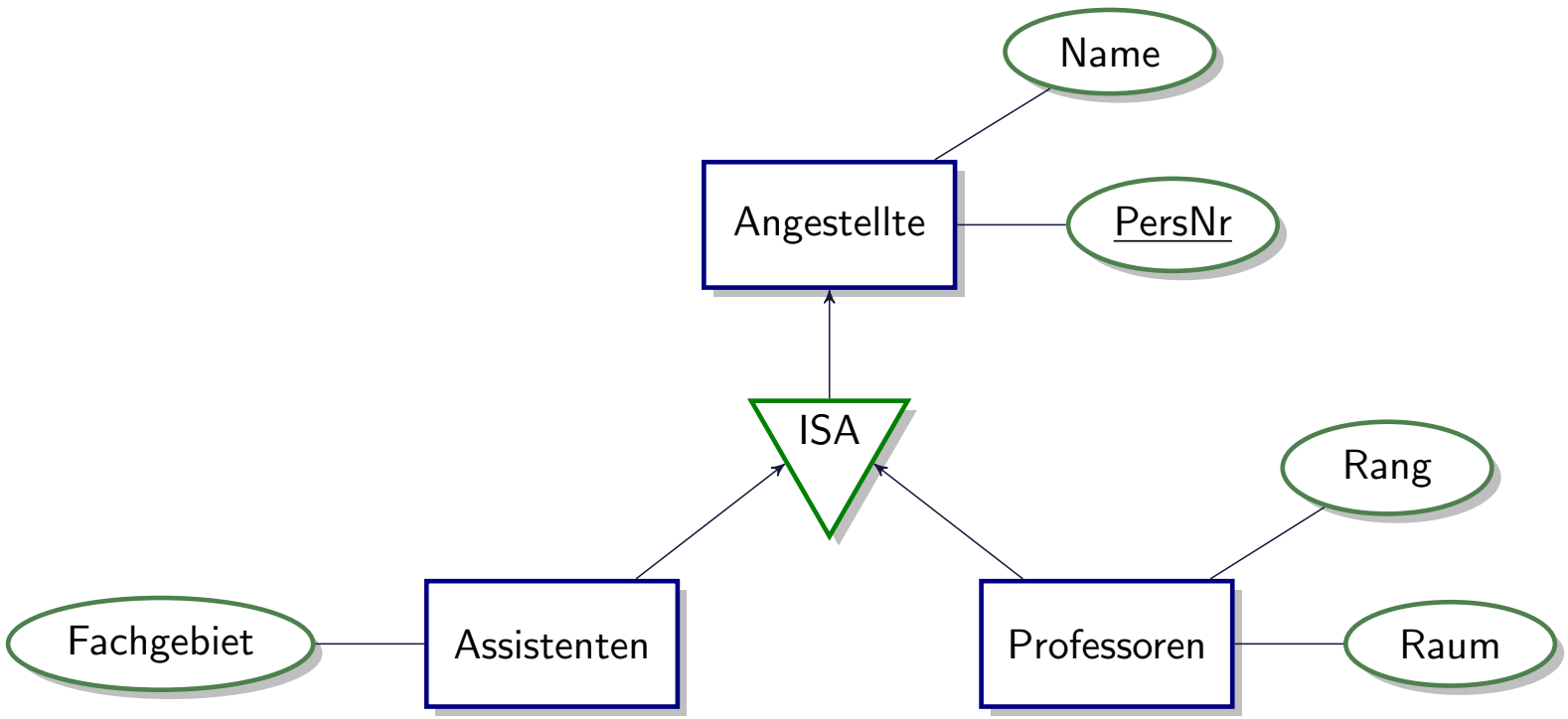
(Dynamische) Sicht

- =Makro für Query
- Ergebnis der Anfrage wird nicht vorberechnet
- Rechenaufwand zum Zeitpunkt der Anfrage (=query time)
- erst wenn die Sicht benutzt wird, wird auch das Ergebnis der Sicht berechnet

Materialisierte Sicht

- Ergebnis der Sicht wird vorberechnet
- wenn eine Anfrage die Sicht benutzt, ist das Ergebnis der Sicht bereits vollständig berechnet
- Rechenaufwand vorher (=index time)
- Problem bei Updates:
  Tabellen, die zur Vorberechnung der Sicht benutzt werden, ändern sich $\Rightarrow$ Materialisierte Sicht muss (oft) angepaßt werden

Wiederholung: Generalisierung



Sichten zur Modellierung von Generalisierung

```
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 AssistentenDaten
        (PersNr      integer not null,
        Fachgebiet   varchar (30),
        Boss         integer);
```

```
create view Professoren as
        select *
        from Angestellte A join ProfDaten D
        on A.PersNr=D.PersNr;

create view Assistenten as
        select *
        from Angestellte A join AssistentenDaten D
        on A.PersNr=D.PersNr;
```

Untertypen als Sicht

- Also Join von Untertyp und Obertyp
- nur bei Zugriff auf Professoren oder Assistenten muss View ausgeführt werden
- D.h. Zugriff auf Obertyp ist bevorzugt (günstig).

Alternative:

```
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 Andere Angestellte
        (PersNr     integer not null,
        Name        varchar (30) not null);
```

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

Obertypen als Sicht

- Union von Ober- und Untertypen
- Also: bevorzugt Zugriff auf Untertypen
- Nur bei Zugriff auf Obertyp muss die View ausgeführt werden

Updates in SQL

```
update studenten
    set semester=semester+1
```

kombiniert mit where:

```
update professoren
    set rang='C4'
where name='Kopernikus'
```

Änderbarkeit von Sichten

Beispiel für nicht veränderbare Sichten

```
create view WieHartAlsPruefer as
    select PersNr, avg(Note)
    from pruefen
    group by PersNr;

update WieHartAlsPruefer
    set Durchschnittsnote=1.0
    where PersNr=(select PersNr
                  from Professoren
                  where Name='Sokrates');
```

Änderbarkeit von Sichten

Beispiel für nicht veränderbare Sichten

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

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

Wie soll das DBMS dieses Tupel den Tabellen Professoren und Vorlesungen zuordnen?

Änderbarkeit von Sichten in SQL

Daten in einer View sind veränderbar (update/insert/delete), wenn ...

in SQL-92

- nur eine Tabelle
- nur Selektion und Projektion
- keine Aggregatfunktionen, Gruppierung und Duplikateliminierung

in SQL-99

- wie SQL-92
- oder: mehrere Tabellen UND Felder, auf die sich das Update bezieht, können mit Hilfe des Primärschlüssels eindeutig zugeordnet werden.
- D.h. auch bei einem Join kann man manchmal ändern.

Änderbarkeit von Sichten in Postgres

Bis zur Version 9.2 waren View in Postgresql generell nicht aktualisierbar!
Ab Version 9.3 im Sinne von SQL-92.
Also in Version ≤ 9.2 hätten auch z.B. inserts auf diese triviale View nicht funktioniert.

```
create view Studis as
    select *
    from studenten;

insert into studis values (42, 'mueller', 11);
```

ERROR: cannot insert into a view.
HINT: You need an unconditional ON INSERT DO INSTEAD rule.

Postgresqls Regel (Rule) System

- Erlaubt Handhabung von INSERT/UPDATE/DELETE auf Views durch spezielle Regeln

Rules (Nicht relevant für Klausur)

create [**or replace**] **rule** name
as on event **to** table [**where** condition]
do [also | instead] action

- event: update/insert/delete
- action:
  - nothing: mache nichts
  - command: eine Aktion
  - command1; command2; ... : mache mehrere Aktionen
- also
- instead

`http://www.postgresql.org/docs/9.3/static/rules-update.html`

View update mit rule **instead** (Nicht relevant für Klausur)

```
create rule StudiInsert
as on insert to studis
do instead
    insert into studenten
        values (NEW.matrnr, NEW.name, NEW.semester);
```

Jetzt OK:

```
insert into studis values (42,'mueller',11);
```

View update mit rule **also** (Nicht relevant für Klausur)

```
create or replace rule StudiInsert
as on insert to studis
do also
    insert into studenten
        values (NEW.matrnr, NEW.name, NEW.semester);
```

```
insert into studis values (42,'mueller',11);
```

ERROR: cannot insert into a view.
HINT: You need an unconditional ON INSERT DO INSTEAD rule.

Endlosrekursion (Nicht relevant für Klausur)

```
create or replace rule StudentenInsert
as on insert to studenten
do  also
    insert into studenten
        values (NEW.matrnr, NEW.name, NEW.semester);
```

```
insert into studenten values (777,'mueller',11);
```

ERROR: infinite recursion detected in rules for relation "studenten"

Ohne Rekursion (Nicht relevant für Klausur)

```
create or replace rule StudentenInsert
as on insert to studenten
do also
    insert into hoeren
    values (New.matrnr, 5001);

select count(*) from studenten;

insert into studenten values (777, 'mueller', 11);

select count(*) from studenten;

select * from hoeren where matrnr=777;
```