

Diskussion: Personal (1)

ER-Diagramm:



Umsetzung ins Relationenmodell?

Diskussion: Personal (2)

Zusätzliche Regel:

In jeder Abteilung (Person) muss mindestens eine beschäftigt sein ([1, N])

Umsetzung ins Relationenmodell?

Zusätzliche Regel:

Jeder Angestellte (Person) muss in einer Abteilung beschäftigt sein ([1, 1])

Umsetzung ins Relationenmodell?

Generated Values

Artificial values, surrogates, no semantic,
mostly as keys:

- Directly in the table definition:

```
create table dept (  
    deptno    serial primary key,  
    deptname  varchar(50) not null);
```

insert with:

```
insert into dept values(default, 'I3') or  
insert into dept values('I3');
```

Sequences to share

```
CREATE [ TEMPORARY | TEMP ] SEQUENCE name
      [ INCREMENT [ BY ] increment ]
      [ MINVALUE minvalue | NO MINVALUE ] [ MAXVALUE
        maxvalue | NO MAXVALUE ]
      [ START [ WITH ] start ] [ CACHE cache ]
      [ [ NO ] CYCLE ]
```

```
CREATE SEQUENCE artificial_key START 101;
```

```
CREATE TABLE Dept (deptno INT DEFAULT
nextval('artificial_key') NOT NULL,...)
```

```
INSERT INTO dept VALUES
(nextval('artificial_key'), 'I3');
```

Die relationale Algebra

σ Selektion

π Projektion

\times Kreuzprodukt

\bowtie Join (Verbund)

ρ Umbenennung

\ltimes Semi-Join (linker)

\rtimes Semi-Join (rechter)

$\ltimes\!\!\!\! \times$ linker äußerer Join

$\rtimes\!\!\!\! \times$ rechter äußerer Join

Allg. Mengenoperationen:

– Differenz

\div Division

\cup Vereinigung

\cap Durchschnitt

Beispiel Mengendurchschnitt

Finde die *PersNr* aller C4-Professoren, die mindestens eine Vorlesung halten.

$$\Pi_{\text{PersNr}}(\rho_{\text{PersNr} \leftarrow \text{gelesenVon}}(\text{Vorlesungen})) \cap \Pi_{\text{PersNr}}(\sigma_{\text{Rang}=\text{C4}}(\text{Professoren}))$$

→ prozedural !

Relationaler Tupelkalkül

Eine Anfrage im Relationenkalkül hat die Form

$$\{t \mid P(t)\}$$

mit t Tupelvariable und P Prädikat

einfaches **Beispiel:**

C4-Professoren

$$\{p \mid p \in \text{Professoren} \wedge p.\text{Rang} = \text{'C4'}\}$$

Relationaler Tupelkalkül: weiteres Beispiel

Studenten mit mindestens einer Vorlesung von Curie

$$\{s \mid s \in \text{Studenten} \\ \wedge \exists h \in \text{hören}(s.\text{MatrNr}=h.\text{MatrNr} \\ \wedge \exists v \in \text{Vorlesungen}(h.\text{VorlNr}=v.\text{VorlNr} \\ \wedge \exists p \in \text{Professoren}(p.\text{PersNr}=v.\text{gelesenVon} \\ \wedge p.\text{Name} = \text{'Curie'})))))\}$$

Dieselbe Anfrage in SQL belegt die Verwandtschaft

```
select s.*
from Studenten s
where exists (
  select h.*
  from hören h
  where h.MatrNr = s.MatrNr and exists (
    select *
    from Vorlesungen v
    where v.VorlNr = h.VorlNr and exists (
      select *
      from Professoren p
      where p.Name = 'Curie' and
            p.PersNr = v.gelesenVon )))
```

Relationaler Domänenkalkül

Anfrage im Domänenkalkül hat die Form:

$$\{[v1, v2, \dots, vn] \mid P(v1, \dots, vn)\}$$

mit $v1, \dots, v2$ Domänenvariablen und P Prädikat

Beispiel:

MatrNr und Namen der Prüflinge von Sokrates

$$\{[m, n] \mid \exists ([m, n, s] \in \text{Studenten} \\ \wedge \exists p, v, g ([m, p, v, g] \in \text{prüfen} \\ \wedge \exists a, r, b ([p, a, r, b] \in \text{Professoren} \\ \wedge a = \text{'Sokrates'})))]\}$$

Ausdruckskraft

Die drei Sprachen

- relationale Algebra
 - relationaler Tupelkalkül, eingeschränkt auf sichere Ausdrücke
 - relationaler Domänenkalkül, eingeschränkt auf sichere Ausdrücke
- sind gleich mächtig

$\{n \mid \neg(n \in \text{Professoren})\}$ z.B. ist nicht sicher, da das Ergebnis unendlich ist

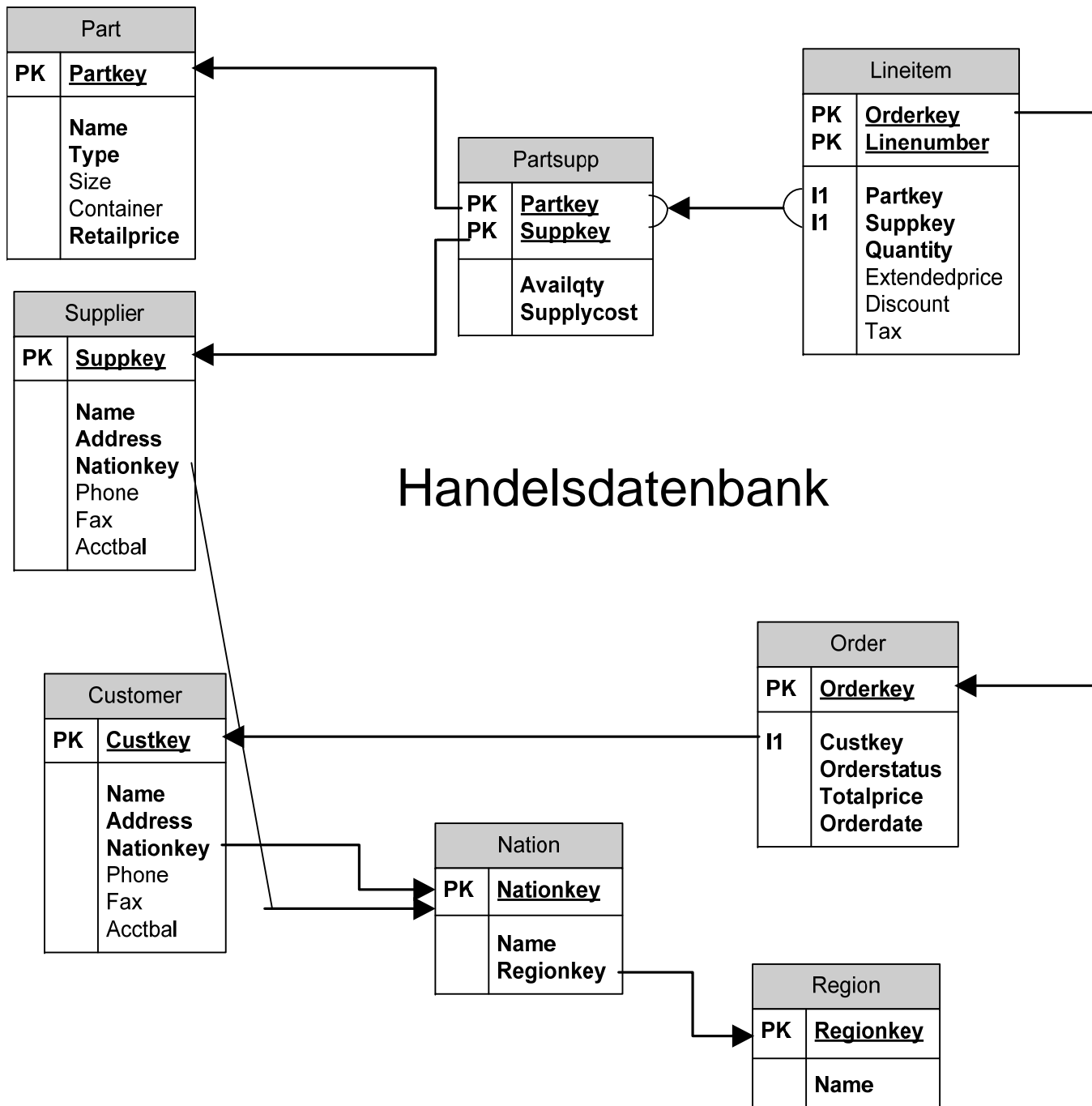
SQL - DRL

Tutorials für erste Einblicke in SQL:

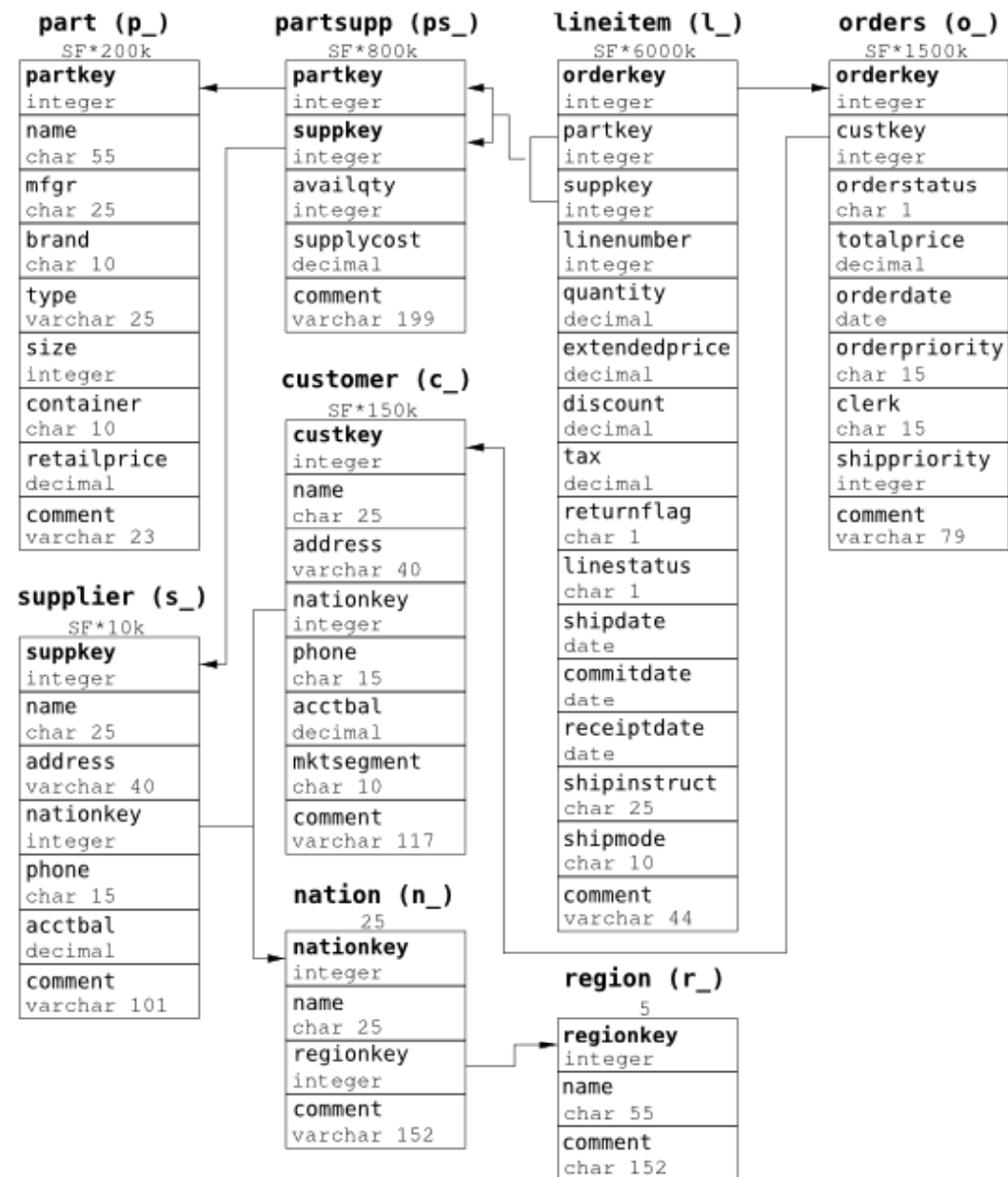
- sql.lernenhoch2.de/lernen/
- www.w3schools.com/sql

Webschnittstellen für SQL:

- sqlfiddle.com (MySQL, Oracle, PostgreSQL, SQLite, MS SQL):
auch Tabellen anlegen möglich
- hyper-db.com/interface.html (HyPer):
Universitätsdatenbank, TPC-H Schema
Query-Ausführungspläne



TPC-H Schema



Gerüst SQL-Anfrage

select	<Attributliste>	5
from	<Relationenliste>	1
[where	<Prädikatsliste>	2
group by	<Attributliste>	3
having	<Prädikatsliste>	4
order by	<Attributliste>	6
fetch first	<Anzahl Ergebnistupel>]	7

Einfaches Beispiel

Anfrage:

"Gib mir die gesamte Information über alle Professoren,,

Professoren

```
select *  
from Professoren
```

PersNr	Name	Rang
2136	Curie	C4
2137	Kant	C4
2126	Russel	C4
2125	Sokrates	C4
2134	Augustinus	C3
2127	Kopernikus	C3
2133	Popper	C3

Ergebnis

PersNr	Name	Rang
2136	Curie	C4
2137	Kant	C4
2126	Russel	C4
2125	Sokrates	C4
2134	Augustinus	C3
2127	Kopernikus	C3
2133	Popper	C3

Attribute selektieren

Anfrage:

"Gib mir die PersNr und den Namen aller Professoren,,

```
select PersNr, Name  
from Professoren
```

Professoren

PersNr	Name	Rang
2136	Curie	C4
2137	Kant	C4
2126	Russel	C4
2125	Sokrates	C4
2134	Augustinus	C3
2127	Kopernikus	C3
2133	Popper	C3

Ergebnis

PersNr	Name
2136	Curie
2137	Kant
2126	Russel
2125	Sokrates
2134	Augustinus
2127	Kopernikus
2133	Popper

Duplikateliminierung

- Im Gegensatz zur relationalen Algebra (Mengen!) eliminiert SQL keine Duplikate
- Falls Duplikateliminierung erwünscht, muss das Schlüsselwort **distinct** benutzt werden

- Beispiel:

Anfrage: „Welche Ränge haben Professoren?“

```
select distinct Rang
```

```
from Professoren
```

Ergebnis:

Rang
C3
C4

Where Klausel: Tupel selektieren

Anfrage:

"Gib mir die PersNr und den Namen aller Professoren, die den Rang C4 haben,,

```
select  PersNr, Name
from    Professoren
where   Rang= 'C4';
```

Ergebnis:

PersNr	Name
2125	Sokrates
2126	Russel
2136	Curie
2137	Kant

Where Klausel: Prädikate

- Prädikate in der where-Klausel können logisch kombiniert werden mit:

AND, OR, NOT

- Als Vergleichsoperatoren können verwendet werden:

=, <, <=, >, >=, between, like

Beispiel für between

Anfrage:

"Gib mir die Namen aller Studenten, die zwischen 1987-01-01 und 1989-01-01 geboren wurden,,

```
select Name  
from Student  
where Geburtstag between 1987-01-01 and 1989-01-01;
```

Anfrage äquivalent zu:

```
select Name  
from Student  
where Geburtstag >= 1987-01-01  
          and Geburtstag <= 1989-01-01;
```

String-Vergleiche

- Stringkonstanten müssen in einfachen Anführungszeichen eingeschlossen sein

Anfrage:

"Gib mir alle Informationen über den Professor mit dem Namen Kant,,

```
select *  
from Professoren  
where Name = 'Kant';
```


Suche mit Jokern (Wildcards)

Anfrage:

"Gib mir alle Informationen über Professoren,
deren Namen mit einem K anfängt"

```
select *  
from Professoren  
where Name like 'K%';
```

Mögliche Joker:

- `_` steht für ein beliebiges Zeichen
- `%` steht für eine beliebige Zeichenkette (auch der Länge 0)

Nullwerte

- In SQL gibt es einen speziellen Wert **NULL**
- Dieser Wert existiert für alle verschiedenen Datentypen und repräsentiert Werte, die
 - *unbekannt* oder
 - *nicht verfügbar* oder
 - *nicht anwendbar* sind.
- Nullwerte können auch im Zuge der Abfrageauswertung entstehen
- Auf NULL wird mit **is NULL** geprüft:

Beispiel:

```
select *  
from Professoren  
where Raum is NULL;
```

Nullwerte cont.

- Nullwerte werden in arithmetischen Ausdrücken durchgereicht: mindestens ein Operand NULL → Ergebnis ebenfalls NULL
- manchmal sehr überraschende Anfrageergebnisse, wenn Nullwerte vorkommen, z.B.:

```
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 dreiwertiger Logik unter Einbeziehung von NULL-Werten:

Auswertung bei Null-Werten

- SQL: dreiwertige Logik, mit den Werten **true**, **false** und **unknown**
- **unknown** liefern Vergleichsoperationen zurück, wenn mindestens eines ihrer Argumente NULL ist.
- In einer **where**-Bedingung werden nur Tupel weitergereicht, für die die Bedingung **true** ist. Insbesondere werden Tupel, für die die Bedingung zu **unknown** auswertet, nicht ins Ergebnis aufgenommen.
- Bei einer Gruppierung wird NULL als ein eigenständiger Wert aufgefasst und in eine eigene Gruppe eingeordnet.
- Logische Ausdrücke werden nach den folgenden Tabellen berechnet:

Dreiwertige Logik-Tabellen

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

not	
true	false
unknown	unknown
false	true

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

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			
VorINr	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

hören	
MatrNr	VorINr
26120	5001
27550	5001
27550	4052
28106	5041
28106	5052
28106	5216
28106	5259
29120	5001
29120	5041
29120	5049
25403	5022
29555	5022
29555	5001

voraussetzen	
Vorgänger	Nachfolger
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5259

prüfen			
MatrNr	VorINr	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

Anfragen über mehrere Relationen: Kreuzprodukt

- Falls mehrere Relationen in der from-Klausel auftauchen, werden sie mit einem Kreuzprodukt verbunden
- Beispiel:
Anfrage: "Gib alle Professoren und Vorlesungen aus,,

```
select *  
from Vorlesung, Professor;
```

Ergebnis???

Anfragen über mehrere Relationen: Joins

- Kreuzprodukte machen meistens keinen Sinn, interessanter sind Joins
- Joinprädikate werden in der where-Klausel angegeben:

```
select *  
from Vorlesung, Professor  
where gelesenVon = PersNr;
```


Anfragen über mehrere Relationen: Joins cont.

Welcher Professor liest "Mäeutik"?

```
select Name, Titel  
from Professoren, Vorlesungen  
where PersNr = gelesenVon  
       and Titel = 'Mäeutik';
```

Beispiel

Professoren				Vorlesungen			
PersNr	Name	Rang	Raum	VorlNr	Titel	SWS	gelesen Von
2125	Sokrates	C4	226	5001	Grundzüge	4	2137
2126	Russel	C4	232	5041	Ethik	4	2125
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2137	Kant	C4	7	5049	Mäeutik	2	2125
				⋮	⋮	⋮	⋮
				4630	Die 3 Kritiken	4	2137

PersN	Name	Rang	Raum	VorlNr	Titel	SWS	gelesen Von
2125	Sokrates	C4	226	5001	Grundzüge	4	2137
1225	Sokrates	C4	226	5041	Ethik	4	2125
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2125	Sokrates	C4	226	5049	Mäeutik	2	2125
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2126	Russel	C4	232	5001	Grundzüge	4	2137
2126	Russel	C4	232	5041	Ethik	4	2125
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2137	Kant	C4	7	4630	Die 3 Kritiken	4	2137

↓ Auswahl

PersN r	Name	Rang	Raum	VorlNr	Titel	SWS	gelesen Von
2125	Sokrates	C4	226	5049	Mäeutik	2	2125

↓ Projektion

Name	Titel
Sokrates	Mäeutik

Namenskollision

- gleichnamige Attribute in verschiedenen Relationen müssen aufgelöst werden

Beispiel:

Welche Studenten hören welche Vorlesungen?

```
select Name, Titel  
from Studenten, hören, Vorlesungen  
where Studenten.MatrNr = hören.MatrNr and  
hören.VorINr = Vorlesungen.VorINr;
```

Namenskollision cont.

Welche Studenten hören welche Vorlesungen?

Alternativ:

```
select s.Name, v.Titel  
from Studenten s, hören h, Vorlesungen v  
where s. MatrNr = h. MatrNr and  
      h.VorlNr = v.VorlNr
```

Mengenoperationen

- In SQL gibt es auch die üblichen Operationen auf Mengen:
Vereinigung, Schnitt und Differenz
- Setzen wie in der relationalen Algebra gleiches Schema der verknüpften Ausgabe-Relationen voraus

(**select** Name
from Assistenten)

union

(**select** Name
from Professoren);

Duplikateliminierung

- Im Gegensatz zu **select** eliminiert **union** automatisch Duplikate
- Falls Duplikate im Ergebnis erwünscht sind, muss der **union all**-Operator benutzt werden

Schnitt, Mengendifferenz

Professoren **und** Assistenten

```
select Name from Professoren
```

```
intersect
```

```
select Name from Assistenten;
```

Professoren, **aber nicht** Assistenten

```
select Name from Professoren
```

```
except
```

```
select Name from Assistenten;
```


Sortierung

- Tupel in einer Relation sind nicht (automatisch) sortiert
- Ergebnis einer Anfrage kann mit Hilfe der **order by**-Klausel sortiert werden
- Es kann aufsteigend oder absteigend sortiert werden
- Default Sortierung: aufsteigend

Beispiel

```
select *  
from Studenten  
order by Name, Semester desc;
```

Geschachtelte Anfragen

- Anfragen können in anderen Anfragen geschachtelt sein, d.h. es kann mehr als eine select-Klausel geben
- Geschachteltes select kann in der where-Klausel, in der from-Klausel und sogar in einer select-Klausel selbst auftauchen
- Im Prinzip wird in der "inneren" Anfrage ein Zwischenergebnis berechnet, das in der "äußeren" benutzt wird

Select in Where-Klausel

- Zwei verschiedene Arten von Unteranfragen: korrelierte und unkorrelierte
- unkorreliert: Unteranfrage bezieht sich nur auf "eigene" Attribute
- korreliert: Unteranfrage referenziert auch Attribute der äußeren Anfrage

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			
VorINr	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

hören	
MatrNr	VorINr
26120	5001
27550	5001
27550	4052
28106	5041
28106	5052
28106	5216
28106	5259
29120	5001
29120	5041
29120	5049
25403	5022
29555	5022
29555	5001

voraussetzen	
Vorgänger	Nachfolger
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5259

prüfen			
MatrNr	VorINr	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

Unkorrelierte Unteranfrage

Namen aller Studenten, die VorlNr 5041 hören

```
select S.Name  
from Studenten S  
where S.MatrNr in  
(select h.MatrNr  
from hoeren h  
where h.VorlNr = 5041);
```

- Unteranfrage wird einmal ausgewertet
- für jedes Tupel der äußeren Anfrage wird geprüft, ob die MatrNr im Ergebnis der Unteranfrage vorkommt

Korrelierte Unteranfrage

Finde alle Professoren, für die Assistenten mit voneinander unterschiedlichen Fachgebieten arbeiten

```
select distinct P.Name
```

```
from Professoren P, Assistenten A
```

```
where A.Boss = P.PersNr
```

```
and exists
```

```
(select *
```

```
from Assistent B
```

```
where B.Boss = P.PersNr and A.Fachgebiet <> B.Fachgebiet);
```

Korrelation

- Für jedes Tupel der äußeren Anfrage hat innere Anfrage verschiedene Werte
- das exists-Prädikat ist wahr, wenn die Unteranfrage mind. ein Tupel enthält

Existenzquantor exists

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


Existenzquantor exists

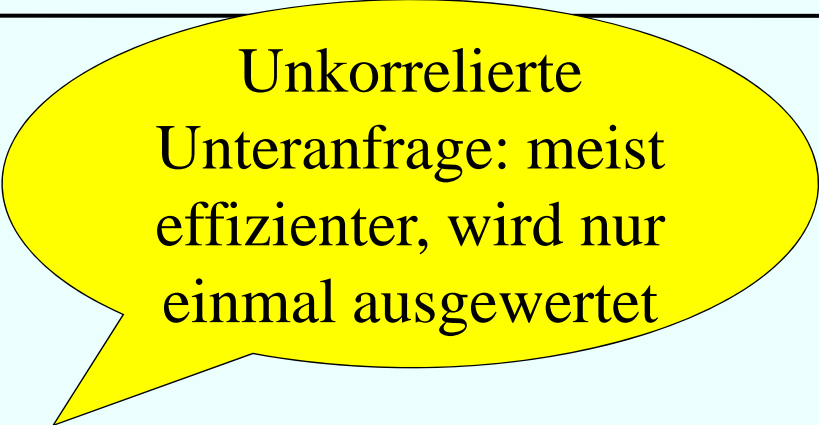
```
select P.Name  
from Professoren P  
where not exists ( select *  
                  from Vorlesungen V  
                  where V.gelesenVon = P.PersNr );
```



Korrelation

Mengenvergleich

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



Unkorrelierte
Unteranfrage: meist
effizienter, wird nur
einmal ausgewertet

Unkorrelierte versus korrelierte Unteranfragen

- korrelierte Formulierung

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

Umformulierung

- Äquivalente unkorrelierte Formulierung

```
select s.*
```

```
from Studenten s
```

```
where s.GebDatum <
```

```
    (select max (p.GebDatum)
```

```
    from Professoren p);
```

- Vorteil: Unteranfrageergebnis kann materialisiert werden
- Unteranfrage braucht nur einmal ausgewertet zu werden

Entschachtelung korrelierter Unteranfragen -- Forts.

```
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;
```

Aggregatfunktion und Gruppierung

Aggregatfunktionen **avg, max, min, count, sum**

```
select avg (Semester)  
from Studenten ;
```

```
select gelesenVon, sum (SWS)  
from Vorlesungen  
group by gelesenVon;
```

Aggregatfunktion und Gruppierung

```
select gelesenVon, Name, sum (SWS)
from Vorlesungen, Professoren
where gelesenVon = PersNr and Rang = 'C4'
group by gelesenVon, Name
having avg (SWS) >= 3;
```

Besonderheiten bei Aggregatoperationen

- SQL erzeugt pro Gruppe ein Ergebnistupel
- alle in der **select**-Klausel aufgeführten Attribute - außer den aggregierten – müssen auch in der **group by**-Klausel aufgeführt werden
- Nur so kann SQL sicherstellen, dass sich das Attribut nicht innerhalb der Gruppe ändert
- NULL Wert ist eigene Gruppe

Anfrage mit group by (Equi-Join, Selektion Rang='C4')

Vorlesung x Professoren							
Vorl Nr	Titel	SWS	gelesen Von	PersNr	Name	Rang	Raum
5001	Grundzüge	4	2137	2125	Sokrates	C4	226
5041	Ethik	4	2125	2125	Sokrates	C4	226
...	C3	...
4630	Die 3 Kritiken	4	2137	2137	Kant	C4	7

↓ **where**-Bedingung

Gruppieren nach gelesen Von, Name

VorlNr	Titel	SWS	gelesen Von	PersNr	Name	Rang	Raum
5001	Grundzüge	4	2137	2137	Kant	C4	7
5041	Ethik	4	2125	2125	Sokrates	C4	226
5043	Erkenntnistheorie	3	2126	2126	Russel	C4	232
5049	Mäeutik	2	2125	2125	Sokrates	C4	226
4052	Logik	4	2125	2125	Sokrates	C4	226
5052	Wissenschaftstheorie	3	2126	2126	Russel	C4	232
5216	Bioethik	2	2126	2126	Russel	C4	232
4630	Die 3 Kritiken	4	2137	2137	Kant	C4	7

↓ Gruppierung

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			
VorINr	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

hören	
MatrNr	VorINr
26120	5001
27550	5001
27550	4052
28106	5041
28106	5052
28106	5216
28106	5259
29120	5001
29120	5041
29120	5049
25403	5022
29555	5022
29555	5001

voraussetzen	
Vorgänger	Nachfolger
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5259

prüfen			
MatrNr	VorINr	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

Nur Gruppen mit mindestens 3 SWS im Schnitt

VorlNr	Titel	SWS	gelesenVon	PersNr	Name	Rang	Raum
5041	Ethik	4	2125	2125	Sokrates	C4	226
5049	Mäeutik	2	2125	2125	Sokrates	C4	226
4052	Logik	4	2125	2125	Sokrates	C4	226
5043	Erkenntnistheorie	3	2126	2126	Russel	C4	232
5052	Wissenschaftstheo.	3	2126	2126	Russel	C4	232
5216	Bioethik	2	2126	2126	Russel	C4	232
5001	Grundzüge	4	2137	2137	Kant	C4	7
4630	Die 3 Kritiken	4	2137	2137	Kant	C4	7

↓ **having** Bedingung

Summenbildung über SWS und Projektion

VorlNr	Titel	SWS	gelesenVon	PersNr	Name	Rang	Raum
5041	Ethik	4	2125	2125	Sokrates	C4	226
5049	Mäeutik	2	2125	2125	Sokrates	C4	226
4052	Logik	4	2125	2125	Sokrates	C4	226
5001	Grundzüge	4	2137	2137	Kant	C4	7
4630	Die 3 Kritiken	4	2137	2137	Kant	C4	7

↓ Aggregation (**sum**) und Projektion

Ergebnis

gelesenVon	Name	sum (SWS)
2125	Sokrates	10
2137	Kant	8

Maximum / Minimum

Gib mir den Studenten mit der größten MatrNr

```
select MatrNr, Name  
from Student  
where MatrNr =  
    (select max(MatrNr)  
    from Student);
```

NICHT

```
select Name, max(MatrNr)  
from Student;
```

Verwertung der Ergebnismenge einer Unteranfrage

```
select tmp.MatrNr, tmp.Name, tmp.VorlAnzahl
from (select s.MatrNr, s.Name, count(*) as VorlAnzahl
      from Studenten s, hoeren h
      where s.MatrNr=h.MatrNr
      group by s.MatrNr, s.Name) tmp
where tmp.VorlAnzahl > 2;
```

MatrNr	Name	VorlAnzahl
28106	Carnap	4
29120	Theophrastos	3

... oder auch

```
select tmp.MatrNr, tmp.Name, tmp.VorlAnzahl
from (select s.MatrNr, s.Name, count(*) as VorlAnzahl
      from Studenten s, hoeren h
      where s.MatrNr = h.MatrNr
      group by s.MatrNr, s.Name
      having count(*) > 2) tmp;
```

Decision-Support-Anfrage mit geschachtelten Unteranfragen

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

Das Ergebnis der Anfrage ?!

VorlNr	AnzProVorl	GesamtAnz	Marktanteil
4052	1	8	0
5001	3	8	0
5022	2	8	0
...

Casting der Integer zu Decimal

```
select h.VorlNr, h.AnzProVorl, g.GesamtAnz,  
        cast(h.AnzProVorl as decimal(6,3)) / g.GesamtAnz  
        as Marktanteil  
  
from ( select VorlNr, count(*) as AnzProVorl  
        from hören  
        group by VorlNr ) h,  
        ( select count (*) as GesamtAnz  
        from Studenten) g;
```

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			
VorINr	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

hören	
MatrNr	VorINr
26120	5001
27550	5001
27550	4052
28106	5041
28106	5052
28106	5216
28106	5259
29120	5001
29120	5041
29120	5049
25403	5022
29555	5022
29555	5001

voraussetzen	
Vorgänger	Nachfolger
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5259

prüfen			
MatrNr	VorINr	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

Das Ergebnis der Anfrage

VorlNr	AnzProVorl	GesamtAnz	Marktanteil
4052	1	8	.125
5001	3	8	.500
5022	2	8	.250
...

Weitere Anfragen mit Unteranfragen

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

```
select Name  
from Studenten  
where Semester > = all ( select Semester  
                        from Studenten );
```

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 prüfen;
```

Die **erste** qualifizierende **when**-Klausel
wird ausgeführt

Joins in SQL-92

- **cross join:** volles Kreuzprodukt (nicht in allen DBS!)
- **natural join:** natürlicher Join, Gleichheitstest auf alle gleichnamigen Attribute in den Relationen, Ausgabe aller Attribute, die gleichnamigen nur jeweils einmal (nicht in allen DBS!)
- **join** oder auch genannt **inner join:** Theta-Join, Theta Prädikat über Attribute
- **left, right** oder **full outer join:** äußerer Join
- **semi-join:** kein Operator in SQL, ausgedrückt mit **exists** oder **in** Konstrukte

(Inner) Join

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

oder auch

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

Äußere Joins (links)

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

Ergebnis

PersNr	p.Name	f.PersNr	f.Note	f.MatrNr	s.MatrNr	s.Name
2126	Russel	2126	1	28106	28106	Carnap
2125	Sokrates	2125	2	25403	25403	Jonas
2137	Kant	2137	2	27550	27550	Schopenhauer
2136	Curie	-	-	-	-	-
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Äußere Joins (rechts)

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

Ergebnis

PersNr	p.Name	f.PersNr	f.Note	f.MatrNr	s.MatrNr	s.Name
2126	Russel	2126	1	28106	28106	Carnap
2125	Sokrates	2125	2	25403	25403	Jonas
2137	Kant	2137	2	27550	27550	Schopen- hauer
-	-	-	-	-	26120	Fichte
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Äußere Joins (full)

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

Veränderung am Datenbestand: Einfügen

Einfügen von Tupeln durch **Anfrage**

insert into hören

select MatrNr, VorlNr

from Studenten, Vorlesungen

where Titel= `Logik`;

Einfügen von Tupeln durch **explizite Wertangabe**

insert into Studenten (MatrNr, Name)

values (28121, `Archimedes`), (4711, `Pythagoras`);

Veränderung am Datenbestand: Einfügen

Einfügen von Tupeln aus **Datei**

Datenbankspezifische Dienstprogramme, z.B. DB2:

- **Import:**

```
IMPORT FROM studis.tbl OF DEL  
INSERT INTO Studenten;
```

Analog: EXPORT TO studis.tbl OF DEL
SELECT * FROM Studenten;

- **Load:**

High-Performance Alternative zu Import

Oracle: Load, Datapump, ...

Veränderung am Datenbestand: Löschen, Verändern

```
delete from Studenten  
where Semester > 13;
```

Achtung: **delete from** Studenten
löscht gesamten **Inhalt** der Relation

```
update Studenten  
set Semester = Semester + 1;
```

Zweistufiges Vorgehen bei Änderungen

1. die Kandidaten für die Änderung werden ermittelt und "markiert"
2. die Änderung wird an den in Schritt 1. ermittelten Kandidaten durchgeführt

Anderenfalls könnte die Änderungsoperation von der Reihenfolge der Tupel abhängen, wie folgendes Beispiel zeigt:

delete from voraussetzen

where Vorgänger **in** (**select** Nachfolger
from voraussetzen);

Beispiel

voraussetzen	
Vorgänger	Nachfolger
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5229

Ohne einen Markierungsschritt hängt das Ergebnis dieser Anfrage von der Reihenfolge der Tupel in der Relation ab. Eine Abarbeitung in der Reihenfolge der Beispielausprägung würde das letzte Tupel (5052, 5229) fälschlicherweise erhalten, da vorher bereits alle Tupel mit 5052 als *Nachfolger* entfernt wurden.

Veränderungen am Schema

- **drop table** <Tabellenname>
- **alter table** <Tabellenname>
 - drop** | **add column** <Attributname> <Datentyp>
 - alter column** <Attributname> **set default** <default>
 - ...

Weitere datenbankspezifisch, z.B. Oracle:

- **alter table** <Tabellenname>
 - **modify** | **add column** <Attributname> <Datentyp>
 - **drop column** <Attributname>
 - **add** | **drop** | **enable** | **disable** <constraint-Klausel>

Sichten ...

- gehören zur DDL:
create view <viewname> **as** <select-statement>
- oft verwendet, um Anfragen übersichtlicher zu gestalten
- stellen eine Art "virtuelle Relation" dar
- zeigen einen Ausschnitt aus der Datenbank
- Vorteile
 - vereinfachen den Zugriff für bestimmte Benutzergruppen
 - können eingesetzt werden, um den Zugriff auf die Daten einzuschränken
- Nachteil
 - nicht auf allen Sichten können Änderungsoperationen ausgeführt werden

Erinnerung

```
select tmp.MatrNr, tmp.Name, tmp.VorlAnzahl
from (select s.MatrNr, s.Name, count(*) as VorlAnzahl
      from Studenten s, hoeren h
      where s.MatrNr=h.MatrNr
      group by s.MatrNr, s.Name) tmp
where tmp.VorlAnzahl > 2;
```

... auch möglich

```
with tmp (MatrNr, Name, VorlAnzahl) as
```

```
(select s.MatrNr, s.Name, count(*)
```

```
  from Studenten s, hoeren h
```

```
  where s.MatrNr=h.MatrNr
```

```
  group by s.MatrNr, s.Name)
```

```
select *
```

```
from tmp
```

```
where VorlAnzahl > 2;
```

→ temporäre Tabelle, nur gültig innerhalb der Query

Vereinfachung mit Sichten

Komplexe Anfrage: Finde die Namen aller Professoren, die Vorlesungen halten, die mehr als der Durchschnitt an Credits wert sind, und die mehr als drei Assistenten beschäftigen.

- nicht alles gleich auf einmal machen → kleinere übersichtlichere Teile
- diese Teile können mit Hilfe von Sichten realisiert werden oder auch mit benannten Zwischenergebnissen

Vereinfachung

1. Finde alle Vorlesungen mit überdurchschnittlich viel Credits:

```
create view ÜberSchnittCredit as  
select Nr, ProfPersNr  
from Vorlesung  
where Credits >  
  (select avg (Credits)  
   from Vorlesung);
```

Vereinfachung

2. Finde alle Professoren mit mehr als drei Assistenten:

```
create view VieleAssistenten as  
select Boss  
from Assistent  
group by Boss  
having count(*) > 3;
```

Vereinfachung

- alles zusammensetzen
- Sichten können wie eine herkömmliche Relation angesprochen werden

```
select Name
from Professor
where PersNr in
  (select PersNr
   from ÜberSchnittCredit) and
  PersNr in
  (select Boss
   from VieleAssistenten);
```

Sichten ...

für den Datenschutz

```
create view prüfenSicht as
```

```
  select MatrNr, VorINr, PersNr
```

```
  from prüfen
```

Sichten ...

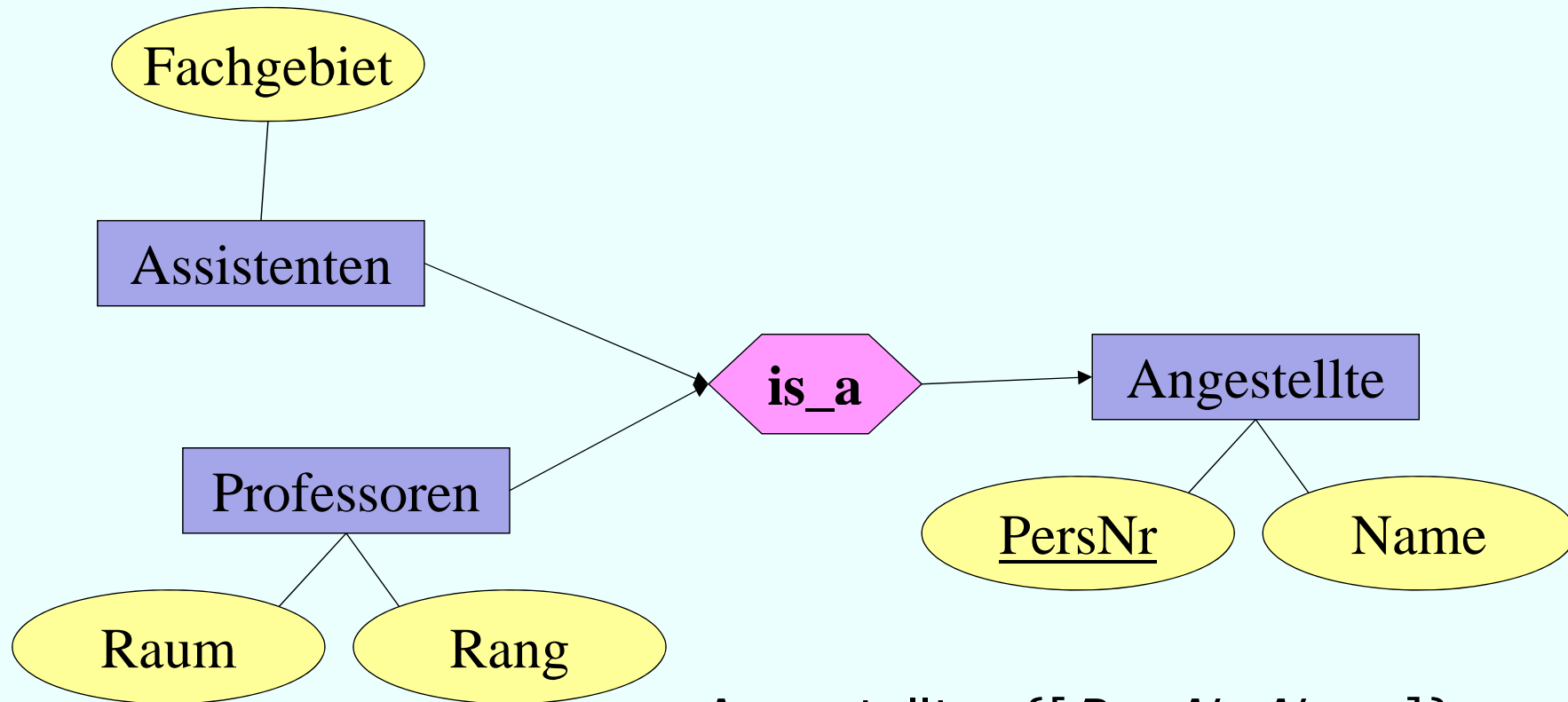
für den Datenschutz

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

für Statistik

```
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)
```

Relationale Modellierung der Generalisierung



Angestellte: {[PersNr, Name]}

ProfDaten: {[PersNr, Rang, Raum]}

AssiDaten: {[PersNr, Fachgebiet]}

Tabellendefinitionen

```
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) );
```


Sichten zur Modellierung von Generalisierung

```
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;
```

→ Untertypen als Sicht

Tabellendefinitionen

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) );
```

create table AndereAngestellte

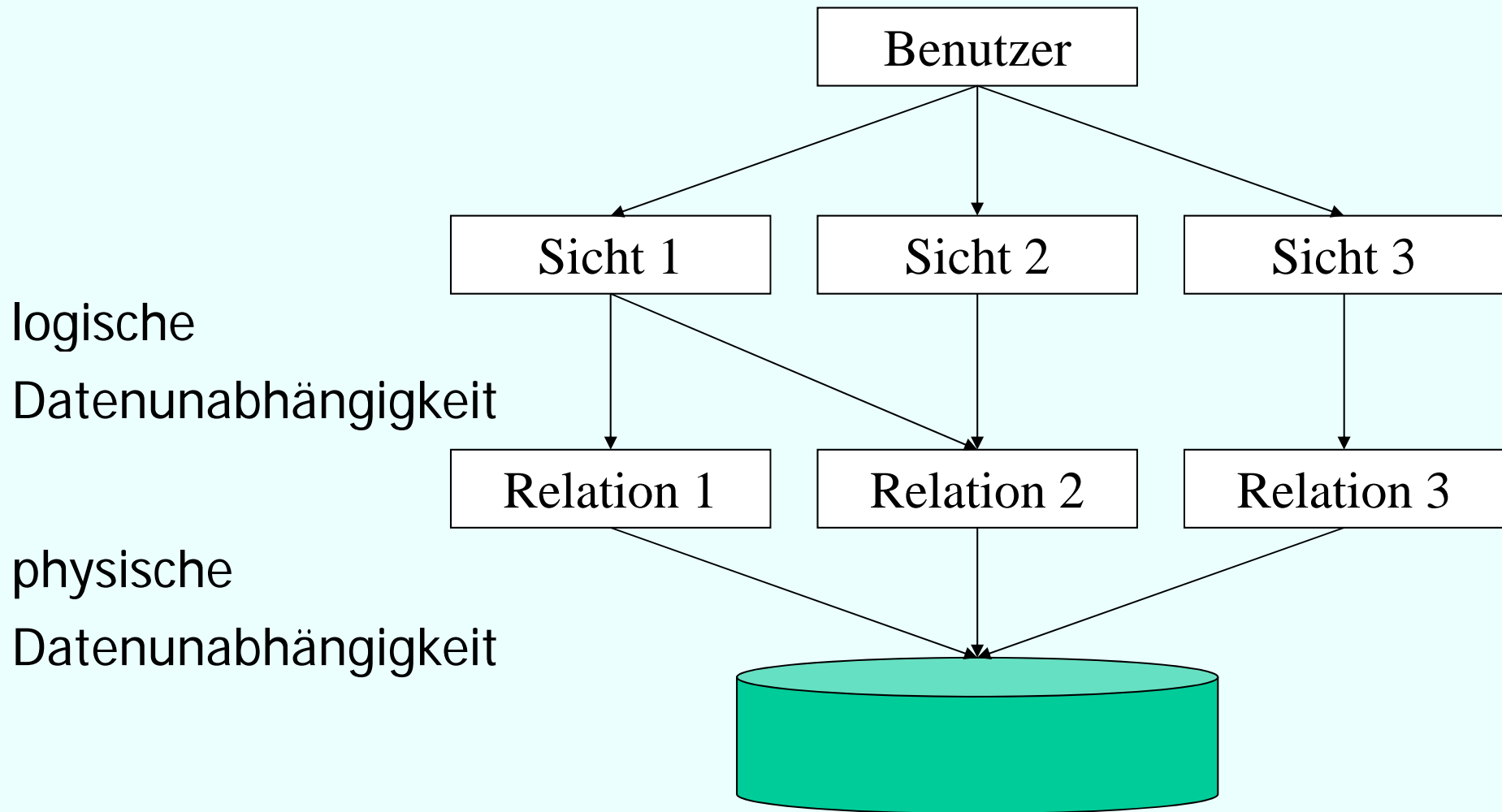
```
(PersNr integer not null,  
Name          varchar (30) not null);
```

Sichten zur Modellierung von Generalisierung

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

→ **Obertyp als Sicht**

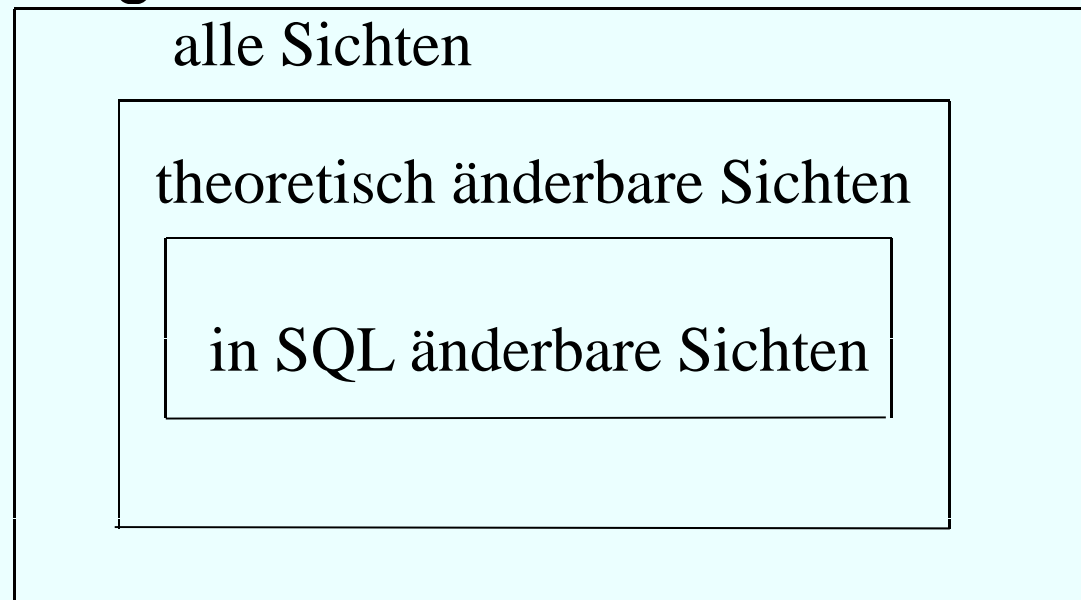
Sichten zur Gewährleistung von Datenunabhängigkeit



Änderbarkeit von Sichten

in SQL

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



Sichten

Lebensdauer, Gültigkeit

Löschen: **DROP VIEW *view-name***

Ungültige (inoperative) Views:

- Basisrelation wird gelöscht

- Rechteverlust des View-Erstellers

- View-Definition bleibt erhalten (ungültig markiert, kann durch Neudefinition reaktiviert werden)

Auswertung:

- Ersetzen der Sicht durch ihre Definition (~ Makro)

- keine** Speicherung (Materialisierung) der Sichtauswertung