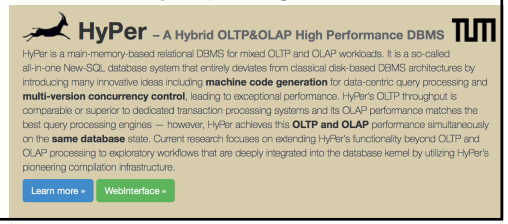
SQL

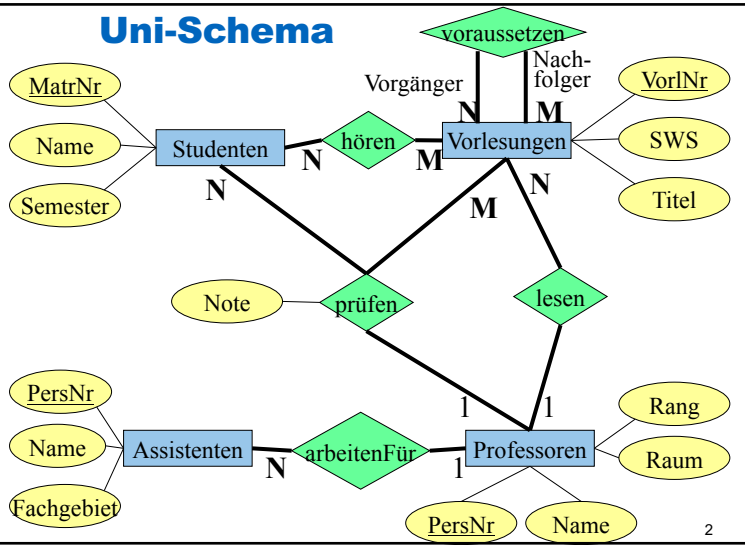
standardisierte
- Datendefinitions (DDL)-
- Datenmanipulations (DML)-
- Anfrage (Query)-Sprache

Fortlaufend erweiterter Standard (objektrelationale Erweiterung, Window-Functions, etc.)

Für praktische Übungen steht eine Web-Seite zur Verfügung: unser eigenes Datenbanksystem HyPer steht für „read-only" SQL-Übungen zur Verfügung:
- http://hyper-db.de



Uni-Schema



2

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

voraussetzen

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

prüfen

MatrNr	VorlNr	PersNr	Note
28106	5001	2126	1
25403	5041	2125	2
27550	4630	2137	2

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

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

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
4630	Die 3 Kritiken	4	2137

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

Die relationale Uni-DB

3

(Einfache) Datendefinition in SQL

Datentypen:
character *(n)*, ***char*** *(n)*
character varying *(n)*, **varchar** *(n)*
numeric *(p,s)*, **integer, decimal**
blob oder **raw** für sehr große binäre Daten
clob für sehr große String-Attribute
date für Datumsangaben
xml für XML-Dokumente

Anlegen von Tabellen/Relationen
create table Professoren
(PersNr **integer not null,**
Name **varchar** (30) **not null**
Rang **character** (2));

4

Veränderung am Datenbestand

Einfügen von Tupeln

insert into hören
  select MatrNr, VorlNr
  from Studenten, Vorlesungen
  where Titel= `Logik' ;

insert into Studenten (MatrNr, Name)
  values (28121, `Archimedes');

5

Studenten		
MatrNr	Name	Semester
⋮	⋮	⋮
29120	Theophrastos	2
29555	Feuerbach	2
28121	Archimedes	-

Null-Wert

6

Veränderungen am Datenbestand

Löschen von Tupeln

delete Studenten
where Semester > 13;

Verändern von Tupeln

update Studenten
  set Semester= Semester + 1;

7

Einfache SQL-Anfrage

select PersNr, Name
from Professoren
where Rang= ´C4´;

PersNr	Name
2125	Sokrates
2126	Russel
2136	Curie
2137	Kant

8

Einfache SQL-Anfragen

Sortierung

select PersNr, Name, Rang

from Professoren

order by Rang **desc**, Name **asc**;

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

Duplikateliminierung

select distinct Rang

from Professoren

Rang
C3
C4

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

voraussetzen

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

prüfen

MatrNr	VorlNr	PersNr	Note
28106	5001	2126	1
25403	5041	2125	2
27550	4630	2137	2

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

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

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
4630	Die 3 Kritiken	4	2137

Assistenten

PersInr	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

Die relationale Uni-DB

Anfragen über mehrere Relationen

Welcher Professor liest "Mäeutik"?

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

$$\prod_{\text{Name, Titel}}(\sigma_{\text{PersNr = gelesenVon} \wedge \text{Titel =' Mäeutik'}}(\text{Professoren} \times \text{Vorlesungen}))$$

Anfragen über mehrere Relationen

Professoren			
PersNr	Name	Rang	Raum
2125	Sokrates	C4	226
2126	Russel	C4	232
⋮	⋮	⋮	⋮
2137	Kant	C4	7

Vorlesungen			
VorlNr	Titel	SWS	gelesen Von
5001	Grundzüge	4	2137
5041	Ethik	4	2125
⋮	⋮	⋮	⋮
5049	Mäeutik	2	2125
⋮	⋮	⋮	⋮
4630	Die 3 Kritiken	4	2137

Verknüpfung X

PersNr	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

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

↓ Projektion

Name	Titel
Sokrates	Mäeutik

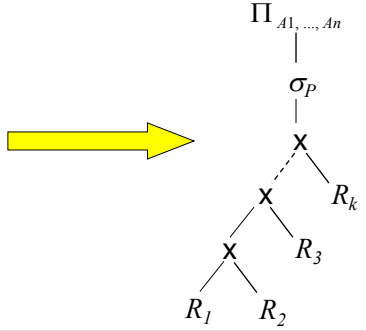
Kanonische Übersetzung in die relationale Algebra

Allgemein hat eine (ungeschachtelte) SQL-Anfrage die Form:

select $A_1, \ldots, A_n$
from $R_1, \ldots, R_k$
where P;

Übersetzung in die relationale Algebra:

$$\Pi_{A1, \ldots, An}(\sigma_P (R_1 \times \ldots \times R_k))$$



Anfragen über mehrere Relationen

Welche Studenten hören welche Vorlesungen?

select Name, Titel
from Studenten, hören, Vorlesungen
where Studenten.MatrNr = hören.MatrNr **and**
hören.VorlNr = Vorlesungen.VorlNr;

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

17

Welche Studenten kennen sich aus Vorlesungen

select s1.Name, s2.Name
from Studenten s1, hoeren h1, hoeren h2, Studenten s2
where h1.VorlNr = h2.VorlNr **and** h1.MatrNr = s1.MatrNr **and** h2.MatrNr = s2.MatrNr

18

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

voraussetzen

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

prüfen

MatrNr	VorlNr	PersNr	Note
28106	5001	2126	1
25403	5041	2125	2
27550	4630	2137	2

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

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

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
4630	Die 3 Kritiken	4	2137

Assistenten

PersInr	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

Die relationale Uni-DB

19

Mengenoperationen und geschachtelte Anfragen

Mengenoperationen **union, intersect, minus**

(**select** Name
from Assistenten)

union

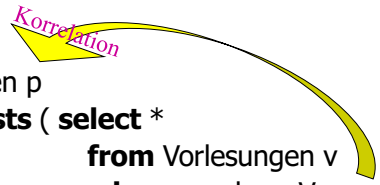
(**select** Name
from Professoren);

20

Existenzquantor **exists**

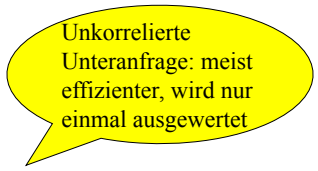
```
select p.Name
from Professoren p
where not exists ( select *
                   from Vorlesungen v
                   where v.gelesenVon = p.PersNr );
```


Existenzquantor **exists**




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


Mengenvergleich




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


Der Vergleich mit "all"

Kein vollwertiger Allquantor!

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

Aggregatfunktion und Gruppierung

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

```
select avg (Semester)
from Studenten;

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

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


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

voraussetzen

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

prüfen

MatrNr	VorlNr	PersNr	Note
28106	5001	2126	1
25403	5041	2125	2
27550	4630	2137	2

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

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

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
4630	Die 3 Kritiken	4	2137

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

Die relationale Uni-DB

Besonderheiten bei Aggregatoperationen

SQL erzeugt pro Gruppe ein Ergebnistupel

Deshalb müssen alle in der **select**-Klausel aufgeführten Attribute - außer den aggregierten – auch in der **group by**-Klausel aufgeführt werden

Nur so kann SQL sicherstellen, dass sich das Attribut nicht innerhalb der Gruppe ändert

Ausführen einer Anfrage mit group by

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
...	...	...	...	...	...	...	...
4630	Die 3 Kritiken	4	2137	2137	Kant	C4	7

⇓ **where**-Bedingung

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

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

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

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

Geschachtelte Anfrage (Forts.)

Unteranfrage in der **where**-Klausel
Welche Prüfungen sind besser als durchschnittlich verlaufen?

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

Geschachtelte Anfrage (Forts.)

Unteranfrage in der **select**-Klausel

Für jedes Ergebnistupel wird die Unteranfrage ausgeführt

Man beachte, dass die Unteranfrage korreliert ist (greift auf Attribute der umschließenden Anfrage zu)

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


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

voraussetzen

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

prüfen

MatrNr	VorlNr	PersNr	Note
28106	5001	2126	1
25403	5041	2125	2
27550	4630	2137	2

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

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

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
4630	Die 3 Kritiken	4	2137

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

Die relationale Uni-DB

Unkorrelierte versus korrelierte Unteranfragen

- korrelierte Formulierung

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


- 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 -- Fortseetzung

```
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;
```

37

Verwertung der Ergebnismenge einer Unteranfrage

```
select tmp.MatrNr, tmp.Name, tmp.VorlAnzahl
from (select s.MatrNr, s.Name, count(*) as VorlAnzahl
        from Studenten s, hören 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

38

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 hören
           group by VorlNr ) h,
         ( select count (*) as GesamtAnz
           from Studenten) g;
```

39

Casting der Integer zu Decimal

```
select   h.VorlNr, h.AnzProVorl, g.GesamtAnz,
         cast(h.AnzProVorl as decimal(6,2)) / g.GesamtAnz as Marktanteil
from     ( select VorlNr, count(*) as AnzProVorl
           from hören
           group by VorlNr ) h,
         ( select count (*) as GesamtAnz
           from Studenten) g;
```

40

Das Ergebnis der Anfrage

VorlNr	AnzProVorl	GesamtAnz	Marktanteil
4052	1	8	.125
5001	4	8	.5
5022	2	8	.25
...	...	...	...

Modularisierung mit „with“
CTE: Common Table Expression

```
with h as (select VorlNr, count(*) as AnzProVorl
           from hoeren
           group by VorlNr) ,
     g as  (select count (*) as GesamtAnz
           from Studenten)

select h.VorlNr, h.AnzProVorl, g.GesamtAnz,
       cast(h.AnzProVorl as decimal(6,2)) / g.GesamtAnz as Marktanteil
from g,h
```


Aufeinander aufbauende CTEs: Bekanntheitsgrad der Professoren aus Vorlesungen

```
with kenntSich(ProfPersNr,StudMatrNr) as
        (select distinct v.gelesenVon,h.MatrNr
         from hoeren h join Vorlesungen v on h.VorlNr=v.VorlNr) ,
  kenntAnzahl(ProfPersNr, AnzStudenten) as
        (select ProfPersNr, count(*) as AnzStudenten
         from kenntSich
         group by ProfPersNr),
  wieviele (GesamtAnz) as (select count(*) from Studenten)

select k.ProfPersNr, p.Name, k.AnzStudenten, w.GesamtAnz,
1.00 * k.AnzStudenten/w.GesamtAnz as Bekanntheitsgrad
from kenntAnzahl k, wieviele w, Professoren p
where k.ProfPErsNr = p.PersNr
order by Bekanntheitsgrad desc
```


Zwei erweiterte Relationen: zum Üben bestens geeignet – sind auch auf der HyPer Webschnittstelle verfügbar

ProfessorenF

PersNr	Name	Rang	Raum	Fakultaet
2125	Sokrates	C4	226	Philosophie
2126	Russel	C4	232	Philosophie
2127	Kopernikus	C3	310	Physik
2133	Popper	C3	52	Philosophie
2134	Augustinus	C3	309	Theologie
2136	Curie	C4	36	Physik
2137	Kant	C4	7	Philosophie

StudentenGF

MatrNr	Name	Semester	Geschlecht	Fakultaet
24002	Xenokrates	18	M	Philosophie
25403	Jonas	12	W	Theologie
26120	Fichte	10	W	Philosophie
26830	Aristoxenos	8	M	Philosophie
27550	Schopenhauer	6	M	Philosophie
28106	Carnap	3	W	Physik
29120	Theophrastos	2	M	Physik
29555	Feuerbach	2	W	Theologie

Beispiel-Anfragen

- Welche Fakultät hat den höchsten Frauenanteil
- Wer hat nur Vorlesungen seiner/ihrer Fakultät gehört
- Wer hat 80% aller Vorlesungen seiner/ihrer Fakultät gehört
- Welche Professoren haben alle Studenten ihrer Fakultät unterrichtet
- ... Und vieles mehr (dann mglw. in der Klausur)

Frauenanteil pro Fakultät

```
select anz.Fakultaet, anz.AnzStudenten, anzw.AnzWeiblich,
(cast(anzw.AnzWeiblich as decimal(5,2))/anz.AnzStudenten *
100) as ProzentWeiblich
from
        (select s.Fakultaet, count(*) as AnzStudenten
         from StudentenGF s
         group by s.Fakultaet) as anz,
        (select sw.Fakultaet,count(*) as AnzWeiblich
         from StudentenGF sw
         where sw.Geschlecht ='W'
         group by sw.Fakultaet) as anzw
where anz.Fakultaet= anzw.Fakultaet
```


Frauenanteil pro Fakultät ... „pretty printed“

```
select anz.Fakultaet, anz.AnzStudenten, anzw.AnzWeiblich,
(cast(anzw.AnzWeiblich as decimal(5,2))/anz.AnzStudenten * 100) as ProzentWeiblich
from
    (select s.Fakultaet, count(*) as AnzStudenten
     from StudentenGF s
     group by s.Fakultaet) as anz,
    (select sw.Fakultaet,count(*) as AnzWeiblich
     from StudentenGF sw
     where sw.Geschlecht ='W'
     group by sw.Fakultaet) as anzw
where anz.Fakultaet= anzw.Fakultaet
```


Weitere Anfragen mit Unteranfragen

```
( select Name
  from Assistenten )
union
( select Name
  from Professoren );

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

Quantifizierte Anfragen in SQL

- Existenzquantor: **exists**

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


Allquantifizierung

SQL-92 hat keinen Allquantor

Allquantifizierung muß also durch eine äquivalente Anfrage mit Existenzquantifizierung ausgedrückt werden

Kalkülformulierung der Anfrage: Wer hat **alle** vierstündigen Vorlesungen gehört?

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

Elimination von $\forall$ und $\Rightarrow$
Dazu sind folgende Äquivalenzen anzuwenden

$$\forall t \in R\ (P(t)) = \neg(\exists t \in R(\neg P(t)))$$
$$R \Rightarrow T = \neg R \vee T$$

Umformung des Kalkül-Ausdrucks ...

Elimination $\forall$

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

Elimination $\Rightarrow$

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

Anwendung von DeMorgan ergibt schließlich:

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

SQL-Umsetzung folgt „kanonisch“:

```
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 ) );
```

53

Allquantifizierung durch count-Aggregation

Allquantifizierung kann immer 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);
```

54

Herausforderung

- Wie formuliert man die komplexere Anfrage: Wer hat alle vierstündigen Vorlesungen gehört
- Grundidee besteht darin, vorher durch einen Join die Studenten/Vorlesungs-Paare einzuschränken und danach das Zählen durchzuführen

55

Nullwerte

- unbekannter Wert
- wird vielleicht später nachgereicht
- Nullwerte können auch im Zuge der Anfrageauswertung entstehen (Bsp. äußere Joins)
- manchmal sehr überraschende Anfrageergebnisse, wenn Nullwerte vorkommen

```
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 **null**-Werten begründet:

56

Auswertung bei Null-Werten

- 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:

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

Diese Berechnungsvorschriften sind recht intuitiv. Unknown or true wird z.B. zu **true** - die Disjunktion ist mit dem **true**-Wert des rechten Arguments immer erfüllt, unabhängig von der Belegung des linken Arguments. Analog ist **unknown and false** automatisch **false** - keine Belegung des linken Arguments könnte die Konjunktion mehr erfüllen.

- 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.

Spezielle Sprachkonstrukte ("syntaktischer Zucker")

```
select *
from Studenten
where Semester > = 1 and Semester < = 4;

select *
from Studenten
where Semester between 1 and 4;

select *
from Studenten
where Semester in (1,2,3,4);
```

String-Vergleiche mit like

Platzhalter "%" ; "_"
- "%" steht für beliebig viele (auch gar kein) Zeichen
- "_" steht für genau ein Zeichen

```
select *
from Studenten
where Name like `T%eophrastos`;

select distinct s.Name
from Vorlesungen v, hören h, Studenten s
where s.MatrNr = h.MatrNr and h.VorlNr = v.VorlNr and
      v.Titel like `%thik%`;
```

Suchbaum-Index
Nicht sinnvoll nutzbar

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: Kreuzprodukt
natural join: natürlicher Join (nicht alle DBMS)
Join oder inner join: Theta-Join
left, right oder full outer join: äußerer Join
union join: Vereinigungs-Join (wird hier nicht vorgestellt)

```
select *
from R1, R2
where  R1.A = R2.B;

select *
from R1 join R2 on R1.A = R2.B;
```


Äußere Joins

```
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;
```

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
2136	Curie	-	-	-	-	-
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Äußere Joins

```
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;
```

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
⋮	⋮	⋮	⋮	⋮	⋮	⋮

65

Äußere Joins

```
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;
```

66

p.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
⋮	⋮	⋮	⋮	⋮	⋮	⋮
2136	Curie	-	-	-	-	-
⋮	⋮	⋮	⋮	⋮	⋮	⋮

67

Rekursion

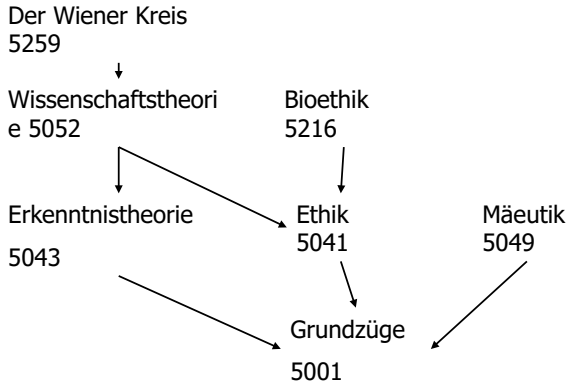
```
select Vorgänger
from voraussetzen, Vorlesungen
where Nachfolger= VorlNr and
      Titel= `Der Wiener Kreis´
```

68

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

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
4630	Die 3 Kritiken	4	2137

69



70

Rekursion

Vor-Vorgänger-Vorlesungen des „Wiener Kreis“

select v1.Vorgänger
from voraussetzen v1, voraussetzen v2, Vorlesungen v
where v1.Nachfolger= v2.Vorgänger **and**
v2.Nachfolger= v.VorlNr **and**
v.Titel=`Der Wiener Kreis´

71

Rekursion

select v1.Vorgänger
from voraussetzen v1, voraussetzen v2, Vorlesungen v
where v1.Nachfolger= v2.Vorgänger **and**
v2.Nachfolger= v.VorlNr **and**
v.Titel=`Der Wiener Kreis´

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

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

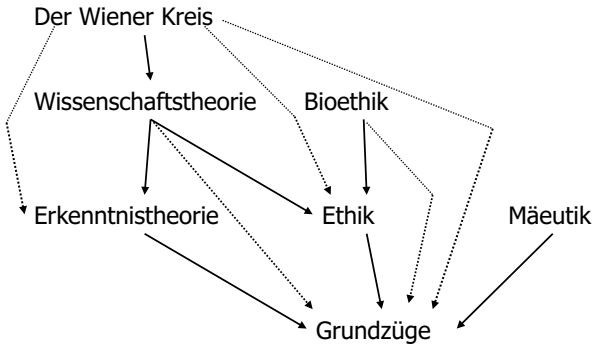
Vorlesungen			
VorlNr	Titel	SWS	gelesen Von
5001	Grundzüge	4	2137
5041	Ethik	4	2125
5043	Erkenntnistheorie	3	2126
5049	Mäeutik	2	2125
4052	Logik	4	2125
5052	Wissenschaftstheorie	3	2126
5216	Bioethik	2	2126
5259	Der Wiener Kreis	2	2133
5022	Glaube und Wissen	2	2134
4630	Die 3 Kritiken	4	2137

72

Vorgänger des „Wiener Kreises“ der Tiefe n

```
select v1.Vorgänger
from voraussetzen v1
        :
        voraussetzen vn_minus_1
        voraussetzen vn,
        Vorlesungen v
where v1.Nachfolger= v2.Vorgänger and
        :
        vn_minus_1.Nachfolger= vn.Vorgänger and
      vn.Nachfolger = v.VorlNr and
        v.Titel= `Der Wiener Kreis´
```


Transitive Hülle: alle (gerichteten) Pfade



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

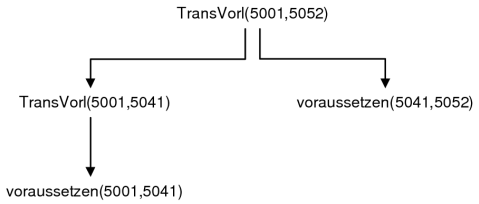
Vorlesungen			
VorlNr	Titel	SWS	gelesen Von
5001	Grundzüge	4	2137
5041	Ethik	4	2125
5043	Erkenntnistheorie	3	2126
5049	Mäeutik	2	2125
4052	Logik	4	2125
5052	Wissenschaftstheorie	3	2126
5216	Bioethik	2	2126
5259	Der Wiener Kreis	2	2133
5022	Glaube und Wissen	2	2134
4630	Die 3 Kritiken	4	2137

Rekursion in Prolog/Datalog


```
kante(1,2).
kante(2,3).
kante(3,4).
kante(4,5).
kante(4,6).
kante(6,7).
kante(3,6).

pfad(V,N) :- kante(V,N).
pfad(V,N) :- kante(V,Z),pfad(Z,N).
```

Transitive Hülle der Relation voraussetzen (in logischer Programmierung: Prolog/Datalog)

TransVorl(V,N) :- voraussetzen(V,N).
TransVorl(V,N) :- TransVorl(V,Z), voraussetzen(Z,N).



77

Rekursion in SQL: gleiche Anfrage

```
with recursive TransVorl (Vorg, Nachf)
as (select Vorgänger, Nachfolger from voraussetzen
      union all
    select t.Vorg, v.Nachfolger
    from TransVorl t, voraussetzen v
    where t.Nachf= v.Vorgänger)

select Titel from Vorlesungen where VorlNr in
   (select Vorg from TransVorl where Nachf in
      (select VorlNr from Vorlesungen
      where Titel= `Der Wiener Kreis´) )
```

78

- zuerst wird eine temporäre Sicht *TransVorl* mit der CTE (also der **with-**Klausel) angelegt
- Diese Sicht *TransVorl* ist rekursiv definiert, da sie selbst in der Definition vorkommt
- Aus dieser Sicht werden dann die gewünschten Tupel extrahiert
- Ergebnis ist natürlich wie gehabt

79

Veränderung am Datenbestand

Einfügen von Tupeln

```
insert into hören
      select MatrNr, VorlNr
      from Studenten, Vorlesungen
      where Titel= `Logik´;

insert into Studenten (MatrNr, Name)
      values (28121,  `Archimedes´);
```

80

Studenten		
MatrNr	Name	Semester
⋮	⋮	⋮
29120	Theophrastos	2
29555	Feuerbach	2
28121	Archimedes	-

Veränderungen am Datenbestand

Löschen von Tupeln

delete Studenten

where Semester > 13;

Verändern von Tupeln

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);

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.

85

Sichten ...

für den Datenschutz

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

86

Sichten ...

für den Datenschutz

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

Statistische Sicht

```
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)

87

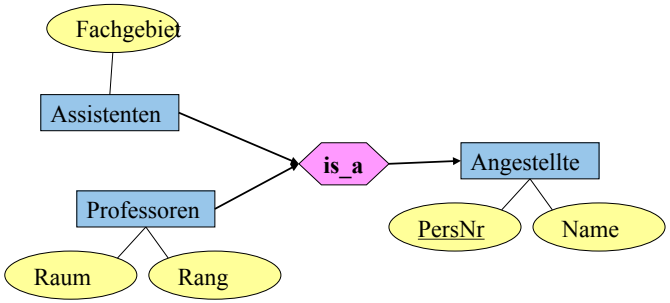
Sichten ...

für die Vereinfachung von Anfagen

```
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';
```

88

Relationale Modellierung der Generalisierung



Angestellte: {[*PersNr, Name*]}

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

Assistenten: {[*PersNr, Fachgebiet*]}

89

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

90

```
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

91

```
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);
```

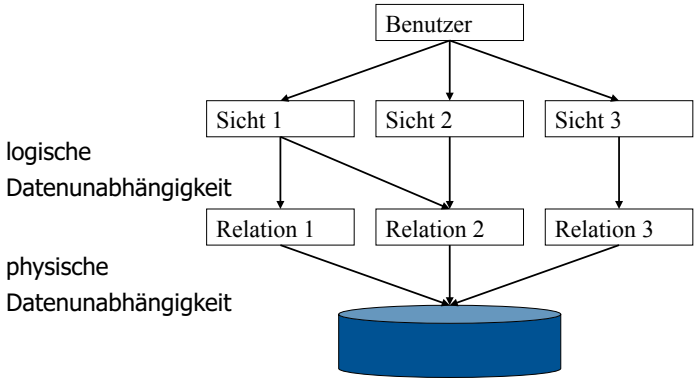
92

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

➔ Obertypen als Sicht

93

Sichten zur Gewährleistung von Datenunabhängigkeit



94

Änderbarkeit von Sichten

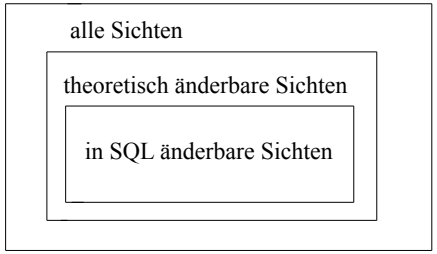
Beispiele für nicht änderbare Sichten

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

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

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

95

Änderbarkeit von Sichten

in SQL ...
- nur eine Basisrelation
- Schlüssel muss vorhanden sein
- keine Aggregatfunktionen, Gruppierung oder Duplikateliminierung



96

Embedded SQL

```
#include <stdio.h>
/*Kommunikationsvariablen deklarieren */
exec sql begin declare section;
        varchar user_passwd[30];
        int exMatrNr;
exec sql end declare section;
exec sql include SQLCA;
main()
{
        printf("Name/Password:");
        scanf("%", user_passwd.arr);
```

97

```
user_passwd.len=strlen(user_passwd.arr);
exec sql wheneversqlerror goto error;
exec sql connect :user_passwd;
while (1) {
        printf("Matrikelnummer (0 zum beenden):");
        scanf("%d", &ecMatrNr);
        if (!exMatrNr) break;
        exec sql delete from Studenten
                where MatrNr= :exMatrNr;
}
exec sql commit work release;
exit(0);
```

98

```
error:
exec sql whenever sqlerror continue;
exec sql rollback work release;
printf("fehler aufgetreten!\n");
exit(-1);
}
```

99

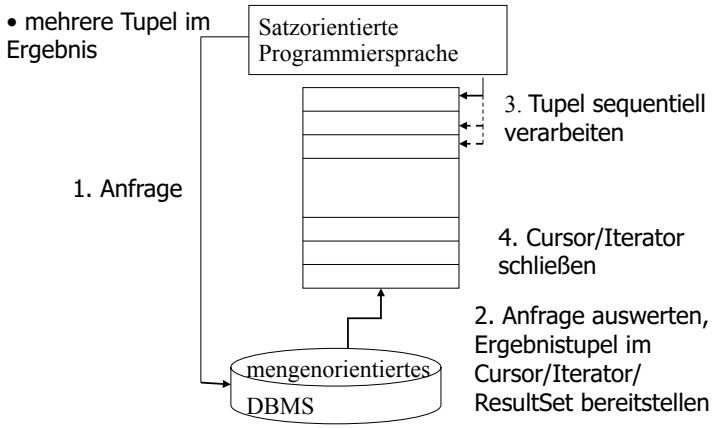
Anfragen in Anwendungsprogrammen

genau ein Tupel im Ergebnis

```
exec sql select avg (Semester)
        into :avgsem
        from Studenten;
```

100

Anfragen in Anwendungsprogrammen



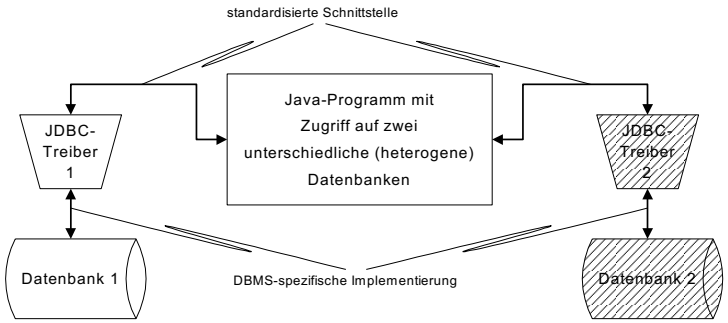
Cursor-Schnittstelle in SQL

1. **exec sql declare** c4profs **cursor for**
   select Name, Raum
   from Professoren
   where Rang='C4';
2. **exec sql open** c4profs;
3. **exec sql fetch** c4profs into :pname, :praum;
4. **exec sql close** c4profs;

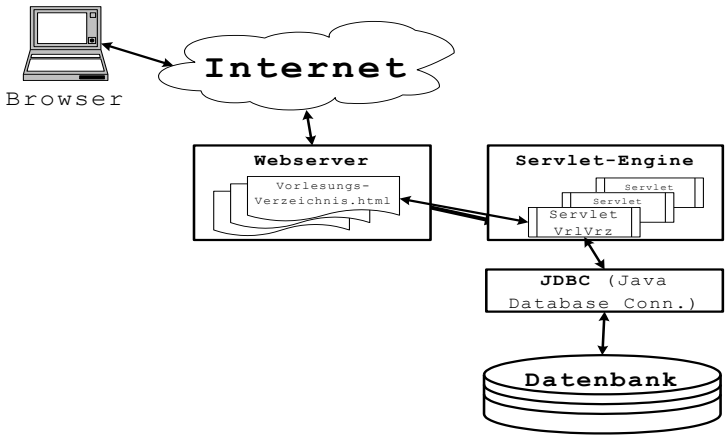
JDBC: Java Database Connectivity

Standardisierte Schnittstelle zur Anbindung von relationalen Datenbanken an Java
Wird heute fast immer für die Anbindung von Datenbanken an das Internet/Web verwendet
- Java Servlets als dynamische Erweiterung von Webservern
- Java Server Pages (JSP): HTML-Seiten mit eingebetteten Java Programmfragmenten

Zugriff auf Datenbanken via JDBC



Web-Anbindung von Datenbanken via Servlets/JDBC



105

JDBC-Beispielprogramm

```
import java.sql.*;   import java.io.*;
public class ResultSetExample {
  public static void main(String[] argv) {
    Statement sql_stmt = null;
    Connection conn = null;
    try {
      Class.forName("oracle.jdbc.driver.OracleDriver");
      conn = DriverManager.getConnection
              ("jdbc:oracle:oci8:@lsintern-db", "nobody", "Passwort");
      sql_stmt = conn.createStatement();
    }
    catch (Exception e) {
      System.err.println("Folgender Fehler ist aufgetreten: " + e);
      System.exit(-1);      }
```

106

```
try {
    ResultSet rset = sql_stmt.executeQuery(
        "select avg(Semester) from Studenten");
    rset.next();   // eigentlich zu prüfen, ob Ergebnis leer
    System.out.println("Durchschnittsalter: " + rset.getDouble(1));
    rset.close();
  }
  catch(SQLException se) {
    System.out.println("Error: " + se);
  }
```

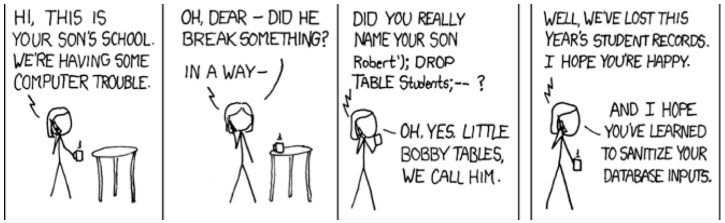
107

```
try {
    ResultSet rset = sql_stmt.executeQuery(
      "select Name, Raum from Professoren where Rang = 'C4'");
    System.out.println("C4-Professoren:");
    while(rset.next())  {
        System.out.println(rset.getString("Name") + " " +
                rset.getInt("Raum"));
    }
    rset.close();
  }
  catch(SQLException se) {System.out.println("Error: " + se); }
  try {
    sql_stmt.close(); conn.close();
  }
  catch (SQLException e) {
    System.out.println("Fehler beim Schliessen der DB: " + e);
  }
 }
}
```

108

Sicherheitsproblem: SQL Injection

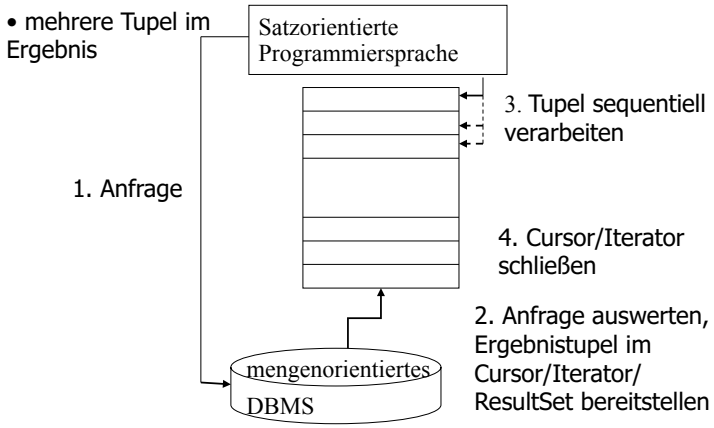
(von: xkcd übernommen)



Vorübersetzung von SQL-Ausdrücken

```
PreparedStatement sql_exmatrikuliere =
   conn.prepareStatement
         ("delete from Studenten where MatrNr = ?");

int VomBenutzerEingeleseneMatrNr;
  // zu löschende MatrNr einlesen
sql_exmatrikuliere.setInt(1,VomBenutzerEingeleseneMatrNr);

int rows = sql_exmatrikuliere.executeUpdate();
if (rows == 1) System.out.println("StudentIn gelöscht.");
  else System.out.println("Kein/e StudentIn mit dieser MatrNr.");
```


Anfragen in Anwendungsprogrammen



SQL/J-Beispielprogramm

```
import java.io.*; import java.sql.*;
import sqlj.runtime.*; import sqlj.runtime.ref.*;

#sql iterator StudentenItr (String Name, int Semester);

public class SQLJExmp {
  public static void main(String[] argv) {
    try {
      Class.forName("COM.ibm.db2.jdbc.app.DB2Driver");
      Connection con = DriverManager.getConnection
                                                ("jdbc:db2:uni");
      con.setAutoCommit(false);
      DefaultContext ctx = new DefaultContext(con);
      DefaultContext.setDefaultContext(ctx);
```

```
StudentenItr Methusaleme;
 #sql Methusaleme = { select s.Name, s.Semester
                      from Studenten s
                      where s.Semester > 13 };
 while (Methusaleme.next()) {
   System.out.println(Methusaleme.Name() + ":" +
                      Methusaleme.Semester());
 }
 Methusaleme.close();
 #sql { delete from Studenten where Semester > 13 };
 #sql { commit };
}
catch (SQLException e) {
 System.out.println("Fehler mit der DB-Verbindung: " + e);
}
catch (Exception e) {
 System.err.println("Folgender Fehler ist aufgetreten: " + e);
 System.exit(-1);    } } }
```


Query by Example

Vorlesungen	VorlNr	Titel	SWS	gelesen Von
		p._t	> 3	

Analog

$$\{[t] \mid \exists v, s, r\ ([v,t,s,r] \in \text{Vorlesungen} \wedge s > 3)\}$$

Join in QBE

Vorlesungen	VorlNr	Titel	SWS	gelesen Von
		Mäeutik		_x

Professoren	PersNr	Name	Rang	Raum
	_x	**p.**_n		

Die Condition Box

Studenten	MatrNr	Name	Semester
		_s	_a

conditions
_a > _b

Studenten	MatrNr	Name	Semester
		_t	_b

Betreuen	potentieller Tutor	Betreuter
p.	_s	_t

Aggregatfunktion und Gruppierung

Vorlesungen	VorlNr	Titel	SWS	gelesen Von
			p.sum.all._x	**p.g.**

conditions
avg.all._x>2

Updates in QBE: Sokrates ist „von uns gegangen“

Professoren	PersNr	Name	Rang	Raum
d.	_x	Sokrates		

Vorlesungen	VorlNr	Titel	SWS	gelesen Von
d.	_y			_x

hören	VorlNr	MatrNr
d.	_y	

Nachfolgend ein paar Notizen zu Übungen ...

... kein Teil der Vorlesung

Uni.PunkteListe: Jede/r erhält einen „Freischuss“

NAME	AUFGABE	MAX	ERZIELT
Bond	1	10	4
Bond	2	10	10
Bond	3	11	4
Maier	1	10	4
Maier	2	10	2
Maier	3	11	3

With Bonus as (…)
...
select * from Bonus

Name	MaxMoeglich	Erzielt
Bond	31	25
Maier	31	17

QuizBonus-Aufgabe (genau ein Bonus pro Student/in)

```
create or replace View QuizBonus as(
  with Schlechteste as (select * from Quiz s  /* schwächsteN pro St */
  where not exists (
       select *  /* es gibt kein anderes schlechteres Quiz dieses Studenten*/
       from Quiz w
       where w.Name = s.Name and w.Max - w.Erzielt > s.Max - s.Erzielt) ),
   AeltestesSchlechtestes as (select * from Schlechteste s
                                  where not exists (
                                       select * from Schlechteste w
                                       where w.Name = s.Name and s.Qno > w.Qno))

select s.Qno,s.Name,s.Max,s.Max as Erzielt from AeltestesSchlechtestes s
      union
select * from Quiz q where not exists (select * from AeltestesSchlechtestes s
                                        where s.Name = q.Name and s.Qno = q.Qno))
```

Qno	Name	Max	Erzielt
1	Meier	80	60
2	Meier	100	88
1	Schmidt	80	60
2	Schmidt	100	80

Medaillengewinner im Zehnkampf:

N	P
Eaton	8869
Suarez	8523
Behrenbruch	8126
Hardee	8671
...	...

SILBER:

```
select * from Z s
where exists (      /* genau einer ist besser */
          select * from Z g
          where g.P > s.P and not exists(
                    select * from Z l
                    where not(l.N=g.N or l.P<= s.P)))
```

Medaillengewinner

N	P
Eaton	8869
Suarez	8523
Behrenbruch	8126
Hardee	8671
...	...

Bronze:

```
select * from Z b
where 2 = (select count(*) from Z gs
           where gs.P > b.P)
```


Medaillengewinner

N	P
Eaton	8869
Suarez	8523
Behrenbruch	8126
Hardee	8671
...	...

Gold:

```
select g.*from Z g where g.N not in (  /*Loser*/
        select l.N
        from Z l join Z b on l.P < b.P)
          /* Man ist Loser l wenn es jemand Besseren b gibt */
```


Rekursive Sicht Pfad mit Pfadlänge

```
with recursive pfad(von,nach,l) as
        ((select v,n,1 from kante)
        union all
        (select k.V, p.nach, p.l+1
        from kante k, pfad p
        where k.N=p.von))

select * from pfad
```


U-Bahn Verbindungen

U6_sued

Q#	nach	Dauer
Forschungszentrum	Garching	2
Garching	Hochbrück	2
Hochbrück	Fröttmaning	4
Fröttmaning	Kieferngarten	1
...	...	

```
with recursive U6_Verbindungen(von,nach,Halte,Dauer) as
        ((select von,nach,1,Dauer from u6_sued)
        union all
        (select k.Von, p.nach, p.Halte+1, k.Dauer+p.Dauer
        from u6_sued k, u6_Verbindungen p
        where k.Nach=p.von))

select * from U6_Verbindungen
```