Formalismus des Relationalen Modells, Transformation eines ER-Modells, Relationale Algebra: Selektion, Projektion, kartesisches Produkt, Umbenennung und Joins

10. DATENBANKSYSTEME: DAS RELATIONALE MODELL

Relationales Modell, Formalismus

- Relation *R*
  - $R \subseteq D_1$ x ... x D_n
  - $D_1, D_2, ..., D_n$ sind Domänen

  Beispiel: Telefonbuch ⊆ string x string x integer
- Tupel: $t \in R$

  Beispiel: t = („Mickey Mouse“, „Main Street“, 4711)
- Relationenschemata werden wie folgt beschrieben

  Telefonbuch: {[*Name: string, Strasse: string, Telefon#:integer*]}

{[...]} deuten an, dass ein Schema eine Menge von Tupeln [] ist

Name des Attributes

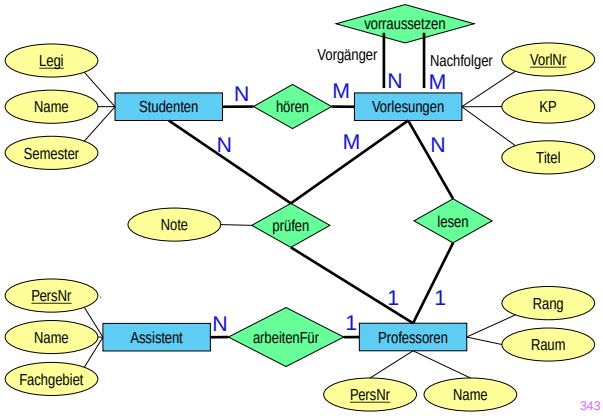
Typ des Attributes

Relationales Modell

Telefonbuch		
Name	Strasse	Telefon#
Mickey Mouse	Main Street	4711
Minnie Mouse	Broadway	94725
Donald Duck	Broadway	95672
...	...	...

- Ausprägung: Zustand der Datenbank
- Schlüssel: minimale Menge von Attributen, welche ein Tupel eindeutig identifizieren
  z.B. {Telefon#} oder {Name, Geburtstag}
- Primärschlüssel (durch Unterstreichung hervorgehoben): Ausgewählter Schlüssel, welcher üblicherweise zur Identifikation eines Tupels in einer Relation verwendet wird.

Uni Schema



Regel #1: Darstellung von Entities

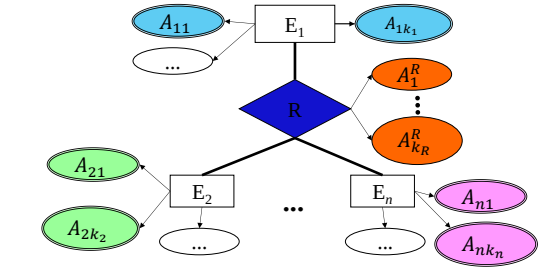
Studenten:
{[Legi:integer, *Name*:*string, Semester*: *integer*]}

Vorlesungen:
{[VorlNr:integer, *Titel*: *string, KP*: *integer*]}

Professoren:
{[PersNr:integer, *Name*: *string, Rang*: *string,Raum*: *integer*]}

Assistenten:
{[PersNr:integer, *Name*: *string, Fachgebiet*: *string*]}

Regel #2: Darstellung von Beziehungen



$$R:\{[\underbrace{A_{11},...,A_{1k_1}}_{\text{Schlüssel } E_1},\underbrace{A_{21},...,A_{2k_2}}_{\text{Schlüssel } E_2},...,\underbrace{A_{n1},...,A_{nk_n}}_{\text{Schlüssel } E_n},\underbrace{A_1^R,...,A_{k_R}^R}_{\text{Attribute von R}}]\}$$

Darstellung von Beziehungen

hören:
{[Legi: integer, VorlNr: integer]}

lesen :
{[PersNr: integer, VorlNr: integer]}

Fremdschlüssel, identifizieren Tupel aus anderen Entitäten

arbeitenFür :
{[AssiPersNr: integer, ProfPersNr: integer]}

voraussetzen:
{[Vorgänger: integer, Nachfolger: integer]}

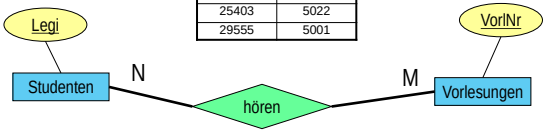
prüfen :
{[Legi: integer, VorlNr: integer, PersNr: integer, Note: decimal]}

Ausprägung von *hören*

Studenten	
Legi	...
26120	...
27550	...
...	...

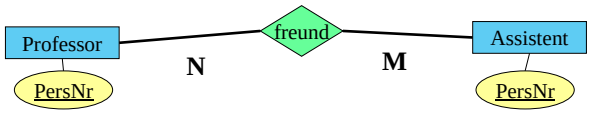
hören	
Legi	VorlNr
26120	5001
27550	5001
27550	4052
28106	5041
28106	5052
28106	5216
28106	5259
29120	5001
29120	5041
29120	5049
29555	5022
25403	5022
29555	5001

Vorlesungen	
VorlNr	...
5001	...
4052	...
...	...



Zur Regel #2: Bennenung der Attribute?

- Spezifiziert das ER-Modell Rollen, dann
  - nimm den Namen der jeweiligen Rolle
- andernfalls
  - benutze die Namen der Schlüsselattribute der Entitäten
  - bei Mehrdeutigkeit erfinde aussagekräftigen Namen
- Beispiel: freund : {[ProfNr: integer, AssiNr: integer]}



Regel #3: Zusammenfassung von Relationen

Fasse nur Relationen mit gleichem Schlüssel zusammen (also auch nur (N:1), (1:N) oder (1:1) Beziehungen)

- Nach Regel #2:
  Vorlesungen : {[VorlNr, *Title, CP*]}
  Professoren: {[*PersNr, Name, Level, Room*]}
  lesen: {[VorlNr, *PersNr*]}
- Zusammenfassen nach Regel #3
  Vorlesungen : {[VorlNr, *Title, CP, PersNr*]}
  Professoren : {[*PersNr, Name, Level, Room*]}

Ausprägung von *Professoren* und Vorlesungen

Professoren			
PersNr	Name	Rang	Raum
2125	Sokrates	FP	226
2126	Russel	FP	232
2127	Kopernikus	AP	310
2133	Popper	AP	52
2134	Augustinus	AP	309
2136	Curie	FP	36
2137	Kant	FP	7

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

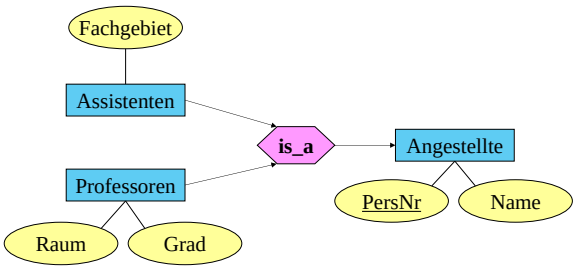
Das funktioniert NICHT

Professoren				
PersNr	Name	Rang	Raum	liest
2125	Sokrates	FP	226	5041
2125	Sokrates	FP	226	5049
2125	Sokrates	FP	226	4052
...	...	...	...	...
2134				
2136				

Vorlesungen		
VorlNr	Titel	KP
5001	Grundzüge	4
5041	Ethik	4
5043	Erkenntnistheorie	3
5049	Mäeutik	2
		4
		3
		2
		2
		2
		4

Problem: Redundanz und Anomalien
PersNr ist kein gültiger Schlüssel für Professoren mehr

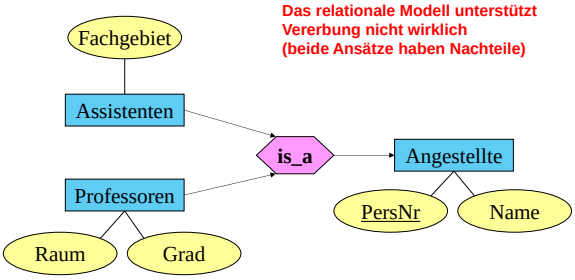
Regel #4: Generalisierung



Angestellte: {[*PersNr, Name*]}
Professoren: {[*PersNr, Grad, Raum*]}
Assistenten: {[*PersNr, Fachgebiet*]}

Regel #4: Generalisierung

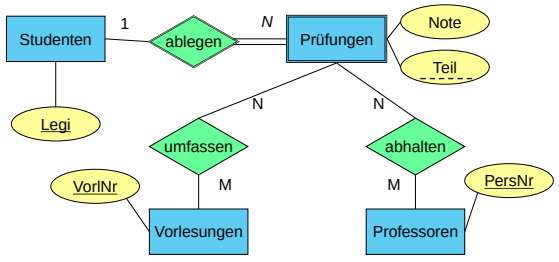
Das relationale Modell unterstützt Vererbung nicht wirklich (beide Ansätze haben Nachteile)



oder so?

Angestellte: {[*PersNr, Name*]}
Professoren: {[*PersNr, Name, Grad, Raum*]}
Assistenten: {[*PersNr, Name, Fachgebiet*]}

Regel #5: Schwache Entitäten



Prüfungen: {[Legi: integer, Teil: string, Grade: integer]}
umfassen: {[Legi: integer, Teil: string, VorlNr: integer]}
abhalten: {[Legi: integer, Teil: string, PersNr: integer]}

Schlüssel von schwachen Entitäten gebildet aus Schlüssel der starken Entität plus Schlüssel der schwachen Entität

Relationales Modell der Uni-DB

Professoren			
PersNr	Name	Rang	Raum
2125	Sokrates	FP	226
2126	Russel	FP	232
2127	Kopernikus	AP	310
2133	Popper	AP	52
2134	Augustinus	AP	309
2136	Curie	FP	36
2137	Kant	FP	7

Studenten		
Legi	Name	Semester
24002	Xenokrates	18
25403	Jonas	12
26120	Fichte	10
26830	Aristoxenos	8
27550	Schopenhauer	6
28106	Carnap	3
29120	Theophrastos	2
29555	Feuerbach	2

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

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

hören	
Legi	VorlNr
26120	5001
27550	5001
27550	4052
28106	5041
28106	5052
28106	5216
28106	5259
29120	5001
29120	5041
29120	5049
29555	5022

prüfen			
Legi	Nr	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

Die relationale Algebra

- σ Selektion
- π Projektion
- x kartesisches Produkt
- ⋈ Join (Verbund)
- ρ Umbenennung

- ∪ Vereinigung
- − Mengendifferenz
- ÷ Division
- ∩ Durchschnitt
- ⋉ Semi-Join (links)
- ⋊ Semi-Join (rechts)
- ⟕ linker äusserer Join
- ⟖ rechter äusserer Join

werden wir nicht diskutieren

Selektion und Projektion

- **Selektion σ**
  Auswahl von Tupeln (Zeilen) der Relation (Tabelle), die das Selektionsprädikat erfüllen

$\sigma_{\text{Semester} > 10}$ (Studenten)

Legi	Name	Semester
24002	Xenokrates	18
25403	Jonas	12

- **Projektion π**
  Extraktion von Attributen (Spalten) der Relation (Tabelle)

π_{Rang}(Professoren)

Level
FP
AP

Kartesisches Produkt

- **Kartesisches Produkt x**
  enthält alle $|L| \cdot |R|$ mögliche Paare von Tupeln aus L und R

L		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

x

R	
D	E
d_1	e_1
d_2	e_2

=

L x R				
L			R	
A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_1	b_1	c_1	d_2	e_2
a_2	b_2	c_2	d_1	e_1
a_2	b_2	c_2	d_2	e_2

Kartesisches Produkt

Professoren x hören					
Professoren				hören	
PersNr	Name	Rang	Raum	Legi	Nr
2125	Sokrates	FP	226	26120	5001
...	...	...	...	...	...
2125	Sokrates	FP	226	29555	5001
...	...	...	...	...	...
2137	Kant	FP	7	29555	5001

Riesiges Ergebnis (|Professoren| * |hören|)

Wird meist in Verbindung mit Selektion eingesetzt
→ Vermeidung riesiger Zwischenergebisse durch Einführung eines separaten Operators (Join)

Umbenennung

Umbenennung von Relationen ρ

- Beispiel: Ermittlung indirekter Vorgänger 2. Stufe der Vorlesung 5216

$\Pi_{\text{V1. Vorgänger}}$(
$\sigma_{\text{V2. Nachfolger=5216} \wedge \text{V1.Nachfolger = V2.Vorgänger}}$
(ρ_{V1} (voraussetzen) x ρ_{V2} (voraussetzen)))

Umbenennung von Attributen ρ

$\rho_{\text{Voraussetzung} \leftarrow \text{Vorgänger}}$ (voraussetzen)

Der natürliche Verbund (Join)

Gegeben seien:

- $R(A_1,..., A_m, B_1,..., B_k)$
- $S(B_1,..., B_k, C_1,..., C_n)$

Dann ist der Join ⋈ definiert als

$$R \bowtie S = \Pi_{A_1,...,A_m, R.B_1,... ,R.B_k, C_1,...,C_n} (\sigma_{R.B_1=S.B_1 \wedge ... \wedge R.B_k=S.B_k} (R \times S))$$

R ⋈ S											
R – S				R ∩ S				S – R			
A_1	A_2	...	A_m	B_1	B_2	...	B_k	C_1	C_2	...	C_n
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Drei-Wege-Join

(Studenten ⋈ hören) ⋈ Vorlesungen						
Legi	Name	Semester	VorlNr	Titel	KP	gelesenVon
26120	Fichte	10	5001	Grundzüge	4	2137
27550	Jonas	12	5022	Glaube und Wissen	2	2134
28106	Carnap	3	4052	Wissenschftstheorie	3	2126
...	...	...	...	...	...	...

NB: der Join-Operator ist assoziativ und kommutativ

Allgemeiner Join (Theta-Join)

Gegeben seien folgende Relationen(-Schemata)

- R(A1, ..., An) und
- S(B1, ..., Bm)

$$R \bowtie_\theta S = \sigma_\theta (R \times S)$$

$R \bowtie_\theta S$							
R				S			
A_1	A_2	...	A_n	B_1	B_2	...	B_m
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Beispiel: Allgemeiner Join

Relationen

- Züge(name, start, ziel, …, länge)
- Gleise(station, nummer, …, länge)

- Finde alle möglichen Gleise für den „CIS Alpino“ in Zürich

$\sigma_{station=„Zürich“}$ (Gleise)

$\bowtie$ Züge.länge < Gleise.länge

$\sigma_{name=„CIS“}$ (Züge)

Datendefinitions-, Manipulations- und Anfrage-Sprache SQL, Datendefinition, Veränderung am Datenbestand, Einfache SQL Abfrage, Anfragen über mehrere Relationen, Mengenfunktionen, Aggregatfunktion und Gruppierung, Geschachtelte Anfragen, Nullwerte, Syntactic Sugar

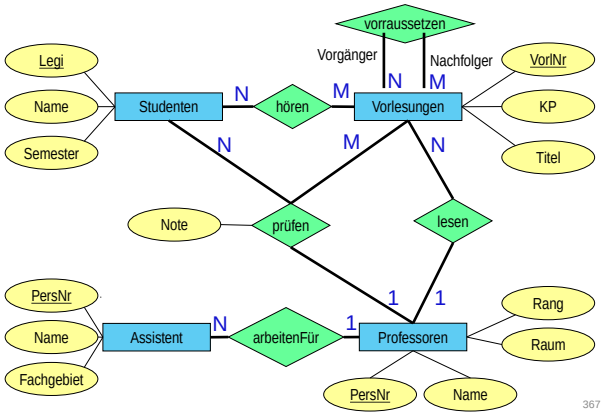
11. DATENBANKSYSTEME: SQL

SQL (Structured Query Language)

- Nachfolger von
  Sequel = Structured English Query Language

- Familie von Standards
  - Datendefinitionssprache (DDL) – Schemas
  - Datenmanipulationssprache (DML) – Updates
  - Anfrage- (Query)-Sprache – Anfragen

Wdh. Uni Schema



Wdh. Relationales Modell der Uni-DB

Professoren

PersNr	Name	Rang	Raum
2125	Sokrates	FP	226
2126	Russel	FP	232
2127	Kopernikus	AP	310
2133	Popper	AP	52
2134	Augustinus	AP	309
2136	Curie	FP	36
2137	Kant	FP	7

Studenten

Legi	Name	Semester
24002	Xenokrates	18
25403	Jonas	12
26120	Fichte	10
26830	Aristoxenos	8
27550	Schopenhauer	6
28106	Carnap	3
29120	Theophrastos	2
29555	Feuerbach	2

Vorlesungen

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

voraussetzen

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

hören

Legi	VorlNr
26120	5001
27550	5001
27550	4052
28106	5041
28106	5052
28106	5216
28106	5259
29120	5001
29120	5041
29120	5049
29555	5022

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

prüfen

Legi	Nr	PersNr	Note
28106	5001	2126	1
25403	5041	2125	2
27550	4630	2137	2

Datendefinition (DDL) 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

DDL (Fortsetzung)

Anlegen einer Tabelle

- create table Professoren
  (PersNr integer not null,
  Name varchar (10) not null
  Rang character (2));

Tabelle löschen

- drop table Professoren;

Tabellenstruktur anpassen

- alter table Professoren add column Raum integer;
- alter table Professoren modify column Name varchar(30);

Veränderung am Datenbestand

Einfügen von Tupeln

insert into Studenten (Legi, Name)
values (28121, `Archimedes‘);

Studenten

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

Null-Wert

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

Veränderungen am Datenbestand

Löschen von Tupeln

```
delete Studenten
   where Semester > 100;
```

Verändern von Tupeln

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


Einfache SQL-Anfrage

```
select   PersNr, Name
from     Professoren
where    Rang= ´FP´;
```

Projektion (ohne Duplikatelimination)

Selektion

PersNr	Name
2125	Sokrates
2126	Russel
2136	Curie
2137	Kant

Einfache SQL-Anfrage: Sortieren

```
select  PersNr, Name, Rang
from    Professoren
order by Rang desc, Name asc;
```

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

Duplikateliminierung

```
select distinct Rang
from Professoren
```

Projektion

Rang
C3
C4

Anfragen über mehrere Relationen

Welcher Professor liest "Mäeutik"?

```
select   Name,  Titel
from     Professoren , Vorlesungen
where    PersNr  =  gelesenVon and  Titel = `Mäeutik‘ ;
⇔
select   Name
from     Professoren join Vorlesungen on PersNr = gelesenVon
where    Titel = `Mäeutik‘ ;
```

Projektion

Kreuzprodukt

Selektion

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

Anfragen über mehrere Relationen

Professoren

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

Vorlesungen

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

Verknüpfung
X

Anfragen über mehrere Relationen (Fortsetzung)

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

↓ Projektion

Name	Titel
Sokrates	Mäeutik

Anfragen über mehrere Relationen

Welche Studenten hören welche Vorlesungen?

```
select Name, Titel
from Studenten, hören, Vorlesungen
where Studenten.Legi = hören.Legi and
    hören.VorlNr = Vorlesungen.VorlNr;
```

Alternative:

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

Umbenennung von Attributen

Titel und Professor aller Vorlesungen

```
select Titel, gelesenVon as PersNr
from Vorlesungen;
```


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.Legi =
   s1.Legi and h2.Legi = s2.Legi
```


Mengenoperationen

Mengenoperationen **union, intersect, minus**

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

union, intersect und minus funktionieren nur auf Relationen mit gleichem Schema

Aggregatfunktion und Gruppierung

Aggregatfunktionen avg, max, min, count, sum

```
select avg(Semester)
from Studenten;

select v.gelesenVon, sum(v.KP)
from Vorlesungen v
group by v.gelesenVon;

select v.gelesenVon, p.Name, sum(v.KP)
from Vorlesungen v, Professoren p
where v.gelesenVon = p.PersNr and v.Rang = 'FP'
group by v.gelesenVon, p.Name
   having avg(v.KP) >= 3;
```

SQL weiss nicht, dass sich der Name innerhalb der Gruppe nicht ändern kann, wenn nur gelesenVon angegeben ist

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

Abarbeitung der Anfrage in SQL

1. Schritt
   from Vorlesungen, Professoren
   where gelesenVon = PersNr and Rang = ´FP´
2. Schritt
   group by gelesenVon, Name
3. Schritt
   having avg (KP) >= 3
4. Schritt
   select gelesenVon, Name, sum (KP)

Ausführen einer Anfrage mit group by

Vorlesung x Professoren

VorlNr	Titel	KP	gelesen Von	PersNr	Name	Rang	Raum
5001	Grundzüge	4	2137	2125	Sokrates	FP	226
5041	Ethik	4	2125	2125	Sokrates	FP	226
...	...	...	...	...	...	...	...
4630	Die 3 Kritiken	4	2137	2137	Kant	FP	7

↓ **where**-Bedingung

nach Selektion

VorlNr	Titel	KP	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

nach Gruppierung

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

↓ having-Bedingung

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

↓ Aggregation (sum) und Projektion

Ergebnis

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

Geschachtelte Anfragen

Existenzquantor exists

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

Korrelation

Mengenvergleich

Unkorrelierte Unteranfrage: meist effizienter, wird nur einmal ausgewertet

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


Der Vergleich mit "all"

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

(Kein vollwertiger Allquantor)

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


Korrelierte versus unkorrelierte 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

Nullwerte (NULL = UNKNOWN)

```
select count (*)
from Student
where Semester < 13 or Semester > =13;
```

vs.

```
select count (*)
from Student;
```

Sind diese Anfragen äquivalent?

Arbeiten mit NULL Werten

1. Arithmetik: null wird propagiert: Ist ein Operand null, dann ist das Resultat null
   - null + 1 -> null
   - null * 0 -> null
2. Vergleiche: Neuer Boolscher Wert unknown. Alle Vergleiche mit null-Werten evaluieren zu unknown.
   - null = null -> unknown
   - null < 13 -> unknown
   - null > null -> unknown
3. Logische Operationen: Boolsche Ausdrücke evaluieren "nach gesundem Menschenverstand" wie folgt

Dreiwertige Logik

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

Arbeiten mit NULL Werten (Forts.)

4. 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.
5. Bei einer Gruppierung wird null als ein eigenständiger Wert aufgefaßt und in eine eigene Gruppe eingeordnet.

Syntactic Sugar

```
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%‘;
```


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;
```

Wie switch in Java oder C/C++: erste qualifizierende when-Klausel wird ausgeführt