SQL (Structured Query Language)

- SQL (hier meist f. Oracle bzw. MySQL)
  - DDL (Data Definition Language)
    - CREATE,ALTER,DROP, Datentypen
  - DML (Data Manipulation Language)
    - SELECT, INSERT UPDATE, DELETE
  - DCL (Data Control Language)
    - GRANT, REVOKE
  - Werkzeuge:
    - TOAD (for MySQL), http://localhost/phpmyadmin, MySql Query Browser
  - Dateien:
    - demo/sql-is_uni/sql-kurs1-oracle.sql (funktioniert auch f. Mysql)
    - demo/sql-is_uni/sql-kurs2-mysql-trigger-views-procs.sql

SQL (http://de.wikipedia.org/wiki/SQL)

- SQL („Structured Query Language“)
  - von IBM in den 1970er Jahren auf der Grundlage des bahnbrechenden Artikels „A Relational Model of Data for Large Shared Data Banks“ (1970) von Edgar F. Codd entworfen.
  - 1986 erster SQL-Standard (ANSI, ISO)
  - 1992 wurde der Standard deutlich überarbeitet und als SQL-92 (oder auch **SQL2**) veröffentlicht.
    - Alle aktuellen Datenbanksysteme halten sich im wesentlichen an diese Standardversion.
  - Die neuere Version SQL:1999 (ISO/IEC 9075:1999, auch **SQL3** genannt) ist noch nicht in allen Datenbanksystemen implementiert. SQL:2003 ist noch weitgehend unimplementiert.

Befehlsübersicht

Datenmanipulation (DML):

SELECT
INSERT
UPDATE
DELETE
Aggregatfunktionen: COUNT, SUM, AVG, MAX,MIN

Datendefinition (DDL):

CREATE SCHEMA
CREATE DOMAIN
CREATE TABLE
CREATE VIEW
ALTER TABLE
DROP SCHEMA
DROP DOMAIN
DROP TABLE
DROP VIEW

Datenkontrolle:

Constraints-Definitionen bei CREATE TABLE
CREATE ASSERTION
DROP ASSERTION
GRANT
REVOKE
COMMIT
ROLLBACK

Eingebettetes SQL:

DECLARE CURSOR
FETCH
OPEN CURSOR
CLOSE CURSOR
SET CONSTRAINTS
SET TRANSACTION
CREATE TEMPORARY TABLE

Datentypen SQL

character(*n*),char(n)	String fester Länge mit *n* Zeichen
varchar2(*n*),varchar(n)	String variabler Länge mit bis zu *n* Zeichen
integer	ganze Zahl
number(*n*,*m*),decimal(n,m)	Festkommazahl mit *n* Stellen, (*m* nach dem Komma
float(m)	Gleitkommazahlen
date	Zeitangabe
long	Zeichenkette bis zu 2 GByte
raw	Binärstrings bis zu 255 Bytes
long raw	Binärobjekte bis zu 2 GByte (Soundfiles, Videos, ...)

- Nicht besetzte Attributwerte werden durch das Schlüsselwort `NULL` gekennzeichnet.
- Die einzelnen DBMS-Hersteller haben diese Liste jedoch um eine Unzahl weiterer Datentypen erweitert, woraus die nützlichsten wohl TEXT und BLOB sein dürften

DDL: CREATE, ALTER TABLE

1. Tabelle anlegen:

```
create table is_professoren (
  PersNr  integer       not null,
  Name    varchar2(10)  not null,
  Rang    character(2) );
```

2. Tabelle erweitern:

```
alter table is_professoren add (Raum integer);
```

3. Tabelle ändern:

```
alter table is_professoren
modify (Name varchar2(30));
```

4. Tabelle verkürzen (nicht in Oracle):

```
alter table is_professoren
drop column Gebdatum;
```

DDL: CREATE INDEX

- Beschleunigung von Anfragen

- Syntax:
  - CREATE [bitmap][unique] INDEX *index* ON *table* (*column*,....)
  - bitmap: wenn kaum Unterschiede

- Beispiel:

```
SELECT * FROM STRAFEN WHERE BETRAG=25;
CREATE INDEX STRAFE_IDX ON STRAFEN (BETRAG);
```

- Ein Index kann jederzeit erstellt/gelöscht werden.
- Bei Insert/update/delete wird der Index automatisch angepasst.
- D.h. das Insert/update/delete wird langsamer.

DDL: DROP von Objekten

```
DROP { TABLE base-table | VIEW view  | DOMAIN domain | SCHEMA schema }
            {RESTRICT | CASCADE}
```

- Falls Objekte (Relationen, Sichten, ...) nicht mehr benötigt werden, können sie durch die DROP-Anweisung aus dem System entfernt werden
- Mit der CASCADE-Option können 'abhängige' Objekte (z.B. Sichten auf Relationen oder anderen Sichten) mitentfernt werden
- RESTRICT verhindert Löschen, wenn die zu löschende Relation noch durch Sichten oder Integritätsbedingungen referenziert wird
- Beispiele:

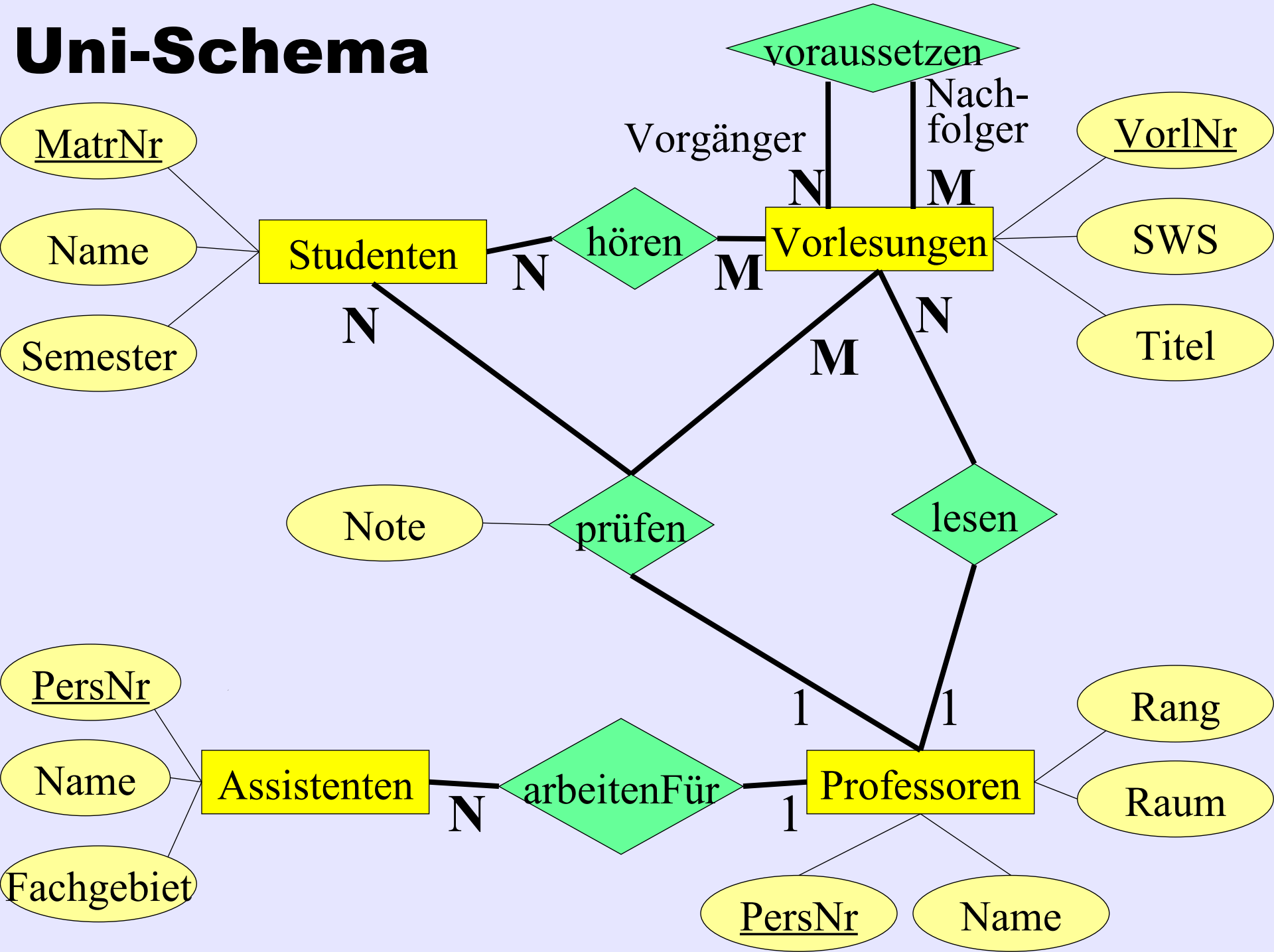
```
DROP   TABLE   PERS RESTRICT
```

PersConstraint sei definiert auf PERS:

```
1. ALTER TABLE  DROP CONSTRAINT  PersConstraint  CASCADE

2. DROP TABLE  PERS  RESTRICT
```

Uni-Schema


voraussetzen
Vorgänger
Nach-
folger
N
M
MatrNr
Name
Semester
Studenten
N
hören
M
Vorlesungen
VorlNr
SWS
Titel
N
M
N
Note
prüfen
lesen
1
1
PersNr
Name
Fachgebiet
Assistenten
N
arbeitenFür
1
Professoren
Rang
Raum
PersNr
Name

Professoren

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

Studenten

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

Vorlesungen

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

voraussetzen

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

hören

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

prüfen

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

Assistenten

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

DML: INSERT, UPDATE, DELETE

- Füge neue Vorlesung mit einigen Angaben ein:

```
insert into is_vorlesungen (VorlNr, Titel, gelesenVon)
values (4711,'Selber Atmen', 2125)
```

- Schicke alle Studenten in die Vorlesung *Selber Atmen*:

```
insert into is_hoeren
select MatrNr, VorlNr
from   is_studenten, is_vorlesungen
where  Titel = 'Selber Atmen'
```

DML: INSERT, UPDATE, DELETE

- Erweitere die neue Vorlesung um ihre Semesterwochenstundenzahl:

```
update is_vorlesungen
set    SWS=6
where  Titel='Selber Atmen'
```

- Entferne alle Studenten aus der Vorlesung *Selber Atmen:*

```
delete from is_hoeren
where vorlnr =
      (select VorlNr from is_vorlesungen
       where  Titel = 'Selber Atmen')
```

- Entferne die Vorlesung *Selber Atmen*:

```
delete from is_vorlesungen
where titel = 'Selber Atmen'
```

DML: SELECT

- Die SELECT-Anweisung startet eine Abfrage.
  - Aufgrund der Syntax kann eine SELECT-Anweisung auch als "**SFW-Block**" (SELECT, FROM, WHERE) bezeichnet werden.

- Syntax (unvollständig):

```
SELECT [DISTINCT] Auswahlliste
FROM Quelle
WHERE Where-Klausel
[GROUP BY (Group-by-Attribut)+
[HAVING Having-Klausel]]
[ORDER BY (Sortierungsattribut)+ [ASC|DESC]]
```

DML: SELECT

- **SELECT** selektiert Spalten
- **FROM** definiert die Ausgangstabellen
- **WHERE** selektiert die Reihen, die der Bedingung genügen

  ```
  AND OR NOT
  = > < != <= >=
  IS NULL , BETWEEN, IN, LIKE
  ```

- **GROUP BY** gruppiert Reihen auf der Basis gleicher Werte in Spalten
- **HAVING** selektiert Gruppen, die der Bedingung genügen

- **ORDER BY** sortiert Reihen auf der Basis von Spalten

Einfaches SELECT

```
select    PersNr, Name
from      is_professoren
where     Rang= ´C4´;
```

PersNr	Name
2125	Sokrates
2126	Russel
2136	Curie
2137	Kant

Einfaches SELECT

Sortierung

```
select PersNr, Name, Rang
from   is_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

Einfaches SELECT

Duplikateliminierung

```
select distinct Rang
from is_professoren
```

Rang
C3
C4

Einfaches SELECT

Zähle alle Studenten:

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

Liste Namen und Studiendauer in Jahren von allen Studenten, die eine Semesterangabe haben:

```
select Name, Semester/2 AS Studienjahr
from   is_Studenten
where  Semester is not null;
```

Einfaches SELECT

Liste alle Studenten mit Semesterzahlen zwischen 1 und 4:

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

select *
  from is_Studenten
  where Semester between 1 and 4;

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

Einfaches SELECT

Liste alle Vorlesungen, die im Titel den String Ethik enthalten, klein oder groß geschrieben:

```
select *
  from is_vorlesungen
  where upper(Titel) like '%ETHIK%';
```

Liste Personalnummer, Name und Rang aller Professoren, absteigend sortiert nach Rang, innerhalb des Rangs aufsteigend sortiert nach Name:

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

Einfaches SELECT

Liste alle Geburtstage mit ausgeschriebenem Monatsnamen:

```
select
to_char(GebDatum,'month DD, YYYY') AS Geburtstag
from is_studenten;
```

Liste das Alter der Studenten in Jahren:

```
select (sysdate - GebDatum) / 365 as Alter_in_Jahren
from is_studenten;
```

Liste die Wochentage der Geburtsdaten der Studenten:

```
select to_char(GebDatum,'day')
from is_studenten;
```

JOINS (Anfragen über mehrere Tabellen)

Welcher Professor liest Mäeutik?

```
select Name, Titel
from   is_Professoren, is_Vorlesungen
where  is_Professoren.PersNr =
       is_Vorlesungen.gelesenVon
       and
       Titel = 'Mäeutik';
```

- Angabe der beteiligten Tabellen in FROM-Klausel

- WHERE-Klausel enthält Join-Bedingung sowie weitere Selektionsbedingungen
- Formulierung der Join-Bedingung erfordert bei gleichnamigen Attributen Hinzunahme der Relationennamen oder von Alias-Namen

JOINS (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	gelesenVon
5001	Grundzüge	4	2137
5041	Ethik	4	2125
⋮	⋮	⋮	⋮
5049	Mäeutik	2	2125
⋮	⋮	⋮	⋮
4630	Die 3 Kritiken	4	2137

↘ Verknüpfung ↙

PersNr	Name	Rang	Raum	VorlNr	Titel	SWS	gelesenVon
2125	Sokrates	C4	226	5001	Grundzüge	4	2137
2125	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

75

↓ Auswahl

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

↓ Projektion

Name	Titel
Sokrates	Mäeutik

JOINS (Anfragen über mehrere Relationen)

Welche Studenten hören welche Vorlesungen?

```
select Name, Titel
from   is_Studenten, is_hoeren, is_Vorlesungen
where  is_hoeren.MatrNr = is_Studenten.MatrNr
       and
       is_hoeren.VorlNr = is_Vorlesungen.VorlNr;
```

Alternativ:

```
select  s.Name, v.Titel
from    is_Studenten s, is_hoeren h, is_Vorlesungen v
where   h.MatrNr = s.MatrNr
        and
        h.VorlNr = v.VorlNr;
```

Anm: Wenn Alias-Namen verwendet werden, kann der Real-Name der Tabelle nicht mehr verwendet werden.

JOINS (Anfragen über mehrere Relationen)

- Hierarchische Beziehung auf einer Relation (PERS)
  - Tabelle:
    - PERS (PNR, NAME, BERUF, GEHALT, ..., *MNR*, *ANR*)
    - PRIMARY KEY (PNR),
    - FOREIGN KEY (*MNR*) REFERENCES PERS (PNR)
- Finde die Angestellten, die mehr als ihre (direkten) Manager verdienen (Ausgabe: NAME,GEHALT, NAME des Managers)

```
SELECT p.name, p.gehalt, m.name as 'Managername'
FROM   pers p, pers m
WHERE  p.mnr = m.pnr
AND    p.gehalt > m.gehalt;
```

- Verwendung von Korrelationsnamen obligatorisch!

CROSS Join

- **cross join:**
  - Berechnet das Kreuzprodukt
  - Ist die Basis aller Join-Arten
  - Auch karthesisches Produkt od. Obermenge genannt.
  - Jeder Datensatz aus R1 wird mit allen Datensätzen aus R2 verknüpft.

  - Select * from R1, R2

    Oder
  - Select * from R1 cross join R2

INNER Join

- Auswahl der Tupel des karthesischen Produkts, die **gleiche Werte** in einem Attribut besitzen.

  - ```
    SELECT a.name as Assi, p.name as Prof
      FROM  is_assistenten a, is_professoren p
      WHERE a.boss = p.persnr
    ```
    entspricht

  - ```
    SELECT a.name as Assi, p.name as Prof
      FROM  is_assistenten a INNER JOIN is_professoren p
      ON    a.boss = p.persnr
    ```

- Es gilt:
- Duplikate sind hier möglich
  - Abhilfe: **SELECT DISTINCT**
- Attribute können durch Voranstellung des Relationennamens eindeutig identifiziert werden

OUTER Join

- `SELECT * FROM R1 LEFT OUTER JOIN R2 ON R1.attr2 = R2.attr1`
- `SELECT * FROM R1 RIGHT OUTER JOIN R2 ON R1.attr2 = R2.attr1`
- `SELECT * FROM R1 FULL OUTER JOIN R2 ON R1.attr2 = R2.attr1.`

- Ergebnismenge:
  - wie beim inner join **und** zusätzlich werden Tupel ausgewählt, die keinen entsprechenden Wert in der jeweils anderen Tabelle besitzen.
  - Diese fehlenden Werte werden in der Ergebnismenge mit NULL aufgefüllt.
  - Das bedeutet, dass beim outer join in der Regel mehr Tupel in der Ergebnismenge enthalten sind.

OUTER Join

- Es werden alle Professoren mit ihren Vorlesungen angezeigt.

```
select * from is_professoren p INNER JOIN
  is_vorlesungen v on p.persnr=v.gelesenvon
```

- Es werden alle Professoren mit ihren Vorlesungen angezeigt und Professoren, die keine Vorlesungen halten

```
select * from is_professoren p LEFT OUTER JOIN
  is_vorlesungen v on p.persnr=v.gelesenvon
```

+LEFT OUTER Join

```
select  p.PersNr, p.Name,     f.PersNr, f.Note, f.MatrNr,  s.MatrNr, s.Name
from    is_professoren p left outer join
    (is_pruefen f left outer join is_studenten s on f.MatrNr= s.MatrNr)
    on p.PersNr=f.PersNr;
```

Es werden auch alle Professoren angezeigt, die keine Prüfungen abgehalten haben.

Die anderen Spalten werden mit NULL aufgefüllt.

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

+RIGHT OUTER Join

```
select p.PersNr, p.Name, f.PersNr, f.Note, f.MatrNr, s.MatrNr, s.Name
from  is_professoren p right outer join
        (is_pruefen f right outer join is_studenten s on f.MatrNr= s.MatrNr)
       on p.PersNr=f.PersNr;
```

Es werden die Studenten angezeigt, die keine Prüfungen absolviert haben

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

+FULL OUTER Join

```
select p.PersNr, p.Name, f.PersNr, f.Note, f.MatrNr, s.MatrNr, s.Name
from is_professoren p full outer join
     (is_pruefen f full outer join is_studenten s on f.MatrNr= s.MatrNr)
     on p.PersNr=f.PersNr;
```

Es wird alles angezeigt!

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	Schopenhauer
-	-	-	-	-	26120	Fichte
2136	Curie	-	-	-	-	-

JOIN Beispiele

- Liste die Namen der Studenten mit ihren Vorlesungstiteln:

```
select  Name, Titel
from    is_studenten, is_hoeren, is_vorlesungen
where   is_hoeren.MatrNr = is_studenten.MatrNr
and     is_hoeren.VorlNr = is_vorlesungen.VorlNr
```

- alternativ:

```
select   s.Name, v.Titel
from     is_studenten s, is_hoeren h, is_vorlesungen v
where    h.MatrNr = s.MatrNr
and      h.VorlNr = v.VorlNr
```

- Liste die Namen der Assistenten, die für denselben Professor arbeiten, für den Aristoteles arbeitet:

```
select  a2.Name
from    is_assistenten a1, is_assistenten a2
where   a2.boss  =  a1.boss
and     a1.name  = 'Aristoteles'
and     a2.name != 'Aristoteles'
```

GROUP BY - Aggregatfunktionen

- Ermittle die durchschnittliche Studendauer

```
select  AVG(Semester)
from is_studenten
```

eingebaute Aggregationsfunktionen:

Name	Beschreibung
MIN	Minimum
MAX	Maximum
SUM	Summe
AVG	Durchschnitt
COUNT	Anzahl
STDDEV	Standardabweichung
VARIANCE	Varianz

```
select  COUNT (DISTINCT name)
from is_studenten
```

- Einsatz bei Datawarehouse-Anwendungen (Auswertungen)

GROUP BY

- Ermittle die gehaltenen SWS pro Professor

```
select      gelesenVon, SUM(SWS)
from        is_vorlesungen
GROUP BY    gelesenVon;
```

- Gib dazu auch den Namen des Professors an (d.h. ein JOIN kommt hinzu), und nur die Profs, die C4 als Rang haben und durchschnittlich mehr als 3 stündige Vlen lesen.

```
select      gelesenVon, Name, SUM(SWS)
from        is_vorlesungen, is_Professoren
where       gelesenVon = PersNr and Rang = 'C4'
GROUP BY    gelesenVon, Name
HAVING      AVG (SWS) > 3;
```

- 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 Attrbiut nicht
  - innerhalb der Gruppe ändert

GROUP BY (Ausführung einer Anfrage)

Vorlesungen × Professoren

VorlNr	Titel	SWS	gelesenVon	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	gelesenVon	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

GROUP BY (Ausführung einer Anfrage)

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	Wissenschaftstheorie	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

Mengen:
IN, NOT IN, ANY (= SOME), ALL, EXISTS

SQL	Bedeutung
A Θ ANY S	$\exists_{x \in S}$ A Θ S
A Θ ALL S	$\forall_{x \in S}$ A Θ S
A IN S	A = ANY S
A NOT IN S	A <> ALL S
EXISTS S	S <> $\emptyset$

wobei $\Theta \in \{=, <>, <, <=, >, >=\}$

Wer studiert am längsten?

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

Mengen:
IN, NOT IN, ANY (= SOME), ALL, EXISTS

Liste alle Professoren, die keine Vorlesung abhalten.

```
select Name
from   is_Professoren
where
PersNr NOT IN(select gelesenVon from is_vorlesungen );
```

Liste folgende Professoren

```
select Name
from   is_professoren
where PersNr IN (123,456,789);
```

Mengen: union, intersect, minus

```
( select Name
  from is_assistenten )
union
( select Name
  from is_professoren);
```

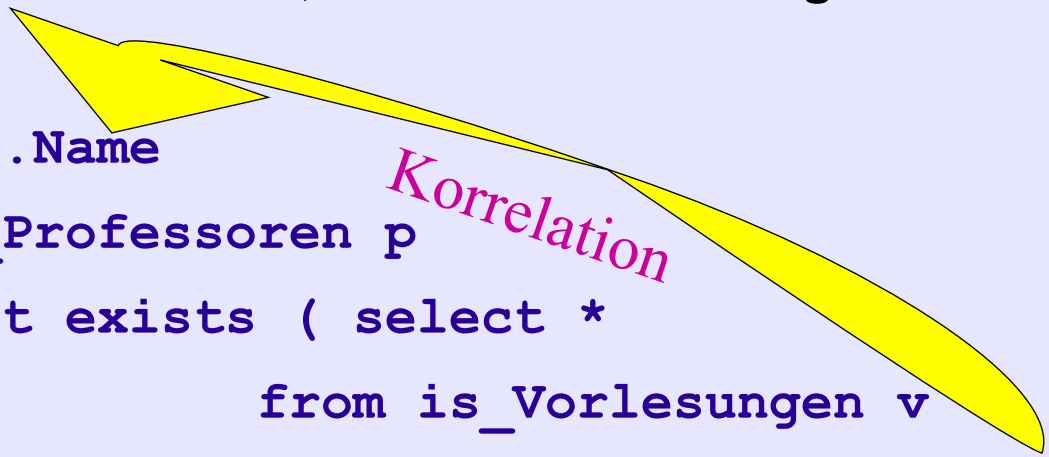
SUBQUERY

- Query-Block in der WHERE-Klausel
- Muss in Klammern geschrieben werden

- 2 Arten:
  - Korrelierte SubQuery
  - Unkorrelierte SubQuery

SUBQUERY: Korreliert

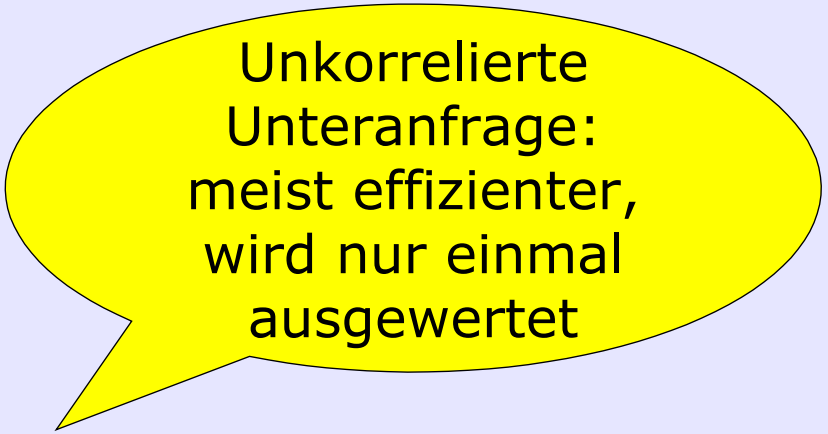
Liste alle Professoren, die keine Vorlesung halten.




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

- Koppelung der Tupelvariable p
- Die Subquery wird für jede Tupelvariable p ausgeführt !!

SUBQUERY unkorreliert mit Mengenvergleich




```
select Name
from is_Professoren
where PersNr not in (select gelesenVon
                     from is_Vorlesungen );
```

SUBQUERY: Operationen

- Subquery liefert **einzelnen Wert**:
  <, <=, ..., BETWEEN

- Subquery liefert **Menge**:
  IN, NOT IN, ANY (= SOME), ALL, EXISTS

+SUBQUERY: (Un)Korreliert

- Korrelierte Formulierung

```
select s.* from is_studenten s
where exists
( select p.* from is_professoren p
where s.GebDatum < p.GebDatum );
```

- Äquivalente, unkorrelierte Formulierung

```
select s.* from is_studenten s
where s.GebDatum <
( select max(p.GebDatum) from is_professoren p );
```

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

+SUBQUERY: Entschachtelung korrelierter Unteranfragen

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

Entschachtelung durch Join

```
select a.*
from is_assistenten a, is_professoren p
where a.Boss = p.PersNr and p.GebDatum > a.GebDatum;
```

+SUBQUERY: Verwertung der Ergebnismenge

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

Anm: Views würde man hier einsetzen. Siehe weiter unten VIEWS/Sichten

MatrNr	Name	VorlAnzahl
28106	Carnap	4
29120	Theophrastos	3

+SUBQUERY: Decision-Support Anfrage

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

(Oracle)

Frage: Was wird hier ermittelt?

+... mit Cast: integer->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 is_hoeren
          group by VorlNr ) h,
        ( select count (*) as GesamtAnz
          from is_Studenten) g;
```

Sichten/Views: Was ist das?

```
CREATE VIEW Lebensalter (SPIELERNR, LEBENSALTER) AS
select SPIELERNR, SYSDATE - GEB_DATUM
from   SPIELER;
```

```
select * from Lebensalter;
```

- Kann wie Tabelle verwendet werden;
- Daten werden dynamisch ermittelt.

Sichten/Views: Warum?

- Verwendung für den **Datenschutz**

```
create view PRUEFEN_SICHT as
    select MatrNr, VorlNr, PersNr
    from is_pruefen
```

Die Note wird nicht angezeigt!

```
Select * from PRUEFEN_SICHT
```

Sichten/Views: Warum?

für die Vereinfachung von Anfagen

```
create view STUD_PROF (Sname, Semester, Titel, Pname) as
    select s.Name, s.Semester, v.Titel, p.Name
    from   is_studenten s, is_hoeren h, is_vorlesungen v,
           is_professoren p
    where  s.Matr.Nr=h.MatrNr and h.VorlNr=v.VorlNr and
           v.gelesenVon = p.PersNr
```

Verwendung:

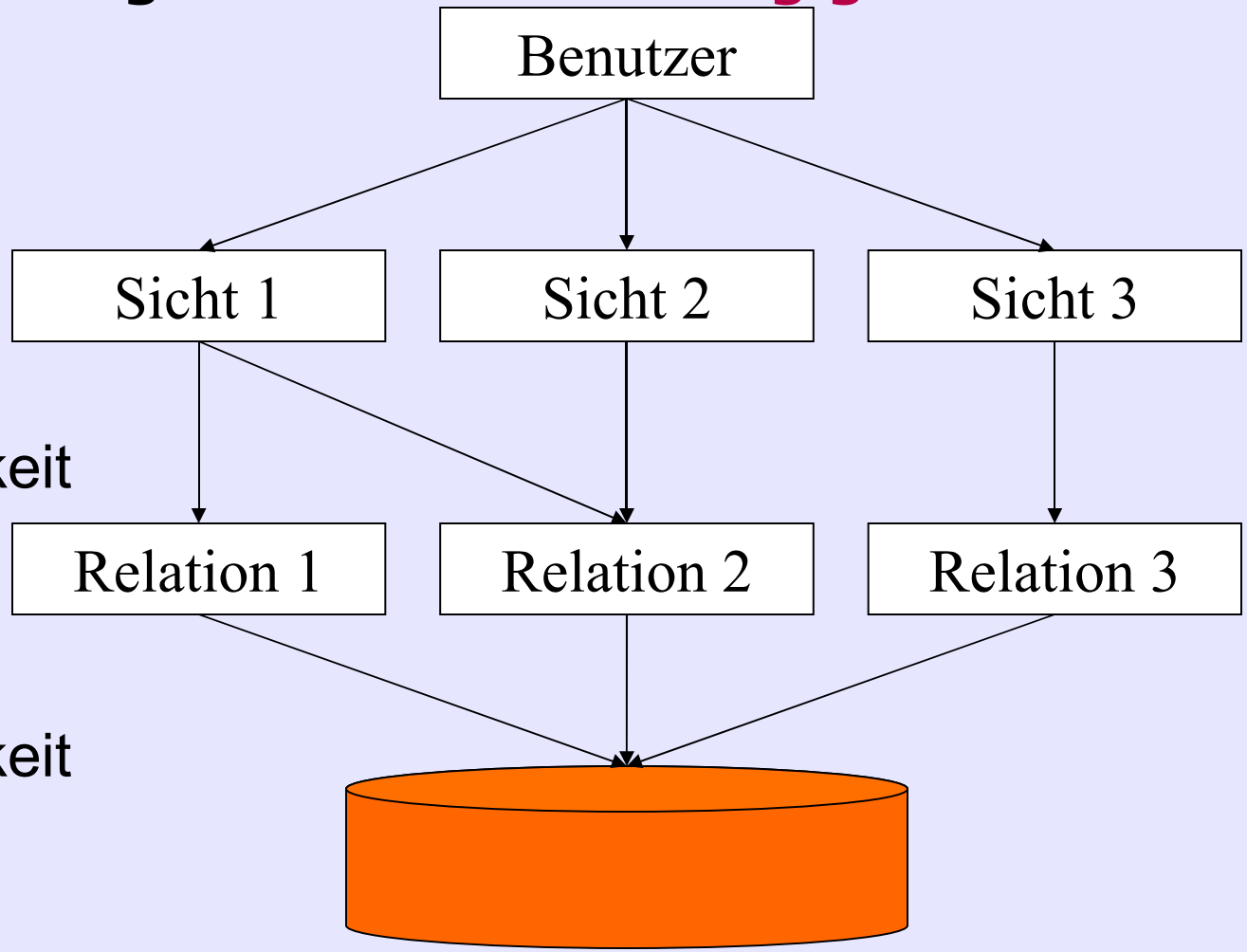
```
select distinct Semester
from STUD_PROF
where PName=`Sokrates';
```

Sichten/Views: Warum?

zur Gewährleistung von Datenunabhängigkeit

logische Datenunabhängigkeit

physische Datenunabhängigkeit



Sichten/Views: ...

Kann eigene Spalten haben:

```
create view PruefGuete(Name, GUETEGRAD) as
       (select prof.Name, avg(pruef.Note)
        from is_Professoren prof join
                is_pruefen pruef on
                prof.PersNr = pruef.PersNr
        group by prof.Name
        having count(*) > 50)
```

Frage: Welche Daten beinhaltet diese View?

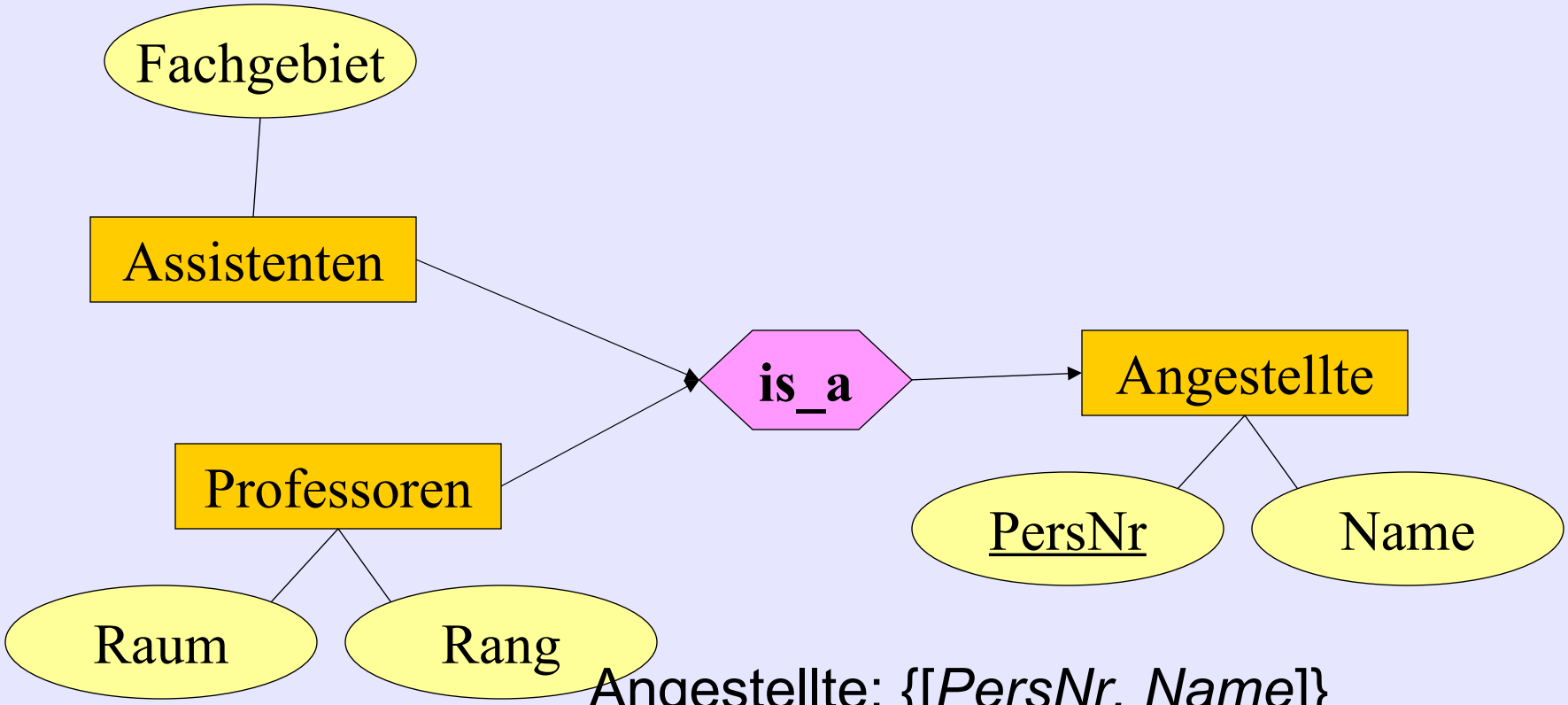
Verwendung der View:

```
select * from PruefGuete;
```

Frage: Werden alle Daten gelöscht?

```
drop view PruefGuete;
```

Relationale Modellierung der Generalisierung



Angestellte: {[*PersNr, Name*]}

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

Assistenten: {[*PersNr, Fachgebiet*]}

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);
```

Untertypen (Professoren, Assistenten) als Sicht

```
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;
```

Obertypen (Angestellte) als Sicht

```
create table Professoren
    (PersNr  integer not null,
    Name        varchar (30) not null,
    Rang        character (2),
    Raum        integer);

create table Assistenten
    (PersNr  integer not null,
    Name        varchar (30) not null,
    Fachgebiet  varchar (30),
    Boss        integer);

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

Obertypen (Angestellte) als Sicht

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

Änderbarkeit von Sichten

Beispiele für nicht änderbare Sichten

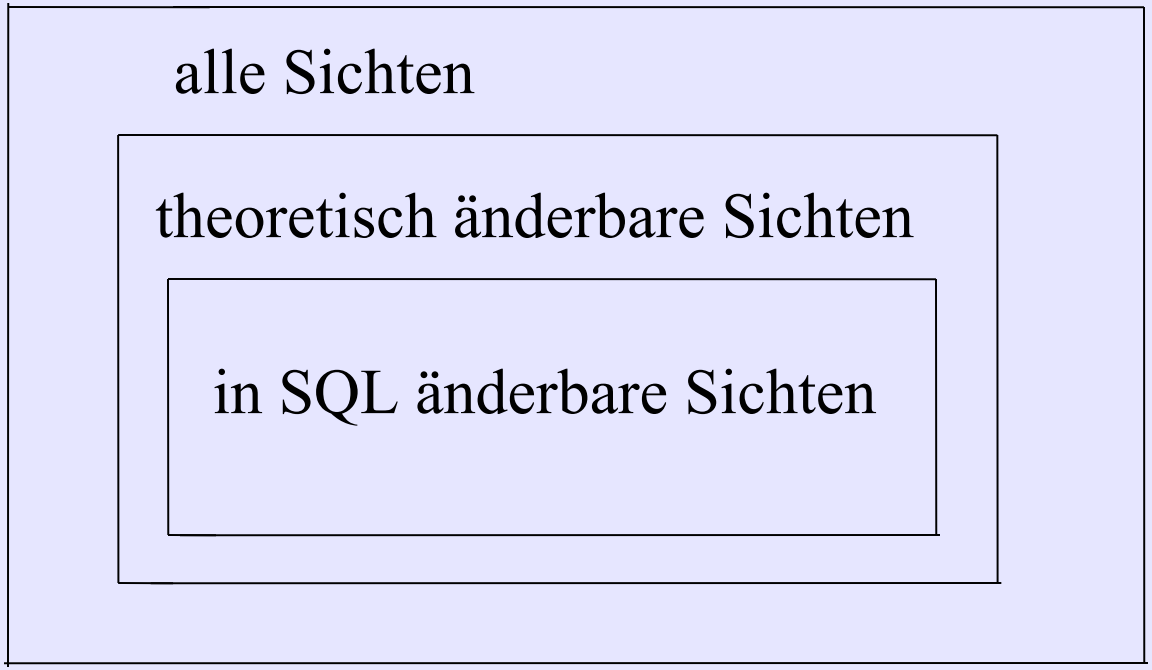
```
create view WieHartAlsPruefer (PersNr, Durchschnittsnote) as
    select PersNr, avg(Note)
    from is_pruefen
    group by PersNr;

create view VorlesungenSicht as
    select Titel, SWS, Name
    from is_Vorlesungen, is_Professoren
    where gelesenVon=PersNr;

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

Änderbarkeit von Sichten

- in SQL
  - nur eine Basisrelation
  - Schlüssel muß vorhanden sein
  - keine Aggregatfunktionen, Gruppierung und Duplikateliminierung



NULL Werte

- 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 is_studenten
where Semester < 13 or Semester > =13
```

[Demo: Professor: insert mit NULL Semester; Wiederhole select]

+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.

3. Logische Ausdrücke werden nach den folgenden Tabellen berechnet:

+Auswertung bei Null-Werten

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.

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.

+Auswertung bei Null-Werten

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

DCL: Grant, Revoke, ...

- Das Vergeben der Benutzerrechte wird mit **GRANT** realisiert.
- Das Entziehen von Benutzerrechten geschieht mit **REVOKE**.

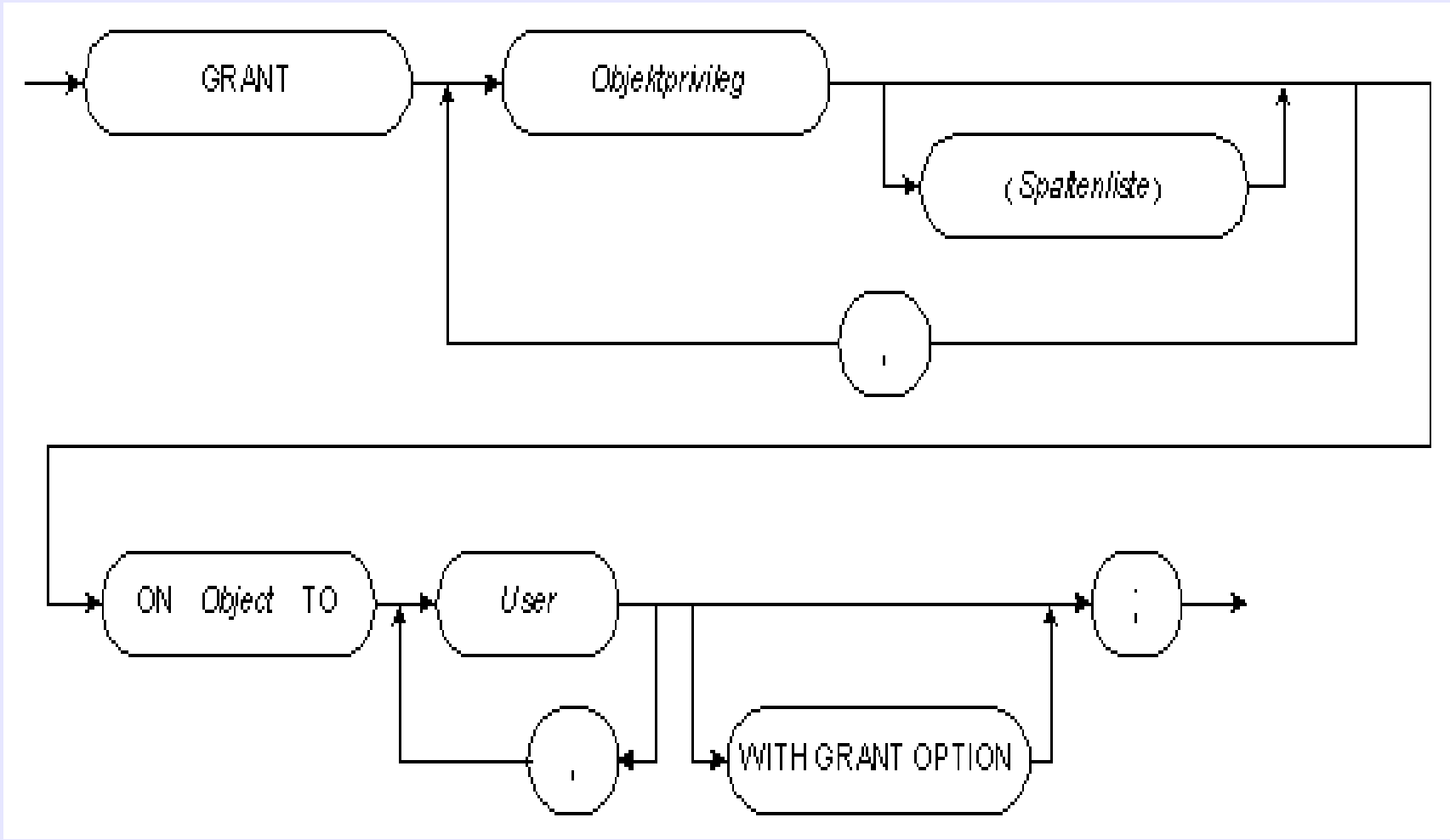
- Oracle unterstützt
  - Systemprivilegien
    - CREATE SESSION, CREATE TABLE, ...
  - Objektprivilegien
    - SELECT, INSERT, ...
  - Rollen
    - Enthalten eine Kombination von System- und Objektrechten.

» [Demo: Oracle, Rollenmanagement: TOAD User Verwaltung]

DCL: Grant, Revoke

- Für alle Implementierungen von GRANT und REVOKE gilt, dass sie festlegen:
  - welche **Privilegien** vergeben/entzogen werden,
  - welches **Objekt** betroffen ist,
  - wem die **Zugriffsrechte** zugeteilt/entzogen werden,
  - und optional, ob Verwaltungsrechte oder nur Zugriffsrechte erteilt oder entzogen werden.
  - Wir zeigen Ihnen hier die vereinfachten Syntaxdiagramme für GRANT und REVOKE.

DCL: Grant, Revoke



DCL: Grant, Revoke

- Hiermit dürfen die drei Benutzer "joerg", "sabine" und "harald" die drei Befehle INSERT, SELECT und DELETE auf die Tabelle "mitarbeiter" anwenden. Sie dürfen außerdem diese Rechte an andere weitervergeben.

```
GRANT INSERT, SELECT, DELETE
ON    mitarbeiter
TO    joerg, sabine, harald
WITH GRANT OPTION;
```

- Es geht auch noch genauer:

```
GRANT UPDATE (gehalt, position)
ON mitarbeiter
TO joerg;
```

DCL: Grant, Revoke

```
REVOKE INSERT, SELECT, DELETE
ON mitarbeiter
FROM joerg, sabine, harald;
```

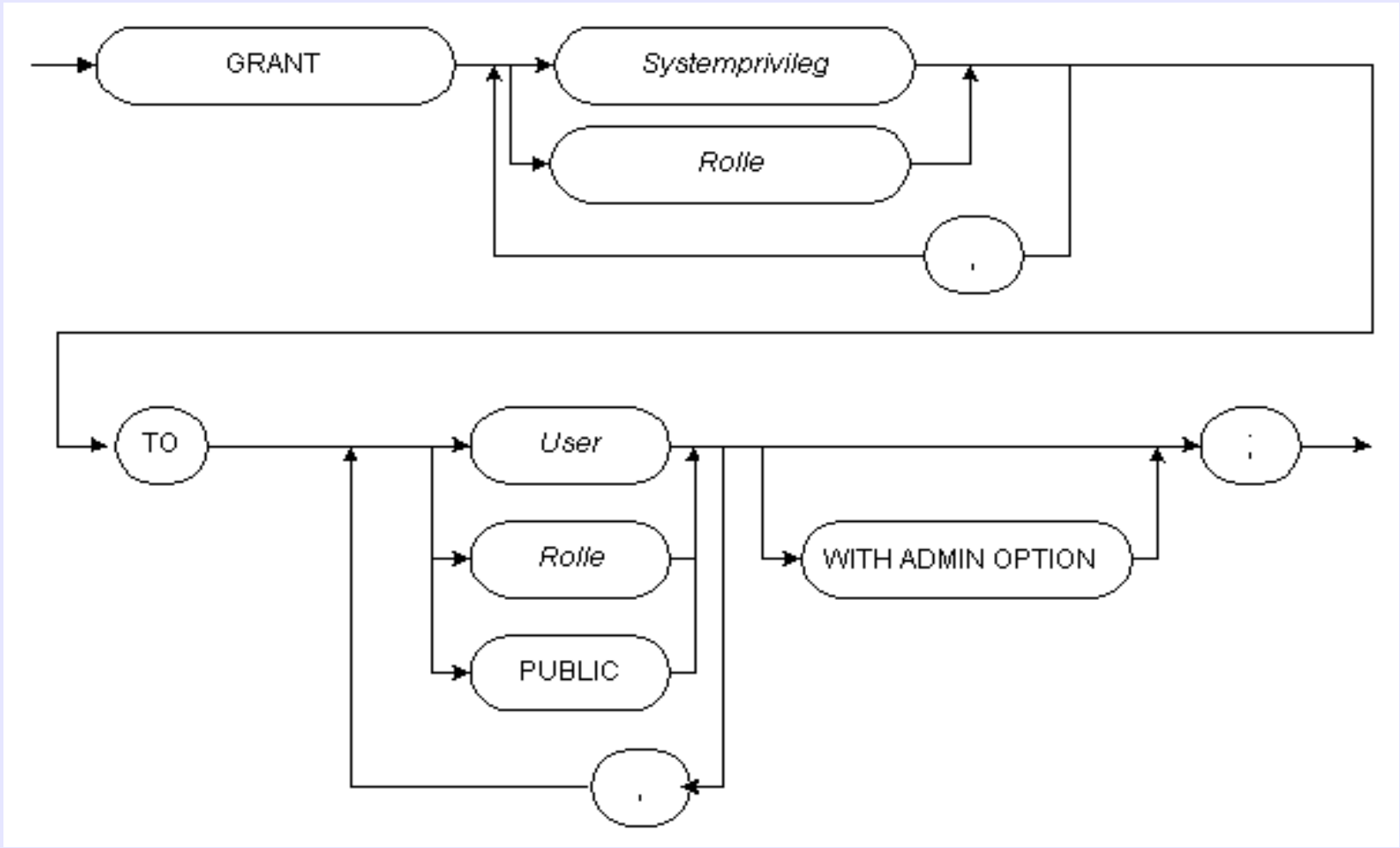
- Mit Public können alle User auf die Tabelle mitarbeiter zugreifen

```
GRANT SELECT
ON mitarbeiter
TO PUBLIC
```

+Rollen bei Oracle

- Im Unterschied zu SQL92 und MySQL ist es in Oracle auch möglich, Rollen zu spezifizieren.
- Dieses Konzept ist dem der Gruppen in Betriebssystemen wie Windows 2000 und UNIX ähnlich und kann bei der Verwaltung von Zugriffsrechten sehr hilfreich sein.
- Es kann eine Anzahl von Zugriffsrechten spezifiziert werden, zu einer Rolle zugeordnet und zu jeder Zeit erweitert werden.
- Benutzer können diese Rolle erteilt bekommen und erhalten die der Rolle zugeteilten Zugriffsrechte - auch Systemprivilegien genannt.
- Oracle enthält über 70 Systemprivilegien, die vergeben werden können.

+Rollen bei Oracle



+Rollen bei Oracle

- `CREATE ROLE verwalter;`

- `GRANT verwalter TO joerg, sabine, harald;`

- ```
  GRANT INSERT, SELECT, UPDATE(gehalt)
    ON mitarbeiter
    TO verwalter
    WITH GRANT OPTION;
  ```
  ```

+Beispiel: Rollen bei Oracle

(demo/sql-is_uni/is_uni_rollen-bei-oracle.sql)

```
-- 1. is_uni vergibt ein objekt-recht an den user schueler
-- ------------------------------------------------------
```

1. User:Schueler:

   Rollen: connect, resource Objekt Rechte: keine

   select * from is_uni.is_professoren **funktioniert nicht**

2. User: is_uni:

   grant select on is_uni.is_professoren to schueler;

3. User: Schueler:

   Rollen: connect,resource Objekt Rechte: is_uni.is_professoren

   select * from is_uni.is_professoren **funktioniert nun**

4. es gibt eine bessere vorgehensweise, deshalb

   User: is_uni

   revoke select on is_uni.is_professoren from schueler;

+Beispiel: Rollen bei Oracle

```
-- 2. is_uni erstellt die rolle is_uni_personalverwalter
--    d.h.: is_uni braucht das recht eine rolle zu erzeugen
1. user: system:
   grant create role to is_uni

2. user: is_uni:
   create role is_uni_personalverwalter;
   grant select, insert, update, delete on
      is_uni.is_professoren to is_uni_personalverwalter;
   grant is_uni_personalverwalter to schueler;

3. user: schueler: (muss sich neu angemeldet haben)
   select * from is_uni.is_professoren
```

funktioniert nun

+Beispiel: Rollen bei Oracle

```
--
-- 3. alles rückgängig machen
--

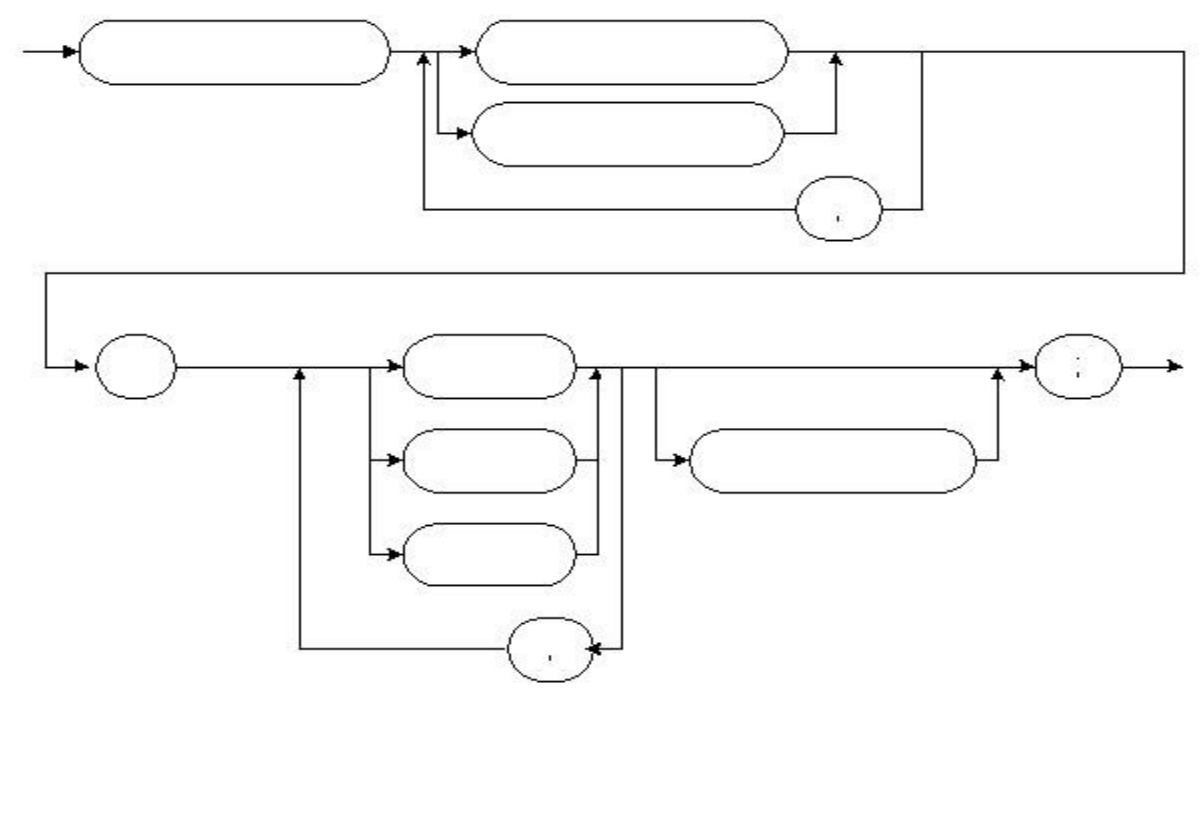
user: is_uni:
   drop role is_uni_personalverwalter

user: system:
   revoke create role from is_uni

commit
```

+Fragen:

- Füge folgende Begriffe ein:
- With admin option, Grant,Systemprivileg,Rolle,To,User, Public



+Übungen/Fragen

- Uni-Beispiel:
  - demo/sql-is_uni/*
- Oracle-Beispiel (Humanresources)
  - demo/sql-hr
- Uni-Fragen:
  - demo/sql-is_uni/fragen_mysql_ohne_antwort.**sql**
    - (in die DB einspielen und beantworten)
  - demo/sql-is_uni/fragen_ohne_antwort.**txt**
    - (in Toad, sql-browser laden und beantworten)

Ausgezeichnete SQL-Übungen

- http://www.sqlzoo.net/