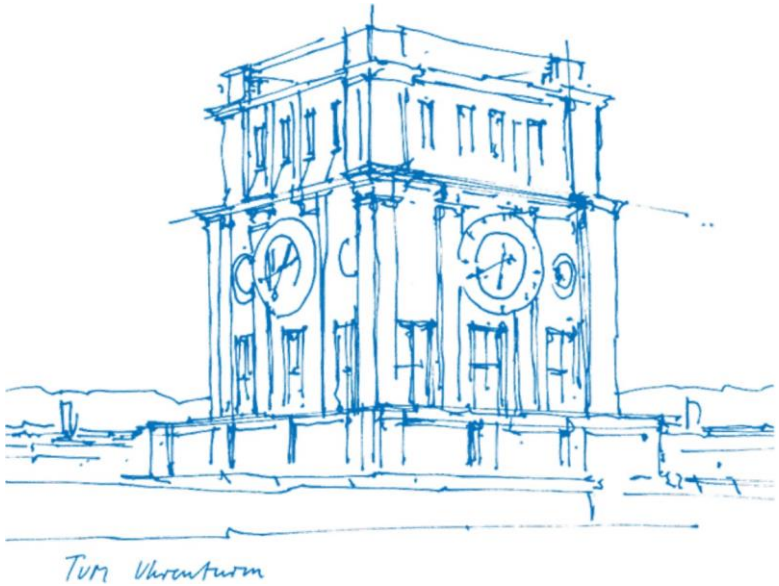
Disclaimer:
Die Slides der Tutorübung dienen nur zur Ergänzung und werden **nicht** fachlich geprüft!
Einige Übungen sind aus der Vorlesung "Grundlagen: Datenbanken" übernommen und dienen ebenfalls nur zur Ergänzung.

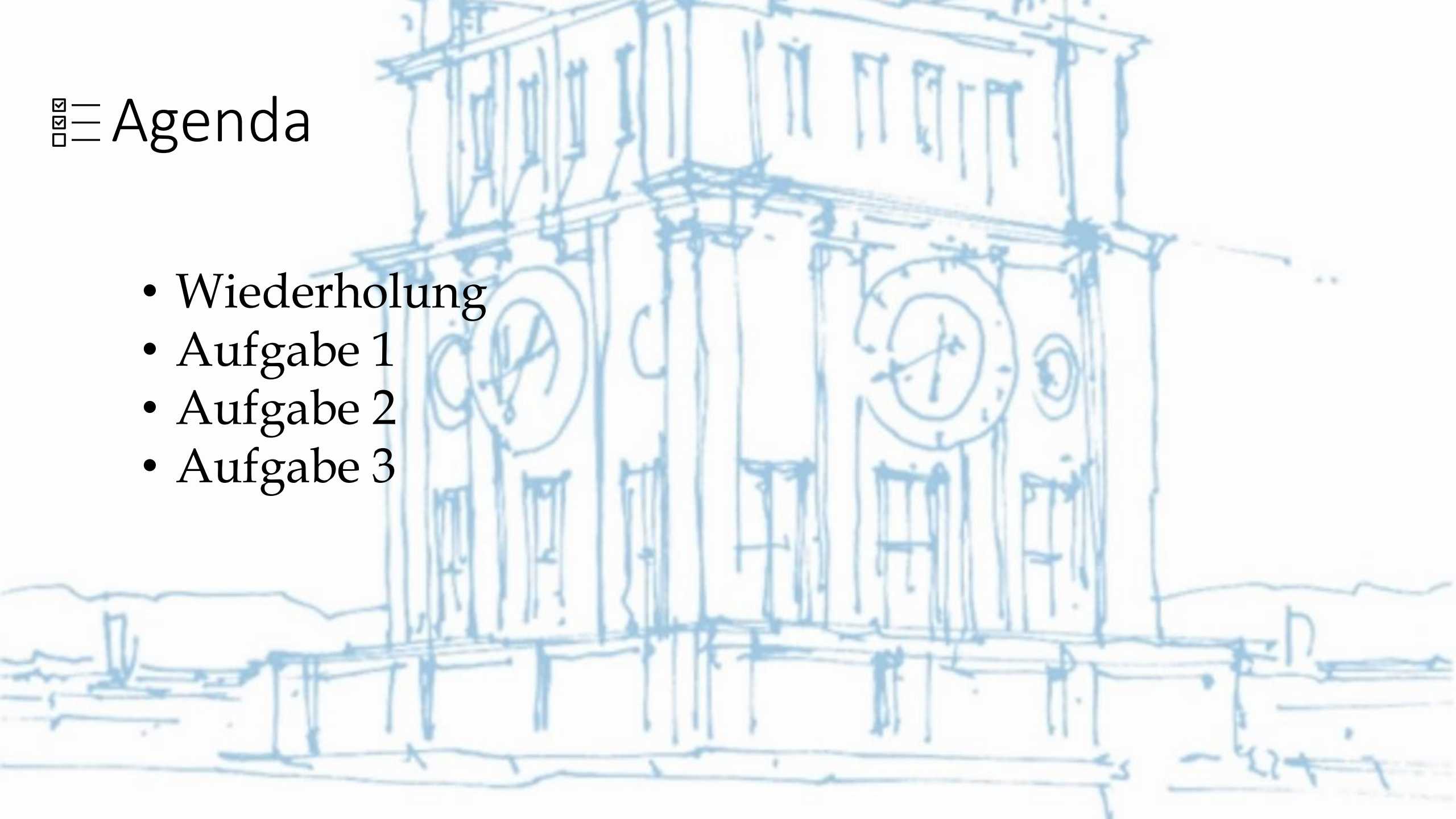
Grundlagen Datenbanken – Tutorium 05

Sebastian Reichbauer (ge64rom@mytum.de)



Agenda

- Wiederholung
- Aufgabe 1
- Aufgabe 2
- Aufgabe 3



Organisatorisches

Download der Folien:
home.in.tum.de/~reichbau
(alle Folien privat ohne vollständigen Stoff)

Hausaufgaben-Upload:
home.in.tum.de/~reichbau

Wiederholung

Structurated Query Language | SQL

Standardisierte (**deklarative**) Anfragesprache für relationale DBMS

Umfasst vier „Gebiete“:

- **DRL**: Data Retrieval Language
- **DML**: Data Manipulation
- **DDL**: Data Definition Language
- **DCL**: Data Control Language

Übungsmöglichkeiten

- HyPer WebInterface http://hyper-db.com/interface.html
- Installation eines DBMS e. g. PostgreSQL, MS-SQL-Server

Anmerkung: (Aktuelle) SQL Standards umfassen eine erweiterte Syntax, mehr Befehle etc. - Hier: Überblick über Vorlesungsinhalte

Wichtige Datentypen

Datentyp	Eigenschaften
char(n), **character**(n)	Character String, **feste** Länge **n** Zeichen
varchar(n), **character varying**(n)	Character String, **variable** Länge **n** Zeichen
integer	**Signed** 32bit int
numeric(p, s)	Allgemeiner numerischer Wert **p** Anzahl Stellen gesamt **s** Anzahl Nachkommastellen
date	Format: yyyy-MM-dd
time	Format: hh:mm:ss
timestamp	Format: yyyy-MM-dd hh:mm:ss

Data Definition Language | DDL

Tabelle anlegen mit **CREATE TABLE**

```
1  create table professoren(
2    persnr integer,
3    name varchar(30) not null,
4    rang char(2),
5    raum integer unique,
6    primary key (persnr)
7  );
```

Der Primärschlüssel wird mit **primary key**(...) angegeben.
Schlüsselkandidaten können mit **unique** gekennzeichnet werden.

Anmerkung

Bezieht sich die Integritätsbedingung nur auf **ein** Attribut, kann sie direkt hinter ihrer Definition stehen (***persnr integer primary key***)

Data Manipulation Language | DML

Daten einfügen mit **INSERT INTO**

Konstante Werte, **Reihenfolge** wichtig!

```
1  INSERT INTO professoren
2  values(3000, 'Proton', 'C4', 101);
```

Konstante Werte - Teil der Attribute (Rest e. g. NULL)

```
1  INSERT INTO professoren(persnr,name)
2  values(3000, 'Proton');
```

Generierung durch Anfrage

```
1  INSERT INTO professoren(persnr,name)
2  SELECT a.persnr, a.name
3  FROM assistenten a
4  WHERE a.persnr = 3006;
```

Data Manipulation Language | DML

Daten ändern mit **UPDATE … SET …**

```
1  UPDATE professoren
2  SET raum = 101, rang = 'C4'
3  WHERE name = 'Kopernikus';
```

Bsp.: Raumnummer von Kant ändert sich zu 101 & Rang zu C4.

Allgemeine Anmerkung
Alle Änderungsoperationen in SQL (e.g. INSERT, UPDATE, DELETE) werden in zwei Schritten ausgeführt. Dies verhindert ein (von der Reihenfolge abhängiges) nichtdeterministisches Ergebnis.

Beispiel: UPDATE führt die SET-Operation erst mit einer temporären Tabelle durch. Anschließend wird die Originaltabelle überschrieben.

Data Manipulation Language | DML

Daten löschen mit **DELETE & DROP**

(Bedingtes) Löschen von Tupeln

```sql
DELETE FROM pruefen
WHERE note = 1 or note > 4;
```

```sql
DELETE FROM pruefen;
```

Löschen von Tabellen, Sichten, Indizes etc.

drop table *Relation*;
drop view *Sicht*;
drop index *Index*;

Data Retrieval Language | DRL

Übersicht

SELECT	*Spalten-/Attributsselektion*
FROM	*Relation-/Tabelleselektion*
WHERE	*Zeilen-/Tupelselektion*
GROUP BY	*Gruppierung*
HAVING	*Gruppierte Filterung*
ORDER BY	*Sortierung*
;	*Anweisungsende*

Logische Kombination: **AND**, **OR**, **NOT**
(Standard-) Vergleichsoperatoren: =, <, >, >=, <=, <>

Erstes SQL Beispiel

```
1  SELECT name
2  FROM studenten
3  WHERE semester >= 3;
```

Ausgabe der Namen von Studenten die mind. im 3 Semester sind (kann Duplikate enthalten).

Im Gegensatz zur relat. Algebra eliminiert SQL keine Duplikate!
Falls erwünscht: **DISTINCT**(bezieht sich i. A. auf die gesamten Ergebniszeilen, d.h. alle angegebenen Spalten als Einheit):

```
1  SELECT distinct v.vorgaenger
2  FROM voraussetzen v;
```

Bsp.: Duplikatfreie Ausgabe der VorlNr. welche Nachfolger haben.

Auch erlaubt: `1  SELECT s.semester+1 ...`

Strings

Stringkonstanten müssen in einfachen Anführungszeichen bzw. Hochkommata eingeschlossen sein: *'Jonas'*

Joker/Wildcards erlauben variable Stringvergleiche:
Beliebiges Zeichen _
Beliebige Zeichenkette (auch Länge 0) %

```
1  SELECT *
2  FROM studenten
3  WHERE name like 'F_%';
```

Studenten deren Namen mit einem F anfängt und mind. 2 Zeichen lang ist.

Umbenennungen

Attribute und Relationen umbennen mit **AS**
Relationen / Attributnamen müssen **eindeutig** sein!

```sql
SELECT *
FROM Studenten, hoeren
WHERE matrnr=matrnr;
```

FALSCH

```sql
SELECT *
FROM Studenten join hoeren
    on matrnr=matrnr;
```

FALSCH

```sql
SELECT *
FROM Studenten, hoeren
WHERE Studenten.matrnr=hoeren.matrnr;
```

RICHTIG

```sql
SELECT *
FROM Studenten as s, hoeren as h
WHERE s.matrnr=h.matrnr;
```

RICHTIG

```sql
SELECT *
FROM Studenten s, hoeren h
WHERE s.matrnr=h.matrnr;
```

RICHTIG

Anmerkung: Gilt insb. für e. g. FROM Studenten s1, Studenten s2

Mengenoperationen

Voraussetzung (siehe Relationale Algebra):
Gleiche Spaltennamen und -anzahl & gleiche Domänen

- Vereinigung: **UNION** oder **UNION ALL**
- Schnitt: **INTERSECT**
- Differenz: **EXCEPT**

```
1  (SELECT *
2   FROM studenten)
3  except
4  (SELECT *
5   FROM studenten
6   WHERE studenten.semester <=6);
```

Bsp.: Verkomplizierte Anfrage für alle Studenten die mind. im 7 Semester sind.

Anmerkung

Im Gegensatz zu **SELECT** eliminiert **UNION** automatisch Duplikate
→ **UNION ALL** erhält Duplikate

Gruppierungen

GROUP BY: Gruppiert Zeilen (e. g. um getrennt zu aggregieren)

Vorsicht

Wird gruppiert, so dürfen alle Attribute die nicht in der GROUP BY-Klausel auftauchen nur aggregiert in der **SELECT**-Klausel stehen!

Angabe mehrerer Attribut möglich → Auswertung „links nach rechts“

```
1  SELECT h.matrnr, s.name, count(*) as Anzahl
2  FROM studenten s join hoeren h on s.matrnr=h.matrnr
3  GROUP BY h.matrnr, s.name;
```

Gibt an, wie viele Vorlesungen jeder Student hört.

Ausgabe: MatrNr., Name und „Anzahl“

Gruppierungsfilter

WHERE-Klausel wird **vor** der Gruppierung ausgewertet.
→ **HAVING**: Filterung nach der Gruppierung

```
1  SELECT v.gelesenVon, count(*) as Anzahl
2  FROM vorlesungen v
3  GROUP BY v.gelesenVon
4  HAVING count(*) > 2;
```

Gibt die Professoren aus (PersNr.), welche mehr als 2 Vorlesungen halten & die jew. Anzahl der Vorlesungen als „Anzahl“.

Anmerkungen:

- **count(*)** in **HAVING** bezieht sich also jew. auf die Gruppe!
- Bei einer Gruppierung werden Nullwerte in eine eigene Gruppe eingeordnet.

Ergebnissortierung

ORDER BY: Sortierung aufsteigend oder absteigend

Keine Angabe von asc/desc → aufsteigende Sortierung
Angabe mehrerer Spalten möglich → Auswertung „links nach rechts“

```
1  SELECT *
2  FROM studenten
3  ORDER BY semester desc, name;
```

Sortierung nach absteigendem Semester - aufsteigende Sortierung des Namens für gleiche Semester.

Aggregatfunktionen

Zählen: **count**(), Aufsummieren: **sum**(), Durchschnitt: **avg**(),
Maximum: **max**(), Minimum: **min**()

```
1  SELECT name, matrnr
2  FROM studenten
3  WHERE matrnr = (SELECT max(matrnr)
4                  FROM studenten);
```

Welcher Student (Name und Matrnr.) hat die größte MatrNr.?

Vorsicht: Aggregatfunktionen reduzieren alle Werte einer Spalte zu einem einzigen Wert!

```
1  SELECT name, max(matrnr)
2  FROM studenten;
```

FALSCH

Aggregatfunktionen

Anmerkungen

Aggregatfunktionen beachten NULL-Werte i. A. nicht!

Ausnahme: **count**(*) gibt die Kardinalität, d. h. inkludiert NULL-Werte.
Unpassende Argumente führen zu Fehlermeldungen.

```
1  SELECT avg(s.name)
2  FROM studenten s;
```

Oft sind Funktionen jedoch **unerwartet** für den Datentyp definiert (**max**() würde hier funktionieren).

Aggregatfunktionen können zusammen mit **DISTINCT** genutzt werden.

```
1  SELECT count(distinct p.matrnr)
2  FROM pruefen p;
```

Auch erlaubt:

```
1  SELECT avg(s.semester + 1)...
```

Unteranfragen

Unteranfragen dürfen in der **SELECT-**, in der **WHERE-** und in der **FROM-**Klausel auftauchen (gilt nicht für jedes DBMS).

Korreliert (s. u.): Unteranfrage referenziert auch „äußere“ Attribute
→ Werden i. A. für jede „umgebende“ Zeile **neu** ausgeführt.

```
1  SELECT s.*
2  FROM studenten s
3  WHERE exists (SELECT *
4                FROM professoren p
5                WHERE p.gebDatum > s.gebDatum);
```

Bsp.: Alle Studenten, die älter als der bzw. die jüngste Professor(in) sind.

Unteranfragen

Unkorreliert (s. u.): Unteranfrage referenziert keine „äußeren“ Attribute
→ Effizient: Wird nur einmal ausgewertet.

```
1  SELECT s.*
2  FROM studenten s
3  WHERE s.gebDatum < (SELECT max(p.gebDatum)
4                      FROM professoren p);
```

Vorsicht

- Bei einem Vergleich in der **WHERE**-Klausel oder einer direkten Verwendung in der **SELECT**-Klausel, darf die Unteranfrage höchstens eine Zeile mit einem Attribut liefern.
- Verwendet man Unteranfragen für „temporäre“ Relationen in der **FROM**-Klausel, so brauchen diese einen Namen (→ **AS**).

Blatt 5 – Aufgabe 1

Aufgabenstellung – Aufgabe 01

Formulieren Sie folgende Anfragen auf dem bekannten Universitätsschema in SQL. Geben Sie alle Ergebnisse duplikatfrei aus.

a) Finden Sie die Studenten, die Sokrates aus Vorlesung(en) kennen.

b) Finden Sie die Studenten, die Vorlesungen hören, die auch Fichte hört.

c) Finden Sie die Assistenten von Professoren, die den Studenten Carnap unterrichtet haben – z.B. als potentielle Betreuer seiner Bachelorarbeit.

d) Geben Sie die Namen der Professoren an, die Theophrastos aus Vorlesungen kennt.

e) Welche Vorlesungen werden von Studenten im Bachelorstudium (1. – 6. Semester) gehört? Geben Sie die Titel dieser Vorlesungen an.

f) Bestimmen Sie für jede Vorlesung wie viele Studenten diese hören. Geben Sie auch Vorlesungen ohne Hörer aus. Sortieren Sie das Ergebnis absteigend nach Anzahl der Hörer.

Lösung – Aufgabe 01

a) Finden Sie die Studenten, die Sokrates aus Vorlesung(en) kennen.

```
select distinct s.Name, s.MatrNr
from Studenten s, hoeren h, Vorlesungen v, Professoren p
where s.MatrNr = h.MatrNr
and h.VorlNr = v.VorlNr
and v.gelesenVon = p.PersNr
and p.Name ='Sokrates';
```

Lösung – Aufgabe 01

b) Finden Sie die Studenten, die Vorlesungen hören, die auch Fichte hört.

```
select distinct s1.Name, s1.MatrNr
from Studenten s1, Studenten s2, hoeren h1, hoeren h2
where s1.MatrNr = h1.MatrNr
and s1.MatrNr != s2.MatrNr
and s2.MatrNr = h2.MatrNr
and h1.VorlNr = h2.VorlNr
and s2.Name ='Fichte';
```

Lösung – Aufgabe 01

c) Finden Sie die Assistenten von Professoren, die den Studenten Carnap unterrichtet haben – z.B. als potentielle Betreuer seiner Bachelorarbeit.

```
select distinct a.Name, a.PersNr
from Assistenten a, Professoren p, Vorlesungen v, hoeren h,
Studenten s
where a.Boss = p.PersNr
and p.PersNr = v.gelesenVon
and v.VorlNr = h.VorlNr
and h.MatrNr = s.MatrNr
and s.Name ='Carnap';
```

Lösung – Aufgabe 01

d) Geben Sie die Namen der Professoren an, die Theophrastos aus Vorlesungen kennt.

```
select distinct p.PersNr, p.Name
from Professoren p, hoeren h, Vorlesungen v, Studenten s
where p.PersNr = v.gelesenVon
and v.VorlNr = h.VorlNr
and h.MatrNr = s.MatrNr
and s.Name ='Theophrastos';
```

Lösung – Aufgabe 01

e) Welche Vorlesungen werden von Studenten im Bachelorstudium (1. – 6. Semester) gehört? Geben Sie die Titel dieser Vorlesungen an.

```
select distinct v.Titel
from Vorlesungen v, hoeren h, Studenten s
where v.VorlNr = h.VorlNr
and h.MatrNr = s.MatrNr
and s.Semester between 1 and 6;
```

Lösung – Aufgabe 01

f) Bestimmen Sie für jede Vorlesung wie viele Studenten diese hören. Geben Sie auch Vorlesungen ohne Hörer aus. Sortieren Sie das Ergebnis absteigend nach Anzahl der Hörer.

```
select v.VorlNr, v.Titel, count(h.MatrNr) as hoerer
from
Vorlesungen v left outer join
hoeren h on (v.VorlNr = h.VorlNr)
group by v.VorlNr, v.Titel
order by hoerer desc;
```

Blatt 5 – Aufgabe 2

Aufgabenstellung – Aufgabe 02

Formulieren Sie die folgenden Anfragen auf dem bekannten Universitätsschema in SQL:

a) Bestimmen Sie das durchschnittliche Semester der Studenten der Universität.
b) Bestimmen Sie das durchschnittliche Semester der Studenten, die mindestens eine Vorlesung bei Sokrates hören.
c) Bestimmen Sie, wie viele Vorlesungen im Schnitt pro Student gehört werden. Beachten Sie, dass Studenten, die keine Vorlesung hören, in das Ergebnis einfließen müssen.

Lösung – Aufgabe 02

a) Bestimmen Sie das durchschnittliche Semester der Studenten der Universität.

```
select avg(semester*1.0) from studenten;
```

Lösung – Aufgabe 02

b) Bestimmen Sie das durchschnittliche Semester der Studenten, die mindestens eine Vorlesung bei Sokrates hören.

```
With vorlesungen_von_sokrates as (
select * from vorlesungen v, professoren p
where v.gelesenVon = p.persnr and p.name = 'Sokrates'
),
studenten_von_sokrates as ( select * from studenten s
where exists (
select * from hoeren h, vorlesungen_von_sokrates v
where h.matrnr = s.matrnr and v.vorlnr = h.vorlnr)
)

select avg(semester) from studenten_von_sokrates
```

Lösung – Aufgabe 02

c) Bestimmen Sie, wie viele Vorlesungen im Schnitt pro Student gehört werden. Beachten Sie, dass Studenten, die keine Vorlesung hören, in das Ergebnis einfließen müssen.

```
select hcount/(scount*1.000)
from (select count(*) as hcount from hoeren) h,
(select count(*) as scount from studenten) s
```

```
select hcount/(cast scount as decimal(10,4)))
from (select count(*) as hcount from hoeren) h,
(select count(*) as scount from studenten) s
```

Blatt 5 – Aufgabe 3

Aufgabenstellung – Aufgabe 3

Gegeben sei die folgende Relation ZehnkampfD mit Athletennamen und den von ihnen erreichten Punkten in den jeweiligen Zehnkampfdisziplinen: ZehnkampfD: {Name; Disziplin; Punkte}

Name	Disziplin	Punkte
Eaton	100 m	450
Eaton	Speerwurf	420
...	...	...
Eaton	Weitsprung	420
Suarez	100 m	850
Suarez	Speerwurf	620
...	...	...

Finden Sie alle ZehnkämpferInnen, die in allen Disziplinen besser sind als der Athlet mit dem Namen Bolt. Formulieren Sie die Anfrage in SQL

a) mit korrelierter Unteranfrage

b) basierend auf Zählen

Lösung – Aufgabe 3

a) Korrelierte Unteranfrage.

```
select distinct a.Name
from ZehnkampfD as a
where not exists (
        select *
        from ZehnkampfD as a2
        where a2.Name = a.Name
        and exists (
                select *
                from ZehnkampfD as b
                where b.Disziplin = a2.Disziplin
                and b.Name = 'Bolt‘
                and b.Punkte >= a2.Punkte
        )
)
```

Lösung – Aufgabe 3

b) Basierend auf Zählen.

```
with besserAlsBolt(name, disziplin) as (
        select a.name, a.disziplin
        from zehnkampfd a, zehnkampfd b
        where b.name = 'Bolt‘
        and a.disziplin = b.disziplin
        and a.punkte > b.punkte
),
disziplinen(anzahl) as (
        select count(distinct disziplin) as anzahl
        from zehnkampfd
)
select name
from besserAlsBolt
group by name
having count(*) = (select anzahl from disziplinen)
```

Eine wunderschöne Woche noch!
Bis nächstes Mal!