



## Übung zur Vorlesung *Einführung in die Informatik 2 für Ingenieure (MSE)*

Alexander van Renen (renen@in.tum.de)

<http://db.in.tum.de/teaching/ss20/ei2/>

### Lösungen zu Blatt 12

#### Aufgabe 1: SQL als DML

Gegeben sei ein erweitertes Universitätsschema mit den folgenden zusätzlichen Relationen *StudentenGF* und *ProfessorenF*:

StudentenGF : {[MatrNr : integer, Name : varchar(20), Semester : integer,  
Geschlecht : char, FakName : varchar(20)]}

ProfessorenF : {[PersNr : integer, Name : varchar(20), Rang : char(2),  
Raum : integer, FakName : varchar(20)]}

Die erweiterten Tabellen sind bereits auf der Webschnittstelle unter

<http://hyper-db.com/interface.html>

angelegt.

#### Lösung 1

- (a) Ermitteln Sie den Männeranteil an den verschiedenen Fakultäten in SQL! Beachten Sie dabei, dass es auch Fakultäten ohne Männer geben kann.

```
WITH FakTotal AS (  
SELECT FakName, COUNT(*) as total  
FROM StudentenGF  
GROUP BY FakName),  
FakMaenner AS (  
SELECT FakName, COUNT(*) as maenner  
FROM StudentenGF  
WHERE geschlecht='M'  
GROUP BY FakName)  
SELECT FakTotal.FakName, (CASE WHEN maenner IS NULL THEN 0  
ELSE maenner END)/(total*1.0)  
FROM FakTotal LEFT JOIN FakMaenner  
ON FakTotal.FakName=FakMaenner.FakName
```

Wir müssen beachten, dass nicht jede Fakultät Männer beherbergt, weswegen diese Fakultäten (in der Standardausprägung im SQL Interface ist dies für Theologie der Fall) dann aus dem Ergebnis herausfallen würden. Aus diesem Grund verwenden wir einen **LEFT OUTER JOIN** um die Zahl der Männer und die Zahl der Studenten insgesamt zu verbinden, wodurch auch die Theologie Fakultät im Ergebnis enthalten ist, auch wenn es keine Männer gibt.

Das **CASE**-Konstrukt dient in der oberen Anfrage dazu, den **NULL** Wert, die durch den **Left Join** für die Anzahl der Männer entstehen, wenn es keine Männer gibt, durch die Zahl 0

zu ersetzen. Alternativ ist dies möglich, indem man `COALESCE(maenner,0)/(total*1.0)` verwendet.

Alternativ können wir das **case**-Konstrukt verwenden, um die Anzahl der Männer an den jeweiligen Fakultäten zu ermitteln. Den Männeranteil erhalten wir dann, indem wir die Anzahl der Männer durch die Gesamtanzahl der Studenten an der Fakultät teilen.

```
select FakName ,
(sum(case when Geschlecht = 'M' then 1 else 0 end)) /
cast (count(*) as float)
from StudentenGF
group by FakName
```

- (b) Ermitteln Sie in SQL die Studenten, die alle Vorlesungen ihrer Fakultät hören. Geben Sie zwei Lösungen an, höchstens eine davon darf auf Abzählen basieren.

```
select s.*
from StudentenGF s
where not exists (select *
from Vorlesungen v, ProfessorenF p
where v.gelesenVon = p.PersNr
and p.FakName = s.FakName
and not exists
(select *
from hoeren h
where h.VorlNr = v.VorlNr
and h.MatrNr = s.MatrNr));
```

Wir fordern hier, dass es keine Vorlesung an der Fakultät des Studenten (d.h. von einem Professor der gleichen Fakultät gelesen) geben darf, die vom Studenten nicht gehört wird.

Alternativ:

```
SELECT * FROM StudentenGF s
WHERE
(SELECT count(*)
FROM Vorlesungen v, ProfessorenF p
WHERE v.gelesenVon = p.PersNr and p.FakName = s.FakName)
=
(SELECT count(*)
FROM hoeren h, Vorlesungen v, ProfessorenF p
WHERE h.MatrNr = s.MatrNr AND h.VorlNr = v.VorlNr AND p.
PersNr = v.gelesenVon AND p.FakName = s.FakName)
```

## Aufgabe 2: Outer Join

In der Vorlesung haben wir den **left outer join** kennen gelernt:

```
select *
from Studenten s left outer join
 hoeren h on s.matrNr = h.matrNr;
```

Ist es möglich eine semantisch äquivalente Anfrage zu formulieren ohne einen **left outer join** zu benutzen (selbstverständlich ist auch der **right** und **full outer join** verboten).

## Lösung 2

Ja:

```
select *
from Studenten s, hoeren h
where s.matrn timer = h.matrn timer
union
select s.*, null, null
from Studenten s
where not exists (select *
from hoeren h
where s.matrn timer = h.matrn timer)
```

## Aufgabe 3: Fleißige Studenten

Formulieren Sie eine SQL-Anfrage, um die Studenten zu ermitteln, die mehr SWS belegt haben als der Durchschnitt. Berücksichtigen Sie dabei auch Totalverweigerer, die gar keine Vorlesungen hören.

## Lösung 3

Folgende SQL-Anfrage ermittelt die fleißigen Studenten:

```
SELECT s.*
FROM Studenten s
WHERE s.MatrNr IN
(SELECT h.MatrNr
FROM hoeren h JOIN Vorlesungen v
ON h.VorlNr = v.VorlNr
GROUP BY h.MatrNr
HAVING SUM(SWS) >
(SELECT SUM(cast(SWS as decimal(5,2)))
/COUNT(DISTINCT(s2.MatrNr))
FROM Studenten s2 LEFT OUTER JOIN hoeren h2
ON h2.MatrNr = s2.MatrNr
LEFT OUTER JOIN Vorlesungen v2
ON v2.VorlNr = h2.VorlNr));
```

Durch die Verwendung temporärer Sichten mit **with** wird die Anfrage übersichtlicher:

```
WITH GesamtSWS AS
(SELECT SUM(cast(SWS as decimal(5,2))) AS AnzSWS
FROM hoeren h2, Vorlesungen v2
WHERE v2.VorlNr = h2.VorlNr),
GesamtStudenten AS
(SELECT count(MatrNr) AS AnzStudenten
FROM Studenten)

SELECT s.*
FROM Studenten s
WHERE s.MatrNr IN (SELECT h.MatrNr
FROM hoeren h JOIN Vorlesungen v ON h.VorlNr = v.VorlNr
GROUP BY h.MatrNr
HAVING SUM(SWS) > (SELECT AnzSWS/AnzStudenten
```

```
FROM GesamtSWS, GesamtStudenten));
```

#### Aufgabe 4: Sichten und Rekursion

Für diese Aufgabe fügen wir eine Spalte (Attribut/Column) zu der Professoren Relation hinzu: doktorElternteilPersNr. Diese gibt die Personennummer der Professorin an bei der eine Professorin ihren Dokortitel erhalten hat. Um eine endlose Kette von Professorinnen zu verhindern erlauben wir in dieser Spalte null Werte.

Erstellen Sie eine Sicht mit folgendem Schema: [persNr, name, generationen]. Die Generation einer Professorin ist durch die Anzahl von bekannten Doktoreltern definiert. Hat eine Professorin zum Beispiel einen null Eintrag in doktorElternteilPersNr, dann ist ihre Generation 0. Sind dagegen 2 Doktoreltern bekannt ist die Generation auch 2.

Das Webinterface beinhaltet diese zusätzliche Spalte nicht, falls Sie dennoch ihre Anfragen testen möchten, können Sie dies in einer lokalen Datenbank tun. Der Folgende Code kann in das die Schemadefinition eingearbeitet werden um die Tabelle mit der zusätzlichen Spalte zu erstellen.

```
CREATE TABLE Professoren
(PersNr          INTEGER PRIMARY KEY,
Name            VARCHAR(30) NOT NULL,
Rang            CHAR(2) CHECK (Rang in ('C2', 'C3', 'C4')),
Raum           INTEGER UNIQUE,
DoktorElternteil INTEGER,
FOREIGN KEY     (DoktorElternteil) REFERENCES Professoren);

INSERT INTO Professoren VALUES
(2125, 'Sokrates', 'C4', 226, null);
INSERT INTO Professoren VALUES
(2134, 'Augustinus', 'C3', 309, 2125);
INSERT INTO Professoren VALUES (2137, 'Kant', 'C4', 007, 2134);
INSERT INTO Professoren VALUES (2126, 'Russel', 'C4', 232, 2137);
INSERT INTO Professoren VALUES (2133, 'Popper', 'C3', 052, 2137);
INSERT INTO Professoren VALUES
(2127, 'Kopernikus', 'C3', 310, null);
INSERT INTO Professoren VALUES (2136, 'Curie', 'C4', 036, 2127);
```

#### Lösung 4

```
with recursive Baum(persNr, name, parent, len) as (
  ( select persNr, name, doktorElternteil, 0 from Professoren )
union
  ( select b.persNr, b.name, p.doktorElternteil, b.len+1
from Professoren p, Baum b where p.persNr = b.parent)
)

select b.persNr, b.name, b.len as generation
from Baum b
where b.len = (select max(t.len)
              from Baum t
              where t.persNr = b.persNr);
```