TU München, Fakultät für Informatik
Lehrstuhl III: Datenbanksysteme
Prof. Alfons Kemper, Ph.D.

Übung zur Vorlesung *Grundlagen: Datenbanken* im WS13/14

Henrik Mühe (muehe@in.tum.de)
http://www-db.in.tum.de/teaching/ws1314/dbsys/exercises/

Blatt Nr. 8

NEUES Tool zum Üben von SQL `http://codematch.muehe.org/`.

Hausaufgabe 1

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

$$\begin{aligned}\text{StudentenGF} &: \{[\underline{\text{MatrNr} : \text{integer}}, \text{Name} : \text{varchar}(20), \text{Semester} : \text{integer},\\ &\quad \text{Geschlecht} : \text{char}, \text{FakName} : \text{varchar}(20)]\}\\ \text{ProfessorenF} &: \{[\underline{\text{PersNr} : \text{integer}}, \text{Name} : \text{varchar}(20), \text{Rang} : \text{char}(2),\\ &\quad \text{Raum} : \text{integer}, \text{FakName} : \text{varchar}(20)]\}\end{aligned}$$

Die erweiterten Tabellen sind auch in der Webschnittstelle angelegt.

(a) Ermitteln Sie den Männeranteil an den verschiedenen Fakultäten in SQL!

(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.

(a)
```
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) 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.

```
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 h"oren h
                        where h.VorlNr = v.VorlNr
                          and h.MatrNr = s.MatrNr));
```

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)
```

Hausaufgabe 2

Gegeben sei die folgende Relation, die Sie auch auf der Webschnittstelle finden:

PunkteListe			
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

Nehmen wir an (im Konjunktiv), dass es in der Klausur eine Freischussregelung gäbe. Nach dieser Regel würde man für die Aufgabe, bei der die Differenz zwischen maximal erzielbarer Punktezahl und eigener Punktezahl am größten ist, einen Bonus erhalten, der die Differenz zwischen maximal erzielbarer Punktezahl und eigener Punktezahl ausgleicht. Dieser Bonus wird allen Studenten aber nur genau einmal zu Gute kommen. Erstellen Sie eine SQL-Sicht, um die PunkteListe nach Verrechnung des Bonus zu bestimmen.

Im unserem Beispiel wäre das:

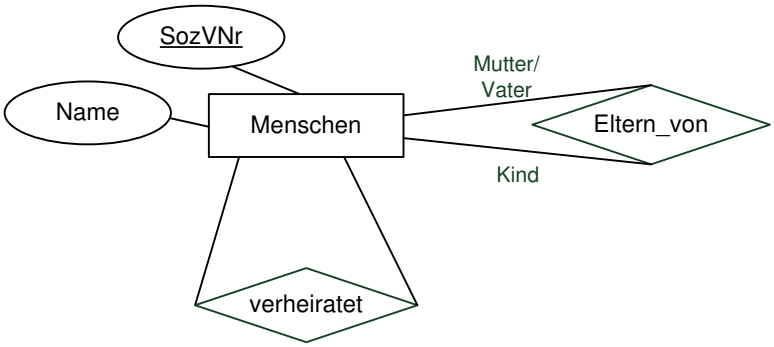
BonusSicht			
Name	Aufgabe	Max	Erzielt
Bond	1	10	4
Bond	2	10	10
Bond	3	11	11
Maier	1	10	4
Maier	2	10	10
Maier	3	11	3

Zunächst erstellen wir eine Sicht *StudBonusaufgabe*, in der für jeden Studenten festgelegt ist, auf welche Aufgabe der Bonus angewendet wird. Für diese Aufgabe muss gelten, dass die erzielte Punktezahl maximal von der zu erreichenden abweicht. Hierfür wird in der temporären Relation t für jeden Studenten die maximale Abweichung der in einer Aufgabe erzielten Punktezahl von der Maximalpunktzahl für diese Aufgabe berechnet. Als zweites muss sichergestellt werden, dass der Bonus nur auf eine Aufgabe angewendet wird. Wir wählen hier unter den in Frage kommenden Aufgaben diejenige mit dem kleinsten Wert (also im Fall von Maier die 2. Aufgabe, und nicht die 3.). Anschließend werden in der View *BonusSicht* die erzielten Punktezahlen pro Aufgabe aufgelistet. Hierbei wird überprüft, ob die aktuell betrachtete Aufgabe in *StudBonusaufgabe* vorkommt und falls ja, wird die erzielte Punktezahl durch die maximal erreichbare ersetzt.

```
create view StudBonusaufgabe as (
select l1.Name, l1.Aufgabe
from PunkteListe l1, (select l2.Name, max(l2.Max-l2.
   Erzielt) as fehlt from PunkteListe l2 group by l2.Name)
   t
where l1.Max-l1.Erzielt =  t.fehlt
  and l1.Name = t.Name
  and not exists (select * from PunkteListe l3 where l3.
     Name = l1.Name and l3.Max-l3.Erzielt = t.fehlt and l3.
     Aufgabe < l1.Aufgabe)
);

create view BonusSicht (Name, Aufgabe, Max, Erzielt) as (
select l.Name, l.Aufgabe, l.Max, (case when l.Aufgabe = (
   select s.Aufgabe from StudBonusaufgabe s where l.Name =
   s.Name) then l.Max else l.Erzielt end)
from PunkteListe l
);
```

Hausaufgabe 3

Gegeben sei das folgende ER-Modell, bei dem wir die Relation *verheiratet* nach dem deutschen Gesetz (d.h. jeder Mensch kann höchstens einen Ehegatten haben) und die Relation *Eltern_von* im biologischen Sinn (d.h. jeder Mensch hat genau eine Mutter und einen Vater) modelliert haben:



Bestimmen Sie sinnvolle Min/Max-Angaben. Geben Sie dann die SQL-Statements zur Erzeugung der Tabellen an, die der Umsetzung des Diagramms in Relationen entsprechen! Verwenden Sie dabei **not null**, **primary key**, **references**, **unique** und **cascade**.

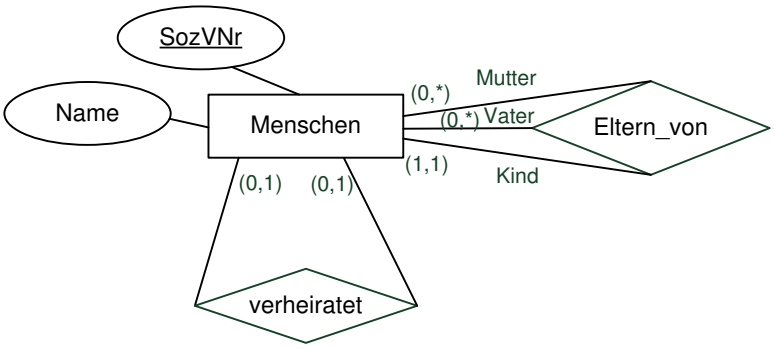
Die folgenden SQL-Statements erzeugen die Tabellen:

```
create table Menschen (
SozVNr      varchar(30) not null primary key,
Name        varchar(30)
);

create table Eltern_von (
MutterVater  varchar(30) not null references Menschen,
Kind          varchar(30) not null references Menschen,
primary key (MutterVater, Kind)
);

create table verheiratet (
Ehegatte1   varchar(30) not null references Menschen on
   delete cascade,
Ehegatte2   varchar(30) not null references Menschen on
   delete cascade,
primary key (Ehegatte1),
unique (Ehegatte2)
);
```

In DB2 müssen alle Attribute, die Teil des Primärschlüssels sind, als **not null** definiert werden. Grundsätzlich ist es auch sinnvoll, **null**-Werte bei Schlüsselattributen auszuschließen. Obwohl der SQL-Standard von 1992 vorschreibt, dass Primärschlüsselattribute implizit als **not null** definiert sind, wird das nicht von allen Datenbanksystemen implementiert (z.B. DB2). In der Tabelle *Menschen* können wir für Namen **null**-Werte zulassen wenn wir davon ausgehen, dass Eltern einige Wochen bis Monate Zeit haben, um einen Namen auszusuchen, das Kind aber zu dieser Zeit schon registriert ist. In *Eltern_von* sollen nur bekannte Eltern-Kind-Beziehungen eingetragen werden, deshalb sind alle Attribute als **not null** deklariert. Sowohl *MutterVater* als auch *Kind* sind *Menschen*, referenzieren also die *Menschen*-Tabelle. Da ein Kind zwei Elternteile hat, setzt sich der Primärschlüssel aus

beiden Attributen *MutterVater* und *Kind* zusammen. Als Primärschlüssel von *verheiratet* kann entweder *Ehegatte1* oder *Ehegatte2* gewählt werden. Der jeweils andere Ehegatte muss als **unique** gekennzeichnet werden. Da mit dem Tod eines Menschen dessen Ehe auch beendet ist, wurden die Fremdschlüssel *Ehegatte1* und *Ehegatte2* mit dem Zusatz **on delete cascade** angelegt. Da davon auszugehen ist, dass sich die Sozialversicherungsnummer niemals ändert, haben wir kein **on update cascade** verwendet. Könnte sie sich ändern, wäre bei allen Attributen, die *Menschen* referenzieren **on update cascade** hinzugefügt werden. (Hinweis: DB2 unterstützt kein **on update cascade**, sondern lediglich **on update restrict**.)

Man hätte den Ehepartner auch in die Relation *Menschen* mit aufnehmen können, da es sich um eine 1:1-Beziehung handelt. Dies würde aber zu vielen **null**-Werten führen (weil sehr viele Menschen keinen Ehepartner haben) und ist daher nicht empfehlenswert. Die Relation *Eltern_von* könnte alternativ auch mit den drei Attributen *Mutter*, *Vater* und *Kind* umgesetzt werden. Dies würde dem folgenden ER-Modell entsprechen:



Die Umsetzung wäre dann:

```
create table Eltern_von (
Mutter  varchar(30) not null references Menschen,
Vater   varchar(30) not null references Menschen,
Kind    varchar(30) not null references Menschen,
primary key (Kind)
);
```