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. 7

Tool zum Üben von SQL `http://www-db.in.tum.de/~muehe/sql/`.

SQL Challenges `http://codematch.muehe.org`.

Mailen Sie Ihre Lösung VOR der Übung an Ihren Tutor, damit Sie diese nicht komplett abtippen müssen, falls Sie in der Übung drankommen.

Hausaufgabe 1

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

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 von **with** und **case** 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));
```

Alternativ:

```
with SWSProStudent as
(select s.MatrNr,
  cast((case when sum(v.SWS) is null
             then 0 else sum(v.SWS)
        end) as float) as AnzSWS
 from Studenten s left outer join hoeren h
   on s.MatrNr = h.MatrNr
   left outer join Vorlesungen v
   on h.VorlNr = v.VorlNr
 group by s.MatrNr
)

select s.*
from Studenten s
where s.MatrNr in (select sws.MatrNr
                   from SWSProStudent sws
                   where sws.AnzSWS > (select avg(AnzSWS)
                                       from SWSProStudent)
                                          );
```

Hausaufgabe 2

Für die nachfolgenden Aufgaben sollten Sie eine größere *prüfen*-Relation benutzen, damit die Anfragen sinnvoll getestet werden können. Im Tool heißt diese Relation `pruefenxl`.

Bestimmen Sie für alle Studenten eine gewichtete Durchschnittsnote ihrer Prüfungen. Die Gewichtung der einzelnen Prüfungen erfolgt gemäß dem Vorlesungsumfang (SWS). Dies entspricht dem Verfahren der Durchschnittsnotenberechnung für Ihr Bachelor-Zeugnis.

Hier sollen die Noten der Studenten gewichtet werden, d.h. dass Noten für Prüfungen über kurze Vorlesungen (z.B. 2 SWS) abgewertet werden, da der Lernaufwand dafür geringer war als für Prüfungen über lange Vorlesungen, die dadurch aufgewertet werden.

```
with gwNoten as(
select s.Name, s.MatrNr, v.Titel, p.Note, v.SWS as Gewicht
   ,
  (p.Note*v.SWS) as gwNote
from pruefen p, Vorlesungen v, Studenten s
where p.VorlNr = v.VorlNr and s.MatrNr = p.MatrNr)

select Name, (sum(gwNote)/sum(Gewicht)) as
   gwDurchschnittsnote
from gwNoten
group by Name, MatrNr;
```

Hausaufgabe 3

Welche Studenten haben alle Vorlesungen, die sie haben prüfen lassen, auch tatsächlich vorher gehört?

Die Anforderung, dass die Studenten im Anfrage-Ergebnis alle Vorlesungen, die sie haben prüfen lassen auch tatsächlich gehört haben, lässt sich umschreiben zu: „Es darf keine Vorlesung geben, die geprüft wurde, zu der es aber keinen Eintrag in *hoeren* gibt.“

```
select s.*
from Studenten s
where not exists (select * from pruefen p
                  where s.MatrNr = p.MatrNr
                    and not exists (select *
                                    from hoeren h
                                    where h.MatrNr = s.
                                       MatrNr
                                      and h.VorlNr = p.
                                         VorlNr));
```

Hausaufgabe 4

Was bringt der Vorlesungsbesuch? Finden Sie heraus, ob es für Prüfungen von Vorteil ist, die jeweiligen Vorlesungen auch gehört zu haben. Ermitteln Sie dazu die Durchschnittsnote der Prüfungen, zu denen die Studenten die Vorlesungen nicht gehört haben und die Durchschnittsnote der Prüfungen, zu denen sie die Vorlesungen gehört haben.

Diese Anfrage lässt sich auf zwei Arten beantworten. Zum einen kann ermittelt werden, wie das Verhältnis der Prüfungen für jede Vorlesung aussieht. Eine mögliche Formulierung hierfür ist folgende:

```
select ngehoert.VorlNr, ngehoert.ds, gehoert.ds
from (select p.VorlNr, avg(p.Note) as ds
      from pruefen p
      where not exists( select *
                        from hoeren h
                        where h.MatrNr = p.MatrNr
                        and h.VorlNr = p.VorlNr)
      group by p.VorlNr) ngehoert,
     (select p.VorlNr, avg(p.Note) as ds
      from pruefen p
      where p.VorlNr in (select h.VorlNr
                         from hoeren h
                         where h.MatrNr = p.MatrNr)
      group by p.VorlNr) gehoert
where ngehoert.VorlNr = gehoert.VorlNr;
```

Alternativ kann aber auch bestimmt werden, wie das Verhältnis der Prüfungsleistungen von gehörten zu nicht gehörten Vorlesungen sich insgesamt darstellt.

```
create view nichtgehoert as
  select avg(Note) as DnoteVLNichtGehoert
  from  pruefen p
  where not exists (select *
                    from hoeren h
                    where h.VorlNr = p.VorlNr
                      and h.MatrNr = p.MatrNr);

create view gehoert as
  select avg(Note) as DnoteVLGehoert
  from pruefen p
  where exists (select *
                from hoeren h
                where h.VorlNr = p.VorlNr
                  and h.MatrNr = p.MatrNr);
```

Da beide Views aus jeweils nur einem Wert bestehen, ergibt sich das Resultat der Anfrage als Kreuzprodukt:

```
select *
from nichtgehoert, gehoert;
```

Hausaufgabe 5

Gegeben (auch im Tool!) sei die Tabelle `ubahn`, die strukturell dem folgenden Beispiel gleicht:

Von	Nach	Dauer
Garching Forschungszentrum	Garching	2
Garching	Garching Hochbrück	2
Garching Hochbrück	Fröttmaning	4
Fröttmaning	Kieferngarten	2
Kieferngarten	Freimann	1
...	...	...
Odeonsplatz	Marienplatz	1
...	...	...
Haderner Stern	Klinikum Großhadern	1

Geben Sie eine Anfrage an, welche für eine gegebene Station, beispielsweise `Garching Forschungszentrum`, ermittelt, welche **anderen** Stationen von hier erreicht werden können. Geben Sie diese **duplikatfrei** aus.

Weitere Übungsvorschläge:

- Ermitteln Sie die Gesamtdauer, die eine Fahrt von einer gegeben zu jeder anderen Station benötigt.
- Geben Sie eine SQL Anfrage an, welche alle von `Freimann` in beide Richtungen rekursiv erreichbaren Stationen ermittelt. Bilden Sie hierzu zunächst die Hülle in beide Richtungen. Was ist das Problem bei der Erstellung der einfachen Hülle auf der symmetrischen Basisrelation?

```
with huelle (von, nach) as (
  select von, nach from ubahn
  union all
  select h.von, u.nach
  from huelle h, ubahn u
  where h.nach = u.von
)
select * from huelle order by von;
```

`ORDER BY` nicht nötig, macht es aber einfacher lesbar.

Dauer

```
with huelle(von,nach,dauer)  as (
  select von,nach,dauer from ubahn
  union all
  select u.von,h.nach,u.dauer+h.dauer
  from ubahn u,huelle h
  where u.nach=h.von
)
  select * from huelle order by von;
```

Die Hülle kann alternativ wie oben "andersrum" aufgebaut werden.

Doppelhülle:

```
with huelle ( von , nach ) as (
select von , nach from ubahn
```

```
UNION ALL
select u . von , h. nach from ubahn u , huelle h
where u . nach =h . von
), doppelhuelle(von,nach) as (
SELECT von, nach FROM huelle
UNION ALL
SELECT nach, von FROM huelle)
select distinct * from doppelhuelle order by von ;
```

Bildet man die Hülle auf der symmetrischen Variante von `ubahn`, so terminiert die Anfrage nicht, da SQL Bag-Semantik nutzt und daher keine Duplikate eliminiert. Würde man Duplikate während der Rekursion ausschließen, wäre dies eine bessere alternative Lösung.