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

Übung zur Vorlesung *Grundlagen: Datenbanken* im WS21/22

Michael Jungmair, Josef Schmeißer, Moritz Sichert, Lukas Vogel (gdb@in.tum.de)
https://db.in.tum.de/teaching/ws2122/grundlagen/

Blatt Nr. 05

Hausaufgabe 1

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

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

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.

Lösung:

a)
```
with
FakultaetTotal as (
  select Fakultaet, count(*) as Total
  from StudentenGF
  group by Fakultaet),
FakultaetMaenner as (
  select Fakultaet,count(*) as Maenner
  from StudentenGF
  where Geschlecht = 'M'
  group by Fakultaet)
select
   FakultaetTotal.Fakultaet,
   (case when Maenner is null then 0 else Maenner end)/(total*1.00)
from FakultaetTotal left outer join FakultaetMaenner
on FakultaetTotal.Fakultaet = FakultaetMaenner.Fakultaet
```

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*1.00)` 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 Fakultaet,
 (sum(case when Geschlecht = 'M' then 1.00 else 0.00 end)) / count(*)
from StudentenGF
group by Fakultaet
```

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.Fakultaet = s.Fakultaet
                    and not exists
                       (select *
                        from hoeren 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.Fakultaet = s.Fakultaet)
=
(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.Fakultaet = s.Fakultaet
)
```

Hausaufgabe 2

Gegeben sei die folgende Relation `ZehnkampfD` mit Athletennamen und den von ihnen

erreichten Punkten in den jeweiligen Zehnkampfdisziplinen:

$$\text{ZehnkampfD} : \{\underline{\text{Name}, \text{Disziplin}}, \text{Punkte}\}$$

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

Sie dürfen davon ausgehen, dass jeder Sportler in jeder Disziplin angetreten ist.

Laden Sie zum Testen entweder die SQL-Datei von der Übungswebseite in ein lokal installiertes Datenbanksystem oder verwenden Sie die Webschnittstelle.

Lösung:

Mit korrelierter 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
    )
)
```

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)
```

Hausaufgabe 3

Gegeben sei die Relation `Fahrplan`, die strukturell dem folgenden Beispiel gleicht:

Von	Nach	Linie	Abfahrt	Ankunft
Garching, Forschungszentrum	Garching	U6	09:06	09:09
Garching	Garching-Hochbrück	U6	09:09	09:11
Garching-Hochbrück	Fröttmaning	U6	09:11	09:15
...	...			
Fröttmaning	Garching-Hochbrück	U6	09:00	09:04
Garching-Hochbrück	Garching	U6	09:04	09:06
Garching	Garching, Forschungszentrum	U6	09:06	09:09
...	...			
Garching, Forschungszentrum	Technische Universität	690	17:56	17:57

Formulieren Sie die folgenden Anfragen auf diese Relation in SQL. Sie können die Typen `TIME` für Uhrzeiten und `INTERVAL` für Zeitintervalle verwenden.

Schreiben Sie z.B. für 10:30 Uhr: `TIME '10:30:00'`.

Das 0-Intervall kann z.B. so konstruiert werden: `INTERVAL '00:00:00'`.

a) Geben Sie eine Anfrage an, welche für alle Stationen ermittelt, welche **anderen** Stationen erreicht werden können. Beachten Sie, dass nur tatsächlich mögliche Verbindungen ausgegeben werden sollen, d.h. die Abfahrt an einer Haltestelle darf nicht vor der Ankunft liegen.

b) Erweitern Sie ihre Anfrage aus Teilaufgabe a), sodass zusätzlich die summierte Fahrtzeit und Wartezeit sowie die gesamte Reisezeit ausgegeben wird. Die Fahrtzeit ist dabei nur die Zeit, in der man sich in einem Verkehrsmittel befindet. Die Wartezeit ist die Zeit, die bei einem Umstieg zwischen Ankunft des alten und Abfahrt des neuen Verkehrsmittels vergeht. Die Reisezeit ist die Zeit zwischen Abfahrt des ersten und Ankunft des letzen Verkehrsmittels.

c) Erweitern Sie ihre Anfrage aus Teilaufgabe a) oder b) nochmals und geben Sie die Anzahl der Umstiege für jede Verbindung aus.

d) Finden Sie die „guten“ Verbindungen, um von Fröttmaning pünktlich zur Vorlesung „Grundlagen: Datenbanken“ um 10:30 Uhr zu kommen. Verwenden Sie dazu Ihre Anfrage aus Teilaufgabe c). Eine Verbindung ist „gut“, wenn sie spätestens um 10:30 in „Garching, Forschungszentrum“ ist und es keine andere Verbindung gibt, die später abfährt aber noch rechtzeitig eintrifft, deren Reisezeit geringer ist und bei der man weniger Umstiege hat.

Lösung:

a) Geben Sie eine Anfrage an, welche für alle Stationen ermittelt, welche **anderen** Stationen erreicht werden können. Beachten Sie, dass nur tatsächlich mögliche Verbindungen ausgegeben werden sollen, d.h. die Abfahrt an einer Haltestelle darf nicht vor der Ankunft liegen.

Für diese Aufgabe werden *rekursive Sichten* mit `with recursive` benötigt. Die Relation `Fahrplan` bildet dort den Rekursionsanfang. Dann werden rekursiv mit `union all` weitere Verbindungen hinzugefügt. Eine neue Verbindung soll nur dann eingefügt werden, wenn die Ankunftshaltestelle gleich der Abfahrtshaltestelle der neuen Teilverbindung ist, wenn diese zeitlich erreichbar ist und wenn die neue Ankunftshaltestelle ungleich der ursprünglichen Abfahrtshaltestelle ist. Dadurch ergeben sich die `where`-Bedingungen.

```
with recursive fahrplan_rec as (
    select von, nach, abfahrt, ankunft from fahrplan
    union all
    select fr.von, f.nach, fr.abfahrt, f.ankunft
    from fahrplan_rec fr, fahrplan f
    where
        fr.nach = f.von and
        fr.ankunft <= f.abfahrt and
        fr.von != f.nach
)
select * from fahrplan_rec
```

b) Erweitern Sie ihre Anfrage aus Teilaufgabe a), sodass zusätzlich die summierte Fahrtzeit und Wartezeit sowie die gesamte Reisezeit ausgegeben wird. Die Fahrtzeit ist dabei nur die Zeit, in der man sich in einem Verkehrsmittel befindet. Die Wartezeit ist die Zeit, die bei einem Umstieg zwischen Ankunft des alten und Abfahrt des neuen Verkehrsmittels vergeht. Die Reisezeit ist die Zeit zwischen Abfahrt des ersten und Ankunft des letzen Verkehrsmittels.

Ausgehend von der Lösung von Teilaufgabe a) muss hier als Rekursionsanfang zusätzlich die Fahrzeit und die Wartezeit gesetzt werden. Bei einer Direktverbindung ergibt sich die Fahrtzeit aus der Differenz zwischen Ankunft und Abfahrt, die Wartezeit ist am Anfang 0. Im Rekursionsschritt muss nun die Fahrtzeit der neuen Teilverbindung der aktuellen Fahrtzeit hinzuaddiert werden. Ähnlich muss zur Wartezeit die Dauer zwischen der Ankunft der aktuellen Verbindung und der Abfahrt der neuen hinzuaddiert werden. Da sich die Reisezeit aus der Summe der Fahrtzeit und der Wartezeit

ergibt, muss diese nicht notwendigerweise schon während der Rekursion berechnet werden, sondern kann durch eine weitere Sicht berechnet werden.

```
with recursive fahrplan_rec_linie as (
    select
        von,
        nach,
        abfahrt,
        ankunft,
        ankunft - abfahrt as fahrtzeit,
        INTERVAL '00:00:00' as wartezeit
    from fahrplan
    union all
    select
        fr.von,
        f.nach,
        fr.abfahrt,
        f.ankunft,
        fr.fahrtzeit + (f.ankunft - f.abfahrt),
        fr.wartezeit + (f.abfahrt - fr.ankunft)
    from fahrplan_rec_linie fr, fahrplan f
    where
        fr.nach = f.von and
        fr.ankunft <= f.abfahrt and
        fr.von != f.nach
),
fahrplan_rec as (
    select
        von,
        nach,
        abfahrt,
        ankunft,
        fahrtzeit,
        wartezeit,
        fahrtzeit + wartezeit as reisezeit
    from fahrplan_rec_linie
)
select * from fahrplan_rec
```

c) Erweitern Sie ihre Anfrage aus Aufgabe a) oder b) nochmals und geben Sie die Anzahl der Umstiege für jede Verbindung aus.

Man muss immer genau dann umsteigen, wenn durch den Rekursionsschritt eine neue Teilverbindung hinzukommt, die zu einer anderen Linie gehört als der aktuellen, oder wenn die Ankunftszeit der bisherigen Verbindung kleiner als die Abfahrtszeit der neuen Teilverbindung ist. Um ersteres zu erkennen, muss in der rekursiven Sicht immer die aktuelle Linie mitgeführt werden. Diese wird im Rekursionsanfang auf die Linie der Teilverbindung gesetzt. Im Rekursionsschritt wird dann mithilfe von `case` ermittelt, ob ein Umstieg stattfindet oder nicht. Im Rekursionsschritt muss auch die aktuel-

le Linie aktualisiert werden, damit die Erkennung von Umstiegen weiterhin korrekt funktioniert.

```
with recursive fahrplan_rec_linie as (
    select
        von,
        nach,
        abfahrt,
        ankunft,
        linie as aktuelle_linie,
        0 as umstiege,
        ankunft - abfahrt as fahrtzeit,
        INTERVAL '00:00:00' as wartezeit
    from fahrplan
    union all
    select
        fr.von,
        f.nach,
        fr.abfahrt,
        f.ankunft,
        f.linie,
        fr.umstiege + case
            when
                f.linie != fr.aktuelle_linie or
                f.abfahrt > fr.ankunft
            then 1
            else 0
        end,
        fr.fahrtzeit + (f.ankunft - f.abfahrt),
        fr.wartezeit + (f.abfahrt - fr.ankunft)
    from fahrplan_rec_linie fr, fahrplan f
    where
        fr.nach = f.von and
        fr.ankunft <= f.abfahrt and
        fr.von != f.nach
),
fahrplan_rec as (
    select
        von,
        nach,
        abfahrt,
        ankunft,
        umstiege,
        fahrtzeit,
        wartezeit,
        fahrtzeit + wartezeit as reisezeit
    from fahrplan_rec_linie
)
```

```
select * from fahrplan_rec
```

d) Finden Sie die „guten“ Verbindungen, um von Fröttmaning pünktlich zur Vorlesung „Grundlagen: Datenbanken“ um 10:30 Uhr zu kommen. Verwenden Sie dazu Ihre Anfrage aus Teilaufgabe c). Eine Verbindung ist „gut“, wenn sie spätestens um 10:30 in „Garching, Forschungszentrum“ ist und es keine andere Verbindung gibt, die später abfährt aber noch rechtzeitig eintrifft, deren Reisezeit geringer ist und bei der man weniger Umstiege hat.

Ausgehend von der Lösung von Teilaufgabe c) kann diese Aufgabe mit `not exists` gelöst werden:

```
select * from fahrplan_rec fr
where
    fr.von = 'Fröttmaning' and
    fr.nach = 'Garching, Forschungszentrum' and
    fr.ankunft <= TIME '10:30:00' and
    not exists (
        select * from fahrplan_rec fr2
        where
            fr2.von = fr.von and
            fr2.nach = fr.nach and
            fr2.ankunft <= TIME '10:30:00' and
            fr2.abfahrt > fr.abfahrt and
            fr2.reisezeit < fr.reisezeit and
            fr2.umstiege < fr.umstiege
    )
```

Hausaufgabe 4

Schreiben Sie eine SQL-Anfrage, die das kleine Einmaleins[1] in Tabellenform ausgibt. Tipp: Verwenden Sie `WITH ... (VALUES ...)`.

Lösung:

Das komplette Einmaleins kann direkt in einen `VALUES`-Ausdruck gefasst werden und ausgegeben werden:

```
with einmaleins as (values
    (  1,   2,   3,   4,   5,   6,   7,   8,   9,  10),
    (  2,   4,   6,   8,  10,  12,  14,  16,  18,  20),
    (  3,   6,   9,  12,  15,  18,  21,  24,  27,  30),
    (  4,   8,  12,  16,  20,  24,  28,  32,  36,  40),
    (  5,  10,  15,  20,  25,  30,  35,  40,  45,  50),
    (  6,  12,  18,  24,  30,  36,  42,  48,  54,  60),
    (  7,  14,  21,  28,  35,  42,  49,  56,  63,  70),
    (  8,  16,  24,  32,  40,  48,  56,  64,  72,  80),
    (  9,  18,  27,  36,  45,  54,  63,  72,  81,  90),
    ( 10,  20,  30,  40,  50,  60,  70,  80,  90, 100)
)
select * from einmaleins
```

[1] https://de.wikipedia.org/wiki/Einmaleins

Alternativ kann das Einmaleins auch von der Datenbank berechnet werden, was ein wenig Schreibarbeit spart:

```
with einmaleins_spalten(a, b, c, d, e, f, g, h, i, k) as (values
    (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
),
einmaleins_zeilen(x) as (values
    (1), (2), (3), (4), (5), (6), (7), (8), (9), (10)
)
select
    a * x,
    b * x,
    c * x,
    d * x,
    e * x,
    f * x,
    g * x,
    h * x,
    i * x,
    k * x
from einmaleins_spalten, einmaleins_zeilen
```