



Übung zur Vorlesung *Grundlagen: Datenbanken* im WS20/21
Christoph Anneser, Josef Schmeißer, Moritz Sichert, Lukas Vogel (gdb@in.tum.de)
<https://db.in.tum.de/teaching/ws2021/grundlagen/>

Blatt Nr. 05

Hausaufgabe 1

Führen Sie die folgenden Änderungen am Datenbestand des bekannten Universitätsschemas in SQL aus. Stellen Sie sicher, dass Ihre SQL-Statements mit jeder beliebigen Ausprägung des Schemas funktionieren.

- Alle Professoren, die den Rang C3 haben, werden auf den Rang C4 befördert. Setzen Sie dazu den Rang aller C3-Professoren auf C4.
- Die Planetenbewegungen sind vollständig erforscht. Löschen Sie alle Assistenten mit diesem Fachgebiet.
- Eine neue Vorlesung mit dem Namen „Grundlagen: Datenbanken“ mit der Nummer 5278 soll erstellt werden. Die Vorlesung wird von der Professorin Curie gehalten und hat die Vorlesung „Logik“ als Voraussetzung. Sie soll 4 SWS umfassen. Tragen Sie den Studenten mit der Matrikelnummer 28106 als Hörer der Vorlesung ein. Erstellen Sie alle notwendigen SQL-Statements.

Lösung:

- Alle Professoren, die den Rang C3 haben, werden auf den Rang C4 befördert. Setzen Sie dazu den Rang aller C3-Professoren auf C4.

```
update Professoren set Rang = 'C4' where Rang = 'C3';
```

- Die Planetenbewegungen sind vollständig erforscht. Löschen Sie alle Assistenten mit diesem Fachgebiet.

```
delete
from Assistenten
where Fachgebiet = 'Planetenbewegung';
```

- Eine neue Vorlesung mit dem Namen „Grundlagen: Datenbanken“ mit der Nummer 5278 soll erstellt werden. Die Vorlesung wird von der Professorin Curie gehalten und hat die Vorlesung „Logik“ als Voraussetzung. Sie soll 4 SWS umfassen. Tragen Sie den Studenten mit der Matrikelnummer 28106 als Hörer der Vorlesung ein. Erstellen Sie alle notwendigen SQL-Statements.

```
insert into Vorlesungen
select 5278, 'Grundlagen: Datenbanken', 4, PersNr
from Professoren
where Name = 'Curie';
```

```
insert into voraussetzen
```

```

select VorlNr, 5278
from Vorlesungen
where Titel = 'Logik';

insert into hoeren values (28106, 5278);

```

Hausaufgabe 2

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

```

StudentenGF : {[MatrNr : integer, Name : varchar(20), Semester : integer,
                Geschlecht : char, Fakultaet : varchar(20)]}
ProfessorenF : {[PersNr : integer, Name : varchar(20), Rang : char(2),
                 Raum : integer, Fakultaet : varchar(20)]}

```

Die erweiterten Tabellen sind auch in der Webschnittstelle angelegt.

- Ermitteln Sie den Männeranteil an den verschiedenen Fakultäten in SQL!
- 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 3

„Bekanntheitsgrad“: Formulieren Sie eine SQL-Anfrage, um den Bekanntheitsgrad von Studenten zu ermitteln. Gehen Sie dabei davon aus, dass Studenten sich aus gemeinsam besuchten Vorlesungen kennen. Sortieren Sie das Ergebnis absteigend nach Bekanntheitsgrad!

Lösung:

Zunächst definieren wir eine View, die für jeden Studenten alle seine Bekannten auflistet. Anschließend müssen wir diese Bekannten zählen, um den Bekanntheitsgrad der Studenten

zu ermitteln. Damit Studenten ohne Bekannte auch im Ergebnis stehen, verwenden wir einen äußeren Join.

```
with Bekannte as (
  select distinct h1.MatrNr as Student, h2.MatrNr as Bekannter
  from hoeren h1, hoeren h2
  where h1.VorlNr = h2.VorlNr
        and h2.MatrNr <> h1.MatrNr
)
select s.MatrNr, s.Name, count(b.Bekannter) as AnzBekannter
from Studenten s left outer join Bekannte b on (s.MatrNr = b.Student)
group by s.MatrNr, s.Name
order by AnzBekannter desc
```

Ohne View sieht die Anfrage entsprechend komplexer aus:

```
select s.MatrNr, s.Name, count(b.Bekannter) as AnzBekannter
from Studenten s left outer join
  (select distinct h1.MatrNr as Student, h2.MatrNr as Bekannter
   from hoeren h1, hoeren h2
   where h1.VorlNr = h2.VorlNr
         and h2.MatrNr <> h1.MatrNr
  ) b on (s.MatrNr = b.Student)
group by s.MatrNr, s.Name
order by AnzBekannter desc
```

Hausaufgabe 4

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'`.

- 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 letzten 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 letzten 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 aktuelle 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
    )

```