TU München, Fakultät für Informatik
Lehrstuhl III: Datenbanksysteme
Prof. Dr. Thomas Neumann

Übung zur Vorlesung *Grundlagen: Datenbanken* im WS14/15

Harald Lang (harald.lang@in.tum.de)
http://www-db.in.tum.de/teaching/ws1415/grundlagen/

Blatt Nr. 6

Tool zum Üben von SQL-Anfragen: `http://hyper-db.com/interface.html`.

Hausaufgabe 1

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 2

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

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

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.

HINWEIS: Für den Fall, dass die beiden Relationen *StudentenGF* und *ProfessorenF* in der oben genannten Webschnittstelle noch nicht existiert sollten, verwenden Sie die folgende Syntax um temporäre Relationen zu erzeugen:

```
with
StudentenGF(MatrNr,Name,Semester,Geschlecht,FakName) as (
  values
    ('24002','Xenokrates','18','M','Philosophie'),
    ('25403','Jonas','12','W','Theologie'),
    ('26120','Fichte','10','W','Philosophie'),
    ('26830','Aristoxenos','8','M','Philosophie'),
    ('27550','Schopenhauer','6','M','Philosophie'),
    ('28106','Carnap','3','W','Physik'),
    ('29120','Theophrastos','2','M','Physik'),
    ('29555','Feuerbach','2','W','Theologie')
),
ProfessorenF(PersNr,Name,Rang,Raum,FakName) as (
  values
    ('2125','Sokrates','C4','226','Philosophie'),
    ('2126','Russel','C4','232','Philosophie'),
    ('2127','Kopernikus','C3','310','Physik'),
    ('2133','Popper','C3','52','Philosophie'),
    ('2134','Augustinus','C3','309','Theologie'),
    ('2136','Curie','C4','36','Physik'),
    ('2137','Kant','C4','7','Philosophie')
)

select * from StudentenGF ...
```

(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 outer 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 1.00 else 0.00 end)) / count(*)
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 hoeren h
                        where h.VorlNr = v.VorlNr
                          and h.MatrNr = s.MatrNr));
```

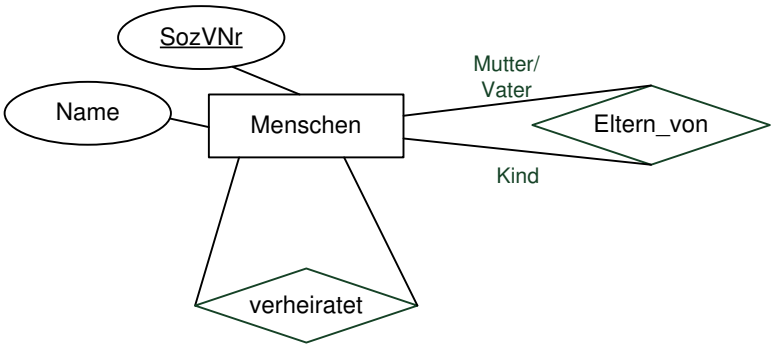
Alternativ:

```
select * from StudentenGF s
where
```

```
(select count(*)
from Vorlesungen v, ProfessorenF p
where v.gelesenVon = p.PersNr and p.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 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**.

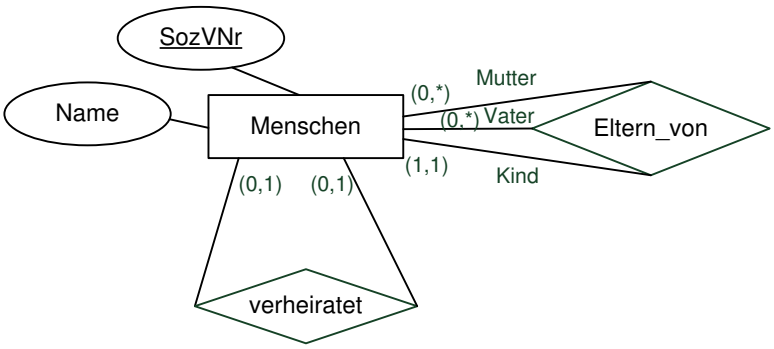
Die folgenden SQL-Statements erzeugen die Tabellen:

```
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)
);
```

Hausaufgabe 4

SQL bietet Datentypen für Festkommazahlen (`DECIMAL(p,s)`) sowie Gleitkommazahlen (`REAL`) mit doppelter Genauigkeit nach IEEE 754. Bei Zweiterem handelt es sich lediglich um eine *approximative* Darstellung von reellen Zahlen.

Konstruieren Sie ein beliebiges Beispiel in SQL, das abhängig von der Verwendung von Festkomma- oder Gleitkommaarithmetik zu verschiedenen Ergebnissen führt.

Lösungsvorschlag von Tutor Moritz Sichert:

a) Beispiel für Absorption:

```
select
  avg(semester*1.0) as avg_semester,
  avg(1000000000000000 + cast(semester as decimal(5,3))) as avg_decimal,
  avg(1000000000000000 + semester*1.0) as avg_implicit_decimal_1,
  avg(1000000000000000 + semester*1.000) as avg_implicit_decimal_3,
  avg(1000000000000000 + cast(semester as real)) as avg_real
from studenten
```

In der Spalte `avg_decimal` steht das korrekte Ergebnis, da für den echten Wert (in der Spalte `avg_semester`) drei Nachkommastellen genau ausreichen.

Bei den Spalten `avg_implict_decimal_1` und `_3` wird der Wert implizit in ein decimal mit jeweils 1 und 3 Nachkommastellen umgewandelt. Deswegen liefert auch nur `avg_implicit_decimal_3` das richtige Ergebnis.

Die Gleitkommaarithmetik hinter `avg_real` liefert ein sehr ungenaues Ergebnis, was auf das Problem der Absorption [1] zurückzuführen ist.

b) Beispiel Auslöschung:

```
select
  1000000000000000.2 - 1000000000000000.1 as implicit_decimal,
  cast(1000000000000000.2 as decimal(16,1)) - cast(1000000000000000.1 as
      decimal(16,1)) as explicit_decimal,
  cast(1000000000000000.2 as real) - cast(1000000000000000.1 as real) as
      diff_real
```

Hier müsste das Ergebnis offensichtlich immer 0.1 sein. Die Subtraktion von Gleitkommanzahlen resultiert hier jedoch aufgrund der Auslöschung [2] zu einem falschen bzw. ungenauen Ergebnis.

Hausaufgabe 5

Gegeben sei eine Relation

$$\mathrm{R} \ : \ \{[\mathrm{A} : \mathrm{integer}, \mathrm{B} : \mathrm{integer}, \mathrm{C} : \mathrm{integer}, \mathrm{D} : \mathrm{integer}, \mathrm{E} : \mathrm{integer}]\},$$

die schon sehr viele Daten enthält (Millionen Tupel). Sie „vermuten“, dass folgendes gilt:

(a) AB ist ein Schlüssel der Relation

(b) $DE \rightarrow B$

Formulieren Sie SQL-Anfragen, die Ihre Vermutungen bestätigen oder widerlegen.

(a) Da jedes Tupel in einer Relation einen eindeutigen Schlüssel besitzt, kann nach der Gruppierung nach A und B anhand der Anzahl der Tupel ermittelt werden, ob hier eine Verletzung der Schlüsseleigenschaft vorliegt. Werden also mindestens zwei Tupel mit den gleichen Werten für A und B als Ergebnis ausgegeben, so bildet AB keinen Schlüssel der Relation, ist das Ergebnis der Anfrage leer, so ist AB ein Schlüssel.

[1] `http://de.wikipedia.org/wiki/Gleitkommazahl#Zahlen_verschiedener_Gr.C3.B6.C3.9Fenordnung_.28engl._absorption.29`

[2] `http://de.wikipedia.org/wiki/Gleitkommazahl#Ausl.C3.B6schung`

```
select A, B
from R
group by A, B
  having count(*) > 1;
```

(b) In diesem Fall muss nur gelten, dass für alle Tupel, die gleiche Werte in D und E besitzen, auch die Werte für das Attribut B gleich sind. D.h. wenn nach D und E gruppiert wird, muss die Anzahl der verschiedenen Werte für B kleiner oder gleich 1 sein. Es gilt wieder, dass das Ergebnis der Anfrage alle Tupel enthält, die die Vermutung verletzen. Ist das Ergebnis leer, so gilt $DE \rightarrow B$.

```
select D, E
from R
group by D, E
  having count(distinct B) > 1;
```

Hausaufgabe 6

Betrachten Sie das Relationenschema

PunkteListe: {[Name, Aufgabe, Max, Erzielt, KlausurSumme, KNote, Bonus, GNote]}

mit der folgenden beispielhaften Ausprägung:

PunkteListe							
Name	Aufgabe	Max	Erzielt	KlausurSumme	KNote	Bonus	GNote
Bond	1	10	4	18	2	ja	1.7
Bond	2	10	10	18	2	ja	1.7
Bond	3	11	4	18	2	ja	1.7
Maier	1	10	4	9	4	nein	4
Maier	2	10	2	9	4	nein	4
Maier	3	11	3	9	4	nein	4

1. Bestimmen Sie die geltenden FDs.
2. Bestimmen Sie die Kandidatenschlüssel.

1. Im Relationenschema gelten die folgenden funktionalen Abhängigkeiten:
   - {KNote, Bonus} → {GNote}
   - {Aufgabe} → {Max}
   - {KlausurSumme} → {KNote}
   - {Name, Aufgabe} → {Erzielt}
   - {Name} → {KlausurSumme, Bonus}

   Natürlich gelten auch alle anderen funktionalen Abhängigkeiten, die mit Hilfe der Armstrong-Axiome daraus hergeleitet werden können.
2. Der Kandidatenschlüssel ist {Name, Aufgabe}. Aus {Name} können die Attribute {KlausurSumme, Bonus}, aus {KlausurSumme} wiederum {KNote}, und aus {KNote, Bonus} dann {GNote} abgeleitet werden. Aus {Aufgabe} kann {Max} abgeleitet werden, und aus {Name, Aufgabe} noch das verbleibende Attribut {Erzielt}.