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

Übung zur Vorlesung *Grundlagen: Datenbanken* im WS17/18

Harald Lang, Linnea Passing (gdb@in.tum.de)
http://www-db.in.tum.de/teaching/ws1718/grundlagen/

Blatt Nr. 07

Tool zum Üben der relationalen Algebra:
`http://db.in.tum.de/people/sites/muehe/ira/`
Tool zum Üben von SQL-Anfragen:
`http://hyper-db.com/interface.html`

Hausaufgabe 1

Gegeben 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

a) 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.

b) Ermitteln Sie die Gesamtdauer, die eine Fahrt von einer gegeben zu jeder anderen Station benötigt.

c) 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?

Lösung:

a) **Erreichbarkeit**

```
with recursive 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 distinct nach
  from huelle
 where von = 'Garching Forschungszentrum'
 order by von;
```

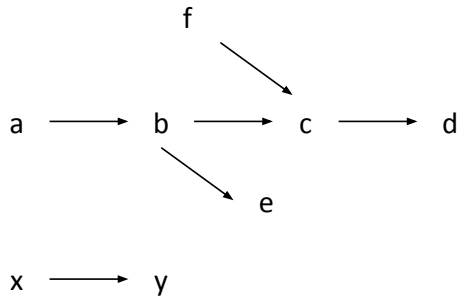
Die `ORDER BY` Klausel ist nicht nötig, macht das Ergebnis aber lesbarer.

b) **Dauer**

```
with recursive 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;
```

c) **"Doppelhülle" (Zusammenhangskomponente)**

Die folgende Lösung berücksichtigt Verzweigungen im U-Bahnnetz sowie mehrere zusammenhangslose Netze (innerhalb einer Relation). Zur Veranschaulichung sei folgendes U-Bahnnetz gegeben – als gerichteter azyklischer Graph (engl. DAG für *directed acyclic graph*):



Um alle in beide Richtungen erreichbaren Stationen zu ermitteln, genügt es nicht die transitive Hülle und deren "Umkehrung" zu betrachten. – Beispielweise wären dann ausgehend von e ausschließlich a und b erreichbar. – Stattdessen wird die Zusammenhangskomponente benötigt, zu welcher die Ausgangsstation (Freimann) gehört. Diese kann ähnlich zur transitiven Hülle rekursiv berechnet werden:

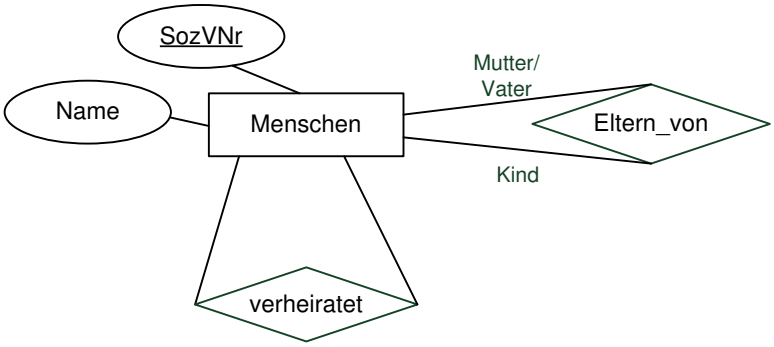
```
with recursive
ubahn(von, nach) as ( -- Basisrelation
  values ('a', 'b'), ('b','c'), ('c','d'), ('b','e'),
         ('f', 'c'),
         ('x', 'y') -- zweite Zusammenhangskomponente
),
undir_ubahn(von, nach) as ( -- ungerichteter Graph
 (select von, nach from ubahn)
  union all
 (select nach, von from ubahn)
),
huelle(von, nach) as (
 (select von, nach from undir_ubahn)
  union -- Mengensemantik
 (select h.von,   u.nach from huelle h, ubahn u
   where h.nach = u.von)
),
zusammenhang(von, nach) as (
 (select nach, von from huelle)
```

```
  union -- Mengensemantik
 (select h.von,   z.nach from huelle h, zusammenhang z
   where h.nach = z.von)
)
select nach from zusammenhang
where von = 'b' -- Freimann
order by nach;
```

Bildet man die Hülle auf der symmetrischen Relation – also wenn für jedes Tupel (a, b) auch (b, a) in der Relation enthalten ist – mit UNION ALL, so terminiert die Rekursion nicht, da aufgrund der Bag-Semantik keine Duplikate eliminiert werden. UNION hingegen eliminiert Duplikate.

Hausaufgabe 2

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**.

Lösung:

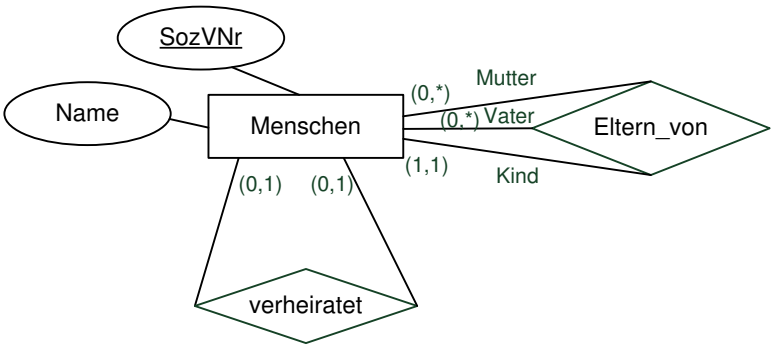
Die folgenden SQL-Statements erzeugen die Tabellen:

```
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 3

Gegeben sei eine Relation

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

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

(a) AB ist ein Superschlüssel der Relation

(b) $DE \rightarrow B$

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

Lösung:

(a) Durch Gruppierung nach A und B kann 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 jedoch leer, so ist AB ein Superschlüssel.

```
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 4

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.

Lösung:

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}.