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

Übung zur Vorlesung *Grundlagen: Datenbanken* im WS16/17

Harald Lang, Linnea Passing (gdb@in.tum.de)
http://www-db.in.tum.de/teaching/ws1617/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 folgende (erweiterte) Relation `ZehnkampfD` mit Athletennamen und den von ihnen erreichten Punkten in den jeweiligen Zehnkampfdisziplinen:

ZehnkampfD : {Name, Disziplin, 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 der relationalen Algebra,
- im relationalen Tupelkalkül,
- im relationalen Domänenkalkül und
- in SQL.

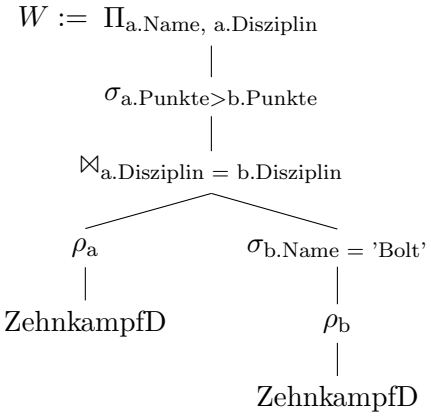
HINWEIS: Beachten Sie, dass die Relation `ZehnkampfD` in der SQL-Webschnittstelle nicht existiert. Verwenden Sie die folgende Syntax um eine temporäre Relationenausprägung zu erzeugen:

```
with zehnkampfd(name,disziplin,punkte) as (
  values
   ('Bolt', '100m', 50),
   ('Bolt', 'Weitsprung', 50),
   ('Eaton', '100m', 40),
   ('Eaton', 'Weitsprung', 60),
   ('Suarez', '100m', 60 ),
   ('Suarez', 'Weitsprung', 60),
   ('Behrenbruch', '100m', 30),
   ('Behrenbruch', 'Weitsprung', 50)
)
select * from zehnkampfd order by disziplin, punkte desc
```

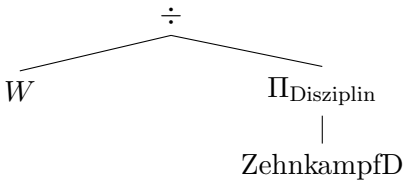
Lösung:

Formulierung in relationaler Algebra

1. Wir ermitteln zunächst alle Wertungen W der Athleten, die eine höhere Punktzahl als *Bolt* erreicht haben:



2. Durch Anwendung des Divisionsoperators bekommen wir diejenigen Athleten, die in *allen* Disziplinen einen höhere Wertung als Bolt haben:



Formulierung im Tupelkalkül

$$
\begin{aligned}
\{[\mathrm{a.Name}] \mid\ & \mathrm{a} \in \mathrm{ZehnkampfD} \wedge \\
& \forall \mathrm{a}' \in \mathrm{ZehnkampfD}(\mathrm{a}'.\mathrm{Name} = \mathrm{a.Name} \\
& \qquad \Rightarrow \\
& \neg \exists \mathrm{b} \in \mathrm{ZehnkampfD}(\mathrm{b.Disziplin} = \mathrm{a}'.\mathrm{Disziplin} \wedge \mathrm{b.Name} = \text{'Bolt'} \wedge \\
& \qquad \mathrm{b.Punkte} \geq \mathrm{a}'.\mathrm{Punkte}) \\
&)\}
\end{aligned}
$$

Formulierung im Domänenkalkül

$$
\begin{aligned}
\{[\mathrm{a}] \mid & \exists \mathrm{d,p}\ ([\mathrm{a,d,p}] \in \mathrm{ZehnkampfD} \wedge \\
& \forall \mathrm{d}', \mathrm{p}'\ ([\mathrm{a}, \mathrm{d}', \mathrm{p}'] \in \mathrm{ZehnkampfD} \\
& \qquad \Rightarrow \\
& \qquad \neg \exists \mathrm{bp}([\text{'Bolt'},\mathrm{d}',\mathrm{bp}] \in \mathrm{ZehnkampfD} \wedge \mathrm{bp} \geq \mathrm{p}') \\
& \quad) \\
&)\}
\end{aligned}
$$

Formulierung in SQL

Da SQL auf dem Tupelkalkül basiert, kann der oben stehende Ausdruck nahezu 1:1 in SQL übersetzt werden. Da SQL allerdings über keine Implikation und über keinen Allquantor verfügt, müssen diese zunächst ersetzt werden. Dazu verwenden wir die beiden Äquivalenzen (i) $a \Rightarrow b \equiv \neg a \vee b$ um Implikationen zu entfernen und (ii) $\forall x(P(x)) \equiv \neg\exists x(\neg P(x))$ um Allquantoren durch Existenzquantoren zu ersetzen.

Im obigen Ausdruck muss also

$$\begin{aligned}&\text{a}'.\text{Name} = \text{a.Name} \Rightarrow\\ &\neg\exists \text{b} \in \text{ZehnkampfD}(\text{b.Disziplin} = \text{a}'.\text{Disziplin} \wedge \text{b.Name} = \text{'Bolt'} \wedge\\ &\qquad\qquad \text{b.Punkte} \geq \text{a}'.\text{Punkte})\end{aligned}$$

umgeformt werden.

Wir entfernen zunächst die Implikation mithilfe der Äquivalenz $a \Rightarrow b \equiv \neg a \vee b$ und erhalten:

$$\begin{aligned}&\text{a}'.\text{Name} \neq \text{a.Name} \vee\\ &\neg\exists \text{b} \in \text{ZehnkampfD}(\text{b.Disziplin} = \text{a}'.\text{Disziplin} \wedge \text{b.Name} = \text{'Bolt'} \wedge\\ &\qquad\qquad \text{b.Punkte} \geq \text{a}'.\text{Punkte})\end{aligned}$$

Da der Ausdruck allquantifiziert ist, wird dieser gemäß (ii) negiert:

$$\begin{aligned}&\text{a}'.\text{Name} = \text{a.Name} \wedge\\ &\exists \text{b} \in \text{ZehnkampfD}(\text{b.Disziplin} = \text{a}'.\text{Disziplin} \wedge \text{b.Name} = \text{'Bolt'} \wedge\\ &\qquad\qquad \text{b.Punkte} \geq \text{a}'.\text{Punkte})\end{aligned}$$

Der vollständige Ausdruck ohne Allquantoren und Implikationen ist dann:

$$\begin{aligned}&\{[\text{a.Name}] \mid \text{a} \in \text{ZehnkampfD} \wedge\\ &\qquad \neg\exists \text{a}' \in \text{ZehnkampfD}(\text{a}'.\text{Name} = \text{a.Name} \wedge\\ &\qquad\qquad \exists \text{b} \in \text{ZehnkampfD}(\text{b.Disziplin} = \text{a}'.\text{Disziplin} \wedge \text{b.Name} = \text{'Bolt'} \wedge\\ &\qquad\qquad\qquad\qquad \text{b.Punkte} \geq \text{a}'.\text{Punkte})\\ &\qquad)\}\end{aligned}$$

Übersetzt in SQL ergibt sich:

```
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
    )
)
```

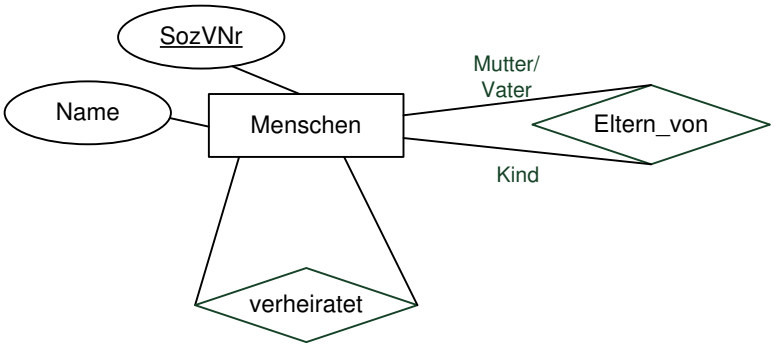
Aufgrund der Multimengensemantik von SQL ist die explizite Angabe von `distinct` erforderlich um Duplikate zu eliminieren.

Alternative Formulierung in SQL 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 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:

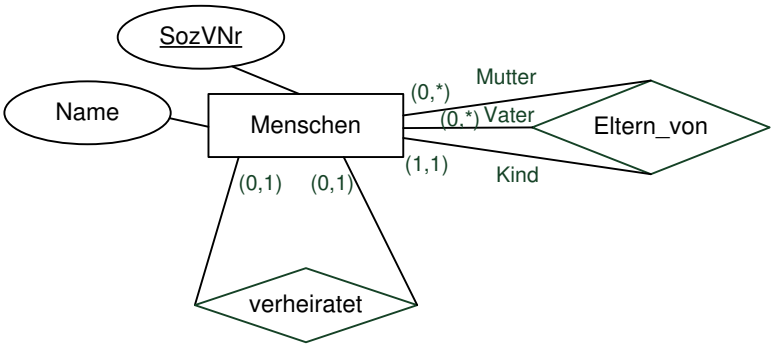
Die folgenden SQL-Statements erzeugen die Tabellen:

```
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}.