Übung zur Vorlesung *Einführung in die Informatik 2 für Ingenieure (MSE)*

Alexander van Renen (renen@in.tum.de)

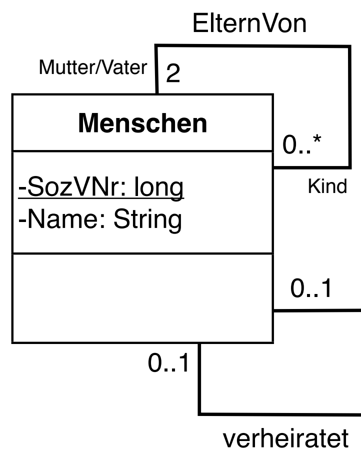
`http://db.in.tum.de/teaching/ss16/ei2/`

Lösungen zu Blatt 11

Tool zum Üben der relationalen Algebra: `http://www-db.in.tum.de/~muehe/ira/`.
SQL-Schnittstelle: `http://hyper-db.com/interface.html`.

Aufgabe 1: SQL als DDL

Gegeben sei das folgende UML-Modell, bei dem wir die Relation *verheiratet* nach dem deutschen Gesetz (d.h. jeder Mensch kann höchstens einen Ehegatten haben) und die Relation *ElternVon* im biologischen Sinn (d.h. jeder Mensch hat genau eine Mutter und einen Vater) modelliert haben:



Bestimmen Sie sinnvolle Funktionalitäts-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 1

Das UML-Diagramm wurde um sinnvolle Funktionalitäts-Angaben erweitert. Die folgenden SQL-Statements erzeugen die Tabellen:

```
create table Menschen (
SozVNr      varchar(30) not null primary key,
Name        varchar(30)
);

create table ElternVon (
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 *ElternVon* 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.

Aufgabe 2: SQL als DML

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

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

Die erweiterten Tabellen sind bereits auf der Webschnittstelle unter

`http://hyper-db.com/interface.html`

angelegt.

Lösung 2

(a) Ermitteln Sie den Männeranteil an den verschiedenen Fakultäten in SQL! Beachten Sie dabei, dass es auch Fakultäten ohne Männer geben kann.

```
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 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 else 0 end)) /
  cast (count(*) as float)
from StudentenGF
group by FakName
```

(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.

```
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));
```

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.

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)
```

Aufgabe 3: Relationenalgebra 1

Beantworten Sie mittels relationaler Algebra.

Lösung 3

(a) Geben Sie einen Ausdruck an, der die Relation $\neg hoeren$ erzeugt. Diese enthält für jeden Studenten und jede Vorlesung, die der Student **nicht** hört einen Eintrag mit Matrikelnummer und Vorlesungsnummer.

$$(\Pi_{MatrNr} Studenten \times \Pi_{VorlNr} Vorlesungen) - hoeren$$

(b) Finden Sie alle Studenten, die keine Vorlesung hören. Geben Sie dabei zwei verschiedene Lösungen an.

$$Studenten \triangleright hoeren$$

oder

$$Studenten - (Studenten \ltimes hoeren)$$

Aufgabe 4: Relationenalgebra 2

Formulieren Sie folgende Anfrage auf dem Universitätsschema in der Relationenalgebra:

Finden Sie die *Studenten*, die *Vorlesungen* hören (bzw. gehört haben), für die ihnen die direkten Voraussetzungen fehlen.

Lösung 4

Wir konstruieren eine hypothetische Ausprägung der Relation *hören*, die gelten müsste, wenn alle Studenten alle benötigten Vorgängervorlesungen hören. Von dieser Menge ziehen wir die tatsächliche Ausprägung von hören ab, so dass diejenigen Einträge übrig bleiben, bei denen ein Student die Vorgängervorlesung nicht hört (bzw. gehört hat).

$$\begin{aligned} R \;:=\; & (\rho_{\text{VorlNr}\leftarrow\text{Vorgänger}}(\Pi_{\text{MatrNr,Vorgänger}}(\text{hören} \bowtie_{\text{VorlNr=Nachfolger}} \text{voraussetzen})) \\ & -\text{hören}) \bowtie \text{Studenten} \end{aligned}$$

Abbildung 1 zeigt den zugehörigen Operatorbaum.

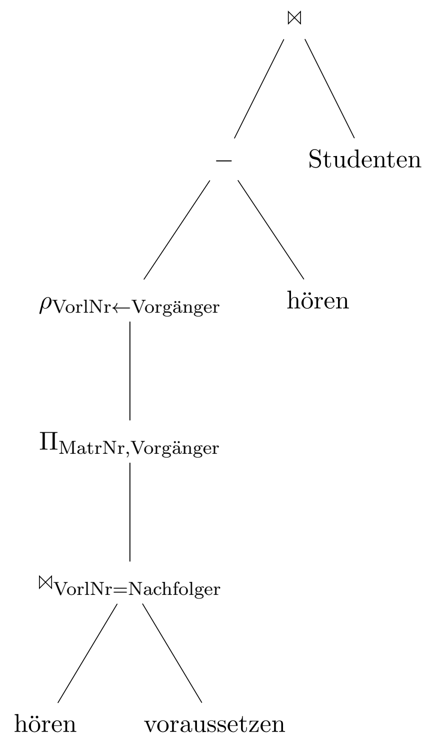


Abbildung 1: Operatorbaum

Optional: Aufgabe 5: SQL

Hinweis: Aufgabe 3 und 4 stellen eine gute Möglichkeit da SQL zu üben.

Lösung 3 in SQL

(a) Die Anfrage kann ähnlich wie in der Relationenalgebra formuliert werden:

```
select s.matrNr, v.vorlNr
from Vorlesungen v, Studenten s
where not exists (select *
                  from hoeren h
                  where h.vorlNr = v.vorlNr
                  and h.matrNr = s.matrNr)
```

(b) Die eine Möglichkeit ist analog zur Relationenalgebra:

```
select *
from Studenten s
where not exists (select *
                  from hoeren h
                  where h.matrNr = s.matrNr)
```

Die andere basiert auf einem **outer join**, da es keinen **anti join** in SQL gibt:

```
select s.*
from Studenten s left outer join hoeren h on s.matrNr = h.
   matrnr
where h.matrNr is null
```

Lösung 4 in SQL

Analog zur Lösung in der Relationenalgebra:

```
select distinct s.*
from hoeren h1, voraussetzen v, Studenten s
where h1.vorlNr = v.nachfolger
and s.matrNr = h1.matrNr
and not exists (select *
                from hoeren h2
                where h2.matrNr = h1.matrNr
                and h2.vorlNr = v.vorgaenger)
```

Optional: Aufgabe 6: Outer Join

In der Vorlesung haben wir den **left outer join** kennen gelernt:

```
select *
from Studenten s left outer join
hoeren h on s.matrNr = h.matrNr;
```

Ist es möglich eine semantisch equivalente Anfrage zu formulieren ohne einen **left outer join** zu benutzen (selbverständlich ist auch der **right** und **full outer join** verboten).

Lösung 6

Ja:

```
   select *
   from Studenten s, hoeren h
   where s.matrnr = h.matrnr
union
   select s.*, null, null
   from Studenten s
   where not exists (select *
                     from hoeren h
                     where s.matrnr = h.matrnr)
```