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

Übung zur Vorlesung *Einsatz und Realisierung von Datenbanken* im SoSe23

Alice Rey, Maximilian Bandle, Michael Jungmair (i3erdb@in.tum.de)
http://db.in.tum.de/teaching/ss23/impldb/

Blatt Nr. 05

Hinweise Die Datalogaufgaben können auf `https://souffle.db.in.tum.de/` getestet werden. Auf der Seite kann unter *examples* ein entsprechender Datensatz geladen werden. Die neuen IDB Regeln sollten am Ende der EDB definiert und dann im Query-Eingabefeld abgefragt werden.

Zusätzlich zu der in der Vorlesung vorgestellten Syntax hier noch eine Kurzübersicht der Vergleichsoperatoren: $X < Y, Y > X$ (kleiner, größer), $X <= Y, X >= Y$ (kleiner oder gleich, größer oder gleich), $X = Y, X! = Y$ (gleich, ungleich), $!pred(X, Y)$ (existiert nicht pred(X,Y)).

Gruppenaufgabe 1

Ist folgendes Datalog-Programm stratifiziert?

$$\begin{aligned} p(X,Y) &:- q_1(Y,Z), \neg q_2(Z,X), q_3(X,P). \\ q_2(Z,X) &:- q_4(Z,Y), q_3(Y,X). \\ q_4(Z,Y) &:- p(Z,X), q_3(X,Y). \end{aligned}$$

Ist das Programm sicher – unter der Annahme, dass p, q_1, q_2, q_3, q_4 IDB- oder EDB-Prädikate sind?

Loesung: Vgl. Übungsbuch. Das Programm ist **nicht stratifizert**, aber **sicher**. Es ist nicht stratifiziert, weil q_2 von p abhängt, aber negiert in p vorkommt. Es ist sicher, weil alle Variablen in IDB- oder EDB-Prädikaten gebunden sind.

Zur Wiederholung: Eine Regel ist **sicher** gdw. alle Variablen **eingeschränkt** sind. Variable X ist in einer Regel **eingeschränkt**, falls sie im Rumpf enthalten ist und sie:

- in einem positiven Prädikat vorkommt (nicht Vergleichsprädikat),
- $X = c$ (Konstante) oder
- $X = Y$, wenn Y bereits nachgewiesen ist.

Jede Variable eines negierten Prädikates muss bereits eingeschränkt sein.[a][b]

Ein Datalog-Programm ist **stratifiziert**, wenn in einer Regel p alle negierten Prädikate $not(q_i)$ nicht von p abhängen (kein Zyklus) und alle positiven Prädikate vorher oder unabhängig von der Reihenfolge (wie bei Rekursion) ausgewertet werden. Formal ausgedrückt können wir jedem Prädikat eine sogenannte Stratifikationsnummer zuordnen. Negierte Prädikate müssen eine echt kleinere Stratifikationsnummer besitzen, positive Prädikate eine maximal so große Stratifikationsnummer wie das davon abhängige Prädikat.

[a] `https://www.dbis.informatik.uni-goettingen.de/Teaching/DB/db-datalog.pdf`
[b] `http://pages.cs.wisc.edu/~paris/cs784-s17/lectures/lecture9.pdf`

Hausaufgabe 2

Gehen Sie von folgender kombinierter Fragmentierung der in Abbildung 1 dargestellten Relation *Professoren* aus:

Professoren						
PersNr	Name	Rang	Raum	Fakultät	Gehalt	Steuerklasse
2125	Sokrates	C4	226	Philosophie	85000	1
2126	Russel	C4	232	Philosophie	80000	3
2127	Kopernikus	C3	310	Physik	65000	5
2133	Popper	C3	52	Philosophie	68000	1
2134	Augustinus	C3	309	Theologie	55000	5
2136	Curie	C4	36	Physik	95000	3
2137	Kant	C4	7	Philosophie	98000	1

Abbildung 1: Beispielausprägung der um drei Attribute erweiterten Relation *Professoren*

1. Zuerst erfolgt eine vertikale Fragmentierung in

$$\begin{aligned} \text{ProfVerw} &:= \Pi_{PersNr,Name,Gehalt,Steuerklasse}(\text{Professoren}) \\ \text{Profs} &:= \Pi_{PersNr,Name,Rang,Raum,Fakultt}(\text{Professoren}) \end{aligned}$$

2. Das Fragment Profs wird weiter horizontal fragmentiert in

$$\begin{aligned} \text{TheolProfs} &:= \sigma_{\text{Fakultät = 'Theologie'}}(\text{Profs}) \\ \text{PhysikProfs} &:= \sigma_{\text{Fakultät = 'Physik'}}(\text{Profs}) \\ \text{PhiloProfs} &:= \sigma_{\text{Fakultät = 'Philosophie'}}(\text{Profs}) \end{aligned}$$

Übersetzen Sie aufbauend auf dieser Fragmentierung die folgende SQL-Anfrage in die kanonische Form.

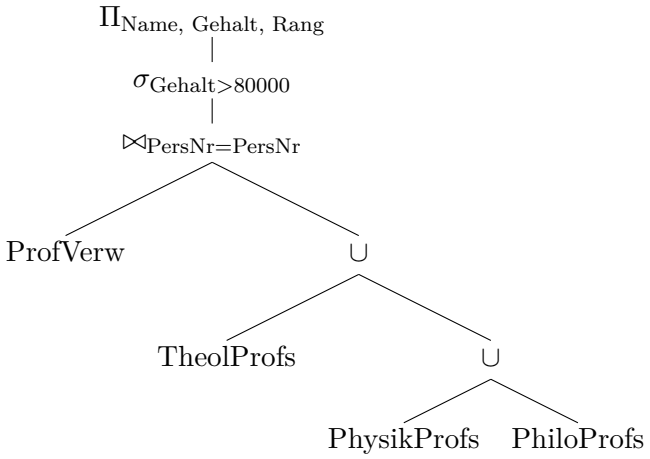
```
select Name, Gehalt Rang
from Professoren
where Gehalt > 80000;
```

Optimieren Sie diesen kanonischen Auswertungsplan durch Anwendung algebraischer Transformationsregeln (Äquivalenzen).

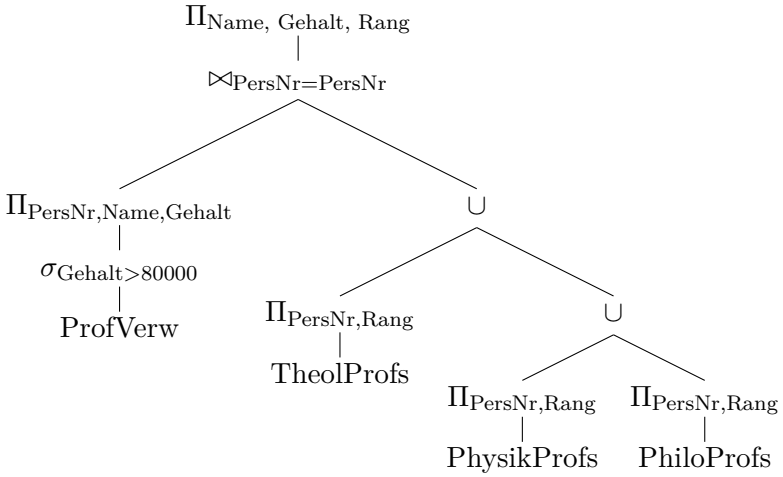
Vgl. Übungsbuch.

$$\Pi_{\text{Name, Gehalt, Rang}}(\Pi_{\text{PersNr,Name, Gehalt}}(\sigma_{\text{Gehalt} > 80000}(\text{ProfVerw})) \bowtie_{\text{PersNr=PersNr}} (\Pi_{\text{PersNr, Rang}}(\text{TheolProfs}) \cup \Pi_{\text{PersNr, Rang}}(\text{PhysikProfs}) \cup \Pi_{\text{PersNr, Rang}}(\text{PhiloProfs})))$$

Der Baum zum kanonischen Auswertungsplan sieht wie folgt aus:



In einem ersten Schritt verschiebt man die Selektion näher an die Datenquellen, die sich nur auf *ProfVerw* bezieht. Anschließend versuchen wir, die Zwischenergebnisse so klein wie möglich zu halten, indem wir zusätzliche Projektionen einfügen. Das Attribut *Name* ist redundant in beiden vertikalen Fragmenten enthalten und wird nicht benötigt.



Hausaufgabe 3

Gegeben sei folgende Relation `Klausur` mit Schlüssel `MatrNr`:

MatrNr	Name	Note	Standort
10101	Philipp	1,0	München
10102	Magdalena	1,0	Garching
10103	Erik	1,0	Garching
10104	Josef	1,0	Garching
10105	Alex	1,0	Garching
10106	Maxmilian	1,0	München

Für eine verteilte Datenbank soll die Tabelle geeignet fragmentiert werden. Ziel ist, Namen mit Standort der Studenten lokal und die Noten getrennt abzuspeichern.

1) Fragmentieren Sie die Relation geeignet *vertikal*.
   a) Geben Sie das Schema für die zwei resultierenden Relationen $KlausurV_1$ und $KlausurV_2$ an. Unterstreichen Sie jeweils den Primärschlüssel.

      $KlausurV_1 : \{[\underline{MatrNr}, Note]\}, KlausurV_2\{[\underline{MatrNr}, Name, Standort]\}$

   b) Geben Sie in SQL-92 die zwei resultierenden Relationen `KlausurV1` und `KlausurV2` als Hilfstabellen (mittels `with`) an.

```
with KlausurV1 as (SELECT MatrNr,Note FROM Personen),
     KlausurV2 as (SELECT MatrNr,Name,Standort FROM Personen)
```

2) Die geeignetere der beiden resultierenden Relationen soll *horizontal* fragmentiert werden.
   a) Geben Sie das Prädikat der Selektion an, mit dem fragmentiert wird.

      Standort='Garching' oder Standort='München'

   b) Geben Sie in SQL-92 die zwei resultierenden Relationen `KlausurH1` und `KlausurH2` als Hilfstabellen (mittels `with`) an.

```
with KlausurH1 as(SELECT * FROM KlausurV2 WHERE Standort='Garching'),
     KlausurH2 as(SELECT * FROM KlausurV2 WHERE Standort<>'Garching')
```

3) Schreiben Sie eine SQL-Abfrage, die die Ursprungsrelation aus den Teilrelationen zusammensetzt.

```
select KlausurV2.*, KlausurV1.Note
from KlausurV1,
  (select * from KlausurH1 union select * from KlausurH2) as KlausurV2
where KlausurV1.MatrNr=KlausurV2.MatrNr
```

Hausaufgabe 4

Für die Rekonstruierbarkeit der Originalrelation R aus vertikalen Fragmenten $R_1, \ldots, R_n$ reicht es eigentlich, wenn Fragmente paarweise einen Schlüsselkandidaten enthalten. Illustrieren Sie, warum es also nicht notwendig ist, dass der Durchschnitt aller Fragmentschemata einen Schlüsselkandidaten enthält. Es muss also nicht unbedingt gelten

$$R_1 \cap \cdots \cap R_n \supseteq \kappa,$$

wobei κ ein Schlüsselkandidat aus R ist.

Geben Sie ein anschauliches Beispiel hierfür – am besten bezogen auf unsere Beispiel-Relation *Professoren*.

Lsg: Vgl. Übungsbuch, Primärschlüssel unterstrichen: RaumF: $\{[\underline{Raum}, Fakultaet]\}$
Professoren: $\{[\underline{PersNr}, Name, Raum]\}$, ProfessorenR: $\{[\underline{PersNr}, Rang]\}$
ProfessorenF: $\{[\underline{PersNr}, Name, Rang, Raum, Fakultaet]\}$
ProfessorenF = RaumF $\bowtie_{\text{Raum=Raum}}$ (Professoren $\bowtie_{\text{PersNr=PersNr}}$ (ProfessorenR))

Gruppenaufgabe 5

Gegeben sei folgende Faktenbasis, die einen direkten azyklischen Graphen (DAG) darstellt.

```
.decl kante(a: number, b: number)
kante(1,2).
kante(2,3).
kante(3,4).
kante(2,5).
kante(5,3).
```

1. Geben Sie in Datalog ein Prädikat `pfad(V,N,L)` an, dass alle möglichen Pfade von `V` nach `N` mit Länge `L` ausgibt.

   ```
   .decl pfad(von: number, nach: number, laenge: number)


   pfad(V,N,L) :- kante(V,N), L=1.
   pfad(V,N,L) :- pfad(V,X,LX), kante(X,N), L=LX+1.
   ```

2. Geben Sie nun das Prädikat `kuerzestePfade(V,N,L)` an, das pro Beginn `V` und Ziel `N` nur den kürzesten Pfad ausgibt.

   ```
   .decl langepfade(von: number, nach: number, laenge: number)
   langepfade(V,N,L2) :- pfad(V,N,L), pfad(V,N,L2), L<L2.
   .decl kuerzestePfade(von: number, nach: number, laenge: number)
   kuerzestePfade(V,N,L) :- pfad(V,N,L), !langepfade(V,N,L).
   .output kuerzestePfade
   ```

3. Bestimmen Sie nun den längsten kürzesten Pfad `.decl laengsterkuerzesterPfad(L: number)`.

   ```
   .decl kurzekuerzestePfade(L: number)
   kurzekuerzestePfade(L) :- kuerzestePfade(_,_,L), kuerzestePfade(_,_,L2), L<L2.
   laengsterkuerzesterPfad(L) :- kuerzestePfade(_,_,L), !kurzekuerzestePfade(L).
   .output laengsterkuerzesterPfad
   ```

4. Erstellen Sie in SQL eine rekursive CTE `pfad(V,N,L)`, die die Länge aller Pfade im DAG ausgibt.

   ```
   with recursive kante(V,N) as (values (1,2), (2,3), (3,4),(2,5),(5,3)),
     pfad(V,N,L) as ( select k.V, k.N, 1 from kante k union
      select k.V, p.N, p.L + 1 from kante k, pfad p where k.N = p.V)
   ```

5. Basierend auf `pfad(V,N,L)`, geben Sie die Länge des längsten kürzesten Pfades aus.

   ```
   select max(min) from (select v,n,min(l) from pfad group by v,n) tmp;
   ```

Hausaufgabe (wird nicht in der Übung besprochen)

Schreiben Sie zu dem U-Bahn-Netz-Beispiel auf der Datalog Seite (unter Examples) folgende Anfragen in Datalog:

1. Erstellen Sie den Stationsplan für den U-Bahnhof Fröttmanning, der alle Stationen, die ohne Umstieg erreichbar sind, auflistet.

```
bidirekt(A,B,L) :- direkt(A,B,L), A!=B.
bidirekt(A,B,L) :- direkt(B,A,L), A!=B.
bidirekt(A,B,L) :- bidirekt(A,X,L), direkt(X,B,L), A!=B.

.decl froettmanning_direkt(B: symbol)
froettmanning_direkt :- bidirekt("froettmanning",B,_)
```

2. Erstellen Sie für Garching-Forschungszentrum einen Plan, der alle erreichbaren Stationen, die minimale Anzahl an Umstiegen und Stops auflistet. Beschreiben Sie Ihren Ansatz ausführlich.

Vereinfachte Lösung (betrachten nur in Fahrtrichtung)

```
.decl aufwand(von: symbol, nach: symbol, linie: symbol,
    stopps: number, umstiege: number)
% Erreichbar naechster Stop
aufwand(A,B,L,S,U) :- direkt(A,B,L), S=0, U=0, A="garching_forschungszentrum".
% Erreichbar auf gleicher Linie
aufwand(A,B,L,S,U) :- aufwand(A,C,L,SX,UX), direkt(C,B,L), S=SX+1, U=UX.
% Erreichbar durch umsteigen
aufwand(A,B,L,S,U) :- aufwand(A,C,LA,SX,UX), direkt(C,B,LB),
   S=SX+1, LA!=LB, L=LB, U=UX+1.
```

Lösung mit Richtungs- und Linienwechsel.

```
% Merke Richtung in die gefahren wird (R=vorwaerts oder rueckwaerts)
.decl bdirekt(von: symbol, nach: symbol, linie: symbol, richtung: symbol)
bdirekt(A,B,L,R) :- direkt(A,B,L), R="v".
bdirekt(A,B,L,R) :- direkt(B,A,L), R="r".

% Maximale Anzahl der Stopps und maximale Anzahl der Umstiege, ist noetig
% falls die Rekursion in einem Kreis im Graph festhaengt.
% Ohne Aggregation einfach 49 statt SMAX bei aufwand(...) einsetzen
.decl smax(x: number)
smax(SMAX) :- SMAX = count: direkt(_,_,_).
.decl linien(x : symbol)
linien(x) :- direkt(_,_,x).
.decl umax(x: number)
umax(UMAX) :- UMAX = count: linien(_).
% Erreichbar naechster Stop
.decl aufwand(von: symbol, nach: symbol, linie: symbol, richtung: symbol,
    stopps: number, umstiege: number)
aufwand(A,B,L,R,S,U) :- bdirekt(A,B,L,R), S=1, U=0, A!=B.
% Erreichbar auf gleicher Linie
aufwand(A,B,L,R,S,U) :- aufwand(A,C,L,R,SX,UX), bdirekt(C,B,L,R),
   S=SX+1, U=UX, A!=B, smax(SMAX), S<SMAX.
% Erreichbar durch Umsteigen auf andere Linien.
% Richtungswechsel erlaubt.
% Maximal 5 Umstiege möglich, da nur 6 verschiedene Linien
aufwand(A,B,L,R,S,U) :- aufwand(A,C,LA,_,SX,UX), bdirekt(C,B,LB,R),
   S=SX+1, LA!=LB, L=LB, U=UX+1, A!=B, umax(UMAX), U<UMAX, smax(SMAX), S<SMAX.
```

Im *aufwand* Prädikat ist ein Richtungswechsel ohne gleichzeitigen Linienwechsel nicht berücksichtigt. Dies ist auch nicht notwendig, da am Startpunkt durch *bdirekt* in beide Richtungen gestartet werden kann, und bei einem Linienwechsel dann auch jedesmal die Möglichkeit besteht, die Richtung frei zu wählen.
Wegen der Struktur der Linien im Beispielgraph (sie treffen sich nur am Sendlinger Tor) ist die Loesung des *aufwand* Prädikats schon jeweils die kürzeste Strecke. Bei einem Netz mit zwei Treffpunkten wären durch die Richtungswechsel Kreise möglich und das Minimum für jede Strecke müsste mit folgendem Prädikat gefunden werden.

```
.decl minaufwand(von: symbol, nach: symbol, stopps: number, umstiege: number)
minaufwand(A,B,S,U) :- aufwand(A,B,_,_,S,_),aufwand(A,B,_,_,_,U),
   U = min ux : aufwand(A,B,_,_,_,ux),
   S = min sx : aufwand(A,B,_,_,sx,_).

.decl minaufwand_gf(nach: symbol, stopps: number, umstiege: number)
minaufwand_gf(B,S,U) :- minaufwand("garching_forschungszentrum",B,S,U).
.output minaufwand_gf
```