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

Übung zur Vorlesung *Einsatz und Realisierung von Datenbanken* im SoSe20

Maximilian {Bandle, Schüle}, Josef Schmeißer (i3erdb@in.tum.de)
http://db.in.tum.de/teaching/ss20/impldb/

Blatt Nr. 04

Hinweise Die Datalogaufgaben können auf `https://datalog.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), $X = Y, X \backslash = Y$ (gleich, ungleich), $not(pred(X, Y))$ (existiert nicht pred(X,Y)).

Hausaufgabe 1

Gegeben sei die folgende Segler-Boots-Reservierung Datenbank:

```
%segler(SID,SNAME,EINSTUFUNG,ALTER)
%boot(BID,BNAME,FARBE)
%reservierung(SID,BID,DATUM)
```

Beantworten Sie die folgenden Anfragen in Datalog und testen Sie unter (`http://datalog.db.in.tum.de/`, Examples => Segler-Boots-Reservierung):

1. Geben Sie die Farben aller Boote, die von 'Lubber' reserviert wurden aus.
   ```
   lubber_farbe(F) :- segler(SID,'Lubber',_,_), reservierung(SID,BID,_),
                      boot(BID, _, F).
   ```
2. Geben Sie alle Segler aus, die eine Einstufung von mindestens 8 oder das Boot 103 reserviert haben.
   ```
   a2(SID,N) :- segler(SID,N,R,_), R>=8.
   a2(SID,N) :- segler(SID,N,_,_), reservierung(SID,103,_).
   ```
3. Geben Sie die Namen aller Segler aus, die mindestens zwei Boote reserviert haben.
   ```
   doppelBoot(S) :- segler(SID,S,_,_), reservierung(SID,BIDA,_),
                    reservierung(SID,BIDB,_), BIDA\=BIDB .
   ```
4. Geben Sie alle Segler aus, die noch nie ein rotes Boot reserviert haben.
   ```
   rotReserviert(SID) :- segler(SID,_,_,_), reservierung(SID,BID,_),
                         boot(BID,_,red).
   nichtRot(SID,S) :- segler(SID,S,_,_), not(rotReserviert(SID)).
   ```
5. Geben Sie alle Segler aus, die mehr als 20 Jahre alt sind und kein rotes Boot reserviert haben.
   ```
   rotReserviert(SID) :- segler(SID,_,_,_), reservierung(SID,BID,_),
                         boot(BID,_,red).
   nichtRotAlt(SID,S,A) :- segler(SID,S,_,A), A>20, not(rotReserviert(SID)).
   ```

6. Geben Sie die Ids der Segler aus, deren Einstufung besser als die eines Seglers mit Namen 'Horatio' ist.

   ```
   nichtSchlecht(SID) :- segler(SID,_,R,_), segler(_,'Horatio',RH,_), R > RH.
   ```

7. Geben Sie die Ids der Segler aus, deren Einstufung besser als die aller Segler mit Namen 'Horatio' ist.

   ```
   dochSchlecht(SID) :- segler(SID,_,R,_), segler(_,'Horatio',RH,_), R=<RH.
   nochBesser(SID) :- segler(SID,_,_,_), not(dochSchlecht(SID)).
   ```

8. Geben Sie den Namen und Alter des ältesten Seglers aus.

   ```
   junger(SID) :- segler(SID,_,_,A), segler(_,_,_,AO), A<AO.
   alter(S,A) :- segler(SID,S,_,A), not(junger(SID)).
   ```

Hausaufgabe 2

Gegeben sei die nachfolgende *KindEltern*-Ausprägung für den Stammbaum-Ausschnitt der griechischen Götter und Helden:

KindEltern		
Vater	Mutter	Kind
Zeus	Leto	Apollon
Zeus	Leto	Artemis
Kronos	Rheia	Hades
Zeus	Maia	Hermes
Koios	Phoebe	Leto
Atlas	Pleione	Maia
Kronos	Rheia	Poseidon
Kronos	Rheia	Zeus

```
ke(zeus,leto,apollon).
ke(zeus,leto,artemis).
ke(kronos,rheia,hades).
ke(zeus,maia,hermes).
ke(koios,phoebe,leto).
ke(atlas,pleione,maia).
ke(kronos,rheia,poseidon).
ke(kronos,rheia,zeus).
ke(poseidon, amphitrite, triton). % muss hinzugefuegt werden
```

Formulieren Sie folgende Anfragen in Datalog und testen Sie unter (`http://datalog.db.in.tum.de/`):

a) Bestimmen Sie alle Geschwisterpaare.

```
parent(P,K) :- kindEltern(P,_,K).
parent(P,K) :- kindEltern(_,P,K).

sibling(A,B) :- parent(P,A),parent(P,B),A\=B.
```

b) Ermitteln Sie Paare von Cousins und Cousinen beliebigen Grades. Die Definition finden Sie auf Wikipedia.

```
cousin(A,B) :- parent(PA,A), parent(PB,B), sibling(PA,PB).
cousin(A,B) :- parent(PA,A), parent(PB,B), cousin(PA,PB).
```

c) Geben Sie alle Verwandtschaftspaare an. Überlegen Sie sich eine geeignete Definition von Verwandtschaft und setzen Sie diese in Datalog um.

```
related(A,B) :- sibling(A,B).
related(A,B) :- related(A,C),parent(C,B).
related(A,B) :- related(C,B),parent(C,A).
```

d) Bestimmen Sie alle Nachfahren von Kronos. Formulieren Sie die Anfrage auch in SQL, so dass sie unter PostgreSQL ausführbar ist (online testen unter: `http://sqlfiddle.com` mit der Datenbank PostgreSQL statt MySQL, das Schema Textfeld können sie leer lassen, müssen aber trotzdem auf 'Build Schema' drücken). Sie können die Daten als Common Table Expression definieren und dann nutzen:

```
WITH RECURSIVE
kindEltern(vater,mutter,kind) as (
  VALUES
   ('Zeus', 'Leto', 'Apollon'),
   ('Zeus', 'Leto', 'Artemis'),
   ('Kronos', 'Rheia', 'Hades'),
   ('Zeus', 'Maia', 'Hermes'),
   ('Koios', 'Phoebe', 'Leto'),
   ('Atlas', 'Pleione', 'Maia'),
   ('Kronos', 'Rheia', 'Poseidon'),
   ('Kronos', 'Rheia', 'Zeus')
),
parent(eltern,kind) as (
  select vater, kind from kindEltern UNION
  select mutter, kind from kindEltern
)
select * from parent where eltern='Zeus'
```

Datalog

```
nachfahr(P,N) :- parent(P,N).
nachfahr(P,N) :- nachfahr(P,X),nachfahr(X,N).
```

Alternativ

```
nachfahr(P,N) :- parent(P,N).
nachfahr(P,N) :- nachfahr(P,X),parent(X,N).
```

Anfrage für die Nachfahren von Kronos

```
nachfahr(kronos,X).
```

SQL

```
WITH RECURSIVE
kindEltern(vater,mutter,kind) as (
  VALUES
   ('Zeus', 'Leto', 'Apollon'),
   ('Zeus', 'Leto', 'Artemis'),
   ('Kronos', 'Rheia', 'Hades'),
   ('Zeus', 'Maia', 'Hermes'),
   ('Koios', 'Phoebe', 'Leto'),
   ('Atlas', 'Pleione', 'Maia'),
   ('Kronos', 'Rheia', 'Poseidon'),
   ('Kronos', 'Rheia', 'Zeus')
),
parent(eltern, kind) as(
  select vater, kind from kindEltern UNION
  select mutter, kind from kindEltern
),
nachfahren(person, nachfahre) AS (
  SELECT * from parent
UNION ALL
  SELECT n.person, p.kind FROM nachfahren n, parent p
  WHERE p.eltern = n.nachfahre
)
select * from nachfahren WHERE person='Kronos'
```

Hausaufgabe 3

Bleiben wir bei dem bekannten Universitätsschema:

```
Assistenten(PersNr, Name, Fachgebiet, Boss)
hoeren(MatrNr, VorlNr)
pruefen(MatrNr,VorlNr, PersNr, Note)
Vorlesungen(VorlNr, Titel, SWS, gelesenVon)
Professoren(PersNr,Name,Rang, Raum)
voraussetzen(Vorg,Nachf)
Studenten(MatrNr, Name, Semester)
```

Formulieren Sie folgende Anfragen in Datalog und testen Sie sie:

a) Geben Sie alle *Professoren* an, die mindestens eine Prüfung abgehalten haben.

```
pruefendeProfs(NAME) :- professoren(PNr,NAME,_,_),pruefen(_,_,PNr,_)
```

b) Übersetzen Sie folgenden Ausdruck des Domänenkalküls in Datalog. Machen Sie sich der Bedeutung des Ausdrucks bewusst.

$$\{[t] \mid \exists\ v,s,g([v,t,s,g] \in \text{Vorlesungen} \wedge \exists\ v2([v,v2] \in \text{voraussetzen} \wedge \exists\ s2,g2([v2,\text{'Wissenschaftstheorie'},s2,g2] \in \text{Vorlesungen})))\}$$

Es sind die Titel der direkten Voraussetzungen für die *Vorlesung* Wissenschaftstheorie.

```
vorWi(Titel) :- vorlesungen(V,Titel,_,_),voraussetzen(V,V2),
                        vorlesungen(V2,'wissenschaftstheorie',_,_).
```

c) Joinen Sie nachfolgende Datalog-Anfrage so, dass Titel ausgegeben werden. Was bedeutet diese Anfrage?

```
geschwisterVL(N1,N2):-voraussetzen(V,N1),voraussetzen(V,N2), N1<N2.
nahverwandtVL(N1,N2):-geschwisterVL(N1,N2).
nahverwandtVL(N1,N2):-geschwisterVL(M1,M2),voraussetzen(M1,N1),
                                    voraussetzen(M2,N2).
```

Es sind entweder „Geschwistervorlesungen“ (ein selber Vorfahre) oder „Vettern“ und „Basen“ (Cousins/Cousinen).

```
nvVT(Titel1, Titel2) :- vorlesungen(N1,Titel1,_,_),
                vorlesungen(N2,Titel2,_,_),naheverwandteVL(N1,N2).
```

Hausaufgabe 4

Geben Sie Datalog Regeln an, die Studenten (Namen angeben) finden, die von einem Prüfer geprüft worden, der selbst nicht die geprüfte Vorlesung gehalten hat. Das korrekte Ergebnis für diese Anfrage ist `Russel`s Prüfling, `Carnap`. Führen Sie die Anfrage im Datalog Tool aus!

```
fremdgeprueft(SN,PID,VPID) :-
    studenten(SID,SN,_), pruefen(SID,V,PID,_),
    vorlesungen(V,_,_,VPID), PID\=VPID.
```

Hausaufgabe 5

U-Bahnen sind toll! Nehmen wir als Faktenbasis die U-Bahnstationen der Linie U2 Richtung Messestadt West und der Linie U6 Richtung Klinikum Großhadern.

```
direkt(Von, Ziel, Linie).
```

Formulieren Sie ein Datalog-Prädikat, das Ihnen von Garching-Forschungszentrum aus kommend die erreichbaren Stationen inklusiver der Anzahl der Stationen angibt. Testen Sie es!

```
indirekt(A,C,S) :- direkt(A,C,_),S=0.
indirekt(A,C,S) :- indirekt(A,B,R),direkt(B,C,_),S=R+1.
indirekt(garching_forschungszentrum,C,S).
```

Hausaufgabe (wird nicht in der Übung besprochen)

Nun fügen wir der EDB folgende Einträge hinzu[1]:

```
ModelParts(model,partname)
Part(partname,maker)
ConsistsOf(partname,partname)
```

`part` ist hierbei ein Bauteil eines Geräts, `marker` ist der Hersteller des Bauteils. `ModelParts` verbindet ein Modell aus den ursprünglichen Daten mit seinem/seinen Bauteilen. `ConsistsOf` beschreibt die Hierarchische Beziehung zwischen Bauteilen.

'kompaktes' Beispiel:

```
ModelParts(workstation,mainboard-hl7).
ModelParts(workstation,hdd30g).
Part(mainboard-hl7,asuz).
Part(gpu7700,nvidio).
Part(hdd30g,sealgate).
Part(transistor,foxcom).
Part(motor,enginesUnited).
Part(wire,theWireCompany).
Part(magnet,theMagnetCompany).
ConsistsOf(hdd30g,transistor).
ConsistsOf(hdd30g,motor).
ConsistsOf(motor,wire).
ConsistsOf(motor,magnet).
...
```

Beantworten Sie in Datalog:

l) Find all models containing parts made by `sealgate`.

```
consistsOfRec(PARTNAME1,PARTNAME2) :-
   consistsOf(PARTNAME1,PARTNAME2).
consistsOfRec(PARTNAME1,PARTNAME2) :-
   consistsOf(PARTNAME1,P),consistsOfRec(P,PARTNAME2).

modelPartsRec(MODEL,PARTNAME,PARTMAKER) :-
   modelParts(MODEL,PARTNAME),part(PARTNAME,PARTMAKER).
modelPartsRec(MODEL,PARTNAME,PARTMAKER) :-
   modelParts(MODEL,P),consistsOfRec(P,PARTNAME),
   part(PARTNAME,PARTMAKER).

sealgatemodels(MODEL) :- modelPartsRec(MODEL,_,sealgate).
```

m) Find all models which contain two different parts by the same maker (regardless of where in the hierarchy).

[1] Inspiriert von `http://people.inf.elte.hu/sila/DB1English/exercise06_products.pdf`.

```
twoparts(MODEL) :-
   modelPartsRec(MODEL,PARTNAME1,PARTMAKER),
   modelPartsRec(MODEL,PARTNAME2,PARTMAKER),
   PARTNAME1<PARTNAME2.
```

Hausaufgabe (wird nicht in der Übung besprochen)

Die Produktdaten einer Firma werden in einer deduktiven Datenbank mit folgenden Relationenschema gehalten:

- Bauteil(**Bauteiltyp**, Gewicht, KonstrukteurID)
- besteht_aus(**Bauteil, Komponennte**, Menge)
- Konstrukteur(**KonstrukteurID**,Name,Geburtsdatum)

Die Relation *Bauteil* beschreibt das Gewicht eines Bauteiltyps und gibt den Konstrukteur an, der diesen Bauteiltyp entworfen hat. Die Relation *besteht* gibt an, aus welchen und jeweils wievielen Einzelkomponenten ein Bauteil besteht. In *Konstrukteur* sind die persönlichen Daten zu den Konstrukteuren gespeichert.

Formulieren Sie die folgenden Anfragen in Datalog:

a) Geben Sie alle Bauteile an, aus denen ein Fahrgestell besteht.

b) Geben Sie alle Bauteile an, an denen der Konstrukteur Schmidt direkt oder indirekt (er hat eine Komponente davon entworfen) beteiligt ist.

```
bauteil(auto,1500,kb).
bauteil(fahrgestell,15,ka).
bauteil(b,20,kb).
bauteil(aa,5,kb).
bauteil(ab,5,ka).
bauteil(aaa,1,ka).
bauteil(aab,2,kc).

konstrukteur(ka,phildunphy,1970).
konstrukteur(kb,jaypritchett,1948).
konstrukteur(kc,schmidt,2000).

besteht_aus(auto,fahrgestell).
besteht_aus(fahrgestell,aa,1).
besteht_aus(fahrgestell,ab,1).
besteht_aus(aa,aaa,1).
besteht_aus(aa,aab,1).


% Teil 1
bauteil_huelle(B,B) :- bauteil(B,_,_).
bauteil_huelle(B,K2) :- bauteil_huelle(B,K), besteht_aus(K,K2,_).

% Ergebis ist
% bauteil_huelle(fahrgestellt,_).
%DES-Datalog> bauteil_huelle(fahrgestell,_).
%
%{
%  bauteil_huelle(fahrgestell,aa),
%  bauteil_huelle(fahrgestell,aaa),
%  bauteil_huelle(fahrgestell,aab),
%  bauteil_huelle(fahrgestell,ab),
%  bauteil_huelle(fahrgestell,fahrgestell)
%}
%Info: 5 tuples computed.



% Teil 2
schmidt(B) :- bauteil(B,_,_), bauteil_huelle(B,K),
  bauteil(K,_,KO), konstrukteur(KO,schmidt,_).

% Ergebnis in schmidt(_).
%DES-Datalog> schmidt(_).
%
%{
%  schmidt(aa),
%  schmidt(aab),
%  schmidt(fahrgestell)
%}
%Info: 3 tuples computed.
```

Hausaufgabe (wird nicht in der Übung besprochen)

Gegeben das folgende Schema der EDB[2]:

```
Product(maker, model, type).
PC(model, speed, ram, hd, price).
Laptop(model, speed, ram, hd, screen, price).
Printer(model, color, type, price).
```

Beantworten Sie in Datalog und testen Sie unter (`http://datalog.db.in.tum.de/`):

a) What PC models have a speed of at least 3.00 GHz?

```
fast_pc(X,Y,P) :- pc(X,Y,_,_,P), Y>3.0.
```

b) Which manufacturers make laptops with a hard disk (hd) of at least 100 GB?

```
manuf(M) :- laptop(P,_,_,D,_,_), D > 100, product(M,P,laptop).
```

c) Find the model number and price of products (of any type) made by manufacturer B.

```
b_prod(M,P) :- product(b, M, pc), pc(M,_,_,_,P).
b_prod(M,P) :- product(b, M, laptop), laptop(M,_,_,_,_,P).
b_prod(M,P) :- product(b, M, printer), printer(M,_,_,P).
```

d) Find the model numbers of all color laser printers.

```
printer(M,color,laser,_)
```

e) Find those manufacturers that sell Laptops, but not PC's.

```
laptop_manuf(M) :- product(M,_,laptop), not(product(M,_,pc)).
```

f) Find those hard-disk sizes that occur in two or more PC's.

```
pop_sizes(D) :- pc(M1,_,_,D,_), pc(M2,_,_,D,_), M1\=M2.
```

g) Find those pairs of PC models that have both the same cpu speed and RAM. A pair should be listed only once, e.g., list (i,j) but not (j,i).

```
sim_pc(M1,M2) :- pc(M1,S,R,_,_), pc(M2,S,R,_,_), M1<M2.
```

Hausaufgabe (wird nicht in der Übung besprochen)

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

1. Erstellen Sie den Stationsplan für den U-Bahnhof Fröttmanning, der alle Station, die ohne umsteigen 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.

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.

[2]Inspiriert von `http://people.inf.elte.hu/sila/DB1English/exercise06_products.pdf`.

Vereinfachte Lösung (betrachten nur in Fahrtrichtung)

```
% Erreichbar naechster Stop
aufwand(A,B,L,S,U) :- direkt(A,B,L), S=0, U=0.
% 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)
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 Stops, ist noetig falls die Rekursion
% in einem Kreis im Graph festhaengt.
% Ohne Aggregation einfach 49 statt SMAX bei aufwand(...) einsetzen
smax(MAX):- count(direkt(_,_,_), MAX).

% Erreichbar naechster Stop
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 Linies.
% Richtungswechsel erlaubt.
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, smax(SMAX), S<SMAX.
```

Im *aufwand* Praedikat ist ein Richtungswechsel ohne gleichzeitigen Linienwechsel nicht beruecksichtigt. Dies ist auch nicht notwendig, da am Startpunkt durch *bdirekt* in beide Richtungen gestartet werden kann, und bei einem Linienwechsel dann auch jedesmal die Moeglichkeit besteht die Richtung frei zu waehlen.
Wegen Struktur der Linien im Beispielgraph (sie treffen sich nur am Sendlinger Tor) ist die Loesung des *aufwand* Praedikats schon jeweils die kuerzeste Strecke. Bei einem Netz mit zwei Treffpunkten waeren durch die Richtungswechsel Kreise moeglich und das Minimum fuer jede Strecke muesste mit folgendem Praedikat gefunden werden.

```
minaufwand(A,B,S,U) :- aufwand(A,B,_,_,S,U),
   min(aufwand(A,B,_,_,ST,U), ST,S),
   min(aufwand(A,B,_,_,S,UM),UM,U).

minaufwand(garching_forschungszentrum,B,S,U)
```