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. 04

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)).

Hausaufgabe 1

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
Poseidon	Amphitrite	Triton

```
.decl kindEltern(vater: symbol, mutter: symbol, kind: symbol)
kindEltern("zeus","leto","apollon").
kindEltern("zeus","leto","artemis").
kindEltern("zeus","maia","hermes").
kindEltern("koios","phoebe","leto").
kindEltern("atlas","pleione","maia").
kindEltern("kronos","rheia","hades").
kindEltern("kronos","rheia","poseidon").
kindEltern("kronos","rheia","zeus").
kindEltern("poseidon", "amphitrite", "triton").
```

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

a) Bestimmen Sie alle Geschwisterpaare.

```
.decl parent(elter: symbol, kind: symbol)
parent(p,k) :- kindEltern(p,_,k).
parent(p,k) :- kindEltern(_,p,k).

.decl sibling(s1: symbol, s2: symbol)
sibling(a,b) :- parent(p,a),parent(p,b),a!=b.
.output sibling
```

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

```
.decl cousin(c1: symbol, c2: symbol)
cousin(a,b) :- parent(pa,a), parent(pb,b), sibling(pa,pb).
cousin(a,b) :- parent(pa,a), parent(pb,b), cousin(pa,pb).
.output cousin
```

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

```
.decl related(r1: symbol, r2: symbol)
related(a,b) :- sibling(a,b);parent(a,b);parent(b,a).
related(a,b) :- related(a,c),parent(c,b).
related(a,b) :- related(c,b),parent(c,a).
.output related
```

d) Bestimmen Sie alle Nachfahren von Kronos. Formulieren Sie die Anfrage auch in SQL, so dass sie unter HyPer ausführbar ist (online testen unter: `http://hyper-db.de/interface.html`). 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'),
   ('Poseidon', 'Amphitrite', 'Triton')
), 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

```
.decl nachfahrVonKronos("kronos",x).
nachfahrVonKronos(x) :- nachfahr("kronos", x).
.output nachfahrVonKronos
```

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'),
('Poseidon', 'Amphitrite', 'Triton')
), 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 2

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.

```
.decl pruefendeProfs(name: symbol)

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

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.

```
.decl vorWi(titel: symbol)
vorWi(Titel) :- vorlesungen(V,Titel,_,_),voraussetzen(V,V2),
                    vorlesungen(V2,"wissenschaftstheorie",_,_).
.output vorWi
```

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

```
.decl geschwisterVL(N1: number, N2: number)
.decl nahverwandtVL(N1: number, N2: number)
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).

```
.decl nvVT(titel1: symbol, titel2: symbol)
nvVT(Titel1, Titel2) :- vorlesungen(N1,Titel1,_,_),
             vorlesungen(N2,Titel2,_,_),nahverwandtVL(N1,N2).
.output nvVT
```

Hausaufgabe 3

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 `Russels` Prüfling, `Carnap`. Führen Sie die Anfrage im Datalog Tool aus! .decl fremdgeprueft(SN: symbol, PID: number, VPID: number)

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

Hausaufgabe 4

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

- .decl bauteil(**Bauteiltyp** : symbol, Gewicht: number, KonstrukteurID: symbol)
- besteht_aus(**Bauteil**: symbol, **Komponente**: symbol, Menge: number)
- Konstrukteur(**KonstrukteurID**: symbol, Name: name, Geburtsdatum: name)

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

```
.decl bauteil(bauteiltyp: symbol, gewicht: number, konstrukteurId: symbol)
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").
.output bauteil

.decl konstrukteur(KonstrukteurID: symbol, Name: symbol, Geburtsdatum: number)
konstrukteur("ka","phildunphy",1970).
konstrukteur("kb","jaypritchett",1948).
konstrukteur("kc","schmidt",2000).
.output bauteil

.decl besteht_aus(Bauteil: symbol,Komponente: symbol, Menge: number)
besteht_aus("fahrgestell","aa",1).
besteht_aus("fahrgestell","ab",1).
besteht_aus("aa","aaa",1).
besteht_aus("aa","aab",1).
.output besteht_aus
```

Formulieren Sie die folgenden Anfragen in Datalog:

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

```
.decl bauteil_huelle(b1: symbol, b2: symbol)
bauteil_huelle(b,b) :- bauteil(b,_,_).
bauteil_huelle(b,k2) :- bauteil_huelle(b,k), besteht_aus(k,k2,_).

.decl fahrgestell_teile(teil: symbol)
fahrgestell_teile(t) :- bauteil_huelle("fahrgestell", t).
.output fahrgestell_teile
% Ergebnis
%
% ---------------
% fahrgestell_teile
% teil
% ===============
% fahrgestell
% aa
% ab
% aaa
% aab
% ===============
%
```

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

```
.decl schmidt(bauteil: symbol)
schmidt(B) :- bauteil(B,_,_), bauteil_huelle(B,K), bauteil(K,_,KO),
              konstrukteur(KO,"schmidt",_).
.output schmidt
% Ergebnis
%
% ---------------
% schmidt
% bauteil
% ===============
% fahrgestell
% aa
% aab
% ===============
%
```

Hausaufgabe 5

Definieren Sie das Prädikat `sg(X,Y)` das für "same generation" steht. Zwei Personen gehören zur selben Generation, wenn Sie mindestens je ein Elternteil haben, das derselben Generation angehört.

Verwenden Sie beispielsweise die folgende Ausprägung einer ElternKind Relation. Das erste Element ist hier das Kind, das zweite ein Elternteil.

```
.decl parent(child: symbol, parent: symbol)
parent("c","a").
parent("d","a").
parent("d","b").
parent("e","b").
parent("f","c").
parent("g","c").
parent("h","d").
parent("i","d").
parent("i","e").
parent("f","e").
parent("j","f").
parent("j","h").
parent("k","g").
parent("k","i").
```

a) Definieren Sie das Prädikat in Datalog.

`.decl sg(h1: symbol, h2: symbol)`

Lösung: Vgl. Übungsbuch. / Tutorfolien

```
sg(x,x) :- parent(_,x). // X als Elternteil
sg(x,x) :- parent(x,_). // X als Kind
// X,Y Kind von U und V, U und V gleiche Generation
sg(x,y) :- sg(u,v),parent(x,u),parent(y,v).
.output sg
```

b) Demonstrieren Sie die naive Ausführung des Prädikats.

Schritt	S
Initialisierung	{}
Schritt 1	[c,c], [a,a], [d,d], [b,b], [e,e], [f,f], [g,g], [h,h], [i,i], [j,j], [k,k]
Schritt 2	[c,c], [a,a], [d,d], [b,b], [e,e], [f,f], [g,g], [h,h], [i,i], [j,j], [k,k] [c,d], [d,c], [d,e], [e,d], [f,g], [g,f], [h,i], [i,h], [i,f], [f,i]
Schritt 3	[c,c], [a,a], [d,d], [b,b], [e,e], [f,f], [g,g], [h,h], [i,i], [j,j], [k,k], [c,d], [d,c], [d,e], [e,d], [f,g], [f,h], [f,i], [g,f], [g,h], [g,i], [h,i], [h,i], [h,f], [h,g], [i,h], [i,f], [i,g], [j,k], [k,j]
Schritt 4	liefert keine zusätzlichen Elemente! Algorithmus terminiert

c) Erläutern Sie das Vorgehen bei der seminaiven Auswertung.

Schritt	ΔS
Initialisierung	[c,c], [a,a], [d,d], [b,b], [e,e], [f,f], [g,g], [h,h], [i,i], [j,j], [k,k]
Schritt 1	[c,d], [d,c], [d,e], [e,d], [f,g], [g,f], [h,i], [i,h], [i,f], [f,i]
Schritt 2	[f,h], [g,h], [g,i], [h,f], [h,g], [i,g], [j,k], [k,j]
Schritt 3	bringt keine zusätzlichen Elemente! Algorithmus terminiert

Hausaufgabe (wird nicht in der Übung besprochen)

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

```
.decl modelParts(model: symbol, bauteil: symbol)
.decl part(teil: symbol, hersteller: symbol)
.decl consistsOf(komponente: symbol, bauteil: symbol)
```

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

'kompaktes' Beispiel:

```
modelParts("workstation","mainboardh-l7").
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").
...
```

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

Beantworten Sie in Datalog:

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

```
.decl consistsOfRec(partname1: symbol, partname2: symbol)
consistsOfRec(partname1,partname2) :-
   consistsOf(partname1,partname2).
consistsOfRec(partname1,partname2) :-
   consistsOf(partname1,p),consistsOfRec(p,partname2).

.decl modelPartsRec(model: symbol, partname: symbol, partmaker: symbol)
modelPartsRec(model,partname,partmaker) :-
   modelParts(model,partname),part(partname,partmaker).
modelPartsRec(model,partname,partmaker) :-
   modelParts(model,p),consistsOfRec(p,partname),
   part(partname,partmaker).

.decl sealgatemodels(model: symbol)
sealgatemodels(model) :- modelPartsRec(model,_,"sealgate").

.output sealgatemodels
```

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

```
.decl twoparts(model :symbol)
twoparts(model) :-
   modelPartsRec(model,partname1,partmaker),
   modelPartsRec(model,partname2,partmaker),
   partname1<partname2.
.output twoparts
```

Hausaufgabe (wird nicht in der Übung besprochen)

Gegeben das folgende Schema der EDB[2]:

```
.decl product(maker: symbol, model: symbol, type: symbol)
.decl pc(model: symbol, speed: float, ram: number, hd: number, price: number)
.decl laptop(model: symbol, speed: float, ram: number,
             hd: number, screen: symbol, price: number)
.decl printer(model: symbol, color: symbol, type: symbol, price: number)
```

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

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

```
.decl fast_pc(model: symbol, speed: float, price: number)
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?

```
.decl manufacturers_100GB(maker: symbol)
```

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

```
manufacturers_100GB(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.

```
.decl b_prod(model: symbol, price: number)
b_prod(m,p) :- product(_, m, "pc"), pc(m,_,_,_,p).
b_prod(m,p) :- product(_, m, "laptop"), laptop(m,_,_,_,_,p).
b_prod(m,p) :- product(_, m, "printer"), printer(m,_,_,p).
```

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

```
.decl color_laser_printers(model: symbol)
color_laser_printers(m) :-printer(m,"color","laser",_).
```

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

```
.decl laptop_manuf(maker: symbol)
laptop_manuf(m) :- product(m,_,"laptop"), !product(m,_,"pc").
```

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

```
.decl pop_sizes(hd: number)
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).

```
.decl sim_pc(m1: symbol, m2: symbol)
sim_pc(m1,m2) :- pc(m1,s,r,_,_), pc(m2,s,r,_,_), m1<m2.
```