



## Übung zur Vorlesung *Einsatz und Realisierung von Datenbanksystemen* im SoSe18

Alexander van Renen, Maximilian E. Schüle (i3erdb@in.tum.de)  
<http://db.in.tum.de/teaching/ss18/impldb/>

### Blatt Nr. 04

**Hinweise** Dieses Übungsblatt ist für die nächsten zwei Wochen gültig.

Die Datalogaufgaben können auf <http://datalog.db.in.tum.de/> getestet werden. Auf der Seite könnt Sie dann unter *examples* einen entsprechenden Datensatz laden. An das Ende der EDB könnt ihr dann neue IDB Regeln definieren und diese dann in dem Query Eingabefeld abfragen. Zusätzlich zu dem in der Vorlesung vorgestellten Syntax hier noch eine kurz Übersicht der Vergleichsoperatoren:  $X < Y, Y > X$  (kleiner, größer),  $X = < Y, X > = Y$  (kleiner gleich, größer gleich),  $X = Y, X \setminus = Y$  (gleich, ungleich),  $not(pred(X, Y))$  (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

```
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, amhitrite, 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 2

Bleiben wir bei dem bekannten Universitätsschema:

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

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

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 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!

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

### Hausaufgabe 4

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.

```
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.

```

sg(X,Y) :- parent(Z,X),X=Y. % X als Elternteil
sg(X,Y) :- parent(X,Z),X=Y. % 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).

```

b) Demonstrieren Sie die naive Ausführung des Prädikats.

c) Erläutern Sie das Vorgehen bei der seminaiven Auswertung.

Siehe Übungsbuch

### Gruppenaufgabe 5

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 stratifiziert**, 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 dann:

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

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.

### Hausaufgabe 6

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

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

2. Das Fragment Profs wird weiter horizontal fragmentiert in

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

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

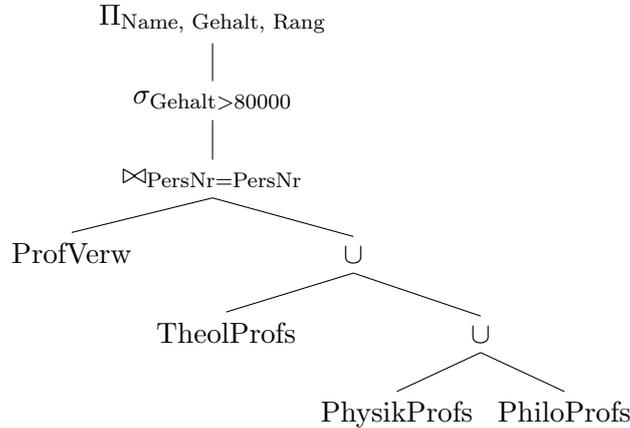
```
select Name, Gehalt, Rang
from Professoren
where Gehalt > 80000;
```

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

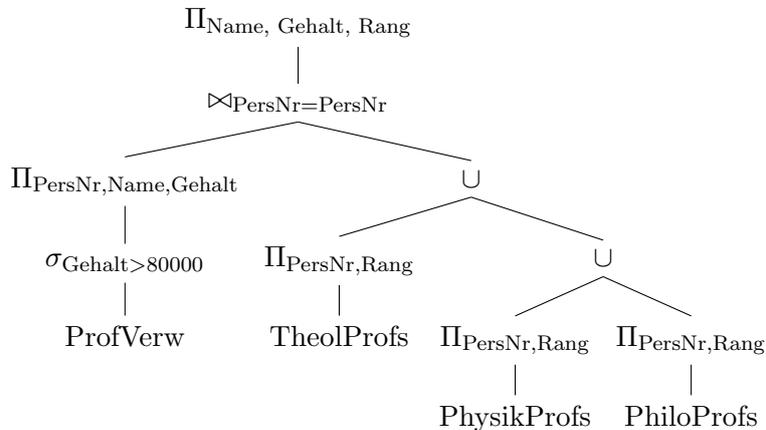
Vgl. Übungsbuch.

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

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 7

Für die Rekonstruierbarkeit der Originalrelation  $R$  aus vertikalen Fragmenten  $R_1, \dots, R_n$  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

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

wobei  $\kappa$  ein Schlüsselkandidat aus  $R$  ist.

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

Siehe Übungsbuch.

### Hausaufgabe (wird nicht in der Übung besprochen)

Beweisen Sie für die Beispielrelationen  $R : \{[A, B, C]\}$  und  $S : \{[C, D, E]\}$  folgende Eigenschaften der Join- / Semi-Join-Operatoren:

$$\begin{aligned}R \bowtie S &= R \bowtie (\Pi_C(R) \bowtie S) \\ R \bowtie S &= (\Pi_C(S) \bowtie R) \bowtie (\Pi_C(R) \bowtie S)\end{aligned}$$

Siehe Übungsbuch

### Hausaufgabe (wird nicht in der Übung besprochen)

Gegeben das folgende Schema der EDB<sup>1</sup>:

```
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/>):

- What PC models have a speed of at least 3.00 GHz?  
`fast_pc(X,Y,P) :- pc(X,Y,_,_,P), Y>3.0.`
- 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).`
- 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).`
- Find the model numbers of all color laser printers.  
`printer(M,color,laser,_)`
- Find those manufacturers that sell Laptops, but not PC's.  
`laptop_manuf(M) :- product(M,_,laptop), not(product(M,_,pc)).`
- Find those hard-disk sizes that occur in two or more PC's.  
`pop_sizes(D) :- pc(M1,_,_,D,_), pc(M2,_,_,D,_), M1\=M2.`
- 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:

<sup>1</sup>Inspiziert von [http://people.inf.elte.hu/sila/DB1English/exercise06\\_products.pdf](http://people.inf.elte.hu/sila/DB1English/exercise06_products.pdf).

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.

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)
```

## Hausaufgabe (wird nicht in der Übung besprochen)

Nun fügen wir der EDB folgende Einträge hinzu<sup>2</sup>:

ModelParts(model,partname)

---

<sup>2</sup>Inspiziert von [http://people.inf.elte.hu/sila/DB1English/exercise06\\_products.pdf](http://people.inf.elte.hu/sila/DB1English/exercise06_products.pdf).

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

part ist hierbei ein Bauteil eines Geräts, maker 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:

- 1) 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).

```
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**, **Komponente**, 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:

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

% Ergebnis ist
% bauteil_huelle(fahrgestell,_).
%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,_,K0), konstrukteur(K0,schmidt,_).

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

```