Einsatz und Realisierung von Datenbanksystemen

ERDB Übungsleitung

Alice Rey, Maximilian Bandle, Michael Jungmair

i3erdb@in.tum.de

Folien erstellt von Maximilian Bandle & Alexander Beischl

Organisatorisches
Disclaimer

Die Folien werden von der Übungsleitung allen Tutoren zur Verfügung gestellt.

Sollte es Unstimmigkeiten zu den Vorlesungsfolien von Prof. Kemper geben, so sind die Folien aus der Vorlesung ausschlaggebend.

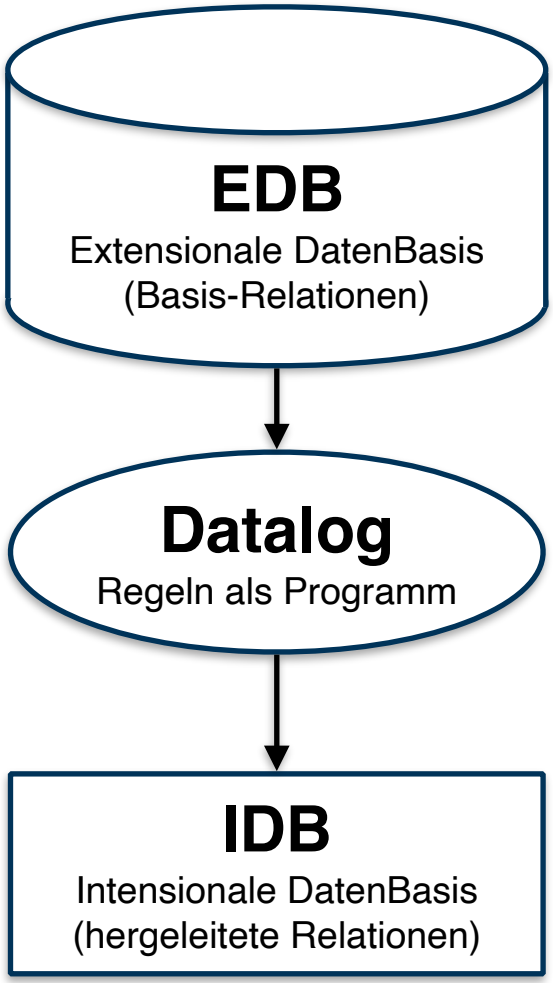
Falls Ihr einen Fehler oder eine Unstimmigkeit findet, schreibt an i3erdb@in.tum.de mit Angabe der Foliennummer.

Deduktive Datenbanken

Deduktive Datenbanken
Einführung

- EDB/Faktenbasis ist die Menge der Relationen
- Deduktion durch Datalog (Data + Prolog -> Datalog)
- Die IDB entsteht durch Anwenden der Datalog-Regeln auf die EDB

➡ Erzeugt weitere Menge von Relationen



Deduktive Datenbanken
Regeln

Deklarationen:

.decl vorlesungen(VorlNr: number, Titel: symbol, SWS: number, PersNr: number)

.decl professoren(PersNr: number, Name: symbol, Rang: symbol, Raum: number)

.decl sokratesVL(Titel: symbol, SWS: symbol)

Basisrelationen:

vorlesungen(5001,"grundzuege",4,2137).

professoren(2125,"sokrates","c4",226).

Regelerzeugung und Join:

sokVL(T,S) :- vorlesungen(_,T,S,P), professoren(P, "sokrates",_,_), S>2.

Deduktive Datenbanken
Syntax

Variablen: Mögliche Typen: **symbol**, **number**, **unsigned**, **float**

Relationen:

- **Deklaration**: .decl relation(feld1: symbol, feld2: number, feld3: float)
- **Fakten**: relation("Wert1", 2, 3.0).
- **Regeln**: relation(Param1, …) **:-** Ausdruck**.** *Punkt immer als Abschluss*

Logische Verknüpfung:

Komma => Und

Semikolon oder Regel mehrfach definieren => Oder

Prädikate: <, =, >, <=, >=, !=

Negation: !relation(Param1, …)

Ausgabe: .output relation

Deduktive Datenbanken
Rekursion

Datenbasis: direkt(Start, Ziel, Linie)

Ziel: indirekt(Start, Ziel, Stops)

1. **Basisfall** => Fülle die Relation mit Anfangswerten

```
indirekt( Start, Ziel, Stops ) :- direkt( Start, Ziel, _), Stops = 0.
```

2. **Rekursion** => Nutze die Relation selbst und erweitere sie

```
indirekt( Start, Ziel, StopsNeu ) :-
        indirekt( Start, Station, Stops ),
        direkt( Station, Ziel, _),
        StopsNeu = Stops + 1.
```

Aufgabe 1

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

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

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

a) Bestimmen Sie alle Geschwisterpaare.

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

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

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:

Aufgabe 1

```
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'
```

Aufgabe 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.

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

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

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).
```

Aufgabe 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!

Aufgabe 4

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.

Aufgabe 5

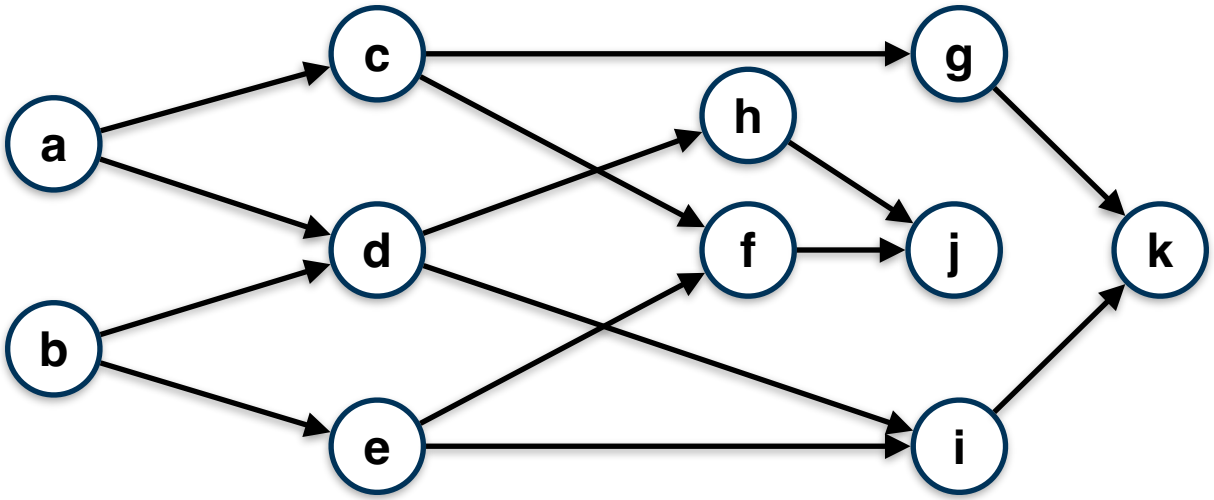
Definieren Sie das Prädikat `sg(X,Y)` das für "same generation" steht. Zwei Personen gehören zur selben Generation, wenn Sie mindestens je ein Elternteil haben, das derselben Generation angehört.

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

a) Definieren Sie das Prädikat in Datalog.

b) Demonstrieren Sie die naive Ausführung des Prädikats.

c) Erläutern Sie das Vorgehen bei der seminaiven Auswertung.




```
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).
```

Aufgabe 5

Naive Auswertung

$S := \{\};$
repeat
$S' := S;$
$S := \Pi_{X,Y}\,(P\,(Z,X) \bowtie_{X=Y} P\,(Z,Y));$
$S := S\,(X,Y) \cup \Pi_{X,Y}\,(P\,(X,Z) \bowtie_{X=Y} P\,(Y,Z));$
$S := S\,(X,Y) \cup \Pi_{X,Y}\,(P\,(X,U) \bowtie (S'\,(U,V) \bowtie P\,(Y,V)));$
until $S' = S$
output $S;$

Aufgabe 5

Semi-naive Auswertung

$S := \{\}; \Delta S := \{\};$
$\Delta S := \Pi_{X,Y}\,(P(Z,X) \bowtie_{X=Y} P(Z,Y));$
$\Delta S := \Delta S(X,Y) \cup \Pi_{X,Y}\,(P(X,Z) \bowtie_{X=Y} P(Y,Z));$
$\Delta S := \Delta S(X,Y) \cup \Pi_{X,Y}\,(P(X,U) \bowtie (S(U,V) \bowtie P(Y,V)));$
$S := \Delta S;$
repeat
  $\Delta S' := \Delta S;$
  $\Delta S := \Pi_{X,Y}\,(P(Z,X) \bowtie_{X=Y} \Delta P(Z,Y))$ /*erste und*/;
    $\cup\, \Pi_{X,Y}\,(\Delta P(Z,X) \bowtie_{X=Y} P(Z,Y))$
    $\cup\, \Pi_{X,Y}\,(P(X,Z) \bowtie_{X=Y} \Delta P(Y,Z))$ /*zweite Regel*/
    $\cup\, \Pi_{X,Y}\,(\Delta P(X,Z) \bowtie_{X=Y} P(Y,Z))$; /*liefern $\emptyset$*/
  $\Delta S := \Delta S$
    $\cup\, \Pi_{X,Y}\,(\Delta P(X,U) \bowtie (S(U,V) \bowtie P(Y,V)))$ /*liefert $\emptyset$*/
    $\cup\, \Pi_{X,Y}\,(P(X,U) \bowtie (\Delta S'(U,V) \bowtie P(Y,V)))$ /*kann $\neq \emptyset$ sein*/
    $\cup\, \Pi_{X,Y}\,(P(X,U) \bowtie (S(U,V) \bowtie \Delta P(Y,V)))$; /*liefert $\emptyset$*/
  $\Delta S := \Delta S - S$; /*entferne Tupel, die schon vorhanden waren*/
  $S := S \cup \Delta S;$
until $\Delta S = \emptyset$

Deduktive Datenbanken
Naive Auswertung der Rekursion

parent <= (K1,K11), (K1,K12), (K11,K111), (K11,K112), (K12,K121), (K12,K122)

verwandte(Vor, Nach) :- parent(Vor, Nach)
verwandte(Vor, Nach) :- verwandte(Vor, Mitte), parent(Mitte, Nach)

Schritt 0: (K1,K11), (K1,K12), (K11,K111), (K11,K112), (K12,K121), (K12,K122)
Schritt 1: (K1,K11), (K1,K12), (K11,K111), (K11,K112), (K12,K121), (K12,K122), (K1, K111), (K1, K112), (K1, K121), (K1, K122)
Schritt 2: (K1,K11), (K1,K12), (K11,K111), (K11,K112), (K12,K121), (K12,K122), (K1, K111), (K1, K112), (K1, K121), (K1, K122)

Deduktive Datenbanken
Semi-Naive Auswertung der Rekursion

parent <= (K1,K11), (K1,K12), (K11,K111), (K11,K112), (K12,K121), (K12,K122)

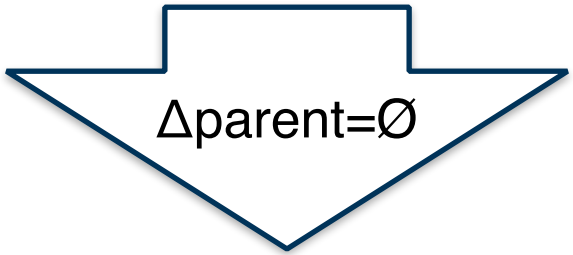
verwandte(Vor, Nach) :- parent(Vor, Nach)
verwandte(Vor, Nach) :- verwandte(Vor, Mitte), parent(Mitte, Nach)

Schritt 0: (K1,K11), (K1,K12), (K11,K111), (K11,K112), (K12,K121), (K12,K122)

Deduktive Datenbanken
Semi-Naive Auswertung der Rekursion

Nur weiteres Auswerten der neu hinzugefügten Knoten (Δ der Relation)

verwandte(Vor, Nach) :- Δparent(Vor, Nach)
verwandte(Vor, Nach) :- Δverwandte(Vor, Mitte), parent(Mitte, Nach)
verwandte(Vor, Nach) :- verwandte(Vor, Mitte), Δparent(Mitte, Nach)



verwandte(Vor, Nach) :- Δverwandte(Vor, Mitte), parent(Mitte, Nach)

Deduktive Datenbanken
Semi-Naive Auswertung der Rekursion

parent <= (K1,K11), (K1,K12), (K11,K111), (K11,K112), (K12,K121), (K12,K122)

verwandte(Vor, Nach) :- Δverwandte(Vor, Mitte), parent(Mitte, Nach)

Schritt 0: (K1,K11), (K1,K12), (K11,K111), (K11,K112), (K12,K121), (K12,K122)
Schritt 1: (K1, K111), (K1, K112), (K1, K121), (K1, K122)
Schritt 2: Ø

Fragen?