Gruppe 3

Tutorübung zu Einsatz und Realisierung von Datenbanksystemen (SS 17)

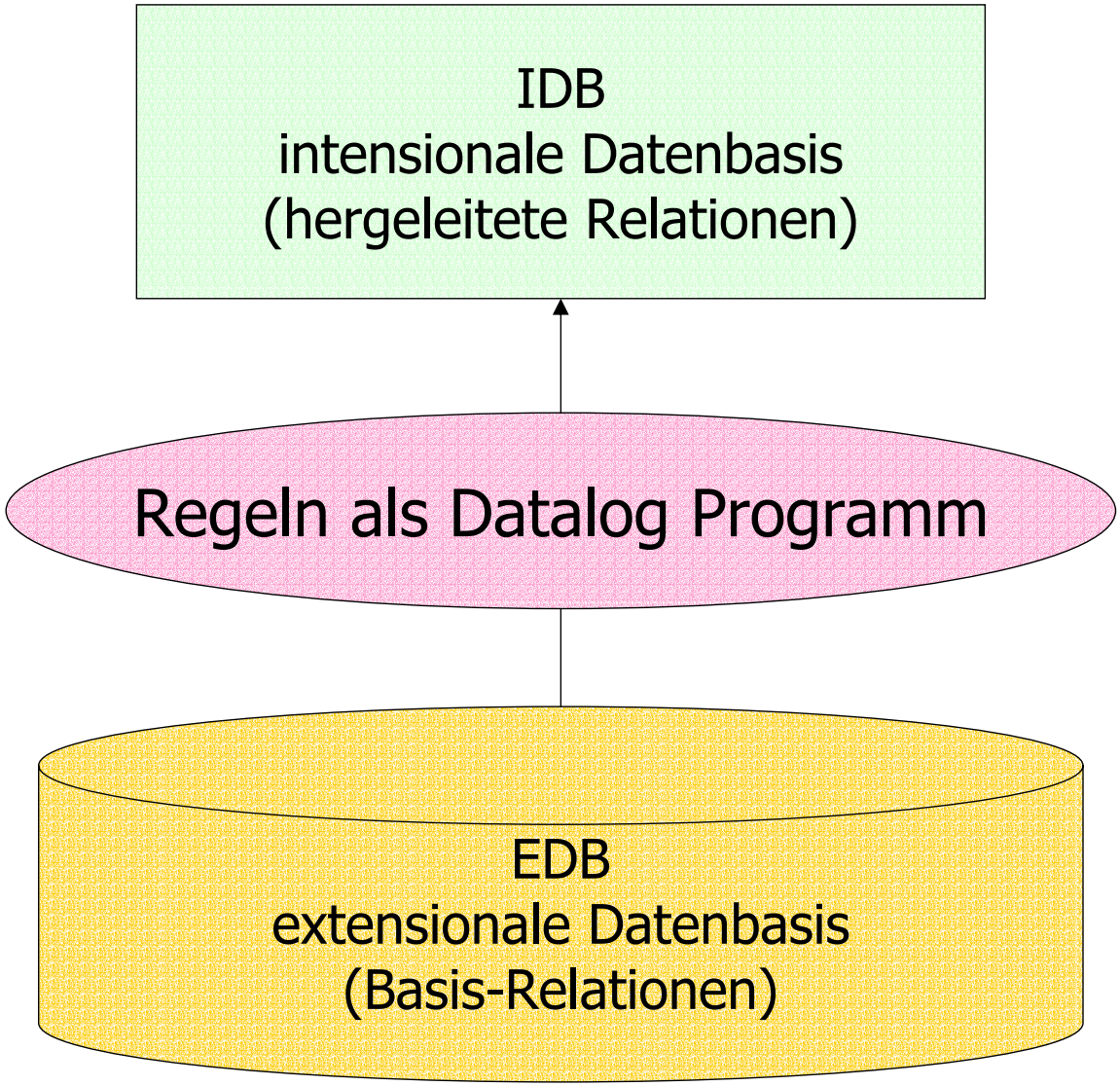
Michael Schwarz

Institut für Informatik
Technische Universität München

24.05.2017

Deduktive Datenbanken

Grundkonzepte einer deduktiven Datenbank



Zusammenfassung

1. Faktenbasis (EDB = extensionale Datenbasis)
2. + logische Herleitungsregeln

$\rightarrow$ Ableitung neuer Fakten (IDB = intensionale Datenbasis)

Datalog Beispiel

EDB

```
% kennt(PersA, PersB, Jahr)
kennt(michael,"Moritz", 2013).
kennt(simon,michael, 2013).
kennt(simon,andreas, 2014).
kennt("Moritz", simon, 2013).
kennt("Moritz", michael, 2013).
```

Herleitungsregeln

```
moritzBekannte(Pers) :- kennt(''Moritz'', Pers, _).
```

IDB

```
moritzBekannte(simon).
moritzBekannte(michael).
```

Datalog Syntax

Atomare Formeln: $q(A_1, ..., A_m).$

Regeln: $p(X_1, ..., X_m) :- q_1(A_{11}, ...A_{1s}), ..., q_n(A_{n1}, ..., A_{nt})$

Variablen: Großbuchstabe + optional weitere Klein-/Großbuchstaben, spezieller Markierung für nicht gebrauchte Werte "_"

Konstanten: Beginnen mit Kleinbuchstaben oder Zahlen

Kommentare: Mit % markiert

Datalog Syntax

Atomare Formeln: $q(A_1, ..., A_m).$

Regeln: $p(X_1, ..., X_m) :- q_1(A_{11}, ...A_{1s}), ..., q_n(A_{n1}, ..., A_{nt})$

Variablen: Großbuchstabe + optional weitere Klein-/Großbuchstaben, spezieller Markierung für nicht gebrauchte Werte "_"

Konstanten: Beginnen mit Kleinbuchstaben oder Zahlen

Kommentare: Mit % markiert

Beispiele

Atomare Formeln: kennt(simon,michael,2013).

Regeln: moritzBekannte(Pers) :- kennt("Moritz",Pers,_).

Variablen: Pers, Jahr, _

Konstanten: "Moritz", simon, 2013

Kommentare: % kennt(A,B,Jahr)

Wichtig: Alle Formeln mit "." beenden!

Datalog Regeln

Konjunktive Prädikate

Wenn mehrere Bedingungen gleichzeitig gelten, können diese mit ”,” verknüpft werden.

verbindungen2014(P1,P2):- kennt(P1,P2,Jahr), Jahr=2014.

Disjunktive Prädikate

Nur eine der Regeldefinitionen muss erfüllt sein, dass ein Wert enthalten ist.

simonMoritzBekannte(P2):- kennt(simon,P2,_).

simonMoritzBekannte(P2):- kennt(”Moritz”,P2,_).

Joins

Datenbasis

% kennt(PersA,PersA,Jahr).

% wohntIn(Pers,Ort).

Regel

```
bekannteImGleichenOrt(A,B,Ort) :-
        kennt(A,B,_), wohntIn(A,Ort), wohntIn(B,Ort).
```

Alle Variablen mit gleichem Namen müssen mit dem selben Wert belegt sein. Somit können die Werte mehrere Prädikate verknüpft werden.

Rekursion

<= 2-hop Freunde

beliebig entfernte Bekannte

Rekursion

<= 2-hop Freunde

zweiHopFreunde(A,B) :- kennt(A,B,_).

zweiHopFreunde(A,C) :- kennt(A,B,_), kennt(B,C,_).

beliebig entfernte Bekannte

Rekursion

<= 2-hop Freunde

zweiHopFreunde(A,B) :- kennt(A,B,_).

zweiHopFreunde(A,C) :- kennt(A,B,_), kennt(B,C,_).

beliebig entfernte Bekannte

bekannte(A,B) :- kennt(A,B,_).

bekannte(A,C) :- bekannte(A,B,_), kennt(B,C,_).

Rekursion

<= 2-hop Freunde

zweiHopFreunde(A,B) :- kennt(A,B,_).

zweiHopFreunde(A,C) :- kennt(A,B,_), kennt(B,C,_).

beliebig entfernte Bekannte

bekannte(A,B) :- kennt(A,B,_).

bekannte(A,C) :- bekannte(A,B,_), kennt(B,C,_).

Datalog erlaubt Rekursion, d.h. Zyklen im Abhängigkeitsgraph. Bei Rekursion werden die Regeln solange ausgewertet, bis keine neuen Fakten mehr abgeleitet werden können.

Naive Auswertung durch Iteration

$$A(V,N) = Vs(V,N) \cup \Pi_{V,N}(A(V,M) \bowtie Vs(M,N))$$

A := {}: /*Initialisierung auf die leere Menge */

repeat

$A' := A;$

$A := Vs(V,N);$ /* erste Regel */

$A := A \cup \Pi_{V,N}(A'(V,M) \bowtie Vs(M,N));$ /* zweite Regel */

until A‘ = A

output A;

Programm zur semi-naiven Auswertung von *aufbauen*

1. $A := \{\ \}; \Delta Vs := \{\ \};$
2. $\Delta A := Vs(V, N);$ /* erste Regel */
3. $\Delta A := \Delta A \cup \Pi_{V,N}(A(V; M) \bowtie Vs(M, N));$ /* zweite Regel */
4. A := ΔA;
5. **repeat**
6. $\Delta A' := \Delta A;$
7. $\Delta A := \Delta Vs(V, N);$ /* erste Regel, liefert∅*/
8. $\Delta A := \Delta A \cup$ /* zweite Regel */
9. $\Pi_{V,N}(\Delta A'(V, M) \bowtie Vs(M, N)) \cup$
10. $\Pi_{V,N}(A(V, M) \bowtie \Delta Vs(M, N));$
11. $\Delta A := \Delta A - A;$ /* entferne "neue" Tupel, die schon vorhanden waren */
12. A := A ∪ ΔA;
13. **until** $\Delta A = \emptyset;$

Negation im Regelrumpf

indirektAufbauen(*V*,*M*) :- aufbauen(*V*,*M*), ¬voraussetzen(*V*,*M*)

Stratifizierte Datalog-Programme

Eine Regel mit einem negierten Prädikat im Rumpf, wie z.B.

$$r \equiv p(\ldots) :- q_1(\ldots), \ldots, \neg q_i(\ldots), \ldots, q_n(\ldots).$$

kann nur dann sinnvoll ausgewertet werden, wenn Q_i schon vollständig materialisiert ist. Also müssen zuerst alle Regeln mit Kopf $q_i(\ldots) :- \ldots$ ausgewertet sein. Das geht nur, wenn q_i nicht abhängig von p ist.

Also darf der Abhängigkeitsgraph keine Pfade von q_i nach p enthalten. Wenn das für alle Regeln der Fall ist, nennt man das Datalog-Programm *stratifiziert.*

Sicherheit von Datalog-Regeln

- Es gibt unsichere Regeln, wie z.B.

  $$\text{ungleich}(X, Y) :- X \neq Y.$$

  Diese definieren unendliche Relationen.
- Eine Datalog-Regel ist sicher, wenn alle Variablen im Kopf beschränkt (range restricted) sind. Dies ist für eine Variable X dann der Fall, wenn:
  - die Variable im Rumpf der Regel in mindestens einem normalen Prädikat – also nicht nur in eingebauten Vergleichsprädikaten – vorkommt oder
  - ein Prädikat der Form X = c mit einer Konstante c im Rumpf der Regel existiert oder
  - ein Prädikat der Form X = Y im Rumpf vorkommt, und man schon nachgewiesen hat, dass Y eingeschränkt ist.