Datenbank-Grundlagen

SS 2005

6. Tupelkalkül und relationale Vollständigkeit
Prof. Dr. Stefan Böttcher
Universität Paderborn

mit Material von
Prof. Dr. Gregor Engels

Beispiele Tupelkalkül

Alle Kunden mit Namen 'Mueller':
$\{ t \mid Kunde(t) \wedge t.Kname = \text{'Mueller'} \}$

gibt alle Attribute von Kunden mit Namen Müller aus.

Die Adressen von Kunden mit Namen 'Mueller':
$\{ t.Adresse \mid Kunde(t) \wedge t.Kname = \text{'Mueller'} \}$

Namen und Adressen von Kunden, die Papier bestellt haben:
$\{ t.Kname, t.Adresse \mid Kunde(t) \wedge \exists b \; (bestellt(b) \wedge b.WName = \text{'Papier'} \wedge t.KName = b.KName) \}$

Erweiterungen in der Notation

Implizite Tupelbildung
analog zum Bereichskalkül:

$\{ u.A1, v.B2 \mid R(u) \wedge S(v) \} =$

$\{ w \mid \exists u \; \exists v: R(u) \wedge S(v) \wedge u.A1 = w[1] \wedge v.B2 = w[2] \}$

Syntax des Tupelkalküls 1 (Atome)

Folgendes sind die Atome des Tupelkalküls:

$t \in R$	mit *t* Tupelvariable und *R* Relation
$t1.A \; \theta \; t2.B$	mit *t1, t2* Tupelvariablen und *A, B* Attribute *t1* muß über *A* und *t2* über *B* definiert sein θ wie in Relationenalgebra: =, ≠, <, >, ≥, ≤
$t.A \; \theta \; c$	mit *t* Tupelvariable, *t* ist über *A* definiert, und $c \in dom(A)$ (*c* hat passenden Typ).

Syntax des Tupelkalküls 2 (Formeln)

Formeln des Tupelkalküls:

Jedes Atom ist eine Formel.

Sind ϕ und $\phi 2$ Formeln,
dann sind auch $\neg\phi$, (ϕ), $\phi \wedge \phi 2$ und $\phi \vee \phi 2$ Formeln.

Ist $\phi(s)$ eine Formel und s eine freie (d.h. noch nicht durch einen Quantor gebundene) Tupelvariable in $\phi(s)$, so sind auch $\forall s : \phi(s)$ und $\exists s : \phi(s)$ Formeln.

Syntax des Tupelkalküls 3 (Anfragen)

Anfragen im Tupelkalkül:

$\{ t \mid \phi(t) \}$	mit *t* Tupelvariable, $\phi(t)$ ist Formel des Tupelkalküls
$\{ p(t1,\dots,tn) \mid \phi(t1,\dots,tn) \}$	mit *t1,…,tn* Tupelvariablen $\phi(t)$ ist Formel des Tupelkalküls p(t1,…,tn) ist eine Liste von projizierten Attributen ti.Ai, wobei ti ∈ {t1,…,tn} und Ai über ti definiert ist

Tupelkalkül im Vergleich zum Bereichskalkül

Anfragen im Tupelkalkül sind wie im Bereichskalkül aufgebaut, mit folgenden Unterschieden:

Variablen sind tupelwertig.
R(t) bedeutet "t ist in Relation R".
Alternative Notation ist $t \in R$.

t.Ai bzw. *t[i]* ermöglichen Zugriff auf die *i*-te Tupelkomponente
Alternative Notation: *t(i)*.

Anfragen haben die Form $\{ t \mid \phi(t) \}$
wobei *t* eine Tupelvariable ist.
Vereinfachte Notation: Beliebige Terme als Zielfunktion

Syntaktisch sichere Anfragen

Anfragen, die syntaktischen Einschränkungen unterliegen,
um die semantische Sicherheit zu erzwingen.

Grundidee:
Jede freie Variable *ti* muss überall in $\phi(t1, \ldots)$ durch positives Auftreten *R(ti)* an endliche Bereiche gebunden werden.
Die Bindung an endliche Bereiche muss für die ganze Bedingung,
also insbesondere für alle Zweige einer Disjunktion, gelten

(Das gilt unter der Annahme, dass Formel in *disjunktiver Normalform* vorliegt, d.h. als Disjunktion von Konjunktionen von positiven und negativen Prädikaten.)

Ausdrucksfähigkeit des Tupelkalküls

Der Tupelkalkül ist **relational vollständig**,
d.h. zu jedem Term τ der Relationenalgebra gibt es
einen äquivalenten (sicheren) Ausdruck A des Tupelkalküls.

Beweis durch Ausdruck aller Grundoperationen der Algebra
(geht ähnlich wie beim Bereichskalkül)

Interpretation am Beispiel

R

bestNr	datum	teil	kunde
21	21.6.04	13	Reich
22	22.6.04	13	Reich

3 verschiedene Schreibweisen für dasselbe

⇕

I : schema(R) --> { true, false }
I (21 , 21.6.04 , 13 , Reich) = true
I (22 , 22.6.04 , 13 , Reich) = true
I (22 , 22.6.04 , 13 , Meier) = false
...

⇕

$\exists t \in R\ \exists u \in R$
t.bestNr=21 $\wedge$ t.datum=21.6.04 $\wedge$ t.teil=13 $\wedge$ t.kunde=Reich
u.bestNr=22 $\wedge$ u.datum=22.6.04 $\wedge$ u.teil=13 $\wedge$ u.kunde=Reich

alles andere (in R nicht gespeicherte) gilt als falsch
(Closed World Assumption)

Interpretation am Beispiel

R

bestNr	datum	teil	kunde
21	21.6.04	13	Reich
22	22.6.04	13	Reich

$P_{(teil,kunde)}R$

teil	kunde
13	Reich
~~13~~	~~Reich~~

⇕ ⇕

I : schema(R) --> { true, false }
I (21 , 21.6.04 , 13 , Reich) = true
I (22 , 22.6.04 , 13 , Reich) = true
I (22 , 22.6.04 , 13 , Meier) = false
...

?

⇕

$\exists t \in R\ \exists u \in R$
t.bestNr=21 $\wedge$ t.datum=21.6.04 $\wedge$ t.teil=13 $\wedge$ t.kunde=Reich
u.bestNr=22 $\wedge$ u.datum=22.6.04 $\wedge$ u.teil=13 $\wedge$ u.kunde=Reich

Interpretation am Beispiel

R

bestNr	datum	teil	kunde
21	21.6.04	13	Reich
22	22.6.04	13	Reich

$P_{(teil,kunde)}R$

teil	kunde
13	Reich
~~13~~	~~Reich~~

⇕ ⇕

I : schema(R) --> { true, false }
I (21 , 21.6.04 , 13 , Reich) = true
I (22 , 22.6.04 , 13 , Reich) = true
I (22 , 22.6.04 , 13 , Meier) = false
...

I : int x char(20) --> { true, false }
I (13 , Reich) = true
~~**I (13 , Reich) = true**~~
I (13 , Meier) = false
...

⇕ ⇕

$\exists t \in R\ \exists u \in R$
t.bestNr=21 $\wedge$ t.datum=21.6.04 $\wedge$ t.teil=13 $\wedge$ t.kunde=Reich
u.bestNr=22 $\wedge$ u.datum=22.6.04 $\wedge$ u.teil=13 $\wedge$ u.kunde=Reich

⇒ **?**

Interpretation am Beispiel

R

bestNr	datum	teil	kunde
21	21.6.04	13	Reich
22	22.6.04	13	Reich

$P_{(teil,kunde)}R$

teil	kunde
13	Reich
~~13~~	~~Reich~~

I : schema(R) --> { true, false }
I (21 , 21.6.04 , 13 , Reich) = true
I (22 , 22.6.04 , 13 , Reich) = true
I (22 , 22.6.04 , 13 , Meier) = false
…

I : int x char(20) --> { true, false }
I (13 , Reich) = true
~~**I (13 , Reich) = true**~~
I (13 , Meier) = false
…

∃t∈R ∃u∈R
t.bestNr=21 ∧ t.datum=21.6.04 ∧ t.teil=13 ∧ t.kunde=Reich
u.bestNr=22 ∧ u.datum=22.6.04 ∧ u.teil=13 ∧ u.kunde=Reich

⇒

∃t∈P(teil,kunde)R
∃u∈P(teil,kunde)R
t.teil=13 ∧ t.kunde=Reich
u.teil=13 ∧ u.kunde=Reich

Anfrage-Auswertung als Beweis,
d.h. Anfrageergebnisse entsprechen abgeleiteten Prädikaten

Semantik der Änderungsoperationen

DAT(S) = Menge aller gültigen Datenbanken zum Schema *S={R1,…,Rp};*
$d \in DAT(S)$ mit *d = {r1, . . . , ri , . . ., rp}* eine Datenbank,
wobei *ri*, die als true interpretierten Tupel aus *Ri* enthält

Die Operation **insert** *t* **into** r_i (R_i) liefert angewandt auf *d*
- $d' := \{r_1, . . . , r_i \cup \{t\}, . . ., r_p\}$ falls $d' \in DAT(S)$
- *d* sonst

Die Operation **delete** *t* **from** r_i (R_i) liefert angewandt auf *d*
- $d' := \{r_1, . . . , r_i - \{t\}, . . ., r_p\}$ falls $d' \in DAT(S)$
- *d* sonst

Die Operation **replace** *t* → *t'* **in** r_i (R_i) liefert angewandt auf *d*
- $d' := \{r_1, . . . , (r_i - \{t\}) \cup \{t'\}, . . ., r_p\}$ falls d' ∈ DAT(S)
- *d* sonst