Vorlesung
Logik und Datenbanken

Nicole Schweikardt

Johann Wolfgang Goethe-Universität Frankfurt am Main

Sommersemester 2008

Relationale Algebra

3.1 Definition und Beispiele
3.2 Anfrageauswertung und Heuristische Optimierung

Relationale Algebra

Grenzen der Ausdrucksstärke konjunktiver Anfragen

Wir haben gesehen:

- konjunktive Anfragen können nur *monotone* Anfragefunktionen beschreiben (Satz **??**) $\rightsquigarrow$ Anfragen mit Negationen der Art

  Welche Regisseure haben noch nie mit “Tom Cruise” gearbeitet?

  können nicht in SPC- bzw. SPJR-Algebra gestellt werden.

- konjunktive Anfragen können keine Ver-ODER-ungen der Art

  In welchen Kinos läuft “Capote” oder “Knallhart”?

  ausdrücken (Übung, Blatt 1, Aufgabe 2(b)).

Jetzt:

Erweitere SPC- bzw. SPJR-Algebra um die Möglichkeit, auch solche Anfragen zu beschreiben.

Vereinigung und Differenz

Operatoren $\cup$ und $-$:

Diese Operatoren können angewendet werden auf Relationen I und J, die dieselbe Sorte bzw. Stelligkeit haben und liefern als Ausgabe die Relationen

$$I \cup J := \{ t : t \in I \text{ oder } t \in J \}$$

bzw.

$$I - J := \{ t \in I : t \notin J \}$$

SPJRU, SPCU und relationale Algebra

Definition 3.1

Sei **R** ein Datenbankschema.

(a) Zur Definition der Klasse der Anfragen der SPJRU[**R**] (bzw. der SPCU[**R**]) werden die Definitionen von SPJR[**R**] (bzw. SPC[**R**]) um die folgende Regel erweitert:

- Sind Q und P zwei SPJRU[**R**]-Anfragen derselben Sorte Σ (bzw. SPCU[**R**]-Anfragen derselben Stelligkeit k), so ist $(Q \cup P)$ eine SPJRU[**R**]-Anfrage der Sorte Σ (bzw. eine SPCU[**R**]-Anfrage der Stelligkeit k).

(b) Zur Definition der Klasse der Anfragen der relationalen Algebra über **R** in der benannten (bzw. der unbenannten) Perspektive werden die Definitionen von SPJRU[**R**] (bzw. SPCU[**R**]) um die folgende Regel erweitert:

- Sind Q und P zwei Anfragen der relationalen Algebra derselben Sorte Σ (bzw. derselben Stelligkeit k), so ist $(Q - P)$ eine Anfrage relationalen Algebra der Sorte Σ (bzw. der Stelligkeit k).

Die Semantik $[\![Q]\!]$ solcher Anfragen Q ist induktiv auf die offensichtliche Art definiert. Mit SPJRU (bzw. SPCU) bezeichnen wir die Klasse aller SPJRU[**R**]-Anfragen (bzw. SPCU[**R**]-Anfragen) für alle Datenbankschemata **R**.

Beispiele

- In welchen Kinos läuft “Capote” oder “Knallhart”?

$$\pi_{\text{Kino}}\left(\sigma_{\text{Titel=”Capote”}}(\mathit{Programm}) \cup \sigma_{\text{Titel=”Knallhart”}}(\mathit{Programm})\right)$$

- Welche Regisseure haben noch nie mit “Tom Cruise” gearbeitet?

$$\pi_{\text{Regie}}(\mathit{Filme}) - \pi_{\text{Regie}}\left(\sigma_{\text{Schauspieler=“Tom Cruise”}}(\mathit{Filme})\right)$$

- Welche derzeit laufenden Filme haben nur Schauspieler, die schon mal in einem Film von “Stephen Spielberg” mitgespielt haben?

$$\pi_{\text{Titel}}(\mathit{Programm}) - \pi_{\text{Titel}}\Big(\mathit{Filme} \bowtie \underbrace{\left(\pi_{\substack{\text{Schau-}\\\text{spieler}}}(\mathit{Filme}) - \pi_{\substack{\text{Schau-}\\\text{spieler}}}\left(\sigma_{\substack{\text{Regie=“Stephen}\\\text{Spielberg”}}}(\mathit{Filme})\right)\right)}_{\text{Schauspieler, die noch nie mit Stephen Spielberg gearbeitet haben}}\Big)$$

Filme mit mind. einem Schauspieler, der noch nie mit Stephen Spielberg gearbeitet hat

Ausdrucksstärke (1/2)

Proposition 3.2

(a) *Jede* SPCU*-Anfrage und jede* SPJRU*-Anfrage ist monoton.*

(b) *Für jede Datenbank* $\mathbf{I}$ *und jede Anfrage* Q *der relationalen Algebra gilt:*

$$adom\big([\![Q]\!](\mathbf{I})\big) \subseteq adom(Q, \mathbf{I}).$$

(c)
$$\begin{array}{ccccc} \text{SPC} & < & \text{SPCU} & < & \textit{relationale Algebra (unbenannte Perspektive)} \\ \equiv & & \equiv & & \equiv \\ \text{SPJR} & < & \text{SPJRU} & < & \textit{relationale Algebra (benannte Perspektive)} \end{array}$$

Beweis:

(a)+(b): Einfache Induktion nach dem Aufbau der Anfragen.

(c): Übung.

Ausdrucksstärke (2/2)

Proposition 3.3

(a) *Benannte Perspektive:*
Keiner der Operatoren σ, π, $\cup$, $-$, $\bowtie$, δ ist redundant.

D.h.: Weglassen jedes einzelnen dieser Operatoren führt zu einer Algebra, die manche in der relationalen Algebra ausdrückbaren Anfragefunktionen nicht beschreiben kann.

(b) *Unbenannte Perspektive:*

(i) *Der Operator σ kann durch Kombination der Operatoren π, $-$, $\times$ ausgedrückt werden.*

Beachte: um dies zu zeigen, muss man nutzen, dass bei der Projektion $\pi_{j_1,\ldots,j_k}$ die Indices j_i nicht paarweise verschieden sein müssen.

(ii) *Keiner der Operatoren π, $\cup$, $-$, $\times$ ist redundant.*

Beweis: Übung.

Theta-Join und Semijoin

Eine positive konjunktive Join-Bedingung ist ein Ausdruck θ der Form $\bigwedge_{\ell=1}^{m} x_{i_\ell}{=}y_{j_\ell}$, für natürliche Zahlen $m \geqslant 0$ und $i_1, \ldots, i_m, j_1, \ldots, j_m \geqslant 1$.

Zwei Tupel $\overline{a} = (a_1, \ldots, a_r) \in \mathbf{dom}^r$ und $\overline{b} = (b_1, \ldots, b_s) \in \mathbf{dom}^s$ mit $r \geqslant \max\{i_1, \ldots, i_m\}$ und $s \geqslant \max\{j_1, \ldots, j_m\}$ erfüllen θ

$$(\text{kurz: } (\overline{a}, \overline{b}) \models \theta, \text{ bzw. } \theta(\overline{a}, \overline{b})),$$

falls für alle $\ell \in \{1, \ldots, m\}$ gilt: $a_{i_\ell} = b_{j_\ell}$.

In der relationalen Algebra (unbenannte Perspektive) lassen sich u.a. die folgenden Operationen ausdrücken:

- Theta-Join $\bowtie_\theta$, wobei θ eine positive konjunktive Join-Bedingung ist.
  Semantik: $I \bowtie_\theta J := \{ (\overline{a}, \overline{b}) : \overline{a} \in I, \overline{b} \in J, \text{ so dass } (\overline{a}, \overline{b}) \models \theta \}$
- Semijoin $\ltimes_\theta$, wobei θ eine positive konjunktive Join-Bedingung ist.
  Semantik: $I \ltimes_\theta J := \{ \overline{a} \in I : \text{ex. } \overline{b} \in J, \text{ so dass } (\overline{a}, \overline{b}) \models \theta \}$

Relationale Algebra

Anfrageauswertung und Heuristische Optimierung

. . . hat viele Aspekte:

- Speicher- und Indexstrukturen
- Betriebssystem
- Seitenersetzungsstrategien
- Statistische Eigenschaften der Daten
- Statistische Informationen über Anfragen
- Implementierung der einzelnen Operatoren
- Ausdrucksstärke der Anfragesprache

Hier: Überblick und einige Teilaspekte.

Details: DBS I+II Vorlesung von Prof. Zicari

Anfrageauswertung allgemein

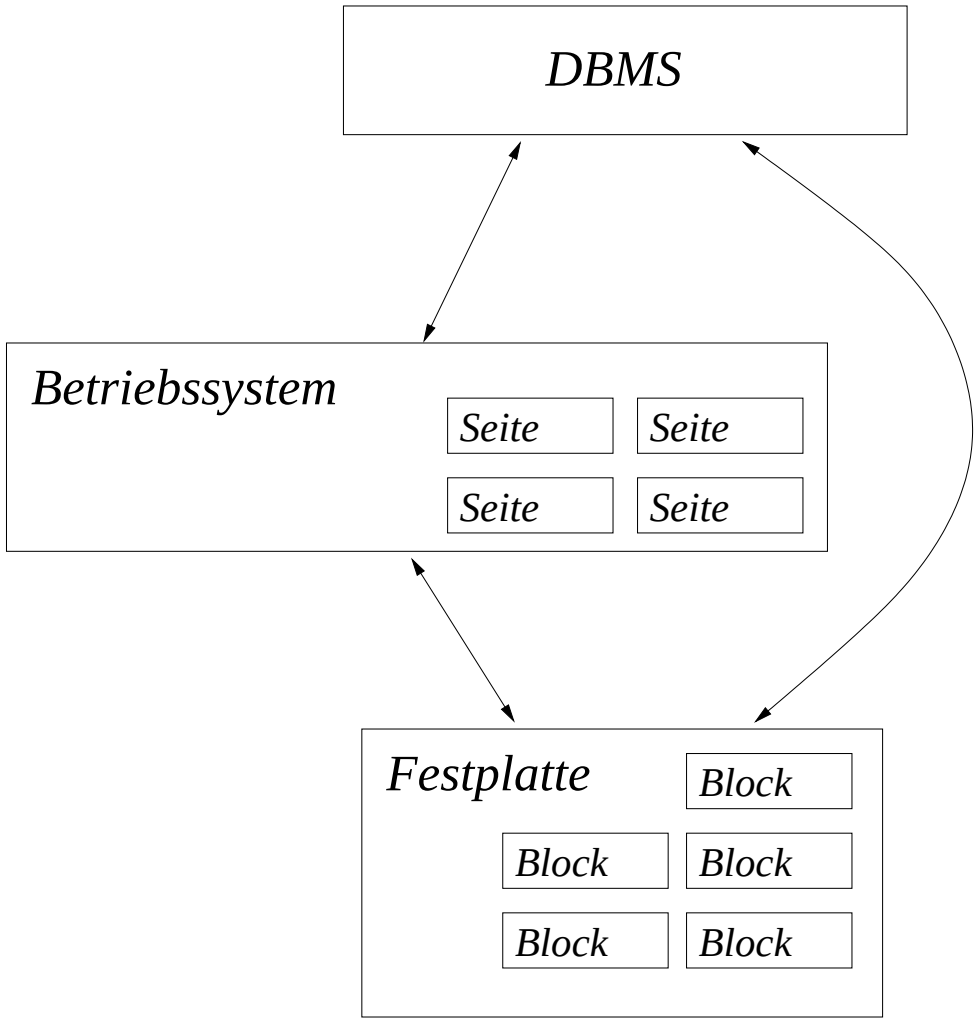
Vorbemerkung:

- Datenbanken sind sehr groß
- werden auf Sekundärspeicher (Festplatte) gespeichert
- Aufwand wird dominiert durch die Anzahl der Plattenzugriffe (“Seitenzugriffe”)
  Denn: In derselben Zeit, die für einen “Random Access” auf der Festplatte benötigt wird, können zigtausende Operationen im Hauptspeicher durchgeführt werden.

Allgemeines Vorgehen:

- durch Erzeugen, Filtern, Manipulieren und Kombinieren von Tupelströmen
- dabei evtl. Verwendung von Indexstrukturen (“Wegweiser”), Hashing und Sortier-Schritten
- wünschenswert: möglichst wenig auf Festplatte zwischenspeichern
- Operationen: Operationen der relationalen Algebra, Sortieren, Duplikatelimination, . . .

Verwaltung des Sekundärspeichers



1 Plattenzugriff $\triangleq$ Lesen eines Blocks bzw. einer Seite

Wichtige Parameter einer Datenbankrelation I

- n_I: Anzahl der Tupel in Relation I
- s_I: (mittlere) Größe eines Tupels aus I
- f_I: Blockungsfaktor ("Wie viele Tupel aus I passen in einen Block?")

$$f_I \approx \frac{\text{Blockgröße}}{s_I}$$

- b_I: Anzahl der Blöcke (Seiten) der Festplatte, die Tupel aus I beinhalten

$$b_I \approx \frac{n_I}{f_I}$$

Lesen eines Blocks (bzw. einer Seite) $\stackrel{\triangle}{=}$ 1 Zugriff auf Platte

Operationen der relationalen Algebra (1/3)

Selektion $\sigma_F(I)$:

- Selektion meist als Filter auf einem Tupelstrom:
  $\mathcal{O}(b_I)$ Zugriffe auf Festplatte; $\mathcal{O}(n_I)$ Schritte insgesamt
- evtl. Verwendung eines Index;
  dann schneller, falls nur sehr wenige Tupel im Ergebnis

Operationen der relationalen Algebra (2/3)

Projektion $\pi_{j_1,\ldots,j_k}(I)$:

- 2 Komponenten:
  - Ändern der einzelnen Tupel (Auswahl und Reihenfolge von Spalten)
  - Duplikatelimination

- Tupeländerung: als Filter auf einem Tupelstrom

  $\mathcal{O}(b_I)$ Zugriffe auf Festplatte; $\mathcal{O}(n_I)$ Schritte insgesamt

  Dabei können Duplikate “entstehen”

- Duplikatelimination:
  - in SQL i.d.R. nicht verlangt (außer bei `SELECT DISTINCT`)
  - sind die Tupel sortiert, so können Duplikate durch einen Scan leicht erkannt werden
  - Sortieren: durch Merge-Sort möglich mit $\mathcal{O}(b_I \cdot \log b_I)$ Plattenzugriffen und $\mathcal{O}(n_I \cdot \log n_I)$ Schritten insgesamt
  - Alternative: Hashing
    - Abbilden der Tupel durch eine Hash-Funktion
    - $\rightsquigarrow$ Duplikate werden auf denselben Wert abgebildet und dadurch erkannt
    - bei idealer Hash-Funktion: lineare Zeit

Operationen der relationalen Algebra (3/3)

Binäre Operationen auf zwei Relationen I und J: $\cup,\ -,\ \times,\ \bowtie_\theta,\ \ltimes_\theta$

- Nested-Loops-Methode: (Schleifeniteration)

  für jedes Tupel $t \in I$ (bzw. jede Seite) wird die gesamte Relation J durchlaufen

  $\mathcal{O}(b_I \cdot b_J)$ Plattenzugriffe; $\mathcal{O}(n_I \cdot n_J)$ Schritte insgesamt

- Merge-Methode: (weniger sinnvoll für $\times$)

  I und J sortiert $\rightsquigarrow$ schrittweise in der vorgegebenen Tupelreihenfolge durchlaufen;
  Für $\ltimes_\theta$: $\mathcal{O}(b_I + b_J)$ Plattenzugriffe; $\mathcal{O}(n_I + n_J)$ Gesamtschritte

  Evtl. vorher nötig: Sortieren von I und/oder J (durch Merge-Sort)
  $\mathcal{O}(b_I \cdot \log b_I)$ und/oder $\mathcal{O}(b_J \cdot \log b_J)$ Plattenzugriffe;
  $\mathcal{O}(n_I \cdot \log n_I)$ und/oder $\mathcal{O}(n_J \cdot \log n_J)$ Gesamtschritte

- Hash-Methode: (weniger sinnvoll für $\times$)

  die kleinere der beiden Relationen in Hash-Tabelle;
  Tupel der zweiten Relation finden ihren Vergleichspartner mittels Hash-Funktion;
  bei idealer Hash-Funktion: Aufwand $\mathcal{O}(n_I + n_J)$

Beispiel für Merge-Technik

Berechne $I \bowtie_\theta J$ für Join-Bedingung $\theta := x_1{=}y_4 \wedge x_2{=}y_1$

1. Sortiere I lexikographisch nach "1-te Spalte; 2-te Spalte"
2. Sortiere J lexikographisch nach "4-te Spalte; 1-te Spalte"
3. Seien t und s die ersten Tupel von I und J
4. Falls $(t_1, t_2) < (s_4, s_1)$, so lies nächstes Tupel t aus I.
5. Falls $(t_1, t_2) > (s_4, s_1)$, so lies nächstes Tupel s aus J.
6. Falls $(t_1, t_2) = (s_4, s_1)$, so gib die Tupel (t, s) und (t, s') für alle Nachfolger s' von s in J mit $(s'_4, s'_1) = (s_4, s_1)$ aus
7. Lies nächstes Tupel t aus I und gehe zu Zeile 4.

Aufwand für Zeilen 3–7:

- falls alle Tupel den gleichen Wert in den Spalten 1,2 bzw. 4,1 haben: ca. $n_I \cdot n_J$ Gesamtschritte
- falls alle Tupel aus J in (s_4, s_1) unterschiedliche Werte haben: ca. $n_I + n_J$ Gesamtschritte :-)
- bei Semijoin $\ltimes_\theta$ statt Theta-Join $\bowtie_\theta$ reichen immer $n_I + n_J$ Gesamtschritte

Anfrageauswertung

Proposition 3.4

Das Auswertungsproblem für die relationale Algebra läßt sich in Zeit $(k+n)^{\mathcal{O}(k)}$ *lösen.*

Beweis:

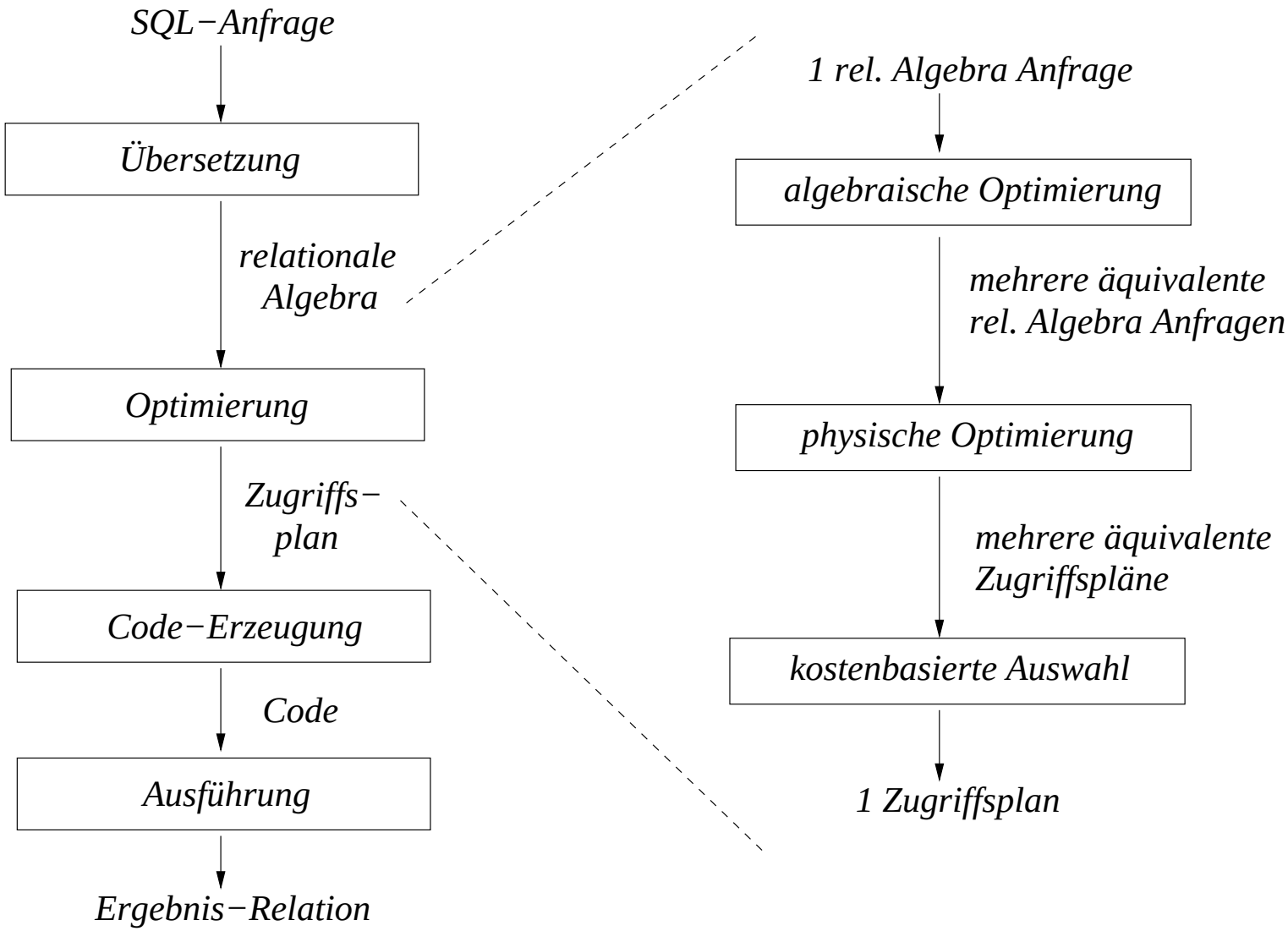
Zeige per Induktion nach dem Aufbau von Anfragen der relationalen Algebra, dass für jede Anfrage Q der Länge k und jede Datenbank $\mathbf{I}$ der Größe n gilt:

(1) $|| [\![Q]\!](\mathbf{I}) || \leqslant (k+n)^k$

(2) Q kann auf $\mathbf{I}$ in $\mathcal{O}\big((k+n)^{2k}\big)$ Elementarschritten ausgewertet werden.

Details: *Übung*.

Anfragebearbeitung durch ein DBMS



Ziel der Optimierung

- möglichst schnelle Auswertung der Anfrage
- möglichst wenige Zugriffe auf Festplatte
- möglichst in allen Operationen so wenig Seiten wie möglich berücksichtigen

Grundregeln:

(1) Selektionen so früh wie möglich

(2) auch Projektionen früh, aber evtl. Duplikatelimination vermeiden

(3) Basisoperationen zusammenfassen und wenn möglich ohne Zwischenspeicherung realisieren
(Bsp: $\bowtie_\theta$ besser als $\times$; $\ltimes_\theta$ besser als $\bowtie_\theta$)

(4) Redundante Operationen oder leere Zwischenrelationen entfernen

(5) Zusammenfassung gleicher Teilausdrücke:
Wiederverwendung von Zwischenergebnissen

Anfrageauswertung an einem Beispiel (1/4)

Anfrage:

Welche Kinos (Name + Adresse) spielen einen Film von “Stephen Spielberg”?

In SQL:

```
SELECT Orte.Kino, Orte.Adresse
FROM Orte, Filme, Programm
WHERE Orte.Kino = Programm.Kino and
      Programm.Titel = Filme.Titel and
      Filme.Regie = “Stephen Spielberg”
```

Direkte Übersetzung in relationale Algebra:

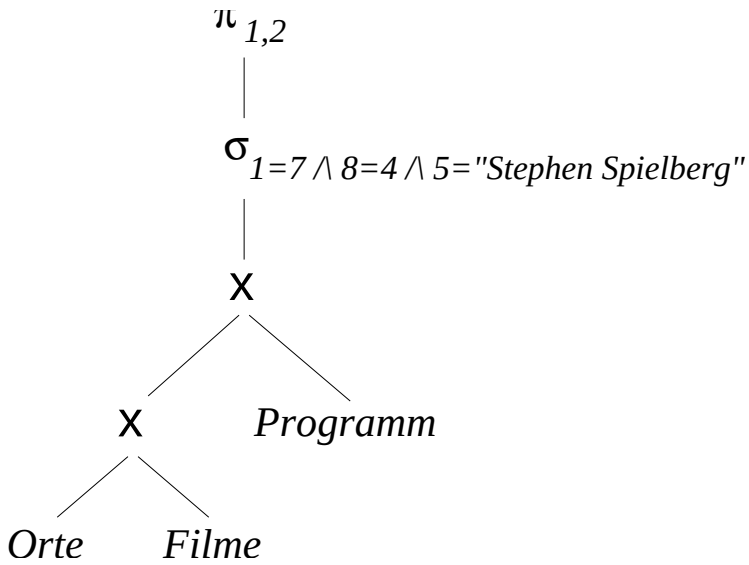
$$\pi_{1,2}\left(\sigma_{\substack{1=7\wedge 8=4\wedge \\ 5=\text{“Stephen Spielberg”}}}\left(\mathit{Orte}\times\mathit{Filme}\times\mathit{Programm}\right)\right)$$

Anfrageauswertung an einem Beispiel (2/4)

Original-Anfrage

$$\pi_{1,2}\left(\sigma_{\substack{1=7\wedge 8=4\wedge \\ 5=\text{“Stephen Spielberg”}}}\left(Orte \times Filme \times Programm\right)\right)$$

dargestellt als Anfrage-Baum:

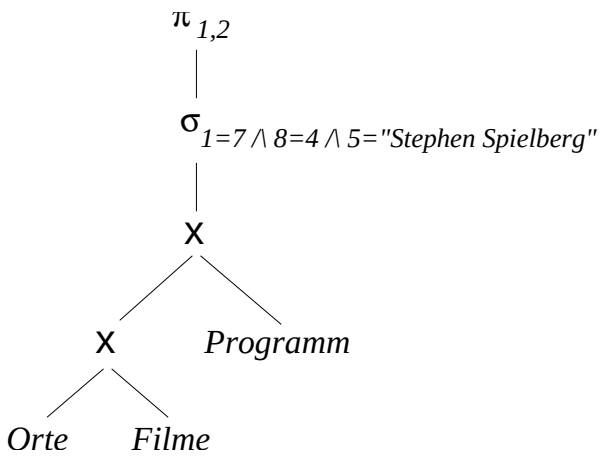


Anfrageauswertung an einem Beispiel (3/4)

Anfrage auswerten auf folgender Beispiel-Datenbank:

- *Filme*: 10.000 Tupel auf 200 Seiten (je 50 pro Seite); je 5 Tupel pro Film, 10 Filme von Stephen Spielberg
- *Programm*: 200 Tupel auf 4 Seiten (je 50 pro Seite); davon 3 Spielberg-Filme in 4 Kinos
- *Orte*: 100 Tupel auf 2 Seiten (je 50 pro Seite)

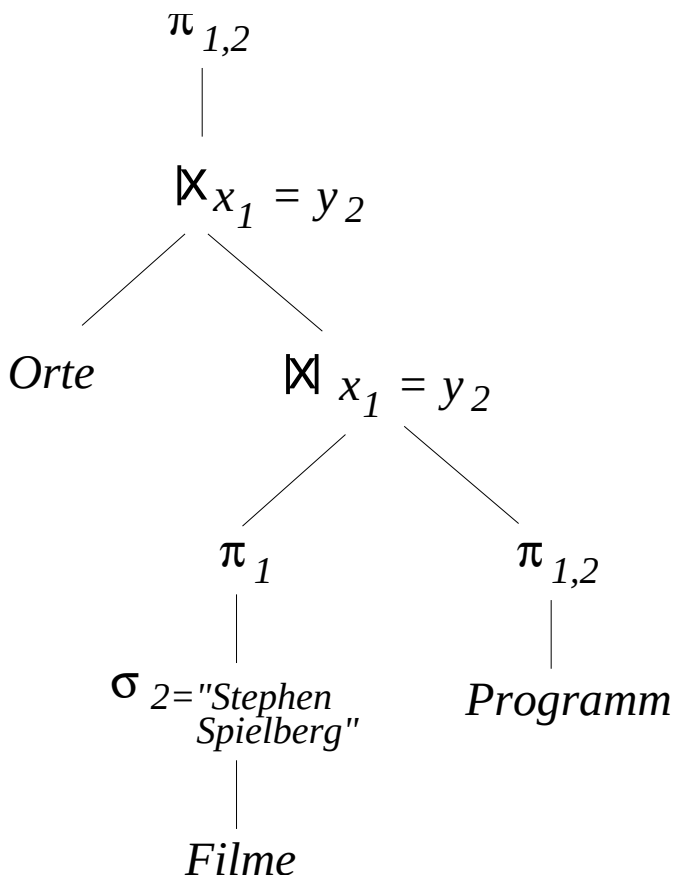
Anfrage-Baum:



Direkte Auswertung dieser Anfrage führt zu über 10.000.000 Plattenzugriffen.

Anfrageauswertung an einem Beispiel (4/4)

Viel besserer Plan:



Auswertung dieses Plans in unserer Beispiel-Datenbank führt zu weniger als 250 Plattenzugriffen.

Heuristische Optimierung (1/3)

- Die heuristische Optimierung wendet allgemeine Regeln zur Umformung einer Anfrage der relationalen Algebra in eine äquivalente Anfrage an, die zu einem vermutlich effizienteren Auswertungsplan führt
- Grundregel: Selektionen so früh wie möglich
- Projektionen auch früh, aber evtl. Duplikatelimination vermeiden
- Anwendung von algebraischen Umformungsregeln
- Ziel: Operationen verschieben, um kleinere Zwischenergebnisse zu erhalten; wenn möglich Redundanzen erkennen

Heuristische Optimierung (2/3)

Einige algebraische Umformungsregeln:

(1) Cartesische Produkte und Joins sind kommutativ und assoziativ:

$Q_1 \bowtie_\theta Q_2 \quad \longleftrightarrow \quad Q_2 \bowtie_{\tilde{\theta}} Q_1$

$(Q_1 \bowtie_{\theta_1} Q_2) \bowtie_{\theta_2} Q_3 \quad \longleftrightarrow \quad Q_1 \bowtie_{\tilde{\theta}_1} (Q_2 \bowtie_{\tilde{\theta}_2} Q_3)$

($\tilde{\theta}$ entsteht aus θ durch "Zurückrechnen" der Spaltennummern)

(2) Ketten von Selektionen (bzw. Projektionen) zusammenfassen:

$\sigma_{F_1}(\sigma_{F_2}(Q)) \quad \longleftrightarrow \quad \sigma_{F_1 \wedge F_2}(Q) \quad \longleftrightarrow \quad \sigma_{F_2}(\sigma_{F_1}(Q))$

$\pi_X(\pi_Y(Q)) \quad \longleftrightarrow \quad \pi_{\tilde{X}}(Q)$

($\tilde{X}$ entsteht aus X durch "Zurückrechnen" der Spaltennummern)

(3) Vertauschen von Selektion und Join:

$\sigma_{F_1 \wedge F_2 \wedge F_3}(Q_1 \times Q_2) \quad \longleftrightarrow \quad \sigma_{F_1}(Q_1) \bowtie_{\theta_3} \sigma_{\tilde{F}_2}(Q_2)$,

wobei die Selektionsbed. F_1 (F_2) sich nur auf Spalten von Q_1 (Q_2) bezieht und F_3 Spalten von Q_1 mit Spalten von Q_2 vergleicht. $\tilde{F}_2$ und θ_3 entstehen aus F_2 und F_3 durch "Zurückrechnen" der Spaltennummern.

(4) Einführung von Semijoins, falls X nur Spalten von Q_1 beinhaltet:

$\pi_X(Q_1 \bowtie_\theta Q_2) \quad \longleftrightarrow \quad \pi_X(Q_1 \ltimes_\theta Q_2)$

Heuristische Optimierung (3/3)

Einige algebraische Umformungsregeln:

(5) Vertauschen von Selektion und Vereinigung bzw. Differenz:

$$\sigma_F(Q_1 \cup Q_2) \quad \longleftrightarrow \quad \sigma_F(Q_1) \cup \sigma_F(Q_2)$$
$$\sigma_F(Q_1 - Q_2) \quad \longleftrightarrow \quad \sigma_F(Q_1) - \sigma_F(Q_2)$$

(6) Analog: Vertauschen von Projektion und Vereinigung bzw. Differenz.

(7) Vertauschen von Projektion und Selektion unter bestimmten Bedingungen

(8) Vertauschen von Projektion und Join unter bestimmten Bedingungen

(9) Löschen von Redundanzen:

$$Q \cup Q \longrightarrow Q, \qquad Q \cap Q \longrightarrow Q, \qquad Q \bowtie Q \longrightarrow Q$$

(10) Löschen leerer Zwischenergebnisse:

$$Q - Q \longrightarrow \emptyset, \qquad Q \cap \emptyset \longrightarrow \emptyset, \qquad Q \cup \emptyset \longrightarrow Q, \qquad Q \bowtie \emptyset \longrightarrow \emptyset,$$
$$Q - \emptyset \longrightarrow Q, \qquad \emptyset - Q \longrightarrow \emptyset$$

(11) . . . usw. . . .

Wunschliste für bessere Optimierung:

- zum “Löschen leerer Zwischenergebnisse”:
  Test, ob eine gegebene (Teil-)Anfrage Q nicht erfüllbar ist (“$Q \equiv \emptyset$”)
- zum “Löschen von Redundanzen”:
  Test, ob zwei (Teil-)Anfragen Q und P äquivalent sind (“$Q \equiv P$”)

In späteren Kapiteln:

- Algorithmen zum Lösen dieser Probleme für konjunktive Anfragen
- Nicht-Entscheidbarkeit dieser Probleme für allgemeinere Anfragen (relationale Algebra)

Vorgehensweise eines Optimierers in einem DBMS

- Erzeugung verschiedener rel. Algebra Ausdrücke:
  unter Verwendung heuristischer Optimierungsregeln
- Erzeugung von verschiedenen Auswertungsplänen:
  Anfrage-Bäume, erweitert um Informationen über zu verwendende Zugriffsstrukturen und Algorithmen für die einzelnen Operationen
- Abschätzung der Kosten für jeden erzeugten Auswertungsplan:
  unter Verwendung von statistischen Informationen über die Daten
  ($\rightsquigarrow$ kostenbasierte Optimierung)
- Auswahl des am günstigsten erscheinenden Plans

Generell:

Je häufiger die Anfrage ausgewertet werden soll, desto mehr Aufwand sollte für die Optimierung verwendet werden.

Join-Reihenfolge

- Joins sind kommutativ und assoziativ
  $\rightsquigarrow$ Änderung der Klammerung bzw. Reihenfolge von Join-Operationen ändert nicht das Ergebnis der Anfrage (modulo Ändern der Spalten-Reihenfolge)
- Aber: Die Größe der Zwischenergebnisse und somit der Auswertungs-Aufwand kann sich drastisch ändern.
- Unter Umständen lässt sich sogar die Anzahl der Joins verkleinern (mehr dazu in Kapitel 5)

- Klassische Vorgehensweise in DBMS: Nur Auswertungspläne betrachten, die Joins von links nach rechts klammern ("left-deep-trees")
  $\rightsquigarrow$ durch Umordnen sind immerhin alle Reihenfolgen möglich (immer noch exponentiell viele Möglichkeiten)
- (Es ist bekannt, dass diese Einschränkung nicht immer sinnvoll und nötig ist)

Jetzt: Heuristische Join-Optimierung bzgl. left-deep-trees

Beispiele (1/3)

Beispiel 3.5

R : 2-stellige Relation mit Attributen A, B und 1.000 Tupeln

S : 2-stellige Relation mit Attributen C, D und 10.000 Tupeln

T : 2-stellige Relation mit Attributen B, C und 100 Tupeln

Pro Tupel $(b, c) \in T$ gibt es ein Tupel $(\cdot, b) \in R$ und ein Tupel $(c, \cdot) \in S$.

Anfrage: $Ans(x) \leftarrow R(x, x_1),\ S(x_2, x_3),\ T(x_1, x_2)$

Aufwand bei Links-nach-rechts-Auswertung:

10.000.000 Tupel nach erstem Join, 100 Tupel nach zweitem Join

---

Andere Join-Reihenfolge: $Ans(x) \leftarrow R(x, x_1),\ T(x_1, x_2),\ S(x_2, x_3)$

Aufwand bei Links-nach-rechts-Auswertung:

100 Tupel nach erstem Join, 100 Tupel nach zweitem Join

Beispiele (2/3)

Beispiel 3.6

R : 2-stellige Relation mit Attributen A, B und 1.000 Tupeln

S : 2-stellige Relation mit Attributen C, D und 10.000 Tupeln

T : 2-stellige Relation mit Attributen B, C und 100 Tupeln

Pro Tupel $(b, c) \in T$ gibt es ein Tupel $(\cdot, b) \in R$ und ein Tupel $(c, \cdot) \in S$.

Für die Konstante d gibt es nur 1 Tupel $(\cdot, d)$ in S.

Anfrage: $Ans(x) \leftarrow R(x, x_1),\ T(x_1, x_2),\ S(x_2, d)$

Aufwand bei Links-nach-rechts-Auswertung:

100 Tupel nach erstem Join, max. 1 Tupel nach zweitem Join

Andere Join-Reihenfolge: $Ans(x) \leftarrow S(x_2, d),\ T(x_1, x_2),\ R(x, x_1)$

Aufwand bei Links-nach-rechts-Auswertung:

max. 1 Tupel nach erstem Join, max. 1 Tupel nach zweitem Join

Beispiele (3/3)

Noch ein Beispiel:

- Auswertung von links nach rechts
- Anfrage: $Ans(z) \leftarrow R(v,w,y),\ v \leqslant x,\ S(x,y,z),\ P(a,v),\ Q(b,w,x)$
- Besserer Auswertungsplan:
  $Ans(z) \leftarrow P(a,v),\ Q(b,w,x),\ R(v,w,y),\ S(x,y,z),\ v \leqslant x,$
- Denn:
  - wahrscheinlich wenige Tupel der Form $(a,\cdot)$ in P
  - wahrscheinlich wenige Tupel der Form $(b,\cdot,\cdot)$ in Q
  - wahrscheinlich wenige Tupel, die $P(a,v),\ Q(b,w,x),\ R(v,w,y)$ erfüllen

Heuristik ("Sideways-Information-Passing"):

- Relations-Atome mit Konstanten zuerst auswerten
- Wenn möglich, Relations-Atome erst dann, wenn weiter links schon eine ihrer Variablen steht.
- Vergleichsoperatoren ($\leqslant$, $<$) möglichst erst dann verwenden, wenn beide Variablen schon verwendet wurden.

SIP-Graph und SIP-Strategie (1/2)

Definitionen:

- Regel $Ans(u) \leftarrow R_1(u_1), \ldots, R_k(u_k), E_1, \ldots, E_\ell, C_1, \ldots, C_m$
  wobei E_i Vergleich mit $=$; C_i Vergleich mit $<$ oder $\leqslant$ (dafür sei $<$ eine lineare Ordnung auf **dom**).
- SIP-Graph
  - Knotenmenge: Rel.-Atome $R_1(u_1), \ldots, R_k(u_k)$ und "="-Atome $E_1, \ldots, E_\ell$
  - Kante zwischen zwei Knoten, falls diese (mind.) eine Variable gemeinsam haben
  - Knoten ist markiert, falls er (mind.) eine Konstante enthält
- Falls der SIP-Graph zusammenhängend ist, so ist eine SIP-Strategie eine Anordnung $A_1, \ldots, A_{k+\ell+m}$ der Atome, so dass für jedes $j > 1$ gilt:
  - A_j ist ein markierter Knoten des SIP-Graphen, oder
  - A_j ist ein Knoten des SIP-Graphen und es gibt ein $j' < j$, so dass es im SIP-Graph eine Kante zwischen A_j und $A_{j'}$ gibt, oder
  - A_j ist ein C_i und für jede Variable x in C_i gibt es $j' < j$, so dass $A_{j'}$ ein Knoten des SIP-Graphen ist, in dem x vorkommt
- Außerdem: Falls ex. $R_i(u_i)$ od. E_i mit einer Konstanten, so ist A_1 ein solches Atom
- Falls der SIP-Graph nicht zusammenhängend ist: SIP-Strategie für jede Zusammenhangskomponente.

SIP-Graph und SIP-Strategie (2/2)

Beispiele:

- $Ans(z) \leftarrow P(a,v),\ Q(b,w,x),\ R(v,w,y),\ S(x,y,z),\ v \leqslant x$
  ist in der Reihenfolge einer SIP-Strategie
- $Ans(z) \leftarrow R(v,w,y),\ v \leqslant x,\ S(x,y,z),\ P(a,v),\ Q(b,w,x)$
  ist nicht in der Reihenfolge einer SIP-Strategie

Bemerkungen:

- Zu jeder regelbasierten konjunktiven Anfrage (evtl. mit $=$ und $\leqslant$; dann aber bereichsbeschränkt) gibt es eine SIP-Strategie.
- Eine SIP-Strategie für eine gegebene Anfrage lässt sich in polynomieller Zeit berechnen.
- Wird eine Variable weiter rechts nicht mehr benötigt, so kann sie aus dem Zwischenergebnis “heraus projiziert” werden.

Relationenkalkül

4.1 CALC_{nat}, CALC_{adom} und CALC_{di}
4.2 Sicherer Relationenkalkül: CALC_{sr}
4.3 Statische Analyse und Auswertungskomplexität
4.4 Grenzen der Ausdrucksstärke

Relationenkalkül

Motivation: Bisher kennengelernte Anfragesprachen

Algebraisch ...	äquivalent dazu ...	Logik-basiert
SPC-Algebra		konjunktiver Kalkül
relationale Algebra		Relationenkalkül

Der konjunktive Kalkül basiert auf einem Fragment der Logik erster Stufe.

Der Relationenkalkül basiert auf der vollen Logik erster Stufe (kurz: FO).

("FO" steht für "first-order logic")

Die Logik erster Stufe — FO[**R**]

Definition 4.1

Sei **R** ein Datenbankschema.
Die Menge FO[**R**] aller Formeln der Logik erster Stufe über **R** ist induktiv wie folgt definiert:

(A) "Relations-Atome": $R(v_1, \ldots, v_r)$ gehört zu FO[**R**],
für alle $R \in \mathbf{R}$, $r := \mathit{arity}(R)$ und $v_1, \ldots, v_r \in \mathbf{var} \cup \mathbf{dom}$.

(G) "Gleichheits-Atome": $v_1 = v_2$ gehört zu FO[**R**], für alle $v_1, v_2 \in \mathbf{var} \cup \mathbf{dom}$.

(BK) "Boolesche Kombinationen": Falls $\varphi_1 \in$ FO[**R**] und $\varphi_2 \in$ FO[**R**], so gehört auch jede der folgenden fünf Formeln zu FO[**R**]:
$\neg\varphi_1$, $(\varphi_1 \wedge \varphi_2)$, $(\varphi_1 \vee \varphi_2)$, $(\varphi_1 \rightarrow \varphi_2)$ und $(\varphi_1 \leftrightarrow \varphi_2)$.

(Q) "Quantoren": Ist $\varphi_1 \in$ FO[**R**] und ist $x \in \mathbf{var}$, so gehören auch die beiden folgenden Formeln zu FO[**R**]: $\exists x\, \varphi_1$ und $\forall x\, \varphi_1$.

Bemerkung: Benutzen wir die Notation aus der Vorlesung *Diskrete Modellierung*, so ist FO[**R**] genau die Menge aller Formeln aus FO[σ], für die Signatur $\sigma := \mathbf{R} \cup \mathbf{dom}$ (wobei jedes Element aus **dom** als "Konstanten-Symbol" aufgefasst wird, das stets "mit sich selbst" interpretiert wird).

Notationen und Anmerkungen

- Manchmal werden wir Formeln der Form
  - $(\varphi_1 \vee \varphi_2)$ als Abkürzung für $\neg(\neg\varphi_1 \wedge \neg\varphi_2)$,
  - $(\varphi_1 \rightarrow \varphi_2)$ als Abkürzung für $(\neg\varphi_1 \vee \varphi_2)$
  - $(\varphi_1 \leftrightarrow \varphi_2)$ als Abkürzung für $(\varphi_1 \wedge \varphi_2) \vee (\neg\varphi_1 \wedge \neg\varphi_2)$
  - $\forall x\, \varphi_1$ als Abkürzung für $\neg\, \exists x\, \neg\varphi_1$

  auffassen.

- $adom(\varphi)$ bezeichnet die Menge aller Konstanten (also Elemente aus **dom**), die in φ vorkommen.
  $Var(\varphi)$ bezeichnet die Menge aller Variablen (also Elemente aus **var**, die in φ vorkommen.
  $frei(\varphi)$ bezeichnet die Menge aller Variablen, die frei in φ vorkommen. D.h.
  - $frei\big(R(v_1,..,v_r)\big) := \{v_1,..,v_r\} \cap \mathbf{var}$
  - $frei\big(v_1{=}v_2\big) := \{v_1, v_2\} \cap \mathbf{var}$
  - $frei\big(\neg\varphi_1\big) := frei\big(\varphi_1\big)$
  - $frei\big((\varphi_1 * \varphi_2)\big) := frei\big(\varphi_1\big) \cup frei\big(\varphi_2\big)$ (für alle $* \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$)
  - $frei\big(\exists x\, \varphi_1\big) := frei\big(\forall x\, \varphi_1\big) := frei\big(\varphi_1\big) \setminus \{x\}$

- Eine Belegung für φ ist eine Abbildung $\beta : frei(\varphi) \rightarrow \mathbf{dom}$.
- Sei $\mathbf{d} \subseteq \mathbf{dom}$. Eine Belegung für φ in $\mathbf{d}$ ist eine Abbildung $\beta : frei(\varphi) \rightarrow \mathbf{d}$.

Nat. Semantik von FO[**R**] und Rel.kalkül CALC

Wir werden verschiedene Varianten der Semantik von FO[**R**] betrachten.
Hier zunächst die natürliche Semantik:

*Definition 4.2 (natürliche Semantik von FO[**R**])*

- Zur Erinnerung: Einer Datenbank **I** vom Schema **R** ordnen wir die logische Struktur $\mathcal{A}_{\mathbf{I}} := \big(\mathbf{dom},\ (\mathbf{I}(R))_{R\in\mathbf{R}},\ (c)_{c\in\mathbf{dom}}\big)$ zu.
- Ist **I** eine Datenbank vom Schema **R** und β eine Belegung für φ (also eine Abbildung $\beta : \mathit{frei}(\varphi) \to \mathbf{dom}$), so sagen wir "**I** erfüllt φ unter β" und schreiben $\mathbf{I} \models \varphi[\beta]$ bzw. $(\mathbf{I}, \beta) \models \varphi$, um auszudrücken, dass $(\mathcal{A}_{\mathbf{I}}, \beta) \models \varphi$.

Definition 4.3 (Relationenkalkül CALC)

Sei **R** *ein Datenbankschema. Eine Anfrage des Relationenkalküls* CALC[**R**] *ist von der Form* $\{\ \langle e_1, \ldots, e_r\rangle\ :\ \varphi\ \}$,
wobei $\varphi \in \text{FO}[\mathbf{R}]$, $r \geqslant 0$ *und* $\langle e_1, \ldots, e_r\rangle$ *ein freies Tupel ist (d.h.* $e_1, \ldots, e_r \in \mathbf{var} \cup \mathbf{dom}$*), so dass* $\mathit{frei}(\varphi) = \{e_1, \ldots, e_r\} \cap \mathbf{var}$.
Wir setzen $\mathit{adom}(Q) := \mathit{adom}(\varphi) \cup (\{e_1, \ldots, e_r\} \cap \mathbf{dom})$.

Semantik: Einer CALC[**R**]*-Anfrage Q der obigen Form ordnen wir die folgende "Anfragefunktion"* $[\![Q]\!]_{nat}$ *zu (f.a.* $\mathbf{I} \in \mathit{inst}(\mathbf{R})$*):*

$$[\![Q]\!]_{nat}(\mathbf{I}) := \{\ \beta(\langle e_1, \ldots, e_r\rangle)\ :\ \beta \text{ ist eine Belegung für } \varphi \text{ mit } \mathbf{I} \models \varphi[\beta]\ \}$$

Beispiele für CALC-Anfragen

- In welchen Filmen hat “George Clooney” mitgespielt, aber nicht selbst Regie geführt?

$$\Big\{ \langle x_{Titel} \rangle \;:\; \exists x_{Regie} \Big(Filme(x_{Titel}, x_{Regie}, \text{“George Clooney”}) \wedge \neg Filme(x_{Titel}, \text{“George Clooney”}, \text{“George Clooney”}) \Big) \Big\}$$

- Welche Filme laufen nur zu 1 Uhrzeit?

$$\Big\{ \langle x_{Titel} \rangle \;:\; \exists x_{Kino}\, \exists x_{Zeit} \Big(Programm(x_{Kino}, x_{Titel}, x_{Zeit}) \wedge \forall y_{Kino}\, \forall y_{Zeit} \big(Programm(y_{Kino}, x_{Titel}, y_{Zeit}) \rightarrow y_{Zeit} = x_{Zeit} \big) \Big) \Big\}$$

- Welche Filme haben nur Schauspieler, die schon mal in einem Film von “Stephen Spielberg” mitgespielt haben?

$$\Big\{ \langle x_T \rangle \;:\; \exists x_R \exists x_S \Big(Filme(x_T, x_R, x_S) \wedge \forall y_S \big(Filme(x_T, x_R, y_S) \rightarrow \exists y_T\, Filme(y_T, \text{“Stephen Spielberg”}, y_S) \big) \Big) \Big\}$$

Frage: Was ist mit . . . ?

$$\Big\{ \langle x_T \rangle \;:\; \forall y_S \Big(\exists x_R\, Filme(x_T, x_R, y_S) \;\rightarrow\; \exists y_T\, Filme(y_T, \text{“Stephen Spielberg”}, y_S) \Big) \Big\}$$

Unsichere CALC-Anfragen

Beispiele:

(1) $Q_1 :=$

$$\Big\{\langle x_T\rangle \;:\; \forall y_S\Big(\exists x_R\, \mathit{Filme}(x_T, x_R, y_S) \;\rightarrow\; \exists y_T\, \mathit{Filme}(y_T, \text{“Stephen Spielberg”}, y_S)\Big)\Big\}$$

(2) $Q_2 := \big\{\langle x_{Schausp}\rangle \;:\; \neg \mathit{Filme}(\text{“Knallhart”}, \text{“Detlev Buck”}, x_{Schausp})\big\}$

(3) $Q_3 := \big\{\langle x_S, y_T\rangle \;:\; \mathit{Filme}(\text{“Knallhart”}, \text{“Detlev Buck”}, x_S) \vee \mathit{Filme}(y_T, \text{“George Clooney”}, \text{“George Clooney”})\big\}$

Auswertung mit $[\![\cdot]\!]_{nat}$: Q_1 liefert alle Filme, die nur solche Schauspieler haben, die schon mal mit “Stephen Spielberg” gearbeitet haben, aber auch **alle Elemente aus dom, die nicht Titel eines Films** sind.
Q_2 liefert alle (unendlich vielen) $a \in$ **dom**, die *nicht* als Schauspieler im Film “Knallhart” mitgespielt haben.
Q_3 liefert alle (unendlich vielen) Tupel der Form $\langle a, b\rangle$, für die a ein Schaupieler in “Knallhart” und b ein beliebiges Element in **dom** ist oder b ein Film von und mit “George Clooney” und a ein beliebiges Element aus **dom**.

Problem: Gemäß der bisherigen Definition der Semantik liefern die Anfragen Q_1, Q_2 und Q_3 stets “unendliche Ergebnisse” die gemäß unserer Definition keine Datenbank-Relationen sind (da die stets endlich sein müssen). Solche Anfrage heißen unsicher.

Eine weitere problematische Anfrage

$Q_4 := \{\langle x \rangle \ : \ \forall y \, R(x, y)\}$

Q_4 liefert als Ergebnis stets die leere Relation, d.h.
$[\![Q_4]\!]_{nat}(\mathbf{I}) = \emptyset$, für alle $\mathbf{I} \in \mathit{inst}(\mathbf{R})$.

Dies ist unschön.

Daher (und weil manche Anfragen unsicher sind in der nat. Semantik), betrachten wir andere Varianten der Semantik ...

Relativierte Semantik von FO[**R**] und CALC[**R**]

Definition 4.4

Sei **R** ein Datenbankschema.

(a) Eine relativierte Datenbank über **R** ist ein Paar $(\mathbf{d}, \mathbf{I})$, wobei $\mathbf{I}$ eine Datenbank vom Schema **R** ist und **d** eine Menge, so dass $adom(\mathbf{I}) \subseteq \mathbf{d} \subseteq \mathbf{dom}$.

(b) Einer relativierten Datenbank $(\mathbf{d}, \mathbf{I})$ über **R** ordnen wir die logische Struktur $\mathcal{A}_{(\mathbf{d},\mathbf{I})} := \big(\mathbf{d},\ (\mathbf{I}(R))_{R\in\mathbf{R}},\ (c)_{c\in\mathbf{d}}\big)$ zu.

(c) Eine FO[**R**]-Formel φ heißt interpretierbar über $(\mathbf{d}, \mathbf{I})$, falls $adom(\varphi) \subseteq \mathbf{d}$.
Eine CALC[**R**]-Anfrage Q heißt interpretierbar über $(\mathbf{d}, \mathbf{I})$, falls $adom(Q) \subseteq \mathbf{d}$.

(d) Ist $(\mathbf{d}, \mathbf{I})$ eine relativierte Datenbank über **R**, ist φ eine FO[**R**]-Formel, die interpretierbar ist über $(\mathbf{d}, \mathbf{I})$, und ist β eine Belegung für φ in **d** (also eine Abbildung $\beta : frei(\varphi) \to \mathbf{d}$), so sagen wir "$\mathbf{I}$ erfüllt φ unter β relativ zu **d**" und schreiben $\mathbf{I} \models_{\mathbf{d}} \varphi[\beta]$ bzw. $(\mathbf{I}, \beta) \models_{\mathbf{d}} \varphi$, um auszudrücken, dass $(\mathcal{A}_{(\mathbf{d},\mathbf{I})}, \beta) \models \varphi$.

(e) Ist Q eine CALC[**R**]-Anfrage der Form $\{\langle e_1, \ldots, e_r\rangle : \varphi\}$ und $(\mathbf{d}, \mathbf{I})$ eine relativierte Datenbank über **R**, über der Q interpretierbar ist, so ist

$$[\![Q]\!]_{\mathbf{d}}(\mathbf{I}) := \{\, \beta(\langle e_1, \ldots, e_r\rangle) : \beta \text{ ist eine Belegung für } \varphi \text{ über } \mathbf{d} \text{ mit } \mathbf{I} \models_{\mathbf{d}} \varphi[\beta] \,\}$$

("Relativiert" soll andeuten, dass Quantifizierung relativiert wird auf Elemente aus **d**.)

Beispiel

- **R** enthalte ein 1-stelliges Relationssymbol R
- **I** sei die Datenbank mit
  $\mathbf{I}(R) = \{\langle a\rangle, \langle b\rangle, \langle c\rangle\}$ und $\mathbf{I}(S) = \emptyset$ für alle $S \in \mathbf{R} \setminus \{R\}$.
- Q sei die CALC[**R**]-Anfrage

$$Q := \left\{ \langle x\rangle \;:\; R(x) \;\wedge\; \exists y \left(\neg R(y) \;\wedge\; \forall z (R(z) \vee z{=}y) \right) \right\}$$

- Dann gilt:
  - $[\![Q]\!]_{\{a,b,c\}}(\mathbf{I}) = \emptyset$
  - $[\![Q]\!]_{\{a,b,c,d\}}(\mathbf{I}) = \{\langle a\rangle, \langle b\rangle, \langle c\rangle\}$
  - $[\![Q]\!]_{\{a,b,c,d,e\}}(\mathbf{I}) = \emptyset$

Active Domain Semantik von FO[**R**] und CALC[**R**]

Definition 4.5

(a) Ist $\mathbf{I} \in inst(\mathbf{R})$ und $\varphi \in \text{FO}[\mathbf{R}]$, so ist $adom(\varphi, \mathbf{I}) := adom(\varphi) \cup adom(\mathbf{I})$.
Ist $\mathbf{I} \in inst(\mathbf{R})$ und $Q \in \text{CALC}[\mathbf{R}]$, so ist $adom(Q, \mathbf{I}) := adom(Q) \cup adom(\mathbf{I})$.

(b) Ist $\varphi \in \text{FO}[\mathbf{R}]$, $\mathbf{I}$ eine Datenbank vom Schema $\mathbf{R}$ und β eine Belegung für φ in $adom(\varphi, \mathbf{I})$ (also eine Abbildung $\beta : frei(\varphi) \to adom(\varphi, \mathbf{I})$), so sagen wir "$\mathbf{I}$ erfüllt φ unter β in der Active Domain Semantik" und schreiben $\mathbf{I} \models_{adom} \varphi[\beta]$ bzw. $(\mathbf{I}, \beta) \models_{adom} \varphi$, um auszudrücken, dass $\mathbf{I} \models_{\mathbf{d}} \varphi[\beta]$ für $\mathbf{d} := adom(\varphi, \mathbf{I})$ gilt.

(c) Ist Q eine CALC[**R**]-Anfrage der Form $\{\langle e_1, \ldots, e_r \rangle \,:\, \varphi\}$, so definiert Q in der Active Domain Semantik die folgende Anfragefunktion

$$[\![Q]\!]_{adom}(\mathbf{I}) \;:=\; [\![Q]\!]_{adom(Q,\mathbf{I})}(\mathbf{I})$$

Bemerkungen zur natürlichen Semantik und zur Active Domain Semantik

- Die Semantik $[\![\cdot]\!]_{nat}$ ist vom Standpunkt der Logik aus "natürlich"; vom Standpunkt des Datenbanknutzers aber eher unnatürlich, da manche Anfragen unendliche Ergebnisse liefern (d.h. unsicher sind).

- Die Active Domain Semantik $[\![\cdot]\!]_{adom}$ liefert immer ein endliches Ergebnis; aber das Ergebnis kann sich (vom Standpunkt des Datenbanknutzers) auf unnatürliche Weise ändern, wenn ein neuer Wert in die Datenbank eingetragen wird (selbst wenn dieser scheinbar nichts mit der Anfrage zu tun hat).

  Beispiel: $Q := \{ \langle x \rangle \ : \ R(x) \ \wedge \ \exists y \big(\neg R(y) \ \wedge \ \forall z (R(z) \vee z{=}y) \big) \ \}$

- Radikaler Schritt: Betrachte nur Anfragen, die bereichsunabhängig sind, d.h. bei denen es egal ist, ob sie in $[\![\cdot]\!]_{nat}$, $[\![\cdot]\!]_{adom}$ oder in $[\![\cdot]\!]_{\mathbf{d}}$ für irgendeine Menge $\mathbf{d}$ mit $adom(Q, \mathbf{I}) \subseteq \mathbf{d} \subseteq \mathbf{dom}$ auswertet werden.

Definition 4.6 (bereichunabhängige CALC[**R**]*-Anfragen)*

Eine Anfrage $Q \in$ CALC[**R**] *heißt bereichsunabhängig (domain independent), falls für jede Datenbank* $\mathbf{I} \in inst(\mathbf{R})$ *und alle* $\mathbf{d}, \mathbf{d}' \subseteq \mathbf{dom}$ *mit* $adom(Q, \mathbf{I}) \subseteq \mathbf{d}, \mathbf{d}'$ *gilt:* $[\![Q]\!]_{\mathbf{d}}(\mathbf{I}) = [\![Q]\!]_{\mathbf{d}'}(\mathbf{I})$.

Beispiele für bereichsunabhängige CALC-Anfragen

- Welche Filme laufen in mindestens 2 Kinos?

$$\left\{ \langle x_T \rangle \;:\; \exists x_K \, \exists x_Z \, \exists y_K \, \exists y_Z \left(Programm(x_K, x_T, x_Z) \wedge Programm(y_K, x_T, y_Z) \wedge \neg x_K = y_K \right) \right\}$$

- In welchen Filmen hat “George Clooney” mitgespielt, aber nicht selbst Regie geführt?

$$\left\{ \langle x_{Titel} \rangle \;:\; \exists x_{Regie} \left(Filme(x_{Titel}, x_{Regie}, \text{“George Clooney”}) \wedge \neg Filme(x_{Titel}, \text{“George Clooney”}, \text{“George Clooney”}) \right) \right\}$$

- Welche Filme laufen nur zu 1 Uhrzeit?

$$\left\{ \langle x_{Titel} \rangle \;:\; \exists x_{Kino} \, \exists x_{Zeit} \left(Programm(x_{Kino}, x_{Titel}, x_{Zeit}) \wedge \forall y_{Kino} \, \forall y_{Zeit} \left(Programm(y_{Kino}, x_{Titel}, y_{Zeit}) \rightarrow y_{Zeit} = x_{Zeit} \right) \right) \right\}$$

- Welche Filme haben nur Schauspieler, die schon mal in einem Film von “Stephen Spielberg” mitgespielt haben?

$$\left\{ \langle x_T \rangle \;:\; \exists x_R \exists x_S \left(Filme(x_T, x_R, x_S) \wedge \forall y_S \left(Filme(x_T, x_R, y_S) \rightarrow \exists y_T \, Filme(y_T, \text{“Stephen Spielberg”}, y_S) \right) \right) \right\}$$

Nicht bereichunabhängig ist . . .

$$\left\{ \langle x_{Titel} \rangle \;:\; \forall y_S \left(\exists x_R \, Filme(x_{Titel}, x_R, y_S) \rightarrow \exists y_T \, Filme(y_T, \text{“Stephen Spielberg”}, y_S) \right) \right\}$$

$CALC_{nat}$, $CALC_{adom}$, $CALC_{di}$

Definition 4.7

$CALC_{nat}[\mathbf{R}]$ bezeichnet die Klasse aller in der natürlichen Semantik $[\![\cdot]\!]_{nat}$ interpretierten CALC[**R**]-Anfragen.

$CALC_{adom}[\mathbf{R}]$ bezeichnet die Klasse aller in der Active Domain Semantik $[\![\cdot]\!]_{adom}$ interpretierten CALC[**R**]-Anfragen.

$CALC_{di}[\mathbf{R}]$ bezeichnet die Klasse aller bereichsunabhängigen CALC[**R**]-Anfragen (interpretiert in $[\![\cdot]\!]_{adom}$ oder, äquivalent dazu, in $[\![\cdot]\!]_{nat}$).
("di" steht für "domain independent")

Satz 4.8 (Äquivalenz von $CALC_{adom}$, $CALC_{di}$ und relationaler Algebra)
Die folgenden Anfragesprachen können genau dieselben Anfragefunktionen ausdrücken:

(a) $CALC_{di}$
(b) $CALC_{adom}$
(c) *relationale Algebra (unbenannte oder benannte Perspektive)*

Und für jedes feste Datenbankschema **R** *gilt: Jede Anfrage aus einer dieser Sprachen kann in polynomieller Zeit in äquivalente Anfragen der anderen Sprachen übersetzt werden.*

Beweis: Siehe Tafel.

Bereichsunabhängigkeit — Pro & Contra $CALC_{di}$

Vorteile:

- logik-basierte Anfragesprache, die die "richtige" Ausdrucksstärke hat (äquivalent zur relationalen Algebra; insbesondere sind alle $CALC_{di}$-Anfragen "sicher")
- keine unerwünschten Effekte, wie sie bei $CALC_{adom}$ auftreten können (vgl. Folie 148)
- Ergebnis der Anfragen ist unabhängig davon, in welcher Semantik ($[\![\cdot]\!]_{adom}$ oder $[\![\cdot]\!]_{nat}$) sie interpretiert werden

Frage:

- Wie kann man bei einer gegebenen CALC-Anfrage Q feststellen, ob Q zu $CALC_{di}$ gehört, d.h. ob Q bereichsunabhängig ist?

Antwort — großer Nachteil von $CALC_{di}$:

- Das ist unentscheidbar, d.h. es gibt keinen Algorithmus, der bei Eingabe einer beliebigen CALC[**R**]-Anfrage Q nach endlich vielen Schritten anhält und entscheidet, ob Q bereichsabhängig ist oder nicht.
- Beweis: auf den nächsten Folien . . .

Der Satz von Trakhtenbrot

Boris A. Trakhtenbrot, russisch-israelischer Mathematiker, geb. 1921;
alternative Schreibweisen: Trachtenbrot bzw. Trahtenbrot

Theorem 4.9 (Trakhtenbrot, 1950) *(hier ohne Beweis)*

Sei σ eine Signatur, die mindestens ein Relationssymbol enthält, dessen Stelligkeit $\geqslant 2$ ist. Dann ist das folgende Problem unentscheidbar:

ERFÜLLBARKEITSPROBLEM FÜR FO[σ] ÜBER ENDLICHEN STRUKTUREN

Eingabe: Ein FO[σ]-Satz ψ (*Zur Erinnerung:* "Satz" heißt: *frei*$(\psi) = \emptyset$)

Frage: Gibt es eine σ-Struktur $\mathcal{A}$, deren Universum **endlich** ist und für die gilt $\mathcal{A} \models \psi$?

Beweis: Siehe Vorlesung Logik in der Informatik, WS 2008/09.

Bereichsunabhängigkeit ist unentscheidbar

Satz 4.10

Sei **R** *ein Datenbankschema, das mindestens ein Relationssymbol R enthält, dessen Stelligkeit* $\geqslant 2$ *ist. Dann ist das folgende Problem unentscheidbar:*

BEREICHSUNABHÄNGIGKEIT FÜR CALC[**R**]

Eingabe: Eine CALC[**R**]-Anfrage Q

Frage: Ist Q bereichsunabhängig (d.h., gehört Q zu CALC_{di}[**R**]) ?

Beweis: Einfache Folgerung aus dem Satz von Trakhtenbrot; siehe Tafel.