Relationale Algebra

Relationale Algebra

Grenzen der Ausdrucksstärke konjunktiver Anfragen

Wir haben gesehen:

- konjunktive Anfragen können nur *monotone* Anfragefunktionen beschreiben (Satz 2.6) $\rightsquigarrow$ Anfragen mit Negationen der Art

  Welche Regisseure haben noch nie mit “Meg Ryan” 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 (vgl. Übungsblatt 1).

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 4.1

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 der 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”?

$$\pi_{\text{Kino}}\Big(\sigma_{\text{Titel="Capote"}}(\mathit{Programm}) \ \cup \ \sigma_{\text{Titel="Knallhart"}}(\mathit{Programm}) \Big)$$

- Welche Regisseure haben noch nie mit “Meg Ryan” gearbeitet?

$$\pi_{\text{Regie}}(\mathit{Filme}) \ - \ \pi_{\text{Regie}}\Big(\sigma_{\text{Schauspieler="Meg Ryan"}}(\mathit{Filme})\Big)$$

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

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

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

Ausdrucksstärke (1/2)

Proposition 4.2

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

(b) *Für jede Datenbank* **I** *und jede Anfrage* Q *der relationalen Algebra gilt:*

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

(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 über den Aufbau der Anfragen.

(c): Übung.

Ausdrucksstärke (2/2)

Proposition 4.3

(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 der 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 Indizes* 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

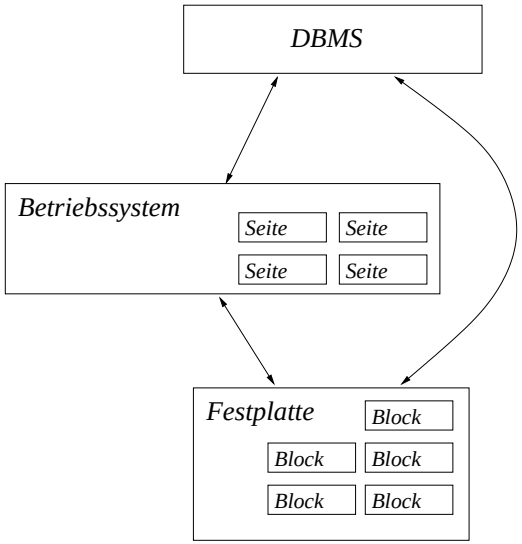
Vorbemerkung:

- 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



Hier (der Einfachheit halber): 1 Plattenzugriff $\widehat{=}$ 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

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

Hier (der Einfachheit halber):
Lesen eines Blocks (bzw. einer Seite) $\widehat{=}$ 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 $\cup, -, \ltimes_\theta$: $\mathcal{O}(b_I + b_J)$ 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: (sinnvoll für $\cup$ und $-$)
  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 4.4

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

Beweis:

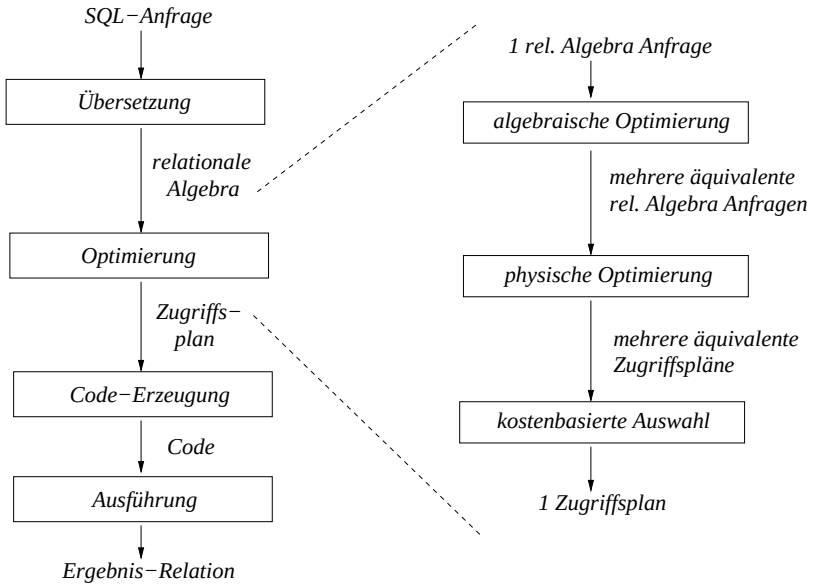
Zeige per Induktion nach dem Aufbau von Anfragen der relationalen Algebra, dass für jede Anfrage Q der Länge k und jede Datenbank $\mathbf{I}$ der Größe n gilt:

(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

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:

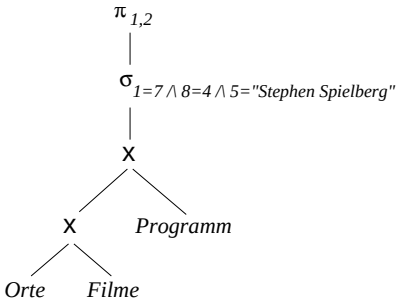
$$\pi_{1,2}\Big(\sigma_{\substack{1=7\wedge 8=4\wedge\\ 5=\text{“Stephen Spielberg”}}}\big(\mathit{Orte}\times\mathit{Filme}\times\mathit{Programm}\big)\Big)$$

Anfrageauswertung an einem Beispiel (2/4)

Original-Anfrage

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

dargestellt als Anfrage-Baum:

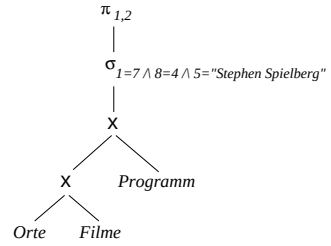


Anfrageauswertung an einem Beispiel (3/4)

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)

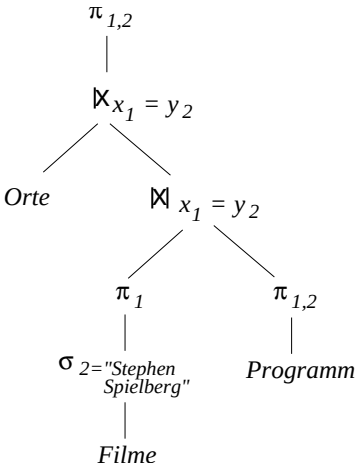
Anfrage-Baum:



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 ("$Q \equiv P$")

- Kapitel 2: Algorithmen zum Lösen dieser Probleme für konjunktive Anfragen
- Nächstes Kapitel: 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 (vgl. Kapitel 2: Anfrage-Minimierung)

- 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 4.5

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

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

S : 2-stellige Relation mit Attributen C, D und 10.000 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 4.6

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

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

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

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

Aufwand bei Links-nach-rechts-Auswertung:
10.000.000 Tupel nach erstem Join, 1 Tupel nach zweitem Join

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

Aufwand bei Links-nach-rechts-Auswertung:
1 Tupel nach erstem Join, 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"; kurz: SIP):

- 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 $=$ oder $\leqslant$; 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.