Datenbanken 2

Anfrageoptimierung

Nikolaus Augsten

nikolaus.augsten@sbg.ac.at

FB Computerwissenschaften
Universität Salzburg

Wintersemester 2015/16

Inhalt

Literatur und Quellen

Lektüre zum Thema "Anfrageoptimierung":

- Kapitel 8 aus Kemper und Eickler: Datenbanksysteme: Eine Einführung. Oldenbourg Verlag, 2013.
- Chapter 13 in Silberschatz, Korth, and Sudarashan: Database System Concepts. McGraw Hill, 2011.

Danksagung Die Vorlage zu diesen Folien wurde entwickelt von:

- Michael Böhlen, Universität Zürich, Schweiz
- Johann Gamper, Freie Universität Bozen, Italien

Inhalt

Schritte der Anfragebearbeitung

1. Parser
   - input: SQL Anfrage
   - output: Relationaler Algebra Ausdruck
2. Optimierer
   - input: Relationaler Algebra Ausdruck
   - output: Auswertungsplan
3. Execution Engine
   - input: Auswertungsplan
   - output: Ergebnis der SQL Anfrage

1. Parser

Parser:

- Input: SQL Anfrage vom Benutzer
  Beispiel:
  ```
  SELECT DISTINCT balance
  FROM account
  WHERE balance < 2500
  ```
- Output: Relationaler Algebra Ausdruck
  Beispiel: $\sigma_{balance<2500}(\Pi_{balance}(account))$
- Algebra Ausdruck nicht eindeutig!
  Beispiel: folgende Ausdrück sind äquivalent
  - $\sigma_{balance<2500}(\Pi_{balance}(account))$
  - $\Pi_{balance}(\sigma_{balance<2500}(account))$
- Kanonische Übersetztung führt zu algebraischer Normalform (eindeutig)

Parser: Kanonische Übersetztung

- SQL Anfrage: `SELECT DISTINCT` $A_1, A_2, \ldots, A_n$
  `FROM` $R_1, R_2, \ldots, R_k$
  `WHERE` θ
- Algebraische Normalform:

$$\pi_{A_1,A_2,\ldots,A_n}(\sigma_\theta(R_1 \times R_2 \times \ldots \times R_k))$$

- Prädikat θ kann sowohl Selektions- als auch Join-Bedingungen enthalten

2. Optimierer

Optimierer:

- Input: Relationaler Algebra Ausdruck
  Beispiel: $\Pi_{balance}(\sigma_{balance<2500}(account))$
- Output: Auswertungsplan
  Beispiel:

$\Pi_{balance}$
| pipeline
$\sigma_{balance<2500}$
use index 1
|
account

- Auswertungsplan wird in drei Schritten konstruiert:
  A) Logische Optimierung: Äquivalenzumformungen
  B) Physische Optimierung: Annotation der relationalen Algebra Operatoren mit physichen Operatoren
  C) Kostenabschätzung für verschiedene Auswertungspläne

A) Logische Optimierung: Äquivalenzumformungen

- Äquivalenz relationaler Algebra Ausdrücke:
  - äquivalent: zwei Ausdrücke erzeugen dieselbe Menge von Tupeln auf allen legalen Datenbankinstanzen
  - legal: Datenbankinstanz erfüllt alle Integritätsbedingungen des Schemas
- Äquivalenzregeln:
  - umformen eines relationalen Ausdrucks in einen äquivalenten Ausdruck
  - analog zur Algebra auf reelle Zahlen, z.B.:
    $a + b = b + a$, $a(b + c) = ab + ac$, etc.
- Warum äquivalente Ausdrücke erzeugen?
  - äquivalente Ausdrücke erzeugen dasselbe Ergebnis
  - jedoch die Ausführungszeit unterscheidet sich signifikant

Äquivalenzregeln – Beispiele

- Selektionen sind untereinander vertauschbar:
  $\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$
  - E relationaler Ausdruck (im einfachsten Fall eine Relation)
  - θ_1 und θ_2 sind Prädikate auf die Attribute von E z.B. $E.sallary < 2500$
  - σ_θ ergibt alle Tupel welche die Bedingung θ erfüllen

- Natürlicher Join ist assoziativ: $(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$
  - das Join Prädikat im natürlichen Join ist "Gleichheit" auf allen Attributen zweier Ausdrücke mit gleichem Namen

    Beispiel: $R[A, B], S[B, C]$, Prädikat ist $R.B = S.B$
  - falls zwei Ausdrücke keine gemeinsamen Attribute haben, wird der natürliche Join zum Kreuzprodukt

    Beispiel: $R[A, B], S[C, D]$, $R \bowtie S = R \times S$

Äquivalenzregeln – Beispiel Anfrage

- Schemas der Beispieltabellen:

  branch(branch-name, branch-city, assets)
  account(account-number, branch-name, balance)
  depositor(customer-name, account-number)

- Fremdschlüsselbeziehungen:

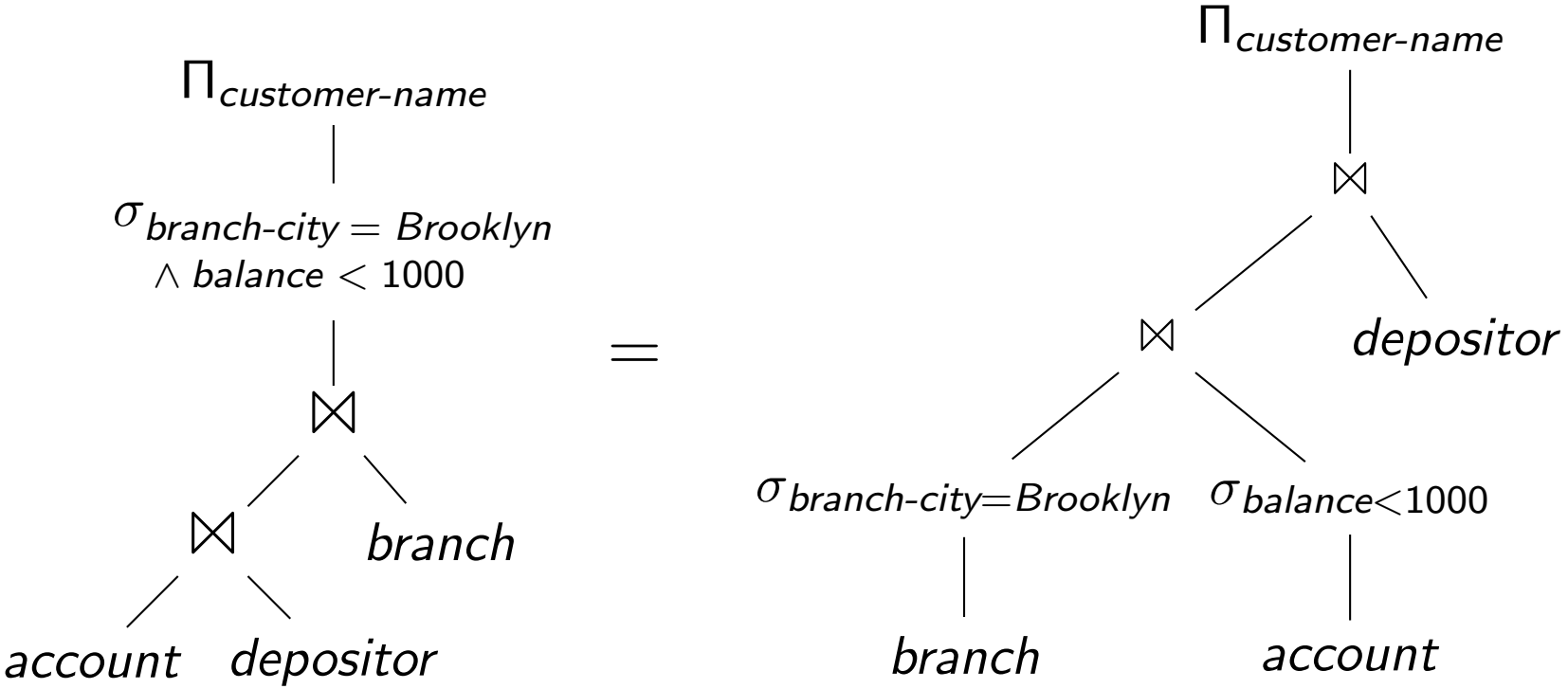
  $\pi_{\textit{branch-name}}(\textit{account}) \subseteq \pi_{\textit{branch-name}}(\textit{branch})$
  $\pi_{\textit{account-number}}(\textit{depositor}) \subseteq \pi_{\textit{account-number}}(\textit{account})$

- Anfrage:

```
SELECT customer-name
FROM branch, account, depositor
WHERE branch-city='Brooklyn' AND
   balance < 1000 AND
   branch.branch-name = account.branch-name AND
   account.account-number = depositor.account-number
```

Äquivalenzregeln – Beispiel Anfrage

- Äquivalente relationale Algebra Ausdrücke (als Operatorbäume dargestellt):

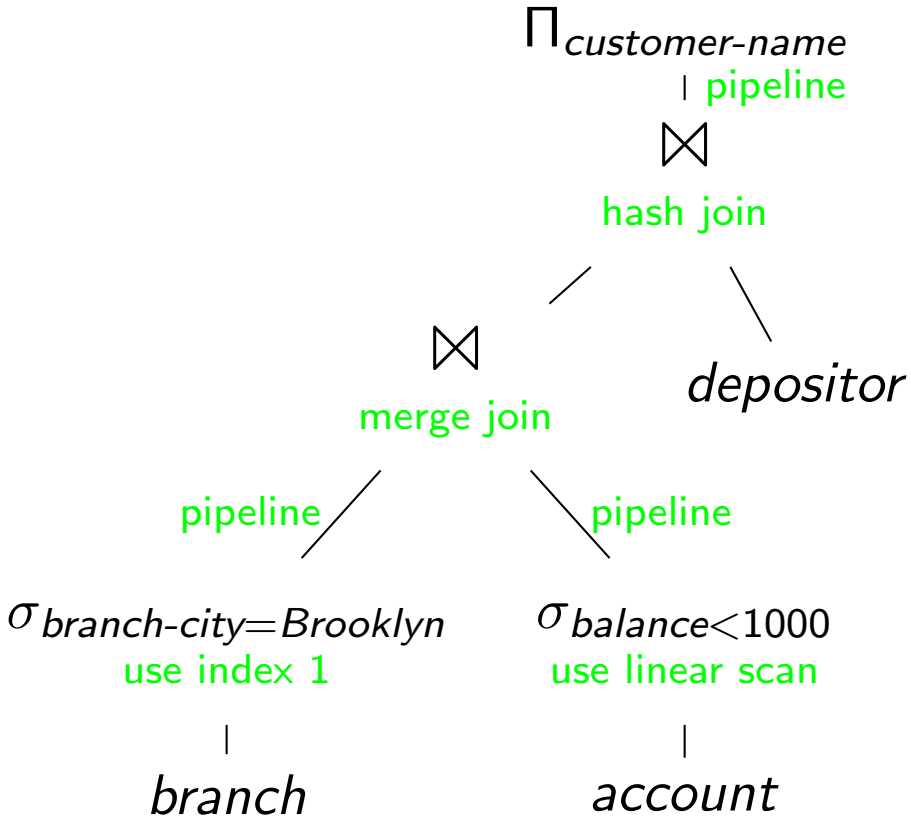


B) Annotation der relationalen Algebra Ausdrücke

- Ein Algebraausdruck ist noch kein Ausführungsplan.
- Zusätzliche Entscheidungen müssen getroffen werden:
  - welche Indices sollen verwendet werden, z.B. für Selektion oder Join?
  - welche Algorithmen sollen verwendet werde, z.B. Nested-Loop oder Hash Join?
  - sollen Zwischenergebnisse materialisiert oder "pipelined" werden?
  - usw.
- Für jeden Algebra Ausdruck können mehrere Ausführungspläne erzeugt werden.
- Alle Pläne ergeben dieselbe Relation, unterscheiden sich jedoch in der Ausführungszeit.

Beispiel: Ausführungsplan

- Ausführungsplan für die vorige Beispielanfrage:
  - *account* ist physisch sortiert nach *branch-name*
  - index 1 ist ein B^+-Baum Index auf *(branch-city,branch-name)*



C) Kostenabschätzung

- Welches ist der beste (=schnellste) Ausführungsplan?
- Schwieriges Problem:
  - Kosten für Ausführungsplan können nur abgeschätzt werden
  - es gibt eine sehr große Zahl von möglichen Ausführungsplänen

Datenbankstatistik für Kostenabschätzung

- Katalog: Datenbanksystem pflegt Statistiken über Daten
- Beispiel Statistiken:
  - Anzahl der Tupel pro Relation
  - Anzahl der Blöcke pro Relation
  - Anzahl der unterschiedlichen Werte für ein Attribut
  - Histogramm der Attributwerte
- Statistik wird verwendet um Kosten von Operationen abzuschätzen, z.B.:
  - Kardinalität des Ergebnisses einer Selektion
  - Kosten für Nested-Loop vs. Hash-Join
  - Kosten für sequentielles Lesen der Tabelle vs. Zugriff mit Index
- Beachte: Statistik wird nicht nach jeder Änderung aktualisiert und ist deshalb möglicherweise nicht aktuell

3. Execution Engine

Die Execution Engine

- erhält den Ausführungsplan vom Optimierer
- führt den Plan aus, indem die entsprechenden Algorithmen aufgerufen werden
- liefert das Ergebnis an den Benutzer zurück

Materialisierung und Pipelining

- Materialisierung:
  - gesamter Output eines Operators (Zwischenergebnis) wird gespeichert (z.B. auf Platte)
  - nächster Operator liest Zwischenergebnis und verarbeitet es weiter
- Pipelining:
  - sobald ein Tupel erzeugt wird, wird es an den nächsten Operator weitergeleitet
  - kein Zwischenspeichern erforderlich
  - Benutzer sieht erste Ergebnisse, bevor gesamte Anfrage berechnet ist
- Blocking vs. Non-Blocking:
  - Blocking: Operator muss gesamten Input lesen, bevor erstes Output Tuple erzeugt werden kann
  - Non-Blocking: Operator liefert erstes Tupel zurück sobald ein kleiner Teil des Input gelesen ist

Integrierte Übung 4.1

- Welche der folgenden Operatoren sind “blocking” bzw. “non-blocking”?
  - Selektion
  - Projektion
  - Sortierung
  - Gruppierung+Aggregation
  - Block Nested-loop Join
  - Index Nested-Loop Join
  - Hash Join
  - Merge Join, Sort-Merge Join

Iteratoren

- Demand-driven vs. Producer-driven Pipeline:
  - Demand-driven: Operator erzeugt Tupel erst wenn von Eltern-Knoten angefordert; Auswertung beginnt bei Wurzelknoten
  - Producer-driven: Operatoren produzieren Tupel und speichern sie in einen Buffer; Eltern-Knoten bedient sich aus Buffer (Producer-Consumer Modell)
- Demand-driven Pipelining: relationale Operatoren werden oft als Iteratoren mit folgenden Funktionen implementiert:
  - open(): initialisiert den Operator
    z.B. Table Scan: Datei öffnen und Cursor auf ersten Datensatz setzen
  - next(): liefert nächstes Tupel
    z.B. Table Scan: Tupel an Cursorposition lesen und Cursor weitersetzten
  - close(): abschließen
    z.B. Table Scan: Datei schließen
- Im Iteratormodell fragt der Wurzelknoten seine Kinder so lange nach Tupeln, bis keine Tupel mehr geliefert werden.

Inhalt

Überblick

- nur eine Auswahl von Äquivalenzregeln (equivalence rules, ER) wird präsentiert
- die Auswahl ist nicht minimal, d.h., einige der Regeln können aus anderen hergeleitet werden
- Notation:
  - $E, E_1, E_2 \ldots$ sind relationale Algebra Ausdrücke
  - $\theta, \theta_1, \theta_2 \ldots$ sind Prädikate (z.B. $A < B \wedge C = D$)

Definition von relationalen Algebra Ausdrücken

- Ein elementarer Ausdruck der relationalen Algebra ist
  - eine Relation in der Datenbank (z.B. Konten)
- Zusammengesetzte Ausdrücke: Falls E_1 und E_2 relationale Algebra Ausdrücke sind, dann lassen sich durch relationale Operatoren weitere Ausdrücke bilden, z.B.:
  - $E_1 \cup E_2$
  - $E_1 - E_2$
  - $E_1 \times E_2$
  - $\sigma_\theta(E_1)$, θ ist ein Prädikat in E_1
  - $\pi_A(E_1)$, A ist eine Liste von Attributen aus E_1
- Geschlossenheit der relationalen Algebra: elementare und zusammengesetzte Ausdrücke können gleich behandelt werden

Äquivalenzregeln/1

Selektion und Projektion:

- ER1 Konjunktive Selektionsprädikate können in mehrere Selektionen aufgebrochen werden:

$$\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$$

- ER2 Selektionen sind untereinander vertauschbar:

$$\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$$

- ER3 Geschachtelte Projektionen können eliminiert werden:

$$\pi_{A1}(\pi_{A2}(\ldots(\pi_{An}(E))\ldots)) = \pi_{A1}(E)$$

  (Ai sind Listen von Attributen)

- ER4 Selektion kann mit Kreuzprodukt und θ-Join kombiniert werden:

  (a) $\sigma_{\theta}(E_1 \times E_2) = E_1 \bowtie_{\theta} E_2$
  (b) $\sigma_{\theta 1}(E_1 \bowtie_{\theta 2} E_2) = E_1 \bowtie_{\theta 1 \wedge \theta 2} E_2$

Äquivalenzregeln/2

Kommutativität und Assoziativität von Joins:

- ER5 Theta-Join und natürlicher Join sind kommutativ:

$$E_1 \bowtie_\theta E_2 = E_2 \bowtie_\theta E_1$$

$$E_1 \bowtie E_2 = E_2 \bowtie E_1$$

- ER6 Joins und Kreuzprodukte sind assoziativ:
  - (a) Natürliche Joins sind assoziativ:

$$(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$$

  - (b) Theta-Joins sind assoziativ:

$$(E_1 \bowtie_{\theta 1} E_2) \bowtie_{\theta 2 \wedge \theta 3} E_3 = E_1 \bowtie_{\theta 1 \wedge \theta 3} (E_2 \bowtie_{\theta 2} E_3)$$

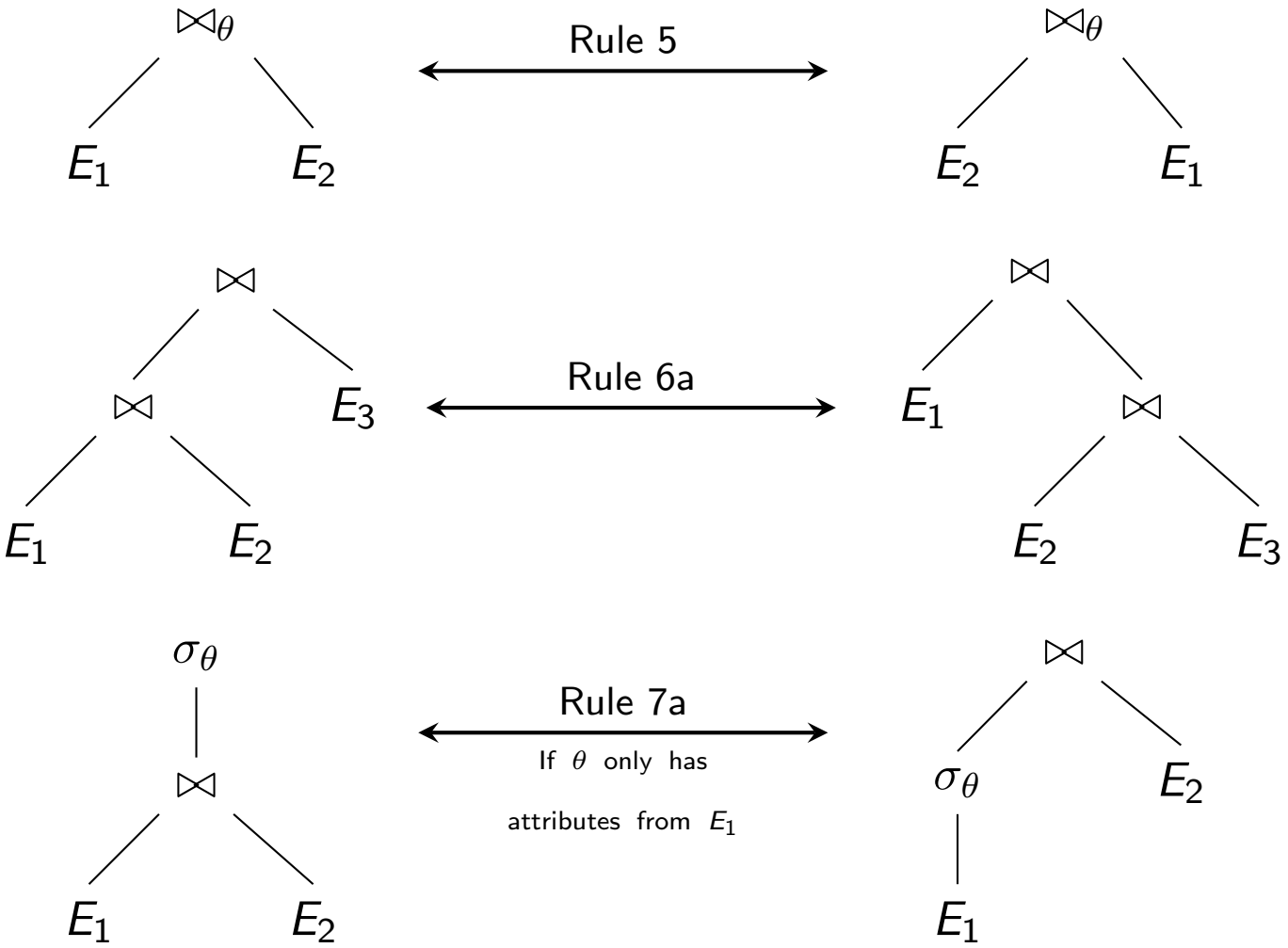
    (θ_2 enthält nur Attribute von E_2 und E_3)
  - (c) Jedes Prädikat θ_i im Theta-Join kann leer sein, also sind auch Kreuzprodukte assoziativ.

Äquivalenzregeln/3

- ER7 Selektion kann bedingt an Join vorbeigeschoben werden:
  - (a) θ_1 enthält nur Attribute eines Ausdrucks (E1):
    $$\sigma_{\theta_1}(E_1 \bowtie_\theta E_2) = \sigma_{\theta_1}(E_1) \bowtie_\theta E_2$$
  - (b) θ_1 enthält nur Attribute von E_1 und θ_2 enthält nur Attribute von E_2:
    $$\sigma_{\theta 1 \wedge \theta 2}(E_1 \bowtie_\theta E_2) = \sigma_{\theta 1}(E_1) \bowtie_\theta \sigma_{\theta 2}(E_2)$$

Beispiel: Äquivalenzregeln

- Darstellung einiger Äquivalenzregeln als Operatorbaum



Äquivalenzregeln/4

- ER8 Projektion kann an Join und Selektion vorbeigeschoben werden:
  - A_1 und A_2 sind jeweils Projektions-Attribute von E_1 und E_2.

(a) Join: θ enthält nur Attribute aus $A_1 \cup A_2$:

$$\pi_{A_1 \cup A_2}(E_1 \bowtie_\theta E_2) = \pi_{A_1}(E_1) \bowtie_\theta \pi_{A_2}(E_2)$$

(b) Join: θ enthält Attribute die nicht in $A_1 \cup A_2$ vorkommen:

- A_3 sind Attribute von E_1 die in θ vorkommen, aber nicht in $A_1 \cup A_2$
- A_4 sind Attribute von E_2 die in θ vorkommen, aber nicht in $A_1 \cup A_2$

$$\pi_{A_1 \cup A_2}(E_1 \bowtie_\theta E_2) = \pi_{A_1 \cup A_2}(\pi_{A_1 \cup A_3}(E_1) \bowtie_\theta \pi_{A_2 \cup A_4}(E_2))$$

(c) Selektion: θ enthält nur Attribute aus A_1:

$$\pi_{A_1}(\sigma_\theta(E_1)) = \sigma_\theta(\pi_{A_1}(E_1))$$

(d) Selektion: θ enthält Attribute A_3 die nicht in A_1 vorkommen:

$$\pi_{A_1}(\sigma_\theta(E_1)) = \pi_{A_1}(\sigma_\theta(\pi_{A_1 \cup A_3}(E_1)))$$

Äquivalenzregeln/5

Mengenoperationen:

- ER9 Vereinigung und Schnittmenge sind kommutativ:

$$E_1 \cup E_2 = E_2 \cup E_1$$
$$E_1 \cap E_2 = E_2 \cap E_1$$

- ER10 Vereinigung und Schnittmenge sind assoziativ.

$$(E_1 \cup E_2) \cup E_3 = E_1 \cup (E_2 \cup E_3)$$
$$(E_1 \cap E_2) \cap E_3 = E_1 \cap (E_2 \cap E_3)$$

Äquivalenzregeln/6

- ER11 Selektion kann an $\cup, \cap$ und $-$ vorbeigeschoben werden:

$$\sigma_\theta(E_1 - E_2) = \sigma_\theta(E_1) - \sigma_\theta(E_2)$$
$$\sigma_\theta(E_1 \cup E_2) = \sigma_\theta(E_1) \cup \sigma_\theta(E_2)$$
$$\sigma_\theta(E_1 \cap E_2) = \sigma_\theta(E_1) \cap \sigma_\theta(E_2)$$

  Für $\cap$ und $-$ gilt außerdem:

$$\sigma_\theta(E_1 \cap E_2) = \sigma_\theta(E_1) \cap E_2$$
$$\sigma_\theta(E_1 - E_2) = \sigma_\theta(E_1) - E_2$$

- ER12 Projektion kann an Vereinigung vorbeigeschoben werden:

$$\pi_L(E_1 \cup E_2) = \pi_L(E_1) \cup \pi_L(E_2)$$

Integrierte Übung 4.2

Stellen Sie die folgenden relationalen Algebra Ausdrücke als Operatorbäume dar:

- $RA1 = \pi_A(R1) \cup \sigma_{X>5}(R2)$
- $RA2 = \pi_A(R1 \bowtie \sigma_{X=Y}(R2 \bowtie \pi_{B,C}(R3 - R4) \bowtie R5))$
  (relationale Operatoren sind linksassoziativ)

Integrierte Übung 4.3

Folgende Äquivalenzregeln sind falsch. Zeigen Sie dies durch ein Gegenbeispiel:

1. $\pi_A(R - S) = \pi_A(R) - \pi_A(S)$

2. $R - S = S - R$

3. $(R - S) - T = R - (S - T)$

4. $\sigma_\theta(E_1 \cup E_2) = \sigma_\theta(E_1) \cup E_2$

Inhalt

Aufzählung Äquivalenter Ausdrücke

- Optimierer verwenden die Äquivalenzregeln um systematisch äquivalente Ausdrücke zu erzeugen.

- Aufzählung aller äquivalenten Ausdrücke von E:
  $X = \{E\}$ (X ist die Menge aller äquivalenten Ausdrücke)
  repeat
    for each $E_i \in X$:
      wende alle möglichen Äquivalenzumformungen an
      speichere erhaltene Ausdrücke in X
  until keine weiteren Ausdrücke können gefunden werden

- Sehr zeit- und speicherintensiver Ansatz.

Effiziente Aufzählungstechniken

- Speicher sparen: Ausdrücke teilen sich gemeinsame Teilausdrücke:
  - Wenn $E2$ aus $E1$ durch eine Äquivalenzumformung entsteht, bleiben die tieferliegenden Teilbäume gleich und brauchen nicht doppelt abgelegt zu werden.
- Zeit sparen: Aufgrund von Kostenabschätzungen werden einige Ausdrücke gar nie erzeugt.
  - Wenn für einen Teilausdruck E' ein äquivalenter Teilausdruck E'' gefunden wird, der schneller ist, brauchen keine Ausdrücke die E' enthalten berücksichtigt werden.
- Heuristik: Wende Heuristiken an um viel versprechende Ausdrücke zu erzeugen:
  - Selektionen möglichst weit nach unten
  - Projektionen möglichst weit nach unten
  - Joins mit kleinem zu erwartenden Ergebnis zuerst berechnen

Heuristische Optimierung/1

- Heuristische Optimierung transformiert den Operatorbaum nach einer Reihe von Heuristiken, welche die Ausführung normalerweise (jedoch nicht in allen Fällen) beschleunigen.
- Ziel der Heuristiken: Größe der Zwischenergebnisse so früh als möglich (d.h. nahe an den Blättern des Operatorbaums) klein machen.
- Einige (alte) Systeme verwenden nur heuristische Optimierung.
- Modern Systeme kombinieren Heuristiken (nur einige Ausdrücke werden betrachtet) mit kostenbasierter Optimierung (schätze die Kosten für jeden betrachteten Ausdruck ab).

Heuristische Optimierung/2

- Typischer Ansatz der heuristischen Optimierung:
  1. Transformiere alle konjunktiven Selektionen in eine Reihe verschachtelter Selektionen (ER1).
  2. Schiebe Selektionen so weit als möglich im Operatorbaum nach unten (ER2, ER7(a), ER7(b), ER11).
  3. Ersetze Kreuzprodukte, welche von einer Selektion gefolgt sind, durch Joins (ER4(a)).
  4. Führe Joins und Selektionen mit starker Selektivität zuerst aus (ER6).
  5. Schiebe Projektionen so weit nach unten als möglich und erzeuge neue Projektionen, sodass kein Attribut weitergeleitet wird, das nicht mehr gebraucht wird (ER3, ER8, ER12).
  6. Identifiziere die Teilbäume, für die Pipelining möglich ist, und führe diese mit Pipelining aus.

Äquivalenzumformung: Beispieltabellen

- Schemas der Beispieltabellen:

  branch(<u>*branch-name*</u>*, branch-city, assets)*
  account(<u>*account-number*</u>*, branch-name, balance)*
  depositor(<u>*customer-name*</u>*, account-number)*

- Fremdschlüsselbeziehungen:

  $\pi_{\textit{branch-name}}(\textit{account}) \subseteq \pi_{\textit{branch-name}}(\textit{branch})$
  $\pi_{\textit{account-number}}(\textit{depositor}) \subseteq \pi_{\textit{account-number}}(\textit{account})$

Beispiele Äquivalenzumformungen/1

- Beispiel 1: Selektion nach unten schieben.
- Anfrage: Finde die Namen aller Kunden die ein Konto in einer Filiale in Brooklyn haben.

  $\pi_{\textit{customer-name}}(\sigma_{\textit{branch-city}='\textit{Brooklyn}'}(\textit{branch} \bowtie (\textit{account} \bowtie \textit{depositor})))$
- Der Join wird für die Konten und Kunden aller Filialen berechnet, obwohl wir nur an den Filialen in Brooklyn interessiert sind.
- Umformung unter Verwendung von ER7(a):

  $\pi_{\textit{customer-name}}$
  $(\sigma_{\textit{branch-city}='\textit{Brooklyn}'}(\textit{branch}) \bowtie (\textit{account} \bowtie \textit{depositor}))$
- Die Selektion wird vorgezogen, damit sich die Größe der Relationen, auf die ein Join berechnet werden muss, reduziert.

Beispiele Äquivalenzumformungen/2

- Beispiel 2: Oft sind mehrere Umformungen notwendig.
- Anfrage: Finde die Namen aller Kunden mit einem Konto in Brooklyn, deren Kontostand kleiner als 1000 ist.

$$\pi_{\textit{customer-name}}(\sigma_{\textit{branch-city}='\textit{Brooklyn}'\wedge\textit{balance}<1000}(\textit{branch}\bowtie(\textit{account}\bowtie\textit{depositor})))$$

- Umformung 1: ER6(a) (Join Assoziativität):

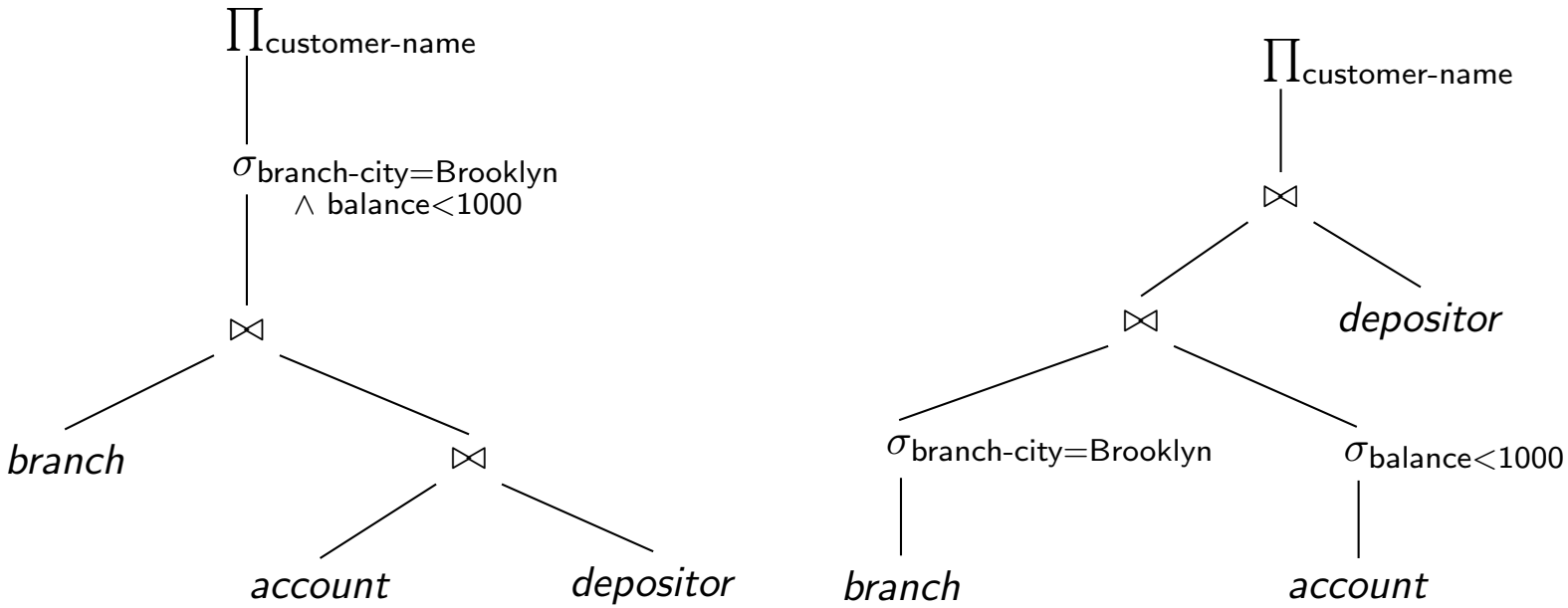
$$\pi_{\textit{customer-name}}(\sigma_{\textit{branch-city}='\textit{Brooklyn}'\wedge\textit{balance}<1000}((\textit{branch}\bowtie\textit{account})\bowtie\textit{depositor}))$$

- Umformung 2: ER7(a) und (b) (Selektion nach unten schieben)

$$\pi_{\textit{customer-name}}(\sigma_{\textit{branch-city}='\textit{Brooklyn}'}(\textit{branch})\bowtie\sigma_{\textit{balance}<1000}(\textit{account})\bowtie\textit{depositor})$$

Beispiele Äquivalenzumformungen/3

- Beispiel 2 (Fortsetzung)
  - Operatorbaum vor und nach den Umformungen.



Beispiele Äquivalenzumformungen/4

- Beispiel 3: Projektion
- Anfrage: (wie Beispiel 1)

  $\pi_{\textit{customer-name}}((\sigma_{\textit{branch-city}='\textit{Brooklyn}'}(\textit{branch}) \bowtie \textit{account}) \bowtie \textit{depositor})$
- Join $\sigma_{\textit{branch-city}='\textit{Brooklyn}'}(\textit{branch}) \bowtie \textit{account}$ ergibt folgendes Schema:

  (branch-name, branch-city, assets, account-number, balance)
- Nur 1 Attribute wird gebraucht: *account-number* für Join mit *depositor*.
- Umformung: ER8(b) (Projektion nach unten schieben):

  $\pi_{\textit{customer-name}}$
  $(\pi_{\textit{account-number}}(\sigma_{\textit{branch-city}='\textit{Brooklyn}'}(\textit{branch}) \bowtie \textit{account})$
  $\bowtie \textit{depositor})$

Integrierte Übung 4.4

- Verwenden Sie die Äquivalenzregeln, um die Projektionen so weit als möglich nach unten zu schieben:

$$\pi_{\textit{customer-name}}\Big(\big(\pi_{\textit{account-number}}(\sigma_{\textit{branch-city}='\textit{Brooklyn}'}(\textit{branch}) \bowtie \textit{account})\big) \bowtie \textit{depositor}\Big)$$

- Lösung:
  - Anwendung von ER8(b): $A_1 = \emptyset$, $A_2 = \{\textit{account-number}\}$, $A3 = A4 = \{\textit{branch-name}\}$

$$\pi_{\textit{customer-name}}\Big(\pi_{\textit{account-number}}\big(\pi_{\textit{branch-name}}(\sigma_{\textit{branch-city}='\textit{Brooklyn}'}(\textit{branch}))\big) \bowtie \pi_{\textit{account-number},\textit{branch-name}}(\textit{account})\big) \bowtie \textit{depositor}\Big)$$

  - Anwendung von ER8(d): $A_1 = \{\textit{branch-name}\}$, $A_3 = \{\textit{branch-city}\}$

$$\pi_{\textit{customer-name}}\Big(\pi_{\textit{account-number}}\big(\pi_{\textit{branch-name}}(\sigma_{\textit{branch-city}='\textit{Brooklyn}'}\big(\pi_{\textit{branch-name},\textit{branch-city}}(\textit{branch})\big)\big) \bowtie \pi_{\textit{account-number},\textit{branch-name}}(\textit{account})\big) \bowtie \textit{depositor}\Big)$$

Beispiele Äquivalenzumformungen/5

- Beispiel 4: Joinreihenfolge
- Für alle Relationen r_1, r_2, r_3 gilt (Assoziativität):
$$(r_1 \bowtie r_2) \bowtie r_3 = r_1 \bowtie (r_2 \bowtie r_3)$$
- Falls $r_2 \bowtie r_3$ groß ist und $r_1 \bowtie r_2$ klein, wählen wir die Reihenfolge
$$(r_1 \bowtie r_2) \bowtie r_3$$
sodass nur ein kleines Zwischenergebnis berechnet und evtl. zwischengespeichert werden muss.

Beispiele Äquivalenzumformungen/6

- Beispiel 5: Joinreihenfolge
- Anfrage:

  $\pi_{\textit{customer-name}}(\sigma_{\textit{branch-city}='\textit{Brooklyn}'}(\textit{branch}) \bowtie \textit{account} \bowtie \textit{depositor})$

- Welcher Join soll zuerst berechnet werden?
  (a) $\sigma_{\textit{branch-city}='\textit{Brooklyn}'}(\textit{branch}) \bowtie \textit{depositor}$
  (b) $\sigma_{\textit{branch-city}='\textit{Brooklyn}'}(\textit{branch}) \bowtie \textit{account}$
  (c) $\textit{account} \bowtie \textit{depositor}$

- (a) ist ein Kreuzprodukt, da *branch* und *depositor* keine gemeinsamen Attribute haben
  $\rightarrow$ sollte vermieden werden

- (b) ist vermutlich kleiner als (c), da (b) nur die Konten in Brooklyn berücksichtigt, (c) jedoch alle Konten.

Integrierte Übung 4.5

Stellen Sie die folgende Anfrage als Operatorbaum dar und führen Sie günstige Äquivalenzumformungen durch:

```
SELECT DISTINCT E.LName
FROM Employee E, WorksOn W, Project P
WHERE P.PName = 'A'
AND P.PNum = W.PNo
AND W.ESSN = E.SSN
AND E.BDate = '31.12.1957'
```

Inhalt

Kostenbasierte Optimierung

- Kostenbasierte Optimierer schätzen die Kosten aller möglichen Anfragepläne ab und wählen den billigsten (=schnellsten).

- Kostenabschätzung erfolgt aufgrund von
  - Datenbankstatistik (im Katalog gespeichert)
  - Wissen über die Kosten der Operatoren (z.B. Hash Join braucht $3(b_r + b_s)$ Blockzugriffe für $r \bowtie s$)
  - Wissen über die Interaktion der Operatoren (z.B. sortiertes Lesen mit einem Index ermöglicht Merge Join statt Sort-Merge Join)

Kombination von Kosten mit Heuristiken

- kostenbasierte Optimierung: durchsuche alle Pläne und suche den billigsten
- heuristische Optimierung: erzeuge einen vielversprechenden Plan nach heuristischen Regeln
- Praktische Optimierer kombinieren beide Techniken:
  - erzeuge eine Menge vielversprechender Pläne
  - wähle den billigsten
  - Plan wird sofort bewertet, sobald er erzeugt wird (und evtl. verworfen)

Teilpläne bewerten

- Optimierer kann Teilpläne bewerten und langsame, äquivalente Teilpläne verwerfen.
  - Dadurch reduziert sich die Menge der Teilpläne, die betrachtet werden müssen.
  - Es reicht jedoch nicht, nur den jeweils schnellsten Teilbaum zu behalten.

- Beispiel:
  - Hash Join ist schneller als Merge Join
  - es kann dennoch besser sein, den Merge Join zu verwenden, wenn die Ausgabe sortiert sein muss
  - der Merge Join liefert ein sortiertes Ergebnis und man spart sich einen zusätzlichen Sortierschritt

Datenbankstatistik/1

- Katalog (Datenbankverzeichnis) speichert u.A. Informationen über die gespeicherten Daten.
- Statistik über Index: Anzahl der Ebenen in Index i
- Statistik über Tabelle $R(A_1, A_2, \ldots, A_n)$:
  - n_R: Anzahl der Tuple in R
  - b_R: Anzahl der Blöcke, auf denen R gespeichert ist
  - $V(R, A) = |\pi_A(R)|$: Anzahl der unterschiedlichen Werte in den Attribut A
- Beispiel: $V(R, A_1) = 1,\ V(R, A_2) = 3,\ V(R, A_3) = 2$

A_1	A_2	A_3
a	b	c
a	x	d
a	y	c

Join Reihenfolgen/1

- Kostenbasierte Optimierung kann verwendet werden, um die beste Join Reihenfolge herauszufinden.
- Join Reihenfolgen der Relationen entstehen durch:
  - Assoziativgesetz: $(R_1 \bowtie R_2) \bowtie R_3 = R_1 \bowtie (R_2 \bowtie R_3)$
  - Kommutativgesetz: $R_1 \bowtie R_2 = R_2 \bowtie R_1$
- Die Join Reihenfolge hat große Auswirkung auf Effizienz:
  - Größe der Zwischenergebnisse
  - Auswahlmöglichkeit der Algorithmen (z.B. vorhandene Indices verwenden)

Join Reihenfolgen/2

Wieviele Reihenfolgen gibt es für $R_1 \bowtie R_2 \bowtie \ldots \bowtie R_m$?

- Assoziativgesetz:
  - Operatorbaum: es gibt C_{m-1} volle binäre Bäume mit m Blättern (anders ausgedrückt: es gibt C_{m-1} Klammerungen von m Operanden)
  - dabei ist C_n die Catalan-Zahl:

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!n!} \qquad n \geq 0$$

- Kommutativgesetz:
  - Blätter des Operatorbaums sind die Relationen $R_1, R_2, \ldots R_m$
  - für jeden Operatorbaum gibt es $m!$ Permutationen
- Anzahl der Join-Reihenfolgen für m Relationen:

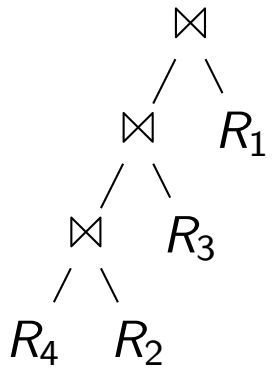
$$m!\,C_{m-1} = \frac{(2(m-1))!}{(m-1)!}$$

Join Reihenfolgen/3

- Anzahl der Join-Reihenfolgen wächst sehr schnell an:
  - $m = 3$: 12 Reihenfolgen
  - $m = 7$: 665.280 Reihenfolgen
  - $m = 10$: $>$ 17.6 Milliarden Reihenfolgen
- Dynamic Programming Ansatz:
  - Laufzeit Komplexität: $O(3^m)$
  - Speicher Komplexität: $O(2^m)$
- Beispiel: $m = 10$
  - Anzahl der Join-Reihenfolgen: 17.6×10^9
  - Dynamic Programming: $3^m = 59049$
- Trotz Dynamic Programming bleibt Aufzählung der Join-Reihenfolgen teuer.

Join Reihenfolgen/4

- Left-deep Join Reihenfolgen
  - rechter Join-Operator ist immer eine Relation (nicht Join-Ergebnis)
  - dadurch ergeben sich sog. left-deep Operatorbäume (im Gegensatz zu "bushy", wenn alle Operatorbäume erlaubt sind)



- Anzahl der left-deep Join Reihenfolgen für m Relationen: $O(m!)$
- Dynamic Programming: Laufzeit $O(m\,2^m)$.
- Vergleich für m Relationen und Beispiel $m = 10$:

	left-deep	bushy
#Baumformen	1	C_{m-1}
#Join Reihenfolgen	$m!$	$\frac{(2(m-1))!}{(m-1)!}$
Dynamic Programming	$O(m\,2^m)$	$O(3^m)$

$m = 10$	left-deep	bushy
	1	4.862
	3.63×10^6	1.76×10^{10}
	10.240	59.049

Greedy Algorithms für Join Reihenfolgen/1

- Ansatz: In jedem Schritt wird der Join mit dem kleinsten Zwischenergebnis verwendet.
- Überblick: Greedy Algorithms für Join Reihenfolge
  - nur left-deep Join Reihenfolgen werden betrachtet
  - Relationen-Paar mit dem kleinsten Join Ergebnis kommt zuerst dran
  - in jedem weiteren Schritt wird jene Relation dazugegeben, die mit dem vorhanden Operatorbaum das kleinste Join-Ergebnis erzeugt
- Algorithmus: Join Reihenfolge von $S = \{R_1, R_2, \ldots, R_m\}$
  1. $O \leftarrow R_i \bowtie R_j$, sodass $|R_i \bowtie R_j|$ minimal ist $(i \neq j)$
  2. $S \leftarrow S - \{R_i, R_j\}$
  3. **while** $S \neq \emptyset$ **do**
     a. wähle $R_i \in S$ sodass $|O \bowtie R_i|$ minimal ist
     b. $O \leftarrow O \bowtie R_i$
     c. $S \leftarrow S - \{R_i\}$
  4. **return** Operatorbaum O

Greedy Algorithms für Join Reihenfolgen/2

- Greedy Algorithms benötigt Abschätzung für Join Kardinalität.
- Abschätzung erfolgt aufgrund der Anzahl der unterschiedlichen Werte für die Join Attribute, z.B. $V(R, A)$.
- Abschätzung für $|R \bowtie S|$ mit dem Join Attribut A:

$$|R \bowtie S| \approx \frac{|R| \cdot |S|}{\max(V(R, A), V(S, A))}$$

- Annahmen über die Werte der Attribute (A ist Join-Attribut):
  - *Geleichverteilung*: Jeder der Werte in $\pi_A(R)$ bzw. $\pi_A(S)$ kommt mit der gleichen Wahrscheinlichkeit vor.
  - *Teilmenge*: $V(R, A) \leq V(S, A) \Rightarrow \pi_A(R) \subseteq \pi_A(S)$
  - *Werterhaltung*: falls Attribut B in R vorkommt aber nicht in S (d.h. B ist kein Join-Attribut), dann gilt: $V(R \bowtie S, B) = V(R, B)$

Greedy Algorithms für Join Reihenfolgen/3

- Beispiel: schätze $|R \bowtie S|$ ab, wobei folgende Statistik gegeben ist.

$R(A,B)$	$S(B,C)$
$n_R = 1000$	$n_S = 2000$
$V(R,B) = 20$	$V(S,B) = 500$

- Abschätzung:

$$|R \bowtie S| \approx \frac{n_R \cdot n_S}{\max(V(R,B), V(S,B))} = \frac{1000 \cdot 2000}{500} = 4000$$

Greedy Algorithms für Join Reihenfolgen/4

- Bisherige Abschätzung ist limitiert auf 1 Join-Attribut zwischen 2 Relationen.
- Für den Greedy Algorithmus muss die Abschätzung verallgemeinert werden:
  - m Relationen $R_1, R_2, \ldots, R_m$
  - beliebig viele Join-Attribute (A ist Join-Attribut wenn es in mindestens zwei Relationen vorkommt)
- Verallgemeinerung der Abschätzung:
  1. starte mit der Größe des Kreuzproduktes $|R_1| \cdot |R_2| \cdot \ldots \cdot |R_m|$
  2. für jedes Join-Attribut: dividiere durch alle $V(R_i, A)$ außer durch das kleinste

Greedy Algorithms für Join Reihenfolgen/5

- Beispiel: schätze $|R \bowtie S \bowtie T|$ ab, wobei folgende Statistik gegeben ist.

$R(A,B,C)$	$S(B,C,D)$	$T(B,E)$
$n_R = 1000$	$n_S = 2000$	$n_T = 5000$
$V(R,A) = 100$		
$V(R,B) = 20$	$V(S,B) = 50$	$V(T,B) = 200$
$V(R,C) = 200$	$V(S,C) = 100$	
	$V(S,D) = 400$	
		$V(T,E) = 500$

- Abschätzung:

$$|R \bowtie S \bowtie T| \approx \frac{n_R \cdot n_S \cdot n_T}{V(S,B) \cdot V(T,B) \cdot V(R,C)} = 5000$$

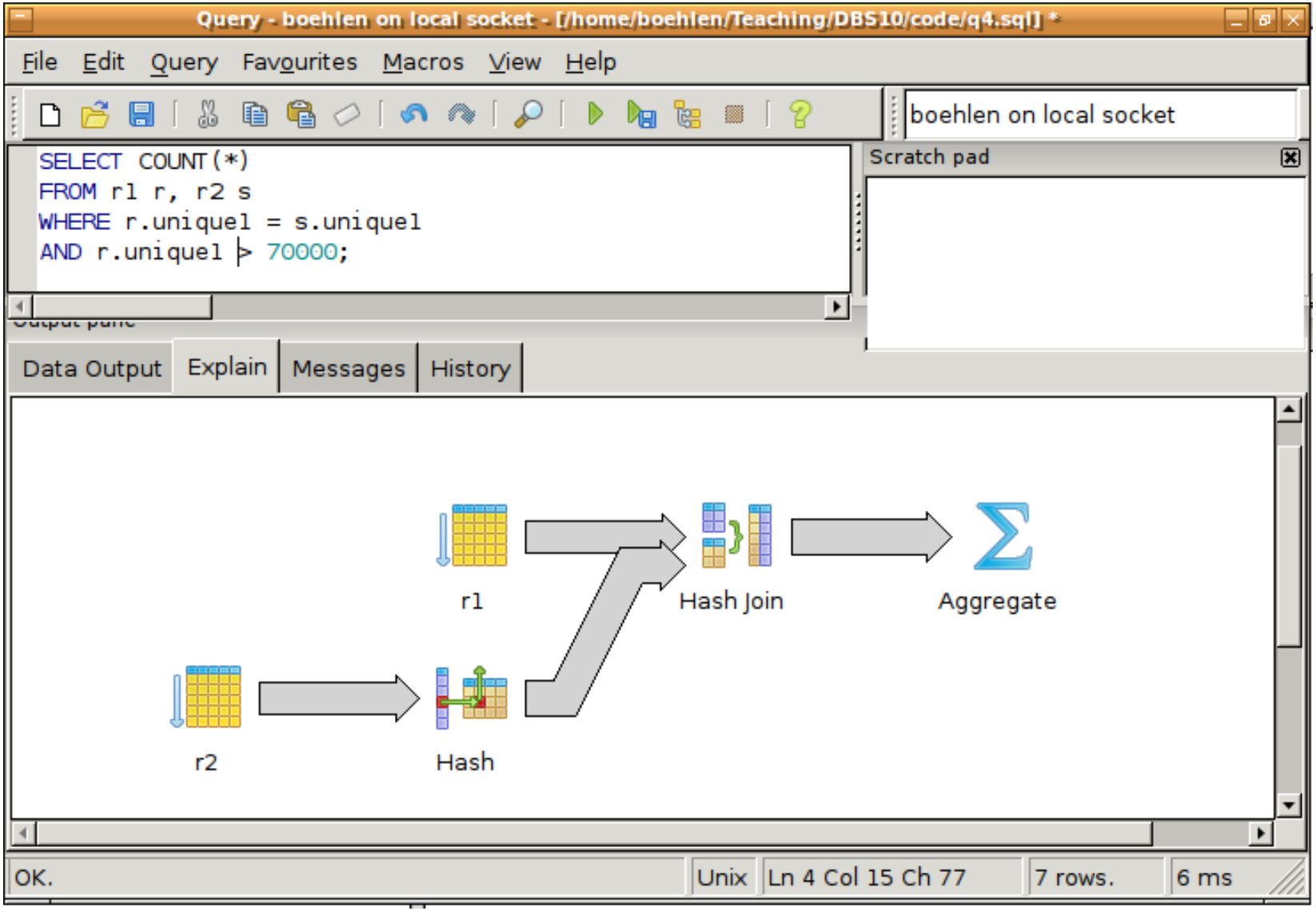
Integrierte Übung 4.6

Eine Datenbank mit folgenden Relationen ist gegeben:

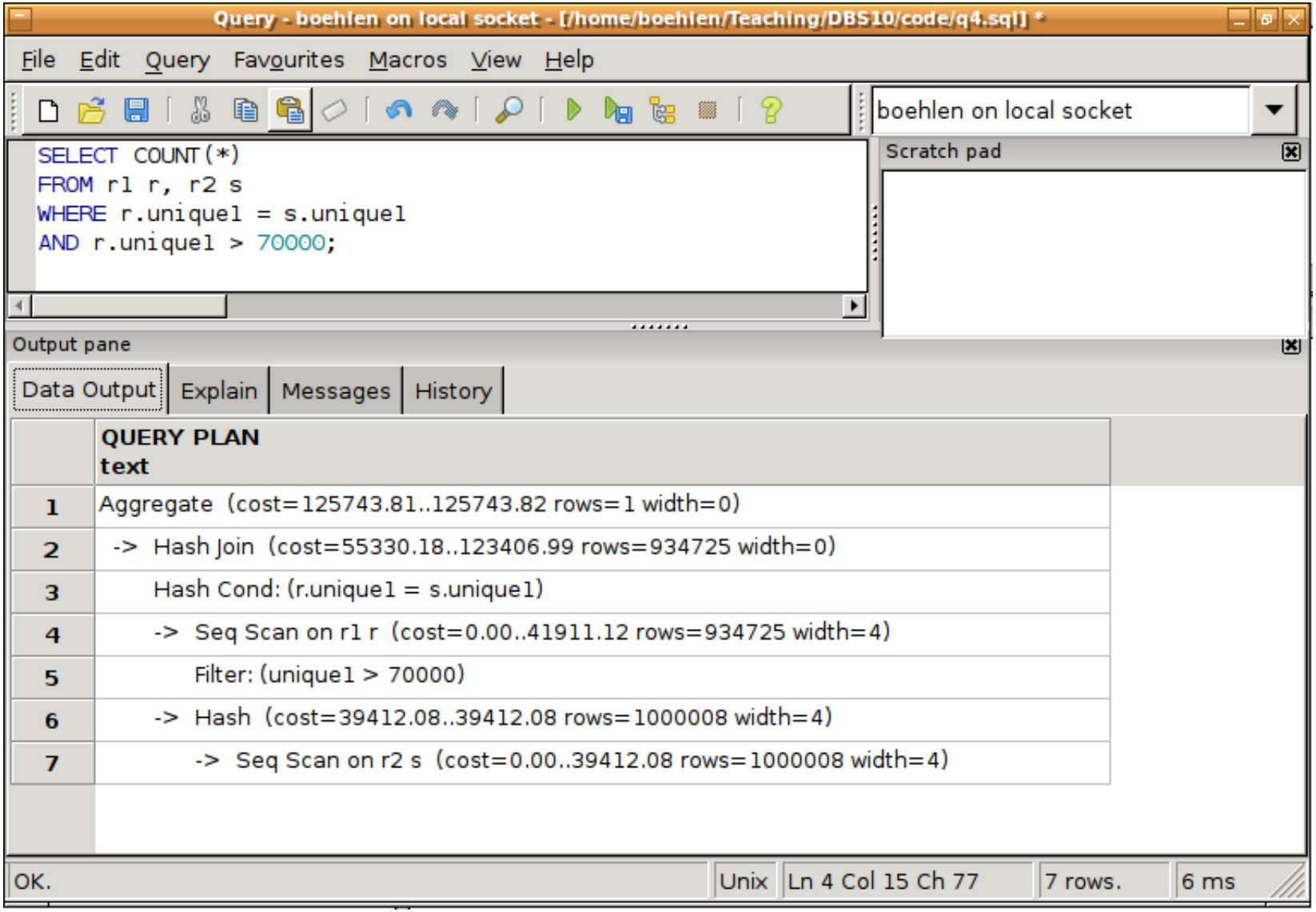
- $|R_1(A, B, C)| = 1000, V(R_1, C) = 900$
- $|R_2(C, D, E)| = 1500, V(R_2, C) = 1100, V(R_2, D) = 50, V(R_2, E) = 50$
- $|R_3(D, E)| = 750, V(R_3, D) = 50, V(R_3, E) = 100$

Finden Sie eine effiziente Join Reihenfolge für den Join $R_1 \bowtie R_2 \bowtie R_3$ und berechnen Sie die Kardinalität des Join-Ergebnisses.

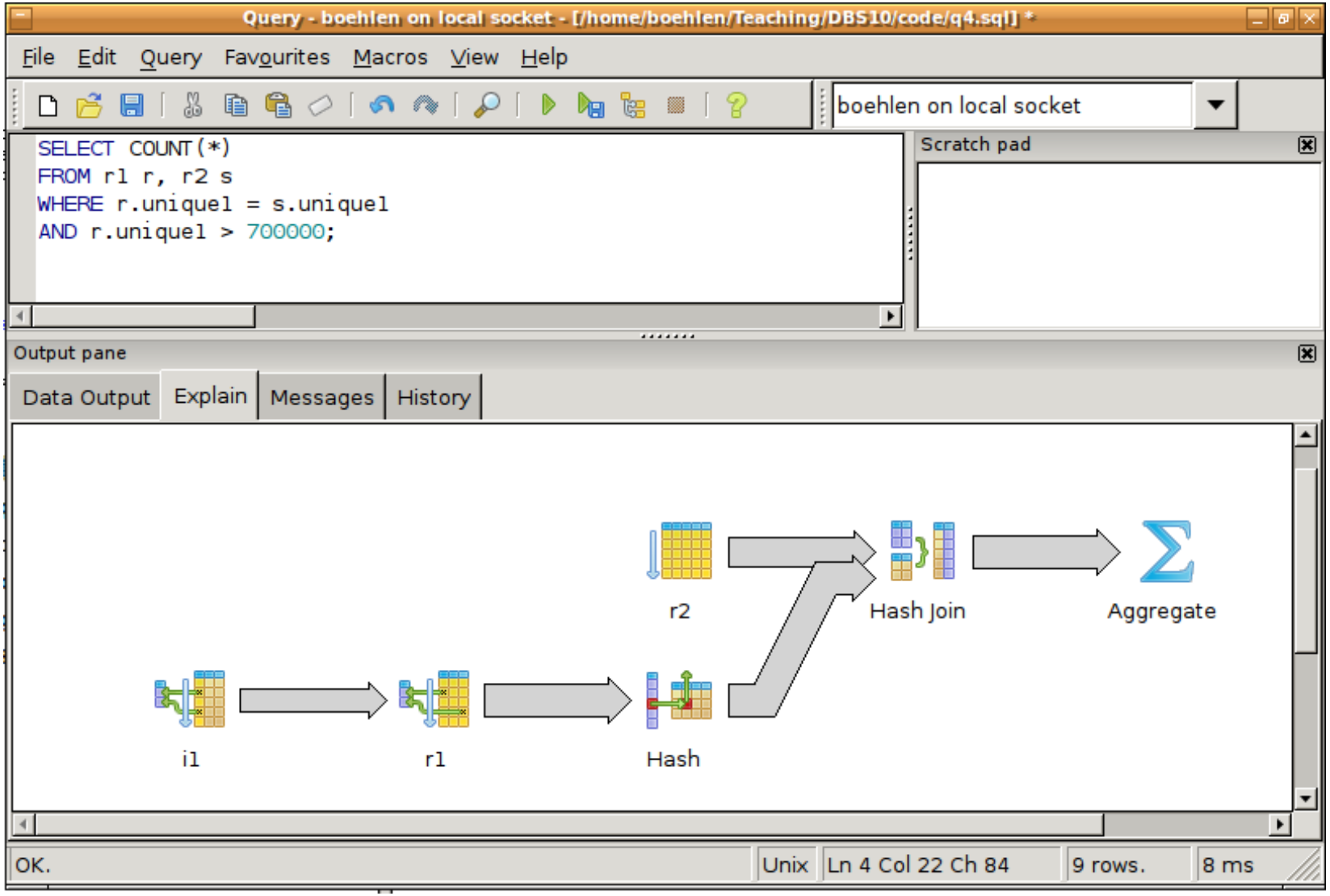
PostgreSQL Anfrageoptimierung/1



PostgreSQL Anfrageoptimierung/2



PostgreSQL Anfrageoptimierung/3



PostgreSQL Anfrageoptimierung/4