TU München, Fakultät für Informatik
Lehrstuhl III: Datenbanksysteme
Prof. Dr. Thomas Neumann

Übung zur Vorlesung *Grundlagen: Datenbanken* im WS19/20

Christoph Anneser, Moritz Sichert, Lukas Vogel (gdb@in.tum.de)
https://db.in.tum.de/teaching/ws1920/grundlagen/

Blatt Nr. 13

Hausaufgabe 1

Gegeben sei die folgende SQL-Anfrage:

```
select distinct a.PersNr, a.Name
from Assistenten a, Studenten s, pruefen p
where s.MatrNr = p.MatrNr
  and a.Boss = p.PersNr
  and s.Name = 'Jonas';
```

Geben Sie die kanonische Übersetzung dieser Anfrage in die relationale Algebra an. Verwenden Sie zur Darstellung des relationalen Algebraausdrucks die Baumdarstellung.

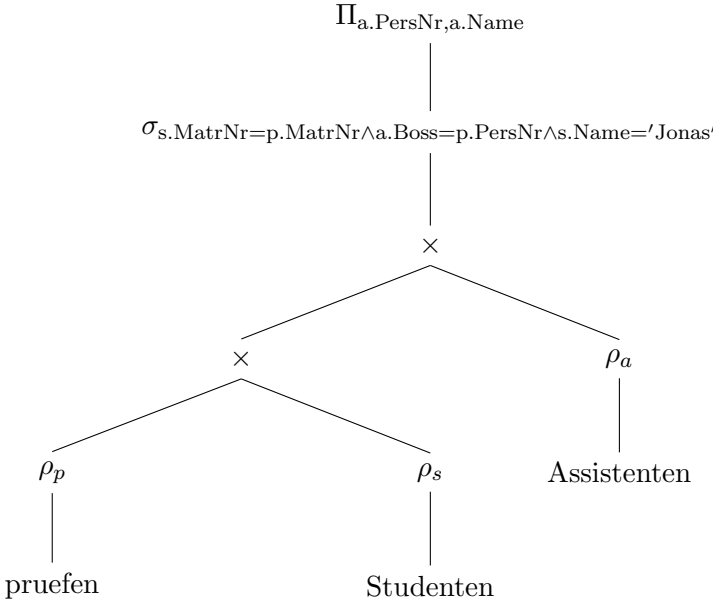
Optimieren Sie Ihren relationalen Algebraausdruck logisch. Gehen Sie dabei von **realistischen** Kardinalitäten für die relevanten Relationen aus.

Verwenden Sie hierfür die folgenden aus der Vorlesung bekannten Optimierungstechniken:

- Aufbrechen von Selektionen
- Verschieben von Selektionen nach "unten" im Plan
- Zusammenfassen von Selektionen und Kreuzprodukten zu Joins
- Bestimmung der Joinreihenfolge

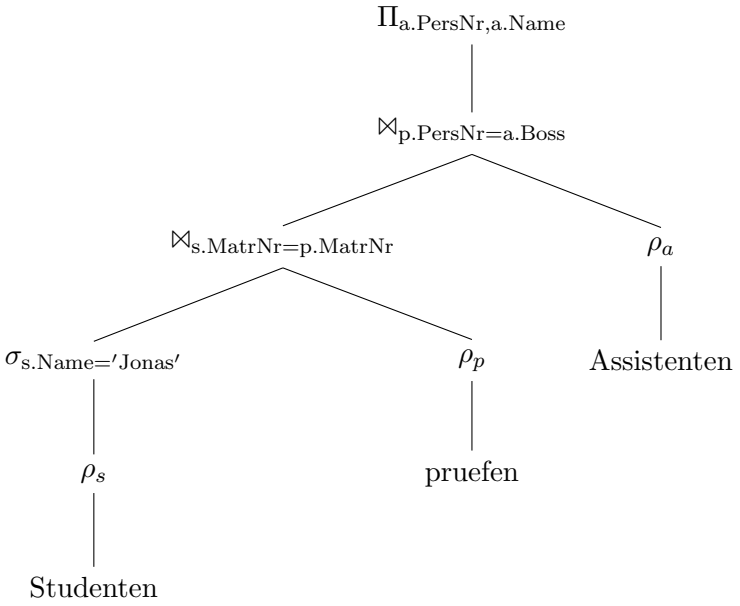
Lösung:

Kanonische Übersetzung:



realistische Kardinalitäten: nur ein Student mit dem Namen 'Jonas', viel mehr Assistenten, deshalb Studenten zuerst, dann über pruefen mit Assistenten joinen

logische Optimierung: Selektionen ganz nach unten, Joins statt Kreuzprodukte & in richtiger Reihenfolge



Hausaufgabe 2

Für einen Join-Baum T sei folgende Kostenfunktion gegeben

$$C_{out}(T) = \begin{cases} 0 & \text{falls } T \text{ eine Basisrelation } R_i \text{ ist} \\ |T| + C_{out}(T_1) + C_{out}(T_2) & \text{falls } T = T_1 \bowtie T_2 \end{cases}$$

Die Kardinaliät sei dabei

$$|T| = \begin{cases} |R_i| & \text{falls } T \text{ eine Basisrelation } R_i \text{ ist} \\ (\prod_{R_i \in T_1, R_j \in T_2} f_{i,j}) |T_1||T_2| & \text{falls } T = T_1 \bowtie T_2 \end{cases}$$

Sei $p_{i,j}$ das Join Prädikat zwischen R_i und R_j, dann sei

$$f_{i,j} = \frac{|R_i \bowtie_{p_{i,j}} R_j|}{|R_i \times R_j|}$$

und die Kardinaliät eines Join-Resultats ist $|R_i \bowtie_{p_{i,j}} R_j| = f_{i,j}|R_i||R_j|$.

Gegeben sei eine Anfrage über die Relationen R_1, R_2, R_3 und R_4 mit $|R_1| = 10, |R_2| = 20, |R_3| = 20, |R_4| = 10$. Die Selektivitäten der Joins seien $f_{1,2} = 0.01, f_{2,3} = 0.5, f_{3,4} = 0.01$, alle nicht gegebenen Selektivitäten sind offensichtlich 1 (Warum?). Berechnen Sie den optimalen (niedrigste Kosten) Join-Tree. Als Vereinfachung reicht es, wenn Sie nur Joins mit Prädikat und keine Kreuzprodukte betrachten.

Lösung:

Es ist kein Algorithmus angegeben. Aufgrund der geringen Anzahl von Relationen ist es möglich, die Kosten aller möglichen Join-Bäume zu berechnen und den kostengünstigsten auszuwählen (Bruteforce).

Zunächst gilt es zu überlegen, für welche Join-Bäume die Kosten tatsächlich zu berechnen sind.

Left-Deep:

$$((R_1 \bowtie R_2) \bowtie R_3) \bowtie R_4 \tag{1}$$
$$((R_4 \bowtie R_3) \bowtie R_2) \bowtie R_1 \tag{2}$$
$$((R_3 \bowtie R_2) \bowtie R_1) \bowtie R_4 \tag{3}$$
$$((R_3 \bowtie R_2) \bowtie R_4) \bowtie R_1 \tag{4}$$

Bushy:

$$(R_1 \bowtie R_2) \bowtie (R_3 \bowtie R_4) \tag{5}$$

Alle anderen Left Deep oder Bushy Trees enthalten Kreuzprodukte oder sind im Bezug auf die Kosten äquivalent. Ersteres entsteht, wenn Relationen in einer Reihenfolge gejoint werden, in der bei einem der Joins kein Prädikt möglich ist, beispielsweise ist dies für den Left-Deep Tree

$$((R_1 \bowtie R_2) \times R_4) \bowtie R_3$$

der Fall. Im Bezug auf die Kosten bei der gegebenen Kostenfunktion äquivalent sind Join-Trees, bei denen die Kinder eines Join Operators vertauscht wurden, etwa

$$(R_1 \bowtie R_2) \bowtie (R_3 \bowtie R_4)$$

und

$$(R_3 \bowtie R_4) \bowtie (R_1 \bowtie R_2).$$

Im Beispiel müssen lediglich die Kosten für die Join-Reihenfolgen 1, 3 und 5 berechnet werden. Dies liegt am Aufbau der Kostenfunktion sowie den symmetrischen Größen der Relationen sowie ihrer Join Selektivitäten.

Die Berechnung von $C_{out}((R_1 \bowtie R_2) \bowtie (R_3 \bowtie R_4))$ sei hier exemplarisch in epischer Breite ausgeführt (Machen Sie es selber; Sie erkennen äußerst schnell ein Muster und müssen keine derartigen Formel-Konvolute schreiben):

$$\begin{aligned}
& C_{out}((R_1 \bowtie R_2) \bowtie (R_3 \bowtie R_4)) \\
= \; & |(R_1 \bowtie R_2) \bowtie (R_3 \bowtie R_4)| + C_{out}(R_1 \bowtie R_2) + C_{out}(R_3 \bowtie R_4) \\
= \; & |(R_1 \bowtie R_2) \bowtie (R_3 \bowtie R_4)| + |R_1 \bowtie R_2| + C_{out}(R_1) + C_{out}(R_2) + |R_3 \bowtie R_4| + C_{out}(R_3) + C_{out}(R_4) \\
= \; & |(R_1 \bowtie R_2) \bowtie (R_3 \bowtie R_4)| + |R_1 \bowtie R_2| + |R_3 \bowtie R_4| \\
= \; & f_{1,3} \cdot f_{1,4} \cdot f_{2,3} \cdot f_{2,4} \cdot |R_1 \bowtie R_2| \cdot |R_3 \bowtie R_4| + |R_1 \bowtie R_2| + |R_3 \bowtie R_4| \\
= \; & 0.5 \cdot (0.01 \cdot 10 \cdot 20) \cdot (0.01 \cdot 20 \cdot 10) + (0.01 \cdot 10 \cdot 20) + (0.01 \cdot 20 \cdot 10) \\
= \; & 2 + 2 + 2 \\
= \; & 6
\end{aligned}$$

Die Ergebnisse der anderen Relevanten Join-Reihenfolgen sind:

$((R_1 \bowtie R_2) \bowtie R_3) \bowtie R_4$	24
$((R_3 \bowtie R_2) \bowtie R_1) \bowtie R_4$	222
$(R_1 \bowtie R_2) \bowtie (R_3 \bowtie R_4)$	6

Der Bushy-Tree 5 ist also der optimale Join-Tree.

Hausaufgabe 3

Gegeben sei die Anfrage:

```
select *
  from R, S, T
 where R.A = S.A and S.B = T.B and T.C = R.A
```

Des Weiteren soll gelten:

- `S.A` und `T.C` seien Fremdschlüssel auf `R`
- `S.B` sei Fremdschlüssel auf `T`
- `R.A`, `T.B` seien Primärschlüssel von `R` respektive `T`
- Ihre Query-Engine unterstützt nur Nested-Loop-Joins
- Kardinalitäten: $|R| = 100, |S| = 1000, |T| = 10$
- Es gibt keine Indexe

Bestimmen Sie, wie in der Vorlesung gezeigt, den optimalen Ausführungsplan als Baum mit Kosten-/Kardinalitätsabschätzungen mit Hilfe von Dynamischem Programmieren.

Lösung:

Für dieses einfache Beispielsytem, welches nur Nested-Loop-Joins unterstützt, kann als Kostenfunktion C verwendet werden:

$$C(T) = \begin{cases} 0 & \text{falls } T \text{ eine Basisrelation } R_i \text{ ist} \\ |T_1| \cdot |T_2| + C(T_1) + C(T_2) & \text{falls } T = T_1 \bowtie^{\text{NL}} T_2 \end{cases}$$

Da ein Nested-Loop-Joins effektiv jedes Tupel aus dem Kreuzprodukt $T_1 \times T_2$ als Zwischenergebnis betrachtet, sind seine Kosten $|T_1 \times T_2| = |T_1| \cdot |T_2|$. Des weiteren müssen natürlich auch die Kosten von T_1 und T_2 mit einberechnet werden. Die Kardinalitäten können wie gewohnt berechnet werden:

$$|T| = \begin{cases} |R_i| & \text{falls } T \text{ eine Basisrelation } R_i \text{ ist} \\ (\prod_{R_i \in T_1, R_j \in T_2} f_{i,j}) |T_1||T_2| & \text{falls } T = T_1 \bowtie^{\text{NL}} T_2 \end{cases}$$

Für Equijoins über den Fremdschlüssel gilt die Abschätzung:

$$f_{i,j} = \frac{1}{|R_i|}, \text{mit Fremdschlüssel in } R_j$$

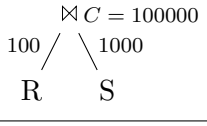
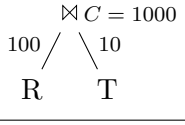
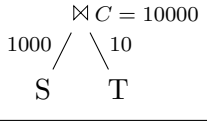
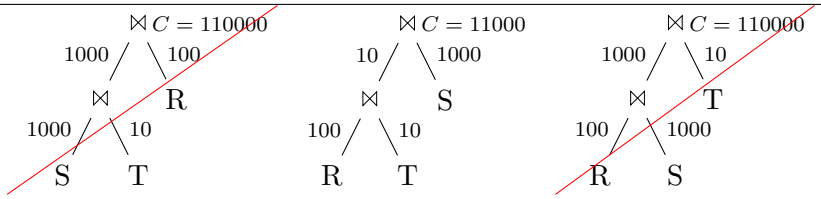
Da alle Relationen über Fremdschlüssel verbunden sind, gelten (vereinfachend) folgende Kardinalitäten:

$|R \bowtie S| = |S \bowtie R| = \frac{1}{|R|} \cdot |R| \cdot |S| = |S| = 1000$

$|R \bowtie T| = |T \bowtie R| = \frac{1}{|R|} \cdot |R| \cdot |T| = |T| = 10$

$|S \bowtie T| = |T \bowtie S| = \frac{1}{|T|} \cdot |S| \cdot |T| = |S| = 1000$

Für die Berechnung der Kosten ergibt sich dann folgende DP-Tabelle:

DP-Tabelle		
Index	Pläne	Kosten
R	R	0
S	S	0
T	T	0
R,S		100000
R,T		1000
S,T		10000
R,S,T		11000

Hausaufgabe 4

Wofür stehen die vier Buchstaben ACID? Erklären Sie für jeden der vier Konzepte, warum es für eine Datenbank wichtig ist. Geben Sie dazu jeweils ein Beispiel an, was passieren könnte, wenn dieses Konzept nicht gelten würde.

Lösung:

- **A**tomicity
- **C**onsistency
- **I**solation
- **D**urability

Wenn eine Datenbank keine Atomarität garantieren kann, kann es zu inkonsistenten Daten kommen (unabhängig von der Konsistenzeigenschaft/Consistency). Hierzu dient eine Überweisung in einer Bank als Beispiel: Wenn eine Transaktion daraus besteht, einen Kontostand zu verringern und einen andern zu erhöhen, entsteht ein inkonsistenter Zustand, wenn nur eine der beiden Operationen tatsächlich gespeichert wird.

Bei einer Datenbank, die nicht konsistent ist, können weitreichende Probleme entstehen. Wenn z.B. garantiert werden muss, dass Kontonummern eindeutig sind, die Datenbank aber inkonsistente Daten zulässt, kann nicht mehr garantiert werden, dass Überweisungen tatsächlich beim richtigen Kontobesitzer ankommen.

Bei dem Beispiel einer Überweisung kann es auch zu Problemen kommen, wenn die Datenbank Transaktionen nicht korrekt voneinander isoliert. Zwei parallele Überweisungen

können gleichzeitig Kontostände ändern, was zu inkorrekten Kontoständen nach der Ausführung beider Transaktionen führen kann.

Wenn eine Datenbank eingesetzt wird, die keine Dauerhaftigkeit garantieren kann, kann nie darauf vertraut werden, dass ein commit eine Transaktion tatsächlich festschreibt. Das ist aber z.B. bei einem Geldautomaten notwendig, der erst Geld ausgeben sollte, sobald die Datenbank garantieren kann, dass die Abhebetransaktion verbucht ist.