TU München, Fakultät für Informatik
Lehrstuhl III: Datenbanksysteme
Prof. Alfons Kemper, Ph.D.

# Übung zur Vorlesung *Grundlagen: Datenbanken* im WS21/22

Michael Jungmair, Josef Schmeißer, Moritz Sichert, Lukas Vogel (gdb@in.tum.de)
https://db.in.tum.de/teaching/ws2122/grundlagen/

**Blatt Nr. 11**

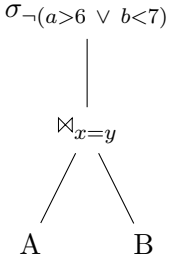
## Hausaufgabe 1

Gegeben seien die Relationen

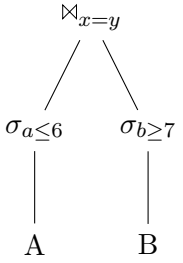
$$A : \{[a, x]\}$$
$$B : \{[b, y]\}$$
$$C : \{[c, z]\}$$

Im Folgenden sehen Sie jeweils zwei Operatorbäume der relationen Algebra. Sind diese äquivalent zueinander? Beweisen oder widerlegen Sie mithilfe der zwölf äquivalenzerhaltenden Transformationsregeln aus der Vorlesung.

a) Ist

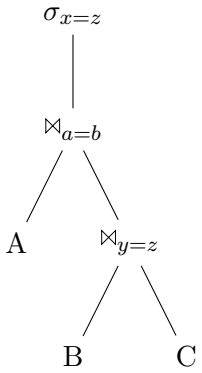



äquivalent zu

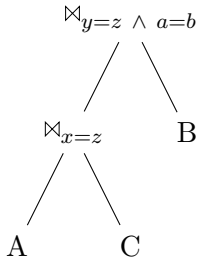



?

b) Ist



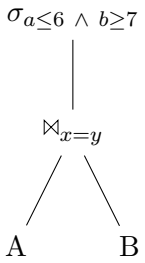

äquivalent zu



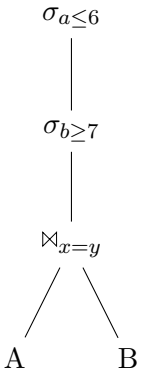

?

**Lösung:**

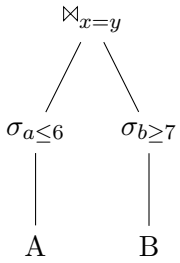
a)




(1)



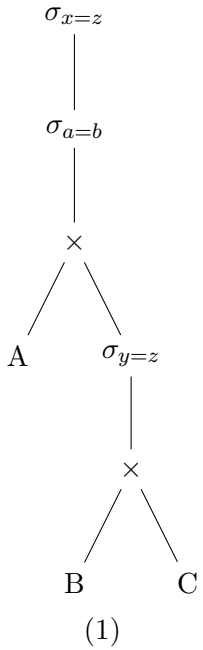

(2)



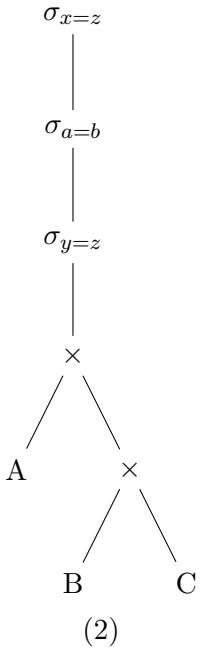

(3)

Das DeMorgansche Gesetz (Regel 11 im Foliensatz) erlaubt uns, die Disjunktion in der Selektion in eine Konjunktion umzuformen (1). Diese Konjunktion können wir nun nach Regel 1 im Foliensatz in zwei getrennte Selektionen aufbrechen (2). Diese Selektionen können wir nun nach unten propagieren (Regel 6 im Foliensatz), da $a$ ein Attribut von $A$ und $b$ ein Attribut von $B$ ist (3). Da wir lediglich Äquivalenzumformungen verwendet haben, ist dieser Operatorbaum äquivalent zum Ausgangsbaum.

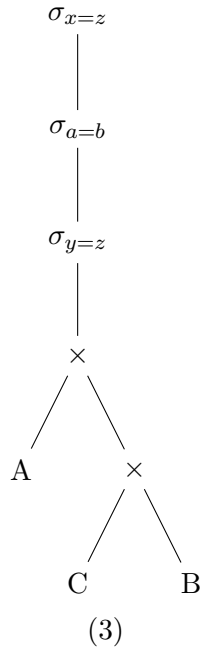
b)



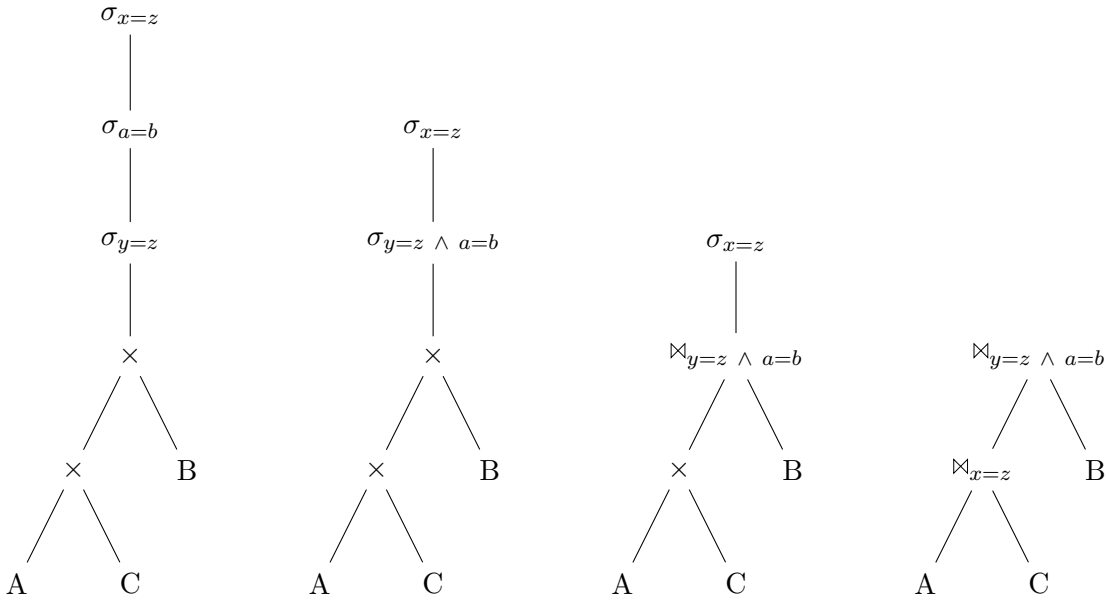

(1)




(2)




(3)




Nach Regel 12 können wir einen Join in ein kartesisches Produktund eine Selektion transformieren (1). Regel 6 erlaubt es uns dann, alle Selektionen nach oben zu propagieren (2). Wir können nun erst die Kommutativität (Regel 5) (3) und dann die Assoziativität (Regel 8) (4) des Kreuzproduktes verwenden. Nach Regel 1 können wir zwei Selektionen zusammenführen (5) und nach Regel 12 erneut in einen Join umwandeln (6) Ein erneutes Anwenden von Regeln 6 und 12 führt uns zum Ziel (7). Auch diese beiden Operatorbäume sind äquivalent.

## Hausaufgabe 2

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.

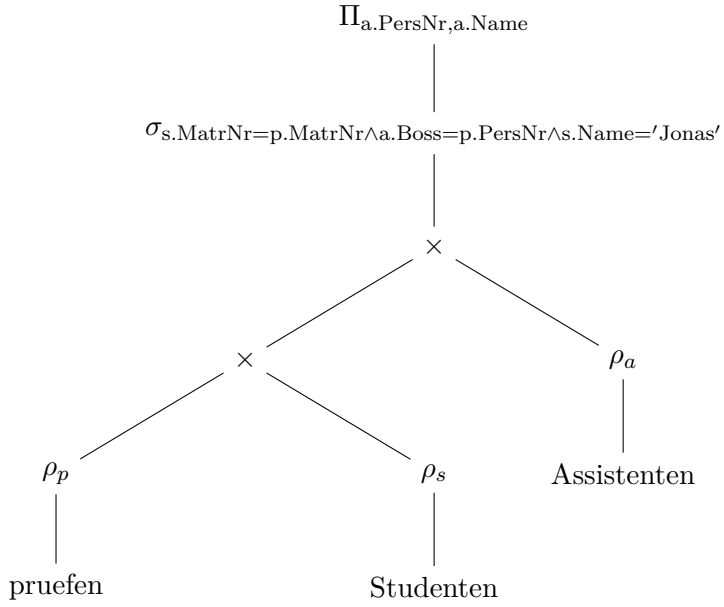
Optimieren Sie Ihren relationalen Algebraausdruck logisch. Gehen Sie dabei von **realistischen** Kardinalitäten für die relevanten Relationen aus.

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

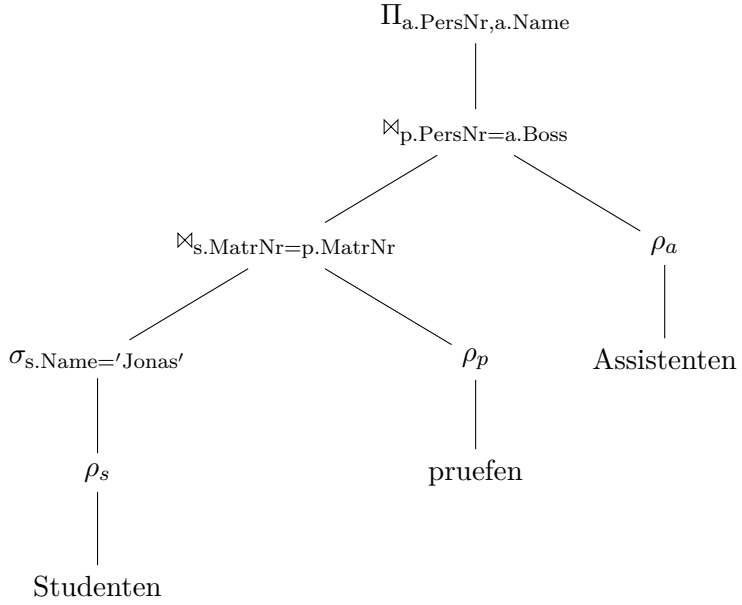
**Lösung:**

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 3

Sie verwenden ein Datenbanksystem, welches die folgenden Joinalgorithmen unterstützt:

- (Blockwise-)Nested-Loop-Join
- Index-Join
- Sort-Merge-Join
- Hash-Join

Geben Sie für jeden der vier Algorithmen ein Beispiel an, bei dem dieses Datenbanksystem diesen Algorithmus verwenden würde. Geben Sie dazu für jedes Beispiel ein Schema und eine SQL-Anfrage an. Begründen Sie, warum der gewählte Joinalgorithmus den anderen vorgezogen wird.

**Lösung:**

- (Blockwise-)Nested-Loop-Join: Schema: Relationen $R$ und $S$. Die Attribute der Relationen sind beliebig.

  Anfrage: `select * from R, S`

  Begründung: Die Anfrage enthält ein Kreuzprodukt. Also ist die Kardinalität der Ergebnismenge zwangsläufig $|R \times S|$. Damit haben alle Joinalgorithmen die selbe (theoretische) Laufzeit. Um sich den unnötigen Overhead des Sortierens oder des Hashings zu sparen, verwendet die Datenbank daher einen Nested-Loop-Join.

- Index-Join: Schema: Relationen: $R : \{[\underline{a}]\}$ und $S : \{[b]\}$.

  Anfrage: `select * from R, S where a = b`

  Begründung: Da $a$ in $R$ ein Schlüssel ist, existiert bereits ein Index. Also kann dieser für einen effizienten Index-Join verwendet werden. Damit wird ebenfalls der Overhead des Sortierens oder Hashings gespart.

- Sort-Merge-Join: Schema: Relationen: $R : \{[a]\}$ und $S : \{[b]\}$.

  Anfrage: `select * from R, S where a = b order by a`

  Begründung: Da das Ergebnis ohnehin nach dem Attribut $a$ sortiert werden muss, kann die Sortierung bereits vor dem Join durchgeführt werden, damit die Sortierung für einen Sort-Merge-Join verwendet werden kann.

- Hash-Join: Schema: Relationen: $R : \{[a]\}$ und $S : \{[b]\}$.

  Anfrage: `select * from R, S where a = b`

  Begründung: Hier wählt das Datenbanksystem den Hash-Join, da seine Komplexität die niedrigste ist.

## Hausaufgabe 4

Gegeben sind die beiden Relationenausprägungen:

| $R$ | |
|---|---|
| | A |
| ... | 0 |
| ... | 5 |
| ... | 7 |
| ... | 8 |
| ... | 8 |
| ... | 10 |
| ⋮ | ⋮ |

| $S$ | |
|---|---|
| B | |
| 5 | ... |
| 6 | ... |
| 7 | ... |
| 8 | ... |
| 8 | ... |
| 11 | ... |
| ⋮ | ⋮ |

Werten Sie den Join $R \bowtie_{R.A=S.B} S$ mithilfe des Nested-Loop- sowie des Sort/Merge-Algorithmus aus. Machen Sie deutlich, in welcher Reihenfolge die Tupel der beiden Relationen verglichen werden und kennzeichnen Sie die Tupel, die in die Ergebnismenge übernommen werden. Vervollständigen Sie hierzu die beiden folgenden Tabellen:

| | | *S.B* | | | | | |
|---|---|---|---|---|---|---|---|
| | | 5 | 6 | 7 | 8 | 8 | 11 |
| *R.A* | 0 | 1 | 2 | 3 | | | |
| | 5 | | | | | | |
| | 7 | | | | | | |
| | 8 | | | | | | |
| | 8 | | | | | | |
| | 10 | | | | | | |

Nested-Loop-Join

| | | *S.B* | | | | | |
|---|---|---|---|---|---|---|---|
| | | 5 | 6 | 7 | 8 | 8 | 11 |
| *R.A* | 0 | 1 | | | | | |
| | 5 | 2✔ | | | | | |
| | 7 | | | | | | |
| | 8 | | | | | | |
| | 8 | | | | | | |
| | 10 | | | | | | |

Sort/Merge-Join

**Lösung:**

| | | *S.B* | | | | | |
|---|---|---|---|---|---|---|---|
| | | 5 | 6 | 7 | 8 | 8 | 11 |
| *R.A* | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| | 5 | 7✔ | 8 | 9 | 10 | 11 | 12 |
| | 7 | 13 | 14 | 15✔ | 16 | 17 | 18 |
| | 8 | 19 | 20 | 21 | 22✔ | 23✔ | 24 |
| | 8 | 25 | 26 | 27 | 28✔ | 29✔ | 30 |
| | 10 | 31 | 32 | 33 | 34 | 35 | 36 |

Nested-Loop-Join

| | | *S.B* | | | | | |
|---|---|---|---|---|---|---|---|
| | | 5 | 6 | 7 | 8 | 8 | 11 |
| *R.A* | 0 | 1 | | | | | |
| | 5 | 2✔ | 3 | | | | |
| | 7 | | 4 | 5✔ | | | |
| | 8 | | | 6 | 7✔ | 10✔ | |
| | 8 | | | | 8✔ | 11✔ | |
| | 10 | | | | 9 | 12 | 13 |

Sort/Merge-Join

Ausführliche Lösung:
`http://www-db.in.tum.de/teaching/ws1415/grundlagen/Loesung11_sort_merge_join.pdf`