Grundlagen Datenbanken: Übung 11

Tanmay Deshpande

Gruppe 20 & 21

ge94vem@mytum.de

QR-Code für die Folien

Wiederholung

Woche 11

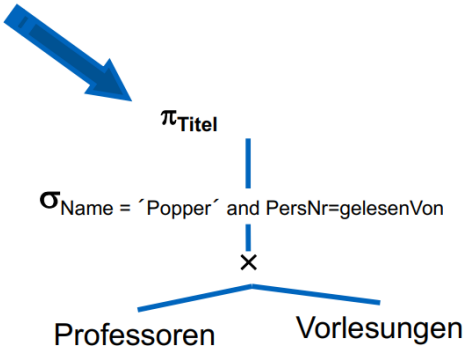
Anfrageoptimierung

- Ziel: Anfrage soll möglichst effizient ausgewertet werden
- 1. **Logische Optimierung:** Relationale Algebra Ausdrücke werden mithilfe von äquivalenzerhaltender Umformungen optimiert
  2. **Physische Optimierung:** Algorithmische Optimierung von der Datenverarbeitung (Bsp: Join-Algorithmen)

Logische Optimierung

- Zuerst, braucht man die kanonische Übersetzung der SQL-Anfrage

```
select Titel
from Professoren, Vorlesungen
where Name = ´Popper´ and
      PersNr = gelesenVon
```

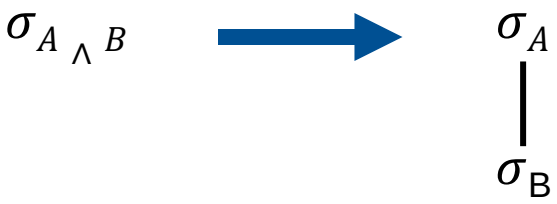


$\pi_{\text{Titel}}(\sigma_{\text{Name = ´Popper´ and PersNr=gelesenVon}}(\text{Professoren} \times \text{Vorlesungen}))$

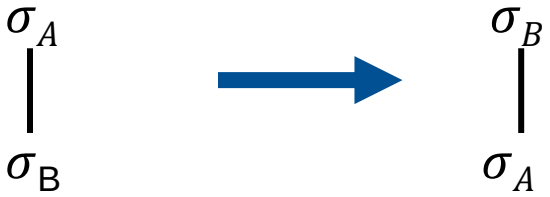
- Danach, äquivalenzerhaltende Transformationsregeln anwenden

Äquivalenzerhaltende Transformationsregeln

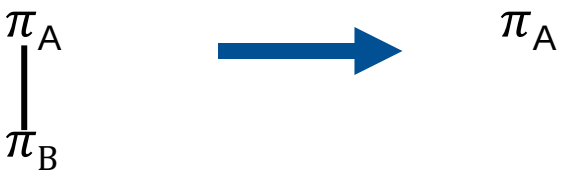
1. Aufspalten von Selektionsprädikaten



2. Kommutative Selektionen



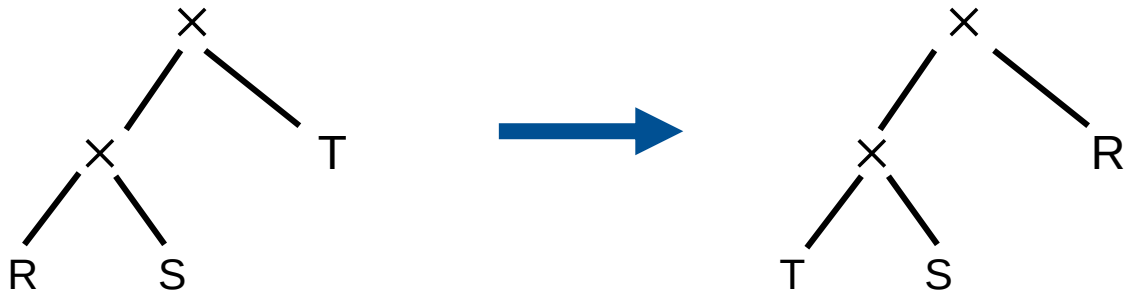
3. π-Kaskaden (falls $A \subseteq B$)



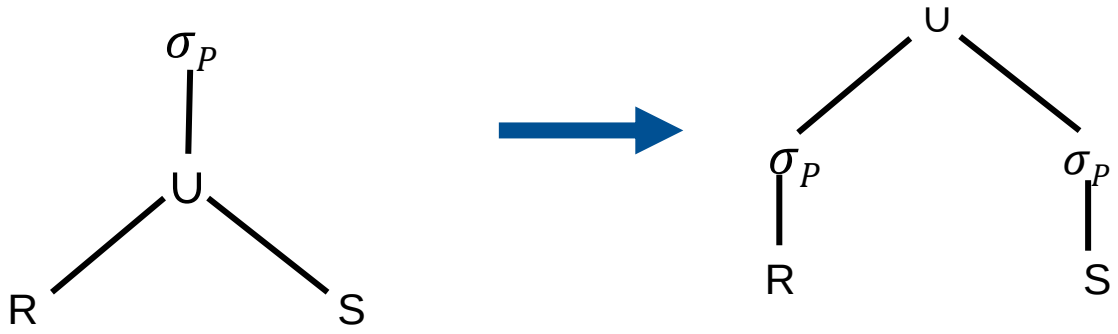
4. σ und π Umtausch (wenn sich die Prädikate auf die gleichen Attribute beziehen)
5. Kommutative $\times$, $\cup$, $\cap$, $\bowtie$
6. σ und $\bowtie$ Umtausch (wenn sich σ nur auf Attribute in *R* bezieht)
7. π und $\bowtie$ Umtausch (wenn sich die Prädikate auf die gleichen Attribute beziehen)

Äquivalenzerhaltende Transformationsregeln

8. Assoziative $\times$, $\cup$, $\cap$, $\bowtie$



9. σ distributiv mit $\cup$, $\cap$, $-$

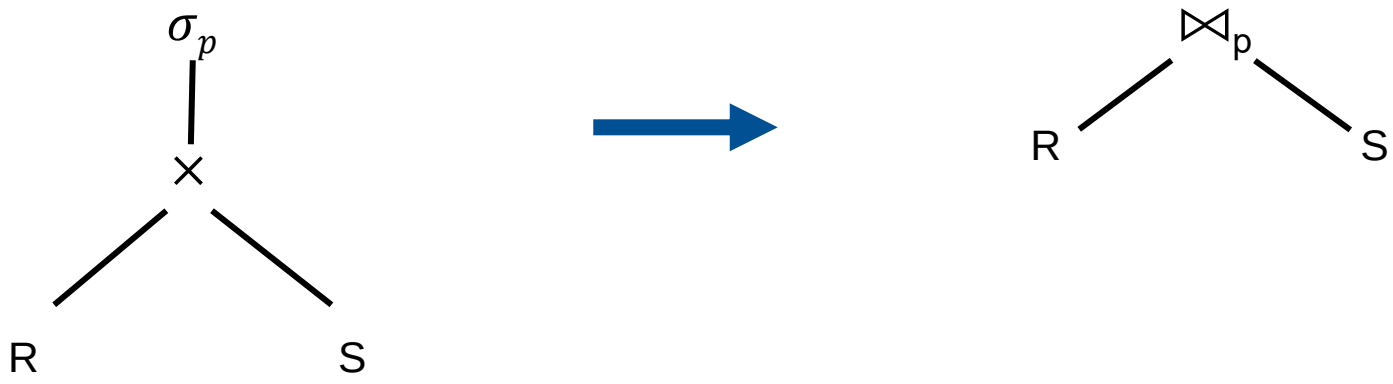


10. π distributiv bezüglich $\cup$

11. Join\Selektionsprädikate mit De-Morgan umformen

Äquivalenzerhaltende Transformationsregeln

12. Selektionen und Kreuzprodukt zu Joins zusammenführen (wenn sie sich auf die gleichen Attribute beziehen)



Logische Optimierung

- Mithilfe der äquivalenzerhaltenden Transformationsregeln können wir die kanonische Übersetzung einer SQL-Anfrage optimieren.
- Ziele:
  - Bearbeitungsreihenfolge der Operationen so zu ändern, dass die kleinstmögliche Zwischenergebnisse entstehen.
  - Zusammenfassen von Operationen
- Vorschrift:
  1. Selektionen aufbrechen
  2. Selektionen nach unten verschieben
  3. Selektionen und Kreuzprodukte zu Joins zusammenfassen
  4. Joinreihenfolge bestimmen
  5. Projektionen Einfügen
  6. Projektionen nach unten verschieben

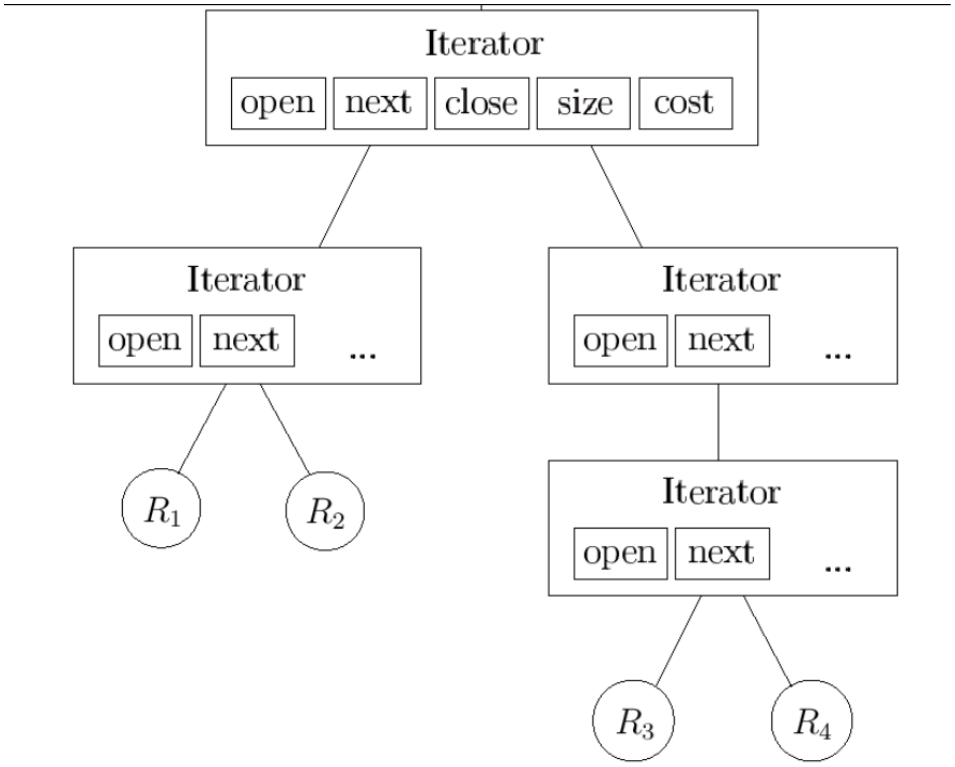
Physische Optimierung

- Wie werden Operationen der relationalen Algebra durchgeführt?
- Wie werden Indizes verwendet (falls vorhanden)?
- Sollen Zwischenergebnisse vollständig berechnet und gespeichert werden?
- Welcher Algorithmus sollte je nach Situation angewendet werden, um die nötigen Daten zu bestimmen?
- Bsp: Join-Algorithmen

- **Iteratoren:**
  * Erlauben uns, verschiedene Implementierungen gegeneinander auszutauschen
  * Zwischenergebnisse können verwendet werden ohne sie komplett zu speichern

Iteratoren

- open: Initialisierung der Eingabe
- next: Berechnet das nächste Tupel des Ergebnisses und gibt es zurück
- close: Schleißt und gibt die Ressource frei
- cost/size: Geschätzte Berechnungskosten bzw. Ergebnisgröße



Nested Loop Join

- Brute-force Algorithmus
- foreach $r \in R$
  foreach $s \in S$
  if $s.B = r.A$
  then $Res := Res\,(r \cdot s)$
- $O(n^2)$ Komplexität

iterator NestedLoop$_p$

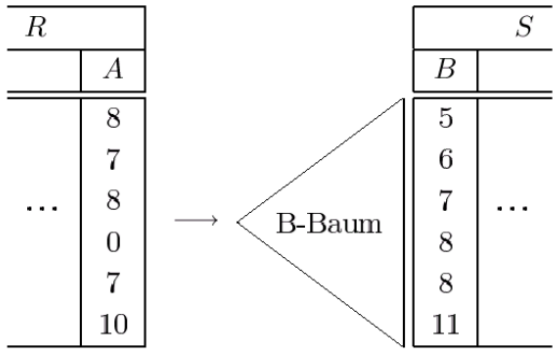
open
- Öffne die linke Eingabe

next
- Rechte Eingabe geschlossen?
  - Öffne sie
- Fordere rechts solange Tupel an, bis Bedingung p erfüllt ist
- Sollte zwischendurch rechte Eingabe erschöpft sein
  - Schließe rechte Eingabe
  - Fordere nächstes Tupel der linken Eingabe an
  - Starte **next** neu
- Gib den Verbund von aktuellem linken und aktuellem rechte Tupel zurück

close
- Schließe beide Eingabequellen

Index Join

- Wie Nested-Loop, aber anstatt für jeden Join-Partner in der linken Relation die ganze rechte Relation durchzugehen, suchen wir nach passenden Werten der rechten Relation mithilfe einer Indexstruktur (Bsp: B-Baum)
- Da diese sortiert sind, erleichtert es unsere Suche im Vergleich zu Brute-Force
- Beispiel:



- Dafür muss so eine Struktur existieren und es kann aufwendig sein sie zu erstellen
- $O(|R| \cdot \log|S|)$ Komplexität

Index Join

- **iterator** IndexJoin$_p$

  open:
  - linke Eingabe öffnen
  - erstes Tupel von der linken Eingabe anfordern

  next:
  - schlage Join-Attributwert im Index nach
  - Match gefunden? Verbinde die Tupel und gibt das Ergebnis zurück
  - Sonst, der nächste Tupel von der linken Eingabe anfordern und next wiederholen
  - Abbruch: linke Eingabe erschöpft

  close: beide Eingaben schleißen

Merge Join

- Voraussetzung: Joinattribute sind sortiert

- Idee: Weil unsere Eingaben sortiert sind, wenn man auf einen großeren Wert als das aktuelle Joinattribut landet, weiß man, dass man alle mögliche Join-Partner erschöpft hat. Beim nächsten Scan muss man die Einträge davor nicht nochmal betrachten

- Setup:
  - Die beiden Eingaben haben jeweils einen Pointer auf ihre Einträge
  - *akt*: Die aktive Eingabe mit dem Pointer auf den kleineren Wert
  -“andere“ Eingabe: Die inaktive Eingabe
  - Markierung: Wir merken uns die Stelle in der „anderen“ Eingabe, wo wir unser Scan für den nächsten Eintrag beginnen müssen

Merge Join

- Voraussetzung: Die Join-Attribute sind sortiert
- **iterator** MergeJoin_p

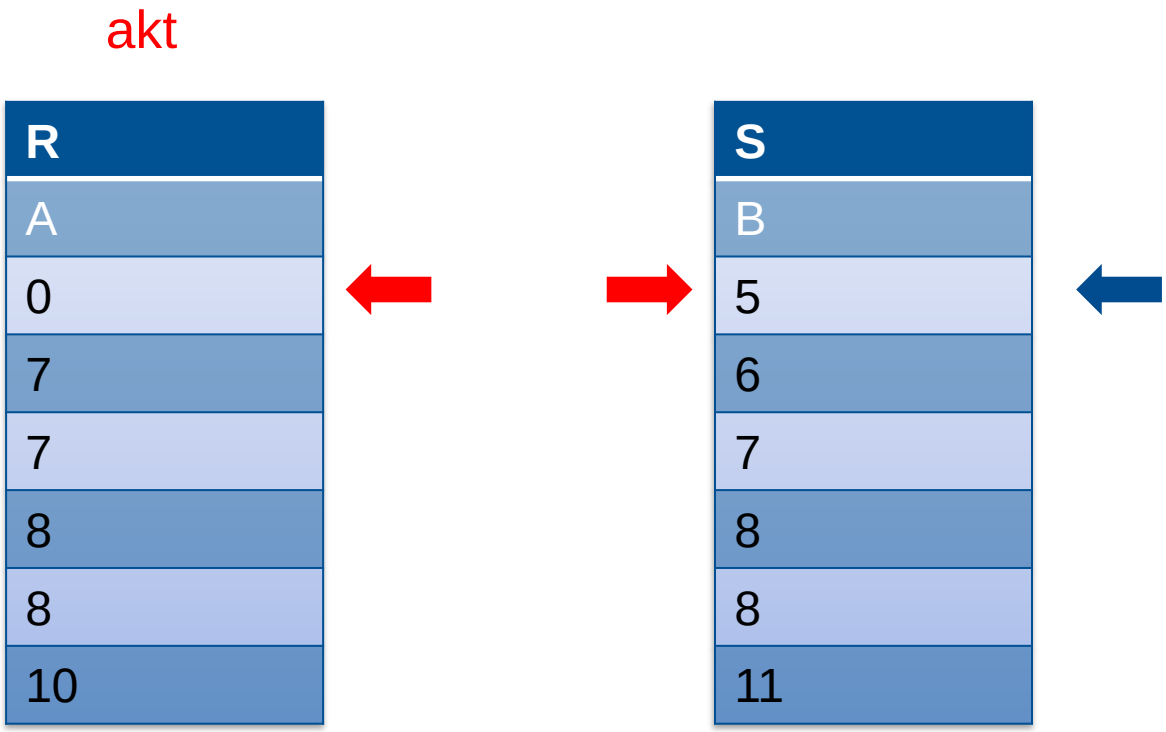
open:

- Öffne beide Eingaben
- Setze *akt* auf linke Eingabe
- Markiere rechte Eingabe

next:

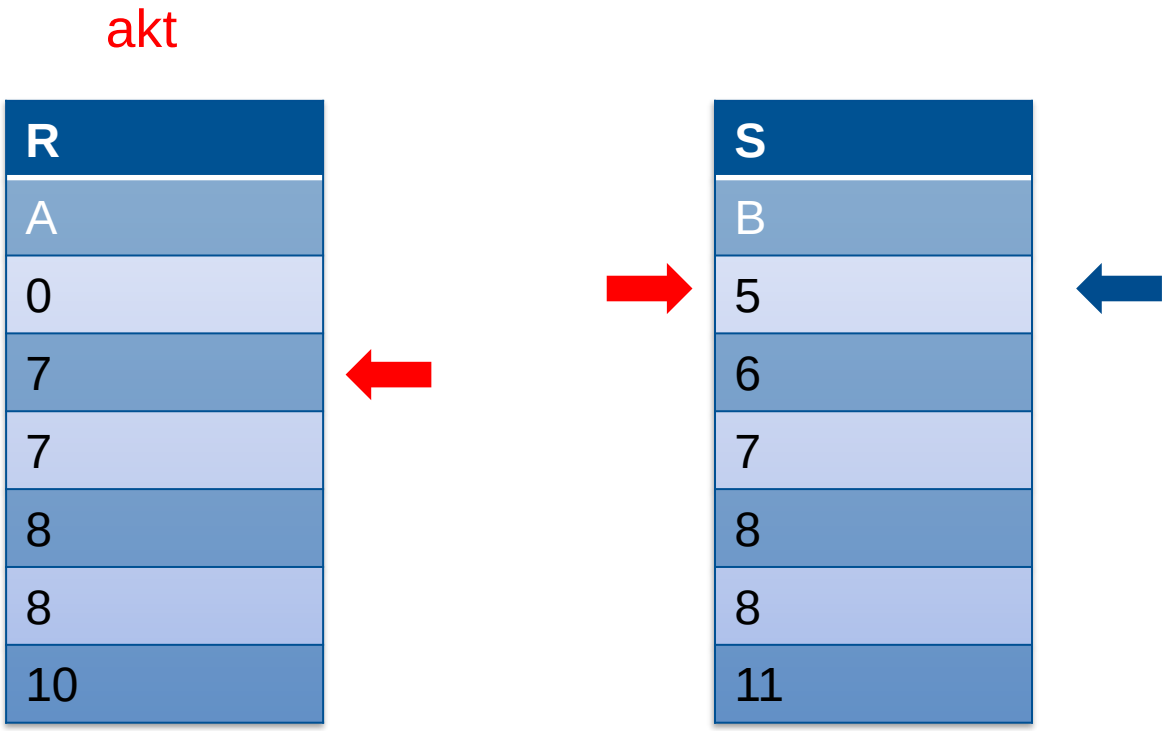
- Solange Bedingung nicht erfüllt
  - Setze *akt* auf Eingabe mit dem kleinsten anliegenden Wert im Joinattribut
  - Bewege *akt* vor
  - Markiere andere Eingabe
- Verbinde linkes und rechtes Tupel
- Bewege andere Eingabe vor
- Ist Bedingung nicht mehr erfüllt oder eine andere Eingabe erschöpft?
  - Bewege *akt* vor
  - Hat sich der Wert des Joinattributs in *akt* verändert?
    - Ja: markiere die andere Eingabe
    - Nein: setze andere Eingabe auf Markierung zurück

Merge Join



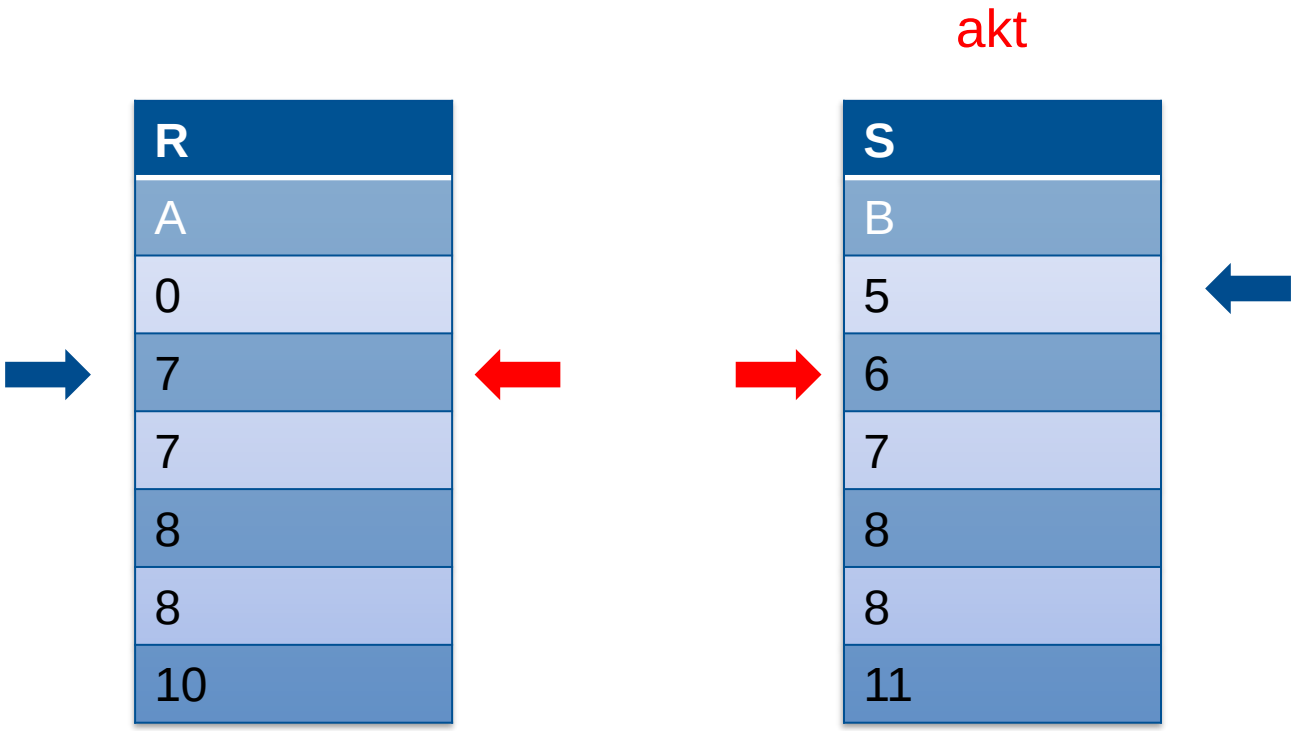
- Bedingung nicht erfüllt
  - *akt* auf kleineren Wert setzen
  - Bewege *akt* vor
  - Markiere andere Eingabe

Merge Join



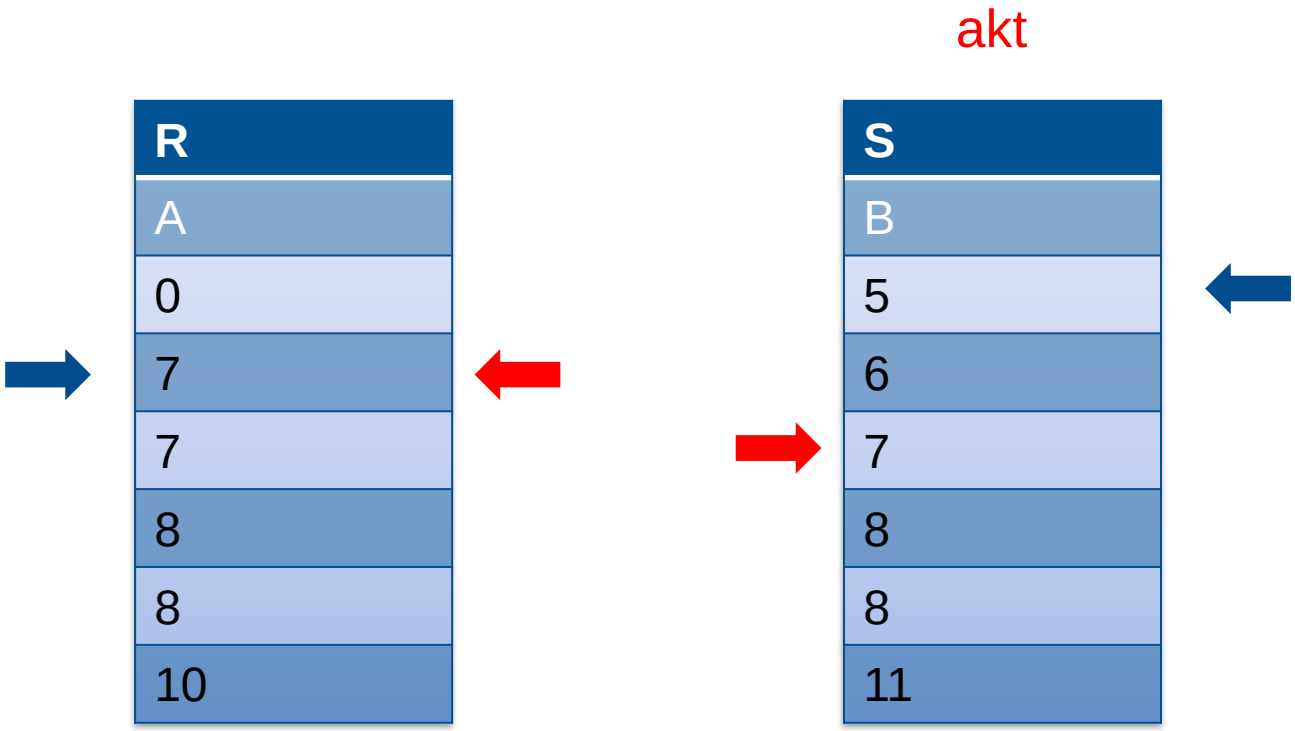
- Bedingung nicht erfüllt
  - *akt* auf kleineren Wert setzen
  - Bewege *akt* vor
  - Markiere andere Eingabe

Merge Join



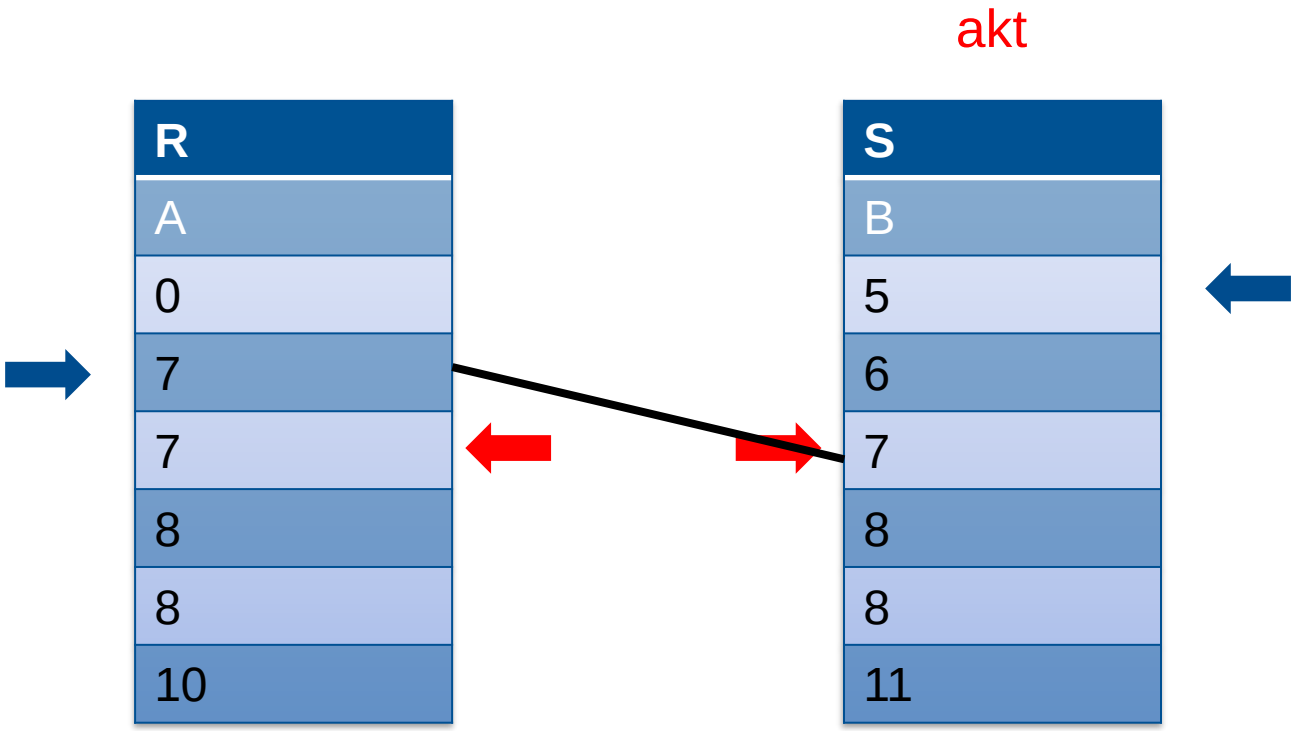
- Bedingung nicht erfüllt
  - *akt* auf kleineren Wert setzen
  - Bewege *akt* vor
  - Markiere andere Eingabe

Merge Join



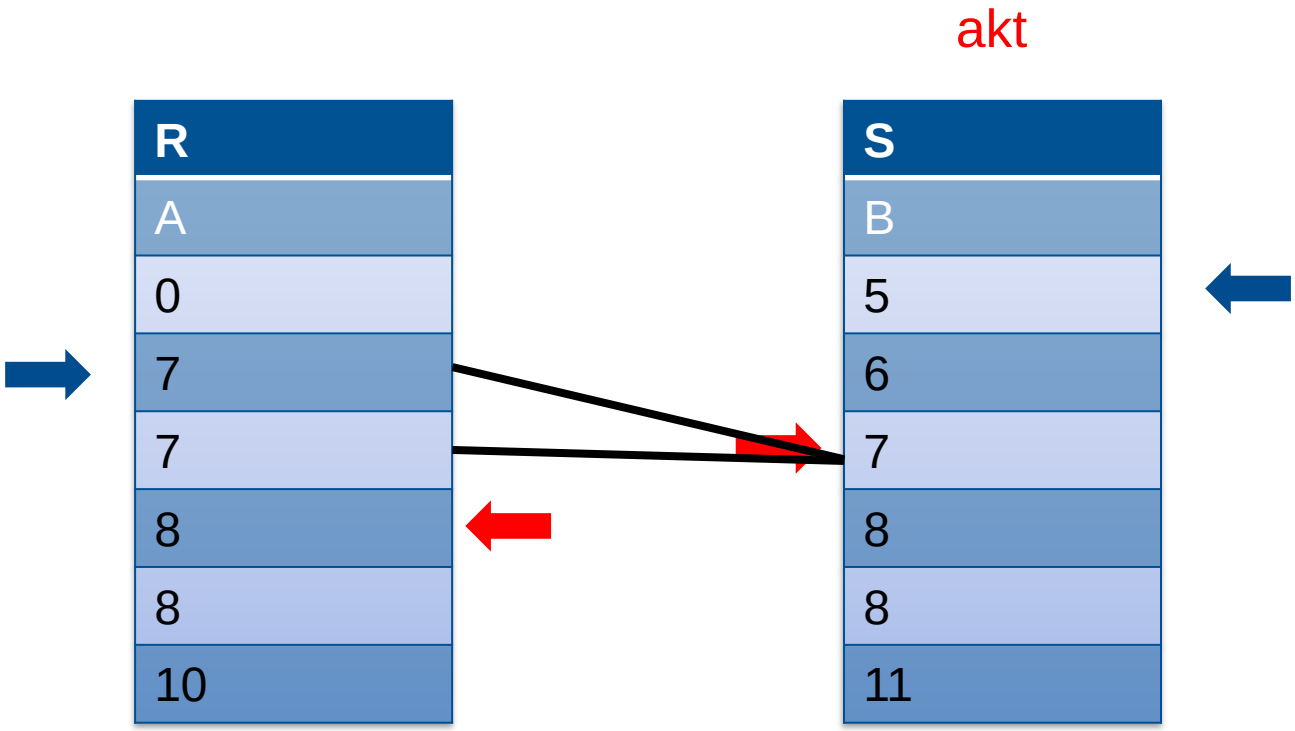
- Bedingung erfüllt
  - Verbinde die Tupel
  - Bewege andere Eingabe vor

Merge Join



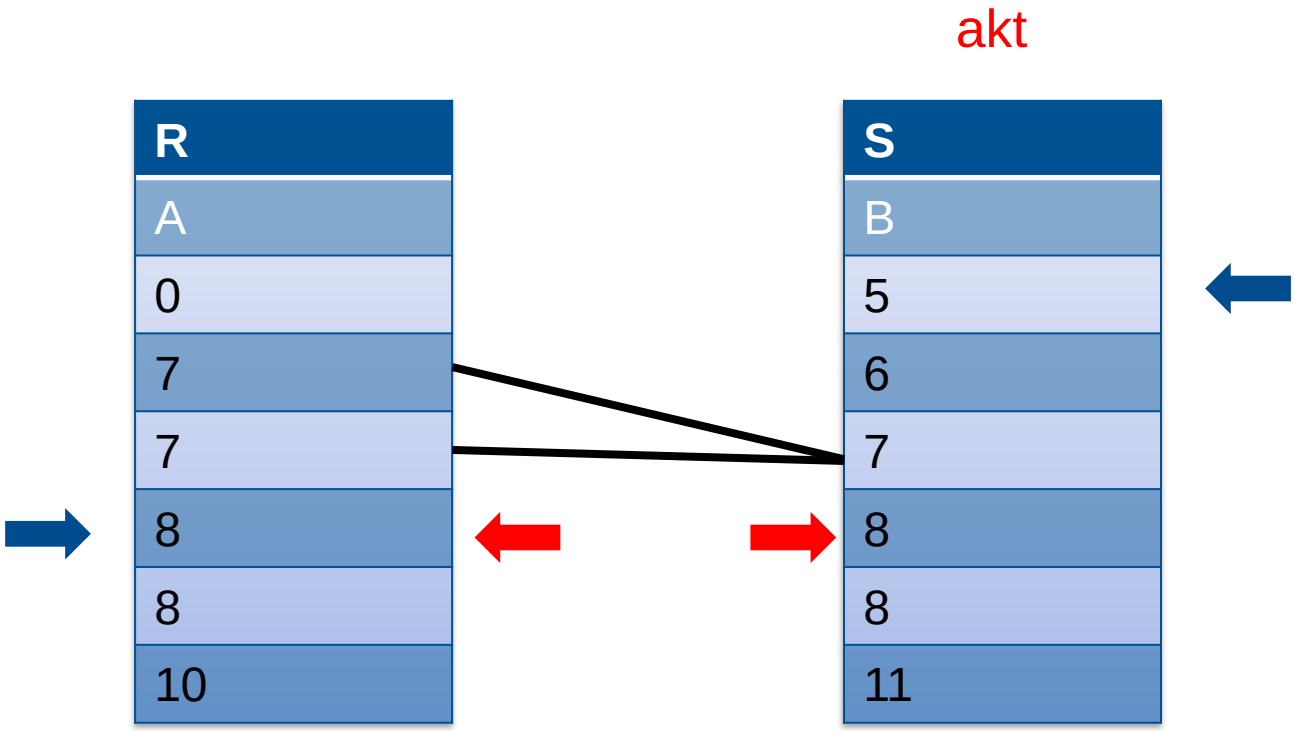
- Bedingung erfüllt
  - Verbinde die Tupel
  - Bewege andere Eingabe vor

Merge Join



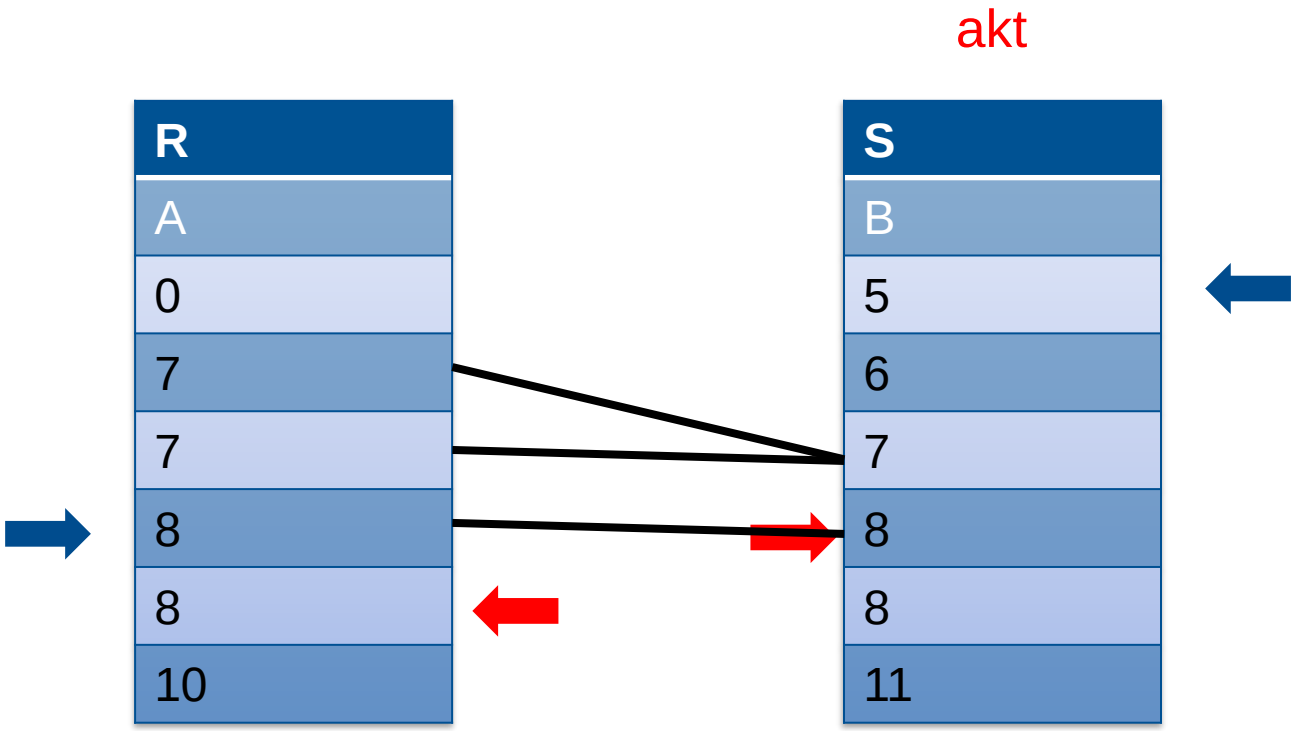
- Bedingung nicht mehr erfüllt
  - Bewege *akt vor*
  - Join-Wert von *akt* hat sich verändert
    - Markiere andere Eingabe

Merge Join



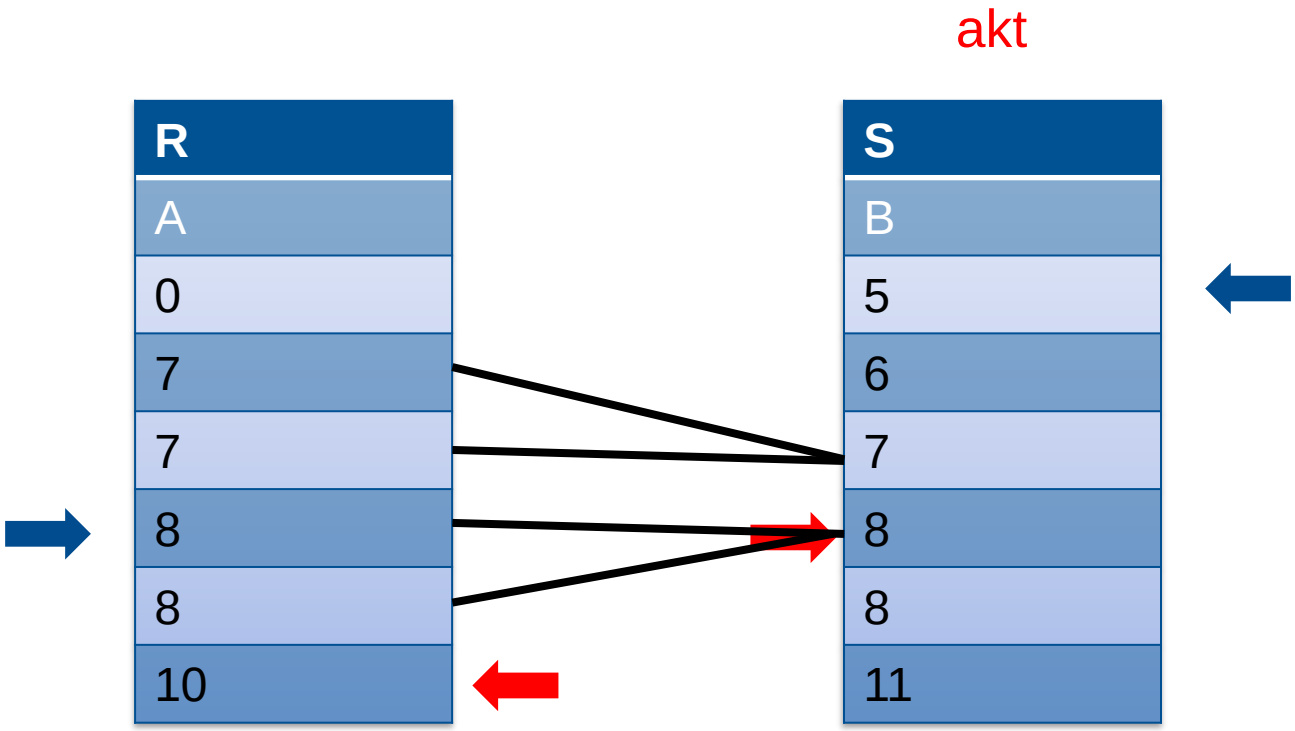
- Bedingung erfüllt
  - Verbinde die Tupel
  - Bewege andere Eingabe vor

Merge Join



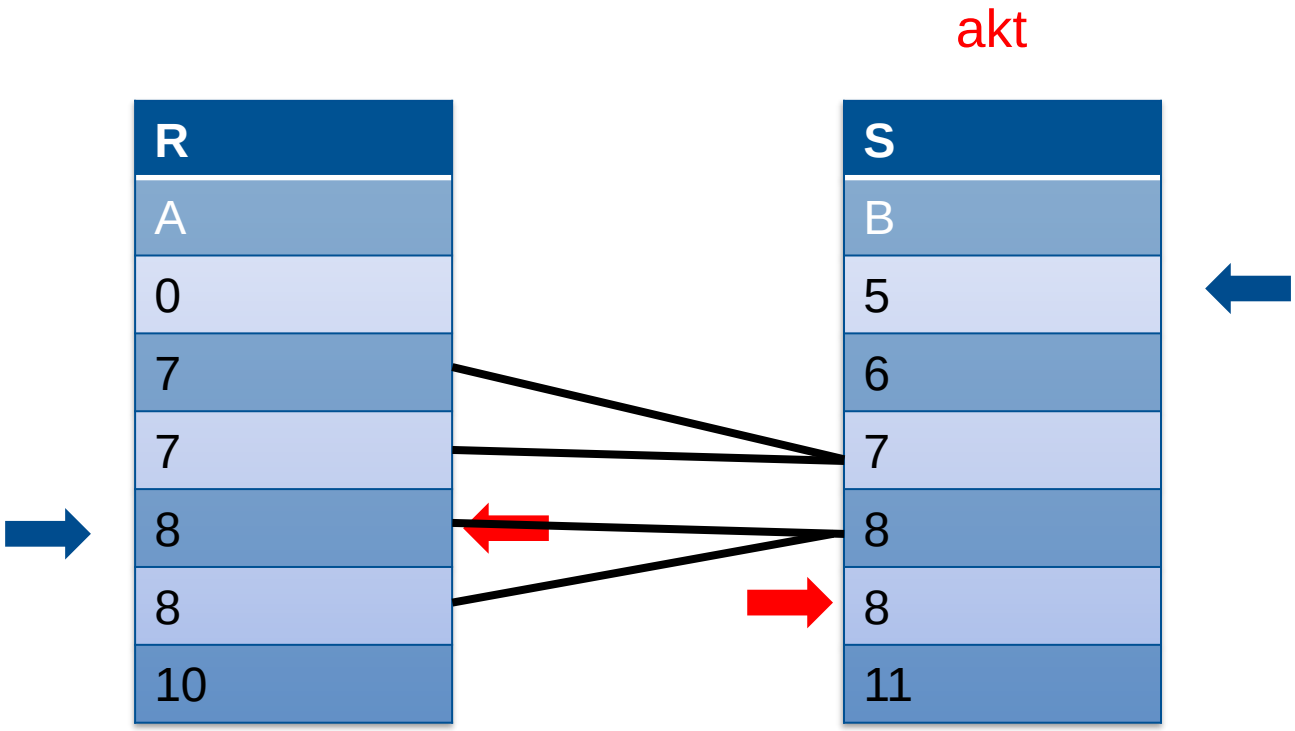
- Bedingung erfüllt
  - Verbinde die Tupel
  - Bewege andere Eingabe vor

Merge Join



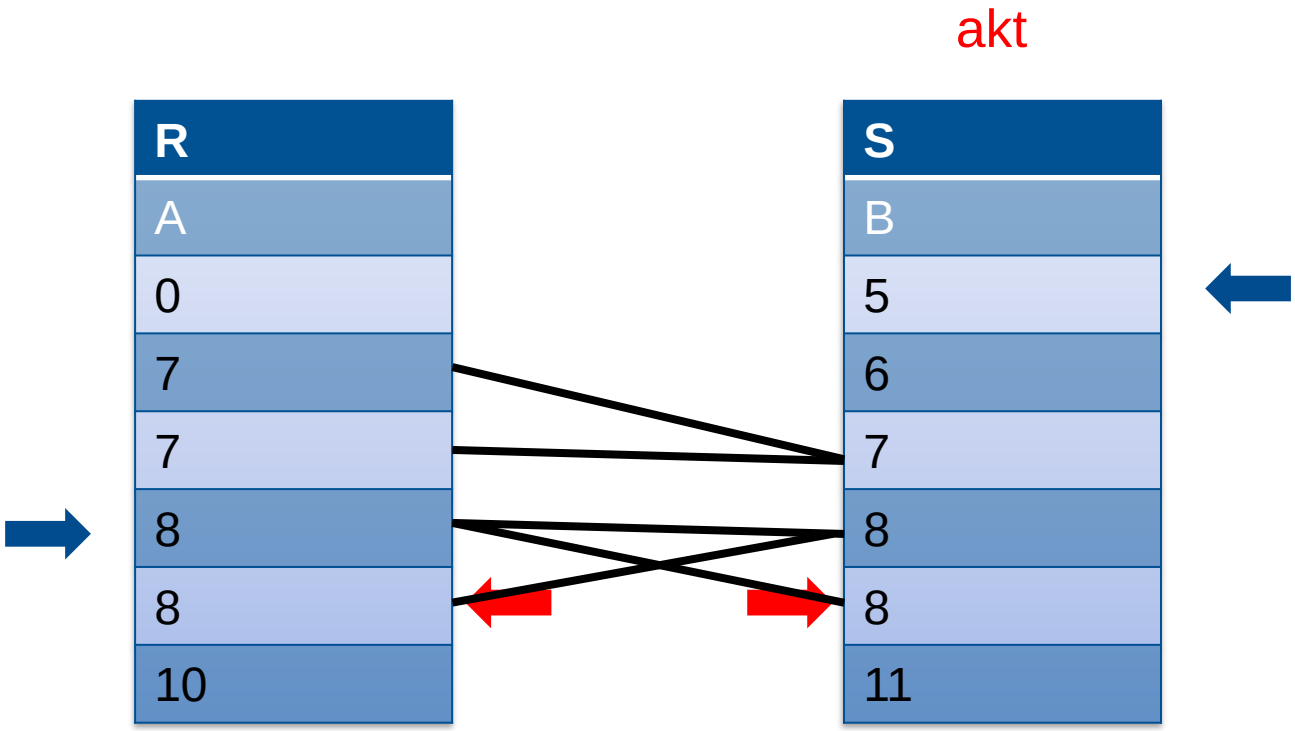
- Bedingung nicht mehr erfüllt
  - Bewege *akt vor*
  - Join-Wert von *akt* hat sich nicht verändert
    - Springe zurück zu Markierung in anderer Eingabe

Merge Join



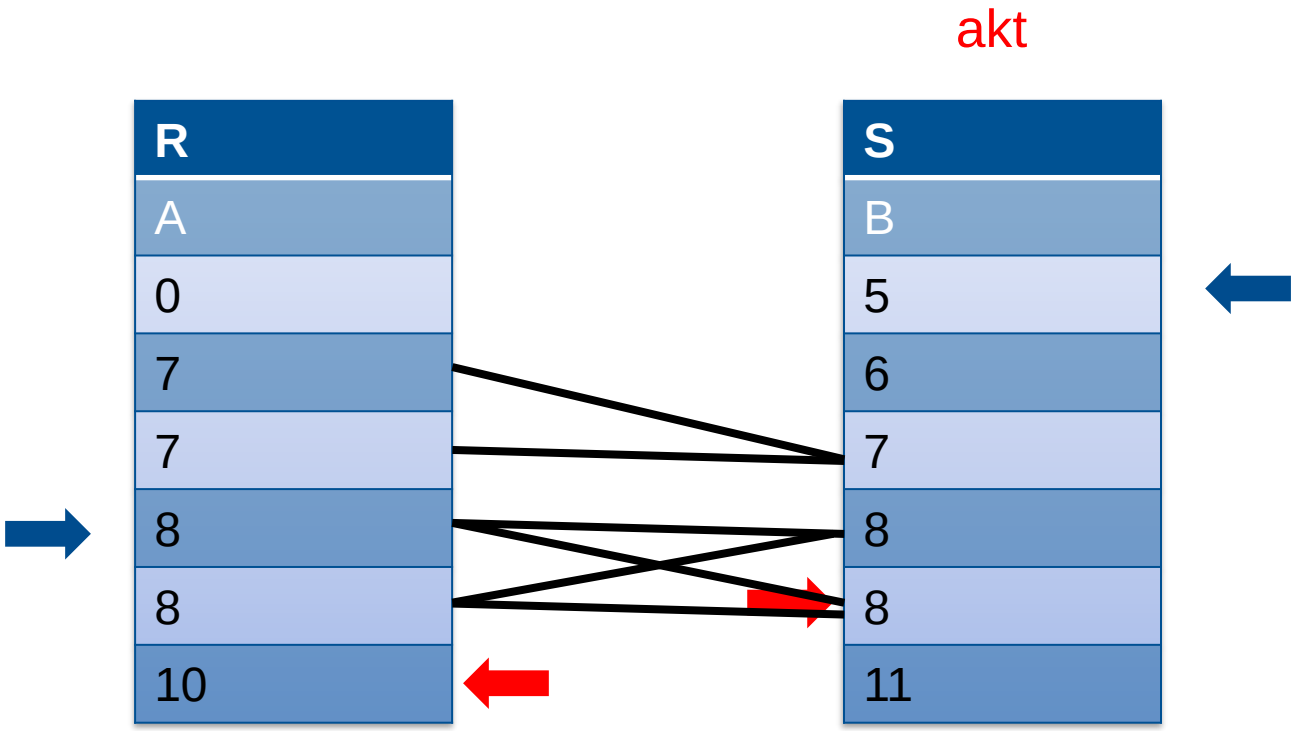
- Bedingung erfüllt
  - Verbinde die Tupel
  - Bewege andere Eingabe vor

Merge Join



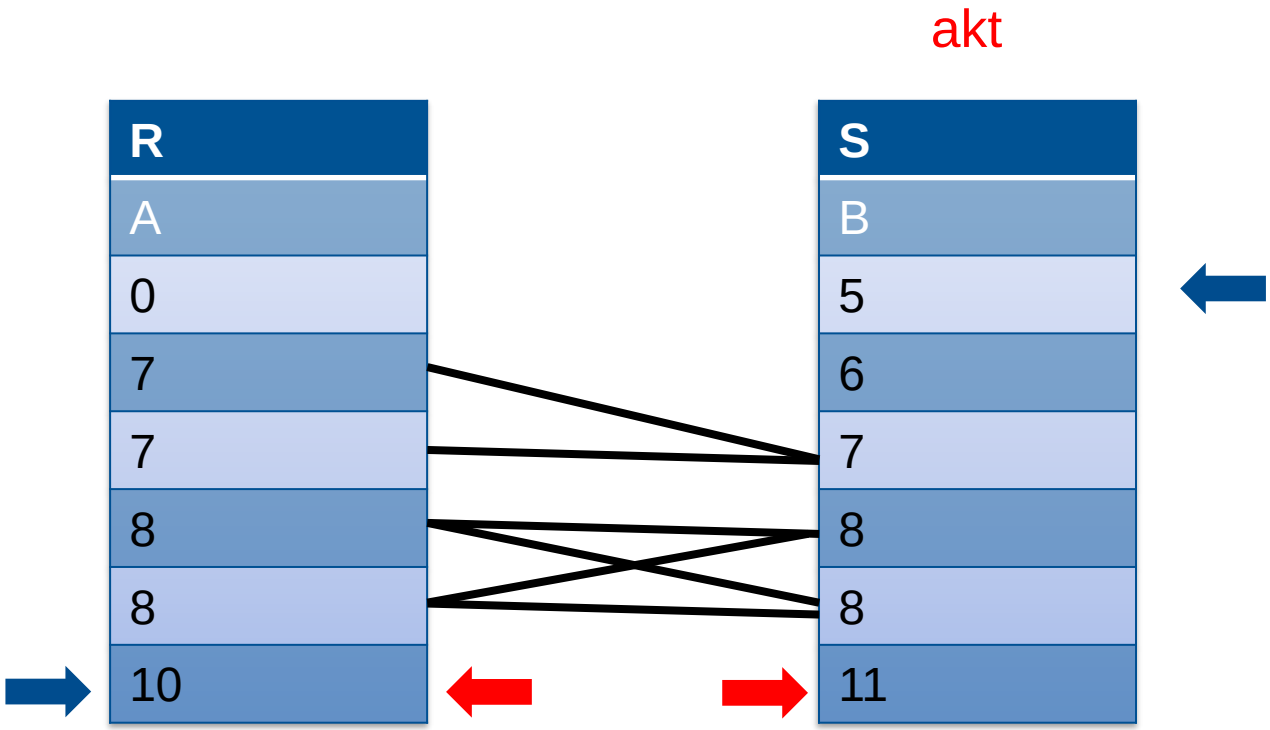
- Bedingung erfüllt
  - Verbinde die Tupel
  - Bewege andere Eingabe vor

Merge Join



- Bedingung nicht mehr erfüllt
  - Bewege *akt vor*
  - Join-Wert von *akt* hat sich verändert
    - Markiere die andere Eingabe

Merge Join



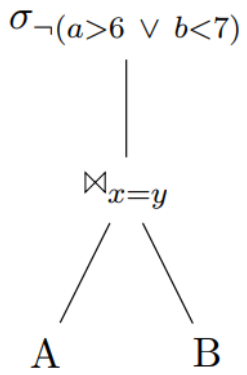
Hash Join

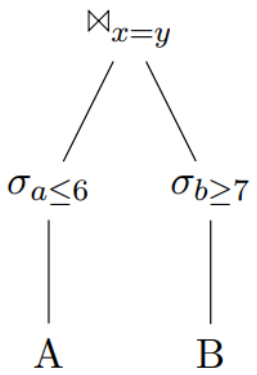
- Die gleiche Hashfunktion wird auf die Join-Attributwerte der beiden Relationen angewendet
- Die Hashfunktion verteilt diese Werte in verschiedenen Buckets. Äquivalente Werte der beiden Relationen landen im selben Bucket. Somit können Join-Partner gefunden werden
- Eignet sich für Equi-Joins
- $O(|R| + |S|)$ Komplexität

Aufgaben

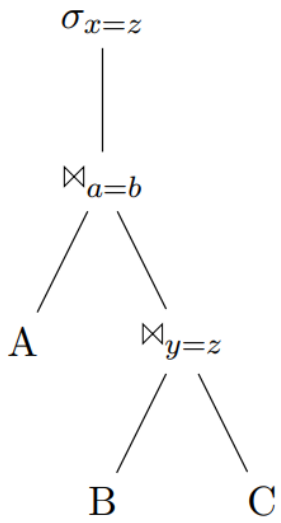
Woche 11

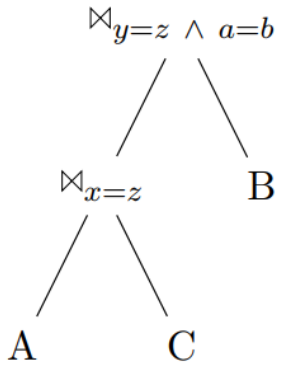
Aufgabe 01

- a) Ist 

äquivalent zu 

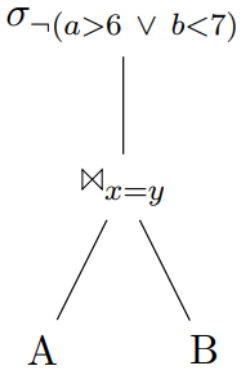
?

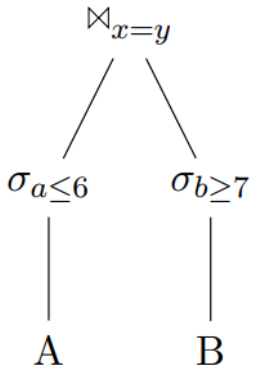
- b) Ist 

äquivalent zu 

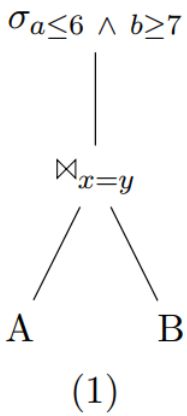
?

Lösungsvorschlag 01 a

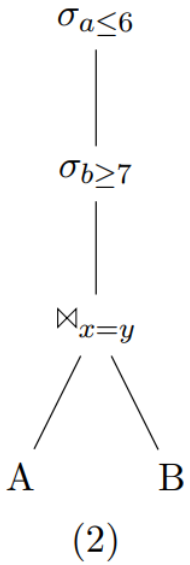
- a) Ist 

äquivalent zu 

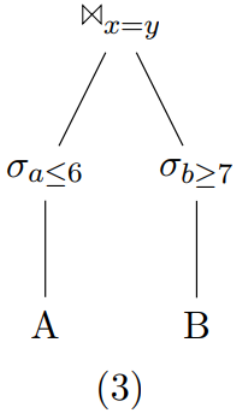
?

- 

(1)

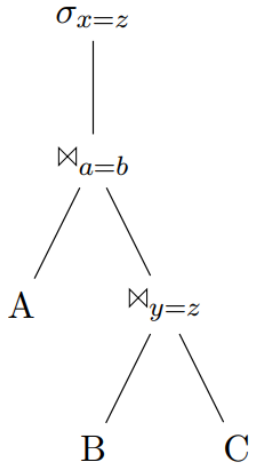


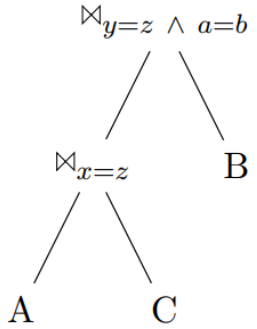
(2)

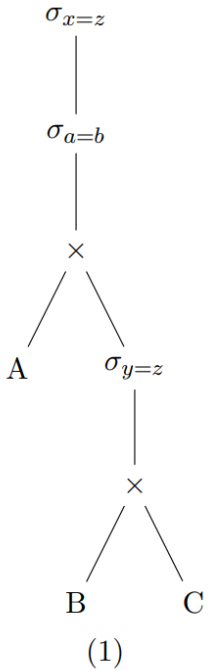


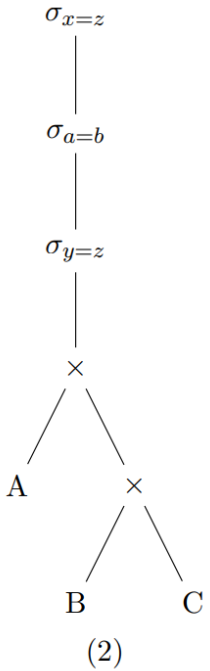
(3)

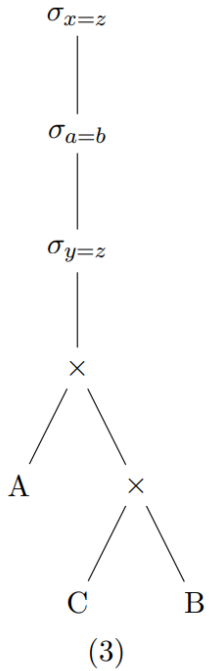
Lösungsvorschlag 01 b

- b) Ist 

äquivalent zu 

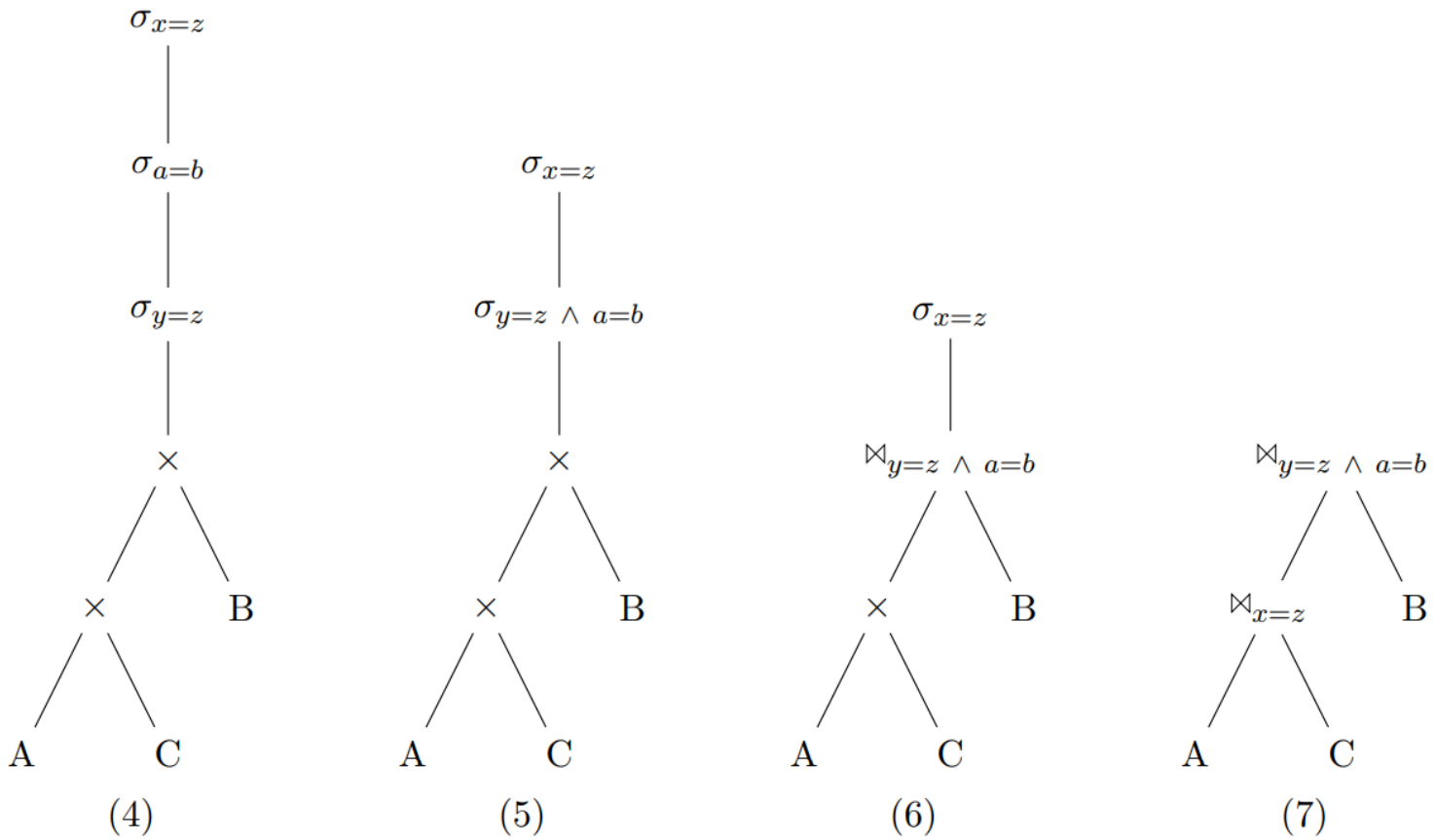
?
- 

(1)


(2)


(3)

Lösungsvorschlag 01 b (contd…)



Aufgabe 02

- a) Was ist ein Equi-Join?
  b) Bei welchen Join-Prädikaten $(<, =, >)$ kann man sinnvoll einen Hashjoin einsetzen?
  c) Gegeben die Relation Profs $= \{\underline{\text{PersNr}}, \text{Name}\}$ und Raeume $= \{\underline{\text{PersNr}}, \text{RaumNr}\}$.
    1) Skizzieren Sie eine geschickte Möglichkeit, den Equi-Join Profs $\bowtie$ Raeume durchzuführen.
    2) In welchem Fall wäre selbst ein Ausdruck wie

$$\text{Profs} \bowtie_{\text{Profs.Persnr} \,<\, \text{Raeume.PersNr}} \text{Raeume}$$

    effizient auswertbar?
  d) Der Student Maier hat einen Algorithmus gefunden, der den Ausdruck $A \times B$ in einer Laufzeit von $O(|A|)$ materialisiert. Was sagen Sie Herrn Maier?

Lösungsvorschlag 02a)-c)

- a) Ein Equi-Join hat eine Äquivalenz als Joinbedingung, etwa die Gleichheit zweier Attribute.

- b) Ein Hash Join bietet sich nur für Equi-Joins an, da lediglich ein Join-Partner mit gleichem Attributwert effizient auffindbar ist. Das Finden eines Partners, dessen Attributwert beispielsweise kleiner sein soll kann mittels Hashing i.A. nicht effizient bearbeitet werden.

- c)

  1) Offenbar ist das Joinattribut gerade der Primärschlüssel, womit von der Existenz eines Indexes ausgegangen werden kann. Somit bietet sich ein Index-basierter Join an, etwa dadurch, dass die eine Relation Element für Element abgearbeitet wird, während Joinpartner aus der anderen Relation mittels des Indexes gefunden werden.

  2) Falls der Index sortiert ist, dies wäre etwa bei einem B-Baum der Fall. Dadurch liegen Joinpartner zumindest nacheinander im Index, anders als bei einer Implementierung des Indexes mittels Hash.

Lösungsvorschlag 02d

- Dies ist mit Sicherheit nicht der Fall, da ein Algorithmus keine bessere Komplexitätsklasse haben kann als sein Ergebnis wächst.
- Mit anderen Worten, $A \times B$ hat eine Ergebnisgröße von $|A| \cdot |B|$ und dieses Ergebnis kann sicher nicht schneller als in $O(|A| \cdot |B|)$ materialisiert werden.

Aufgabe 03

- 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‘;
  ```

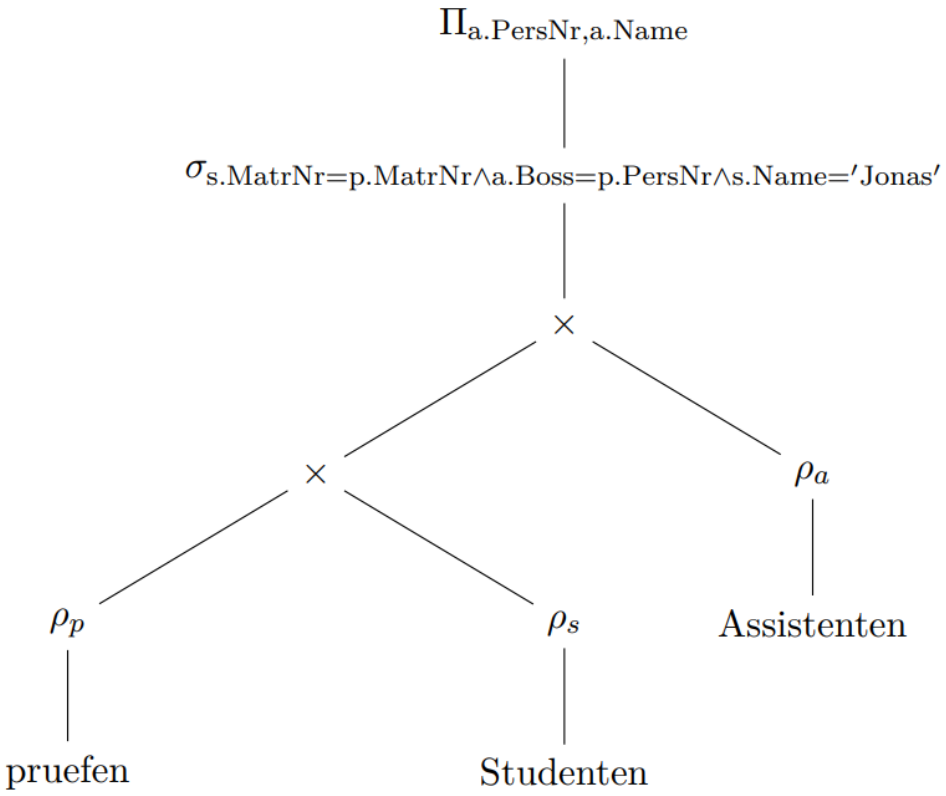
- Skizziere die kanonische Übersetzung in relationaler Algebra als Operatorbaum

- Führe die logische Optimierung aus nach dem Vorschrift aus der Vorlesung (gehe von realistischen Kardinalitäten aus)

- Optimierungstechniken:
  - Aufbrechen von Selektionen
  - Verschieben von Selektionen nach “unten” im Plan
  - Zusammenfassen von Selektionen und Kreuzprodukten zu Joins
  - Bestimmung der Joinreihenfolge

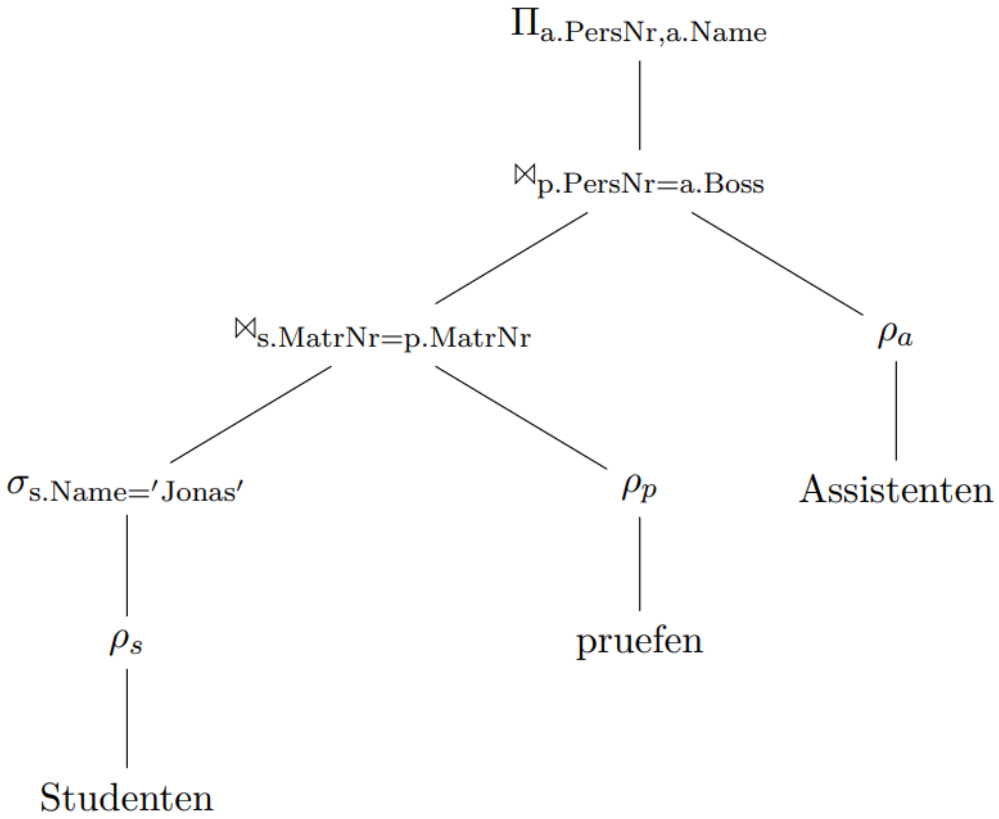
Lösungsvorschlag 03

- Kanonische Übersetzung:



Lösungsvorschlag 03 (contd…)

- Logische Optimierung:



Aufgabe 04

- Gegeben seien die folgenden Relationenausprägungen:

R	
	A
...	0
...	5
...	7
...	8
...	8
...	10
⋮	⋮

S	
B	
5	...
6	...
7	...
8	...
8	...
11	...
⋮	⋮

-

		$S.B$					
		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

Wende Nested Loop und Merge Join auf die Relationen an und vervollständige die Tabellen mit der Bearbeitungsreihenfolge der Einträge

Lösungsvorschlag 04: Nested Loop

		$S.B$					
		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

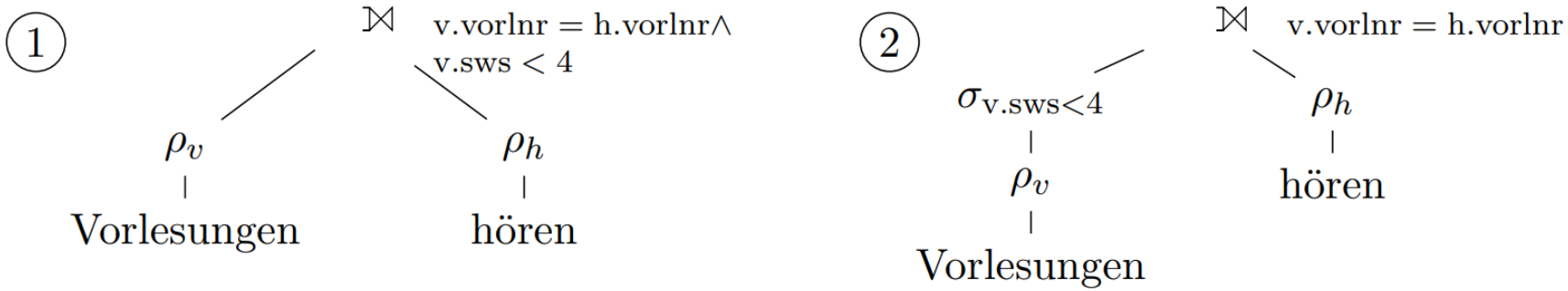
Lösungsvorschlag 04: Merge Join

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

		$S.B$					
		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

Aufgabe 05



Sind die beiden Algebraausdrücke äquivalent? Begründen Sie!

Lösungsvorschlag 05

- Die Ausdrücke sind nicht äquivalent
- Die Selektion in 2. filtriert Vorlesungen heraus, die zu Ergebnis von 1. gehören
- Bsp: Ethik gehört zu Ergebnis von 1. aber nicht von 2.