

## Aufgabe 1

In einer Anfrage werden die Relationen  $R$ ,  $S$  und  $T$  verarbeitet. Die Relationen besitzen die folgenden Schemata.

- $R$  (RID, A)
- $S$  (SID, B)
- $T$  (TID, SID, C)

```
select distinct r.a
from R r, S s, T t
where r.rid = s.sid and s.sid = t.sid and s.b = 3 and t.c = 4;
```

## Aufgabe 1 a)

Überführen Sie die SQL-Anfrage in einen kanonischen Operatorbaum.

## Lösung

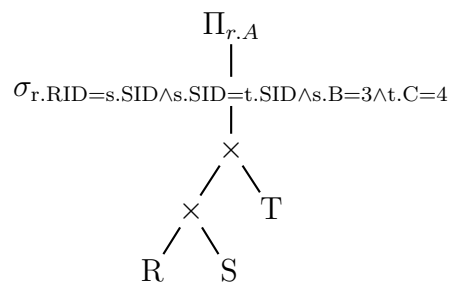


Abbildung 1: Kanonischer Operatorbaum

## Aufgabe 1 b)

Führen Sie eine (logische) Optimierung des in Aufgabenteil a) erstellten, kanonischen

Operatorbaums durch. Verwenden Sie hierfür die in der Vorlesung vorgestellten Heuristiken. Aufgrund fehlender Informationen soll eine Bestimmung der Joinreihenfolge nicht durchgeführt werden.

### Lösung

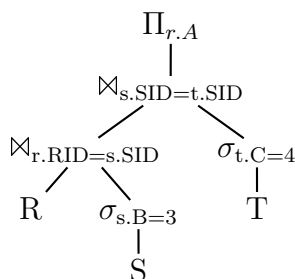
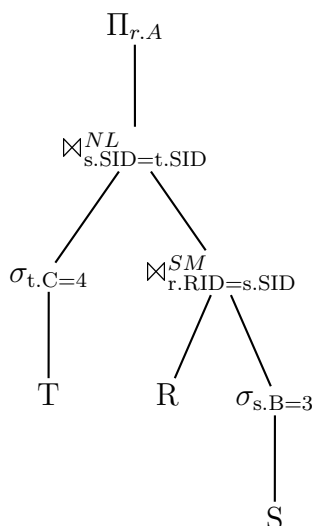


Abbildung 2: Optimierter Operatorbaum

### Aufgabe 1 c)

Zu obiger Anfrage sei der untenstehende Operatorbaum gegeben. Die Superskripte der Join Operatoren geben die jeweils verwendete Joinimplementierung (SM = Sort-Merge Join und NL = Nested-Loop Join) an. Die beiden Eingaben des Sort-Merge Joins sind auf dem jeweiligen Joinattribut sortiert. Weiterhin seien die Kardinalitäten und Selektivitäten einer Datenbank aus der folgenden Tabelle gegeben.



Kardinalitäten		Selektivitäten	
R	1.000	s.B = 3	$\frac{1}{5}$
S	500	r.RID = s.SID	$\frac{1}{100}$
T	200	s.SID = t.SID	$\frac{1}{200}$
		t.C = 4	$\frac{1}{2}$

Bestimmen Sie, für die Ausführung des Operatorsbaum auf der Datenbank, die Anzahl der Tupel auf die insgesamt in den Basisrelationen zugegriffen wird. Beachten Sie, dass mehrfache Zugriffe auf ein Tupel mehrfach gezählt werden.

## Lösung

- Beim Sort-Merge Join wird, da das Join Attribut die jeweiligen Schlüssel der Relation vergleicht, jede Eingabe einmal gelesen ( $1.000 + 500 = 1.500$ ).
- Beim Nested Loop Join wird die linke Eingabe einmal gelesen (200 Tupel) und für  $\frac{1}{2}$  der Tupel wird die rechte Eingabe jeweils einmal gelesen, d.h.  $200 + 100 \times 1.500 = 150.200$ .

---

## Aufgabe 2

---

In einer Anfrage werden die Relationen  $R$ ,  $S$  und  $T$  verarbeitet. Die Relationen besitzen die folgenden Schemata.

- $R$  (A, B, C)
- $S$  (A, D, E, F)
- $T$  (B, G, H)

### Aufgabe 2 a)

Betrachten Sie den (optimierten) Operatorbaum der Anfrage in Abbildung 3. Geben Sie zu diesem Operatorbaum den ursprünglichen, kanonischen Operatorbaum an.

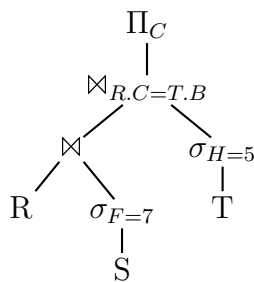


Abbildung 3: Optimierter Operatorbaum

## Lösung

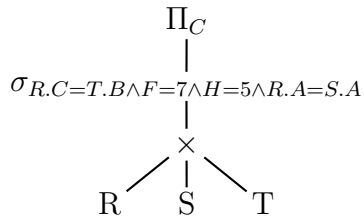


Abbildung 4: Kanonischer Operatorbaum

Aufgabe 2 b)

Betrachten Sie den (optimierten) Operatorbaum der Anfrage in Abbildung 5. Geben Sie zu diesem Operatorbaum den ursprünglichen, kanonischen Operatorbaum an.

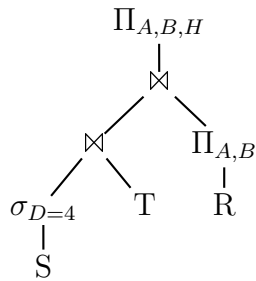


Abbildung 5: Optimierter Operatorbaum

Lösung

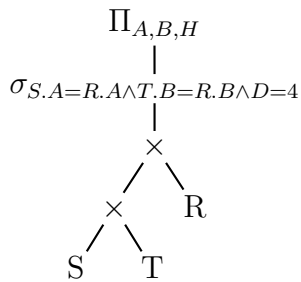


Abbildung 6: Kanonischer Operatorbaum

---

### Aufgabe 3

---

Der Optimierer eines Datenbanksystems ist noch nicht ganz ausgereift und produziert für eine Anfrage folgenden Operatorbaum:

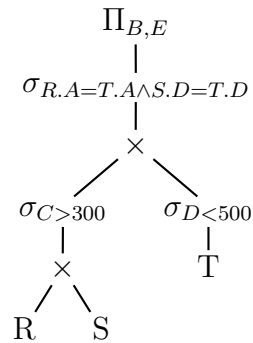


Abbildung 7: Voroptimierter Operatorbaum

Die Relationen besitzen folgendes Schema:

- $R(\underline{A}, B, C)$
- $S(\underline{D}, E, F)$
- $T(\underline{A}, D)$

Weiterhin kennt das Datenbanksystem folgende Statistiken über die Relationen.

Kardinalitäten		Selektivitäten	
R	100	$\sigma_{C>300}$	$\frac{1}{10}$
S	10.000	$\sigma_{D<500}$	$\frac{1}{10}$
T	100.000	$R \bowtie T$	$\frac{1}{1000}$
		$S \bowtie T$	$\frac{1}{1000}$

#### Aufgabe 3 a)

Geben Sie für obigen Plan für jeden Teilausdruck dessen Kardinalität (d.h. die Anzahl der von diesem Teilausdruck produzierten Tupel) an.

#### Lösung

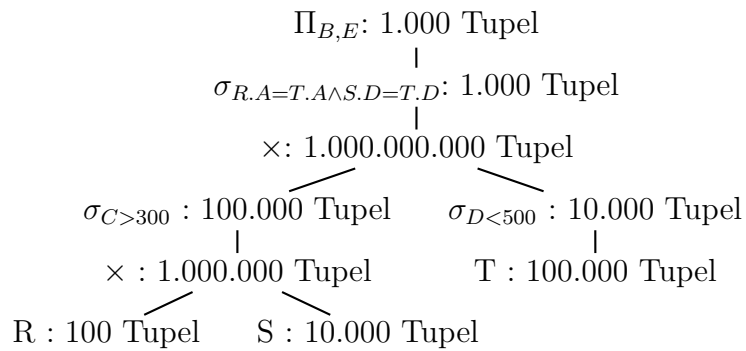


Abbildung 8: Kardinalitäten im voroptimierten Operatorbaum

### Aufgabe 3 b)

Verbessern Sie den oben abgebildeten Operatorbaum mit Hilfe der in der Vorlesung vorgestellten Heuristiken.

### Lösung

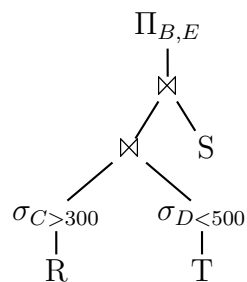


Abbildung 9: Optimierter Operatorbaum

### Aufgabe 3 c)

Geben Sie für den in Aufgabenteil b) erstellten, optimierten Operatorbaum für jeden Teilausdruck die Ergebniskardinalität an.

### Lösung

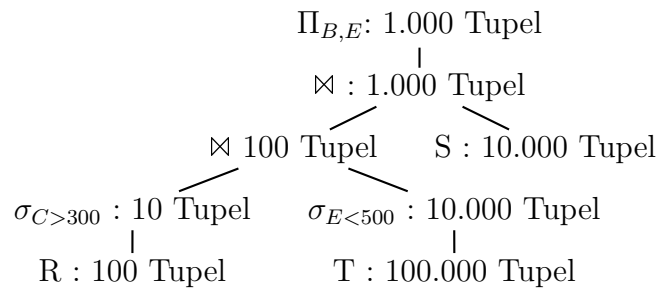


Abbildung 10: Kardinalitäten im optimierten Operatorbaum

---

## Aufgabe 4

---

### Aufgabe 4 a)

Die Eigenschaften des Transaktionskonzepts werden oft (wie in der Vorlesung) mit der Abkürzung ACID zusammengefasst. ACID steht hierbei für vier grundlegende Eigenschaften, die die Transaktionen in einem Datenbanksystem erfüllen müssen. Nennen Sie diese vier Eigenschaften und diskutieren Sie deren Anforderungen bzw. Bedeutung.

### Lösung

#### Atomicity

Alle Operationen einer Transaktion müssen atomar, d.h. ganz oder gar nicht ausgeführt werden.

#### Consistency

Die von einer Transaktion durchgeführten Operationen müssen das Datenbanksystem von einem konsistenten Zustand wieder in einem konsistenten Zustand überführen.

#### Isolation

Mehrere nebenläufige Transaktionen müssen voneinander isoliert werden, d.h. die Transaktionen dürfen sich gegenseitig nicht beeinflussen.

#### Durability

Die von einer erfolgreich abgeschlossenen Transaktion (commit) durchgeführten Änderungen dürfen (auch im Fall von Systemfehlern) nicht verloren gehen, d.h. die Änderungen müssen dauerhaft erhalten bleiben.

#### Aufgabe 4 b)

In der Vorlesung wurden die Strategien  $\text{force} / \neg \text{force}$  und  $\text{steal} / \neg \text{steal}$  besprochen. Diese Strategien bestimmen das Verhalten eines Datenbanksystems in Bezug auf das Einbringen von Änderungsoperationen auf Hintergrundspeicher bzw. das Ersetzen von Pufferseiten. Diskutieren Sie die Unterschiede zwischen den einzelnen Strategien bzw. deren Kombinationen. Worin liegen deren spezifische Vor- und Nachteile?

#### Lösung

- $\text{force}$ : Geänderte Pufferseiten müssen beim Transaktionsabschluss ( $\text{commit}$ ) auf den Hintergrundspeicher geschrieben werden. Hierdurch müssen (theoretisch!) keine Redo-Informationen gespeichert werden, da sichergestellt ist, dass alle Änderungsoperationen auch auf dem Hintergrundspeicher eingelagert sind. Die  $\text{force}$ -Strategie erzwingt dass die von der jeweiligen Transaktion betroffenen Teile des Datenbankpuffers beim  $\text{commit}$  ausgeschrieben werden müssen. Dies kann die Leistung des Systems negativ beeinflussen.
- $\neg \text{force}$ : Bei der  $\neg \text{force}$  Strategie müssen beim Transaktionsabschluss ( $\text{commit}$ ) die geänderten Pufferseiten *nicht* sofort auf den Hintergrundspeicher geschrieben werden. Hierdurch kann es vorkommen dass bei einem Systemabsturz die von einer bereits erfolgreich abgeschlossenen Transaktion durchgeführten Änderungen nicht auf dem Hintergrundspeicher vorhanden sind. Um die Dauerhaftigkeit (*durability*) der Transaktion zu gewährleisten muss also ein Redo-Log gepflegt werden. Mit dessen Hilfe können beim Wiederanlauf (*restart recovery*) die Änderungen rekonstruiert werden.
- $\text{steal}$ : Beim Einsatz der  $\text{Steal}$ -Strategie können geänderte Seiten (sog. *dirty pages*) einer noch aktiven Transaktion aus dem Puffer auf den Hintergrundspeicher verdrängt werden. Hierdurch kann im Puffer Platz für andere Seiten geschaffen werden. Da hierdurch Änderungsoperationen von noch nicht abgeschlossenen Transaktionen auf dem Hintergrundspeicher stehen, muss das System Undo-Informationen speichern, um diese Änderungen im Fall eines Transaktionsabbruchs (*abort*) rückgängig machen zu können.
- $\neg \text{steal}$ : Wird die  $\neg \text{steal}$ -Strategie eingesetzt werden die von noch aktiven Transaktionen geänderten Seiten nicht aus dem Puffer verdrängt. Es kann also kein Pufferplatz durch Verdrängung freigeräumt werden, es wird jedoch garantiert dass keine Änderungsoperationen von noch aktiven Transaktionen auf dem Hintergrundspeicher materialisiert werden. Undo-Informationen müssen deshalb bei einer  $\neg$ -Steal Strategie nicht gespeichert werden.

---

## Freiwillige Zusatzaufgabe 5

---

Implementieren Sie in einer Programmiersprache Ihrer Wahl eine Datenstruktur zur Verwaltung von Transaktionshistorien. Der Einfachheit halber muss Ihre Implementierung nur Lese- und Schreiboperationen für eine bestimmte Transaktion auf einzelnen Datenelementen, soweit die Commit- oder Abort-Operation für eine Transaktion enthalten. Ihre Implementierung sollte entsprechende Historien in textueller Form (z.B. `r1[x]w1[x]c1r2[y]w2[y]a2`) einlesen können.

### Lösung

Siehe Webseite.