

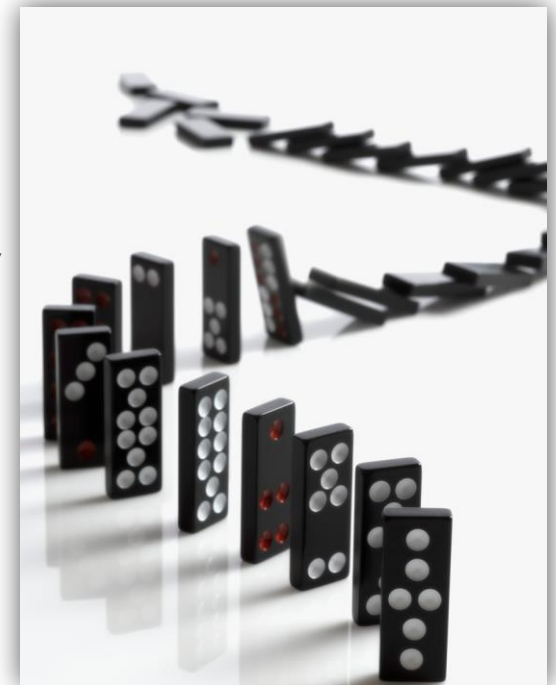


**Hasso
Plattner
Institut**

IT Systems Engineering | Universität Potsdam

Übung Datenbanksysteme I
**Transaktionen,
Selektivität,
XML**

Thorsten Papenbrock



Übersicht: Übungsthemen

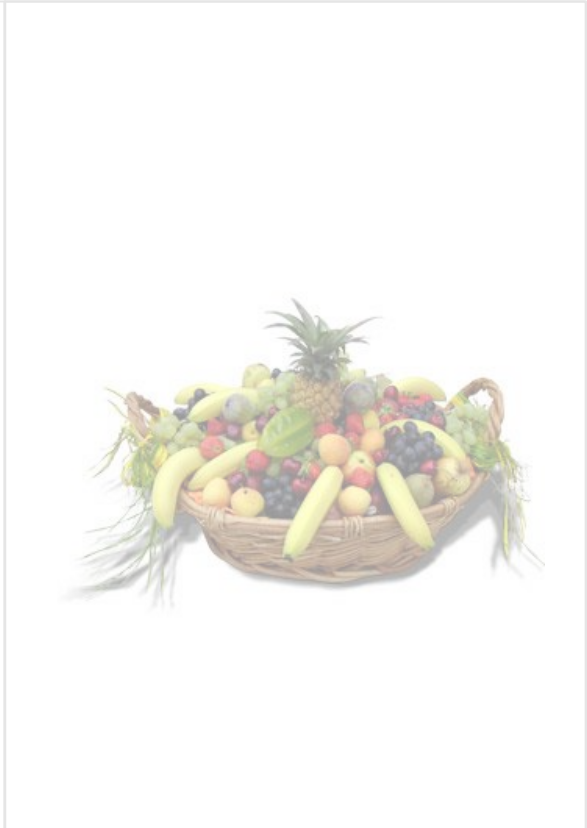
2



Transaktionen



Selektivität



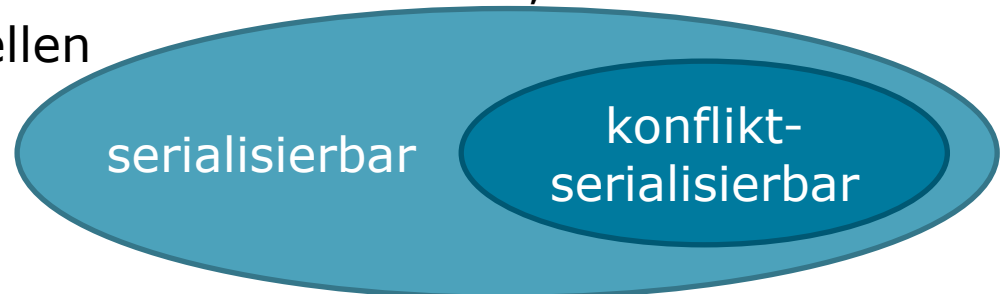
XML

3

Eine **Transaktion** ist eine **Folge von Operationen** (Aktionen), die die Datenbank von einem **konsistenten Zustand** in einen konsistenten (eventuell veränderten) Zustand überführt, wobei das **ACID-Prinzip** eingehalten werden muss.

- **Atomicity** (Atomarität) Transaktion wird entweder ganz oder gar nicht ausgeführt.
- **Consistency** (Konsistenz) Datenbank ist vor Beginn und nach Beendigung einer Transaktion jeweils in einem konsistenten Zustand.
- **Isolation** (Isolation) Transaktion, die auf einer Datenbank arbeitet, sollte den „Eindruck“ haben, dass sie allein auf dieser Datenbank arbeitet.
- **Durability** (Dauerhaftigkeit) Nach erfolgreichem Abschluss einer Transaktion muss das Ergebnis dieser Transaktion „dauerhaft“ in der Datenbank gespeichert werden.

- **Schedule:** „Ablaufplan“ für Transaktionen, bestehend aus einer Abfolge von Transaktionsoperationen
- **Serieller Schedule:** Schedule, in dem Transaktionen vollständig hintereinander ausgeführt werden
- **Serialisierbarer Schedule:** Schedule, dessen Effekt identisch zum Effekt eines (beliebig gewählten!) seriellen Schedules ist
- **Konfliktäquivalente Schedules:** Zwei Schedules, bei denen die Reihenfolge aller konfligierender Aktionen gleich ist
- **Konfliktserialisierbarer Schedule:** Schedule, der konflikt-äquivalent zu einem seriellen Schedule ist



- **Legal Schedule:** Schedule, der kein gesperrtes Objekt erneut sperrt
- **Konsistente Transaktion:** Transaktion, die Aktionen nur auf korrekt gesperrten Objekten ausführt und gesperrte Objekte nach der Verwendung wieder frei gibt
- **2-Phase-Locking:** Alle Sperren einer Transaktion erfolgen vor der ersten Freigabe einer Sperre; ermöglicht Konfliktserialisierbarkeit

6

1. Zeige, dass die beiden seriellen Schedules T_1, T_2 und T_2, T_1 dasselbe Ergebnis liefern (also äquivalent sind).

T_1	T_2
read(A, a_1)	read(B, b_2)
$a_1 := a_1 + 2$	$b_2 := b_2 * 2$
write(A, a_1)	write(B, b_2)
read(B, b_1)	read(A, a_2)
$b_1 := b_1 * 3$	$a_2 := a_2 + 3$
write(B, b_1)	write(A, a_2)

Lösung: Serialisierbarkeit

7

1. Zeige, dass die beiden seriellen Schedules T_1, T_2 und T_2, T_1 dasselbe Ergebnis liefern (also äquivalent sind).

Ergebnis(T_1, T_2):

- $A = (A+2)+3$
- $B = (B*3)*2$

Ergebnis(T_2, T_1):

- $A = (A+3)+2$
- $B = (B*2)*3$

Ergebnisse sind gleich,
da + und * kommutativ
und assoziativ sind!

Aufgabe: Serialisierbarkeit

8

1. Zeige, dass die beiden seriellen Schedules T_1, T_2 und T_2, T_1 dasselbe Ergebnis liefern (also äquivalent sind).
2. Gib einen nicht-seriellen, aber serialisierbaren Schedule an.

T_1	T_2
read(A, a_1)	read(B, b_2)
$a_1 := a_1 + 2$	$b_2 := b_2 * 2$
write(A, a_1)	write(B, b_2)
read(B, b_1)	read(A, a_2)
$b_1 := b_1 * 3$	$a_2 := a_2 + 3$
write(B, b_1)	write(A, a_2)

Lösung: Serialisierbarkeit

9

1. Zeige, dass die beiden seriellen Schedules T_1, T_2 und T_2, T_1 dasselbe Ergebnis liefern (also äquivalent sind).
2. Gib einen nicht-seriellen, aber serialisierbaren Schedule an.

T_1	T_2
read(A, a_1)	read(B, b_2)
	$b_2 := b_2 * 2$
	write(B, b_2)
$a_1 := a_1 + 2$	
write(A, a_1)	
read(B, b_1)	
$b_1 := b_1 * 3$	
write(B, b_1)	
	read(A, a_2)
	$a_2 := a_2 + 3$
	write(A, a_2)

Serialisierbarkeit

10

1. Zeige, dass die beiden seriellen Schedules T_1, T_2 und T_2, T_1 dasselbe Ergebnis liefern (also äquivalent sind).
2. Gib einen nicht-seriellen, aber serialisierbaren Schedule an.
3. Gib einen nicht-seriellen und nicht-serialisierbaren Schedule an

T_1	T_2
read(A, a_1)	read(B, b_2)
$a_1 := a_1 + 2$	$b_2 := b_2 * 2$
write(A, a_1)	write(B, b_2)
read(B, b_1)	read(A, a_2)
$b_1 := b_1 * 3$	$a_2 := a_2 + 3$
write(B, b_1)	write(A, a_2)

Lösung: Serialisierbarkeit

11

1. Zeige, dass die beiden seriellen Schedules T_1, T_2 und T_2, T_1 dasselbe Ergebnis liefern (also äquivalent sind).
2. Gib einen nicht-seriellen, aber serialisierbaren Schedule an.
3. Gib einen nicht-seriellen und nicht-serialisierbaren Schedule an

T_1	T_2
read(A, a_1)	read(B, b_2)
	$b_2 := b_2 * 2$
	write(B, b_2)
$a_1 := a_1 + 2$	read(A, a_2)
write(A, a_1)	
read(B, b_1)	
$b_1 := b_1 * 3$	
write(B, b_1)	
	$a_2 := a_2 + 3$
	write(A, a_2)

Gegeben sind die folgenden drei Schedules:

1. $r_1(A)$ $r_2(A)$ $r_3(B)$ $w_1(A)$ $r_2(C)$ $r_2(B)$ $w_2(B)$ $w_1(C)$

2. $r_1(A)$ $w_1(B)$ $r_2(B)$ $w_2(C)$ $r_3(C)$ $w_3(A)$

3. $w_3(A)$ $r_1(A)$ $w_1(B)$ $r_2(B)$ $w_2(C)$ $r_3(C)$

- Erstelle für jeden Schedule den zugehörigen Konfliktgraph
- Bestimme für jeden Schedule, ob dieser konfliktserialisierbar ist
 - falls "ja", nenne einen konfliktäquivalenten seriellen Schedule
 - falls "nein", nenne alle nicht-serialisierbaren, konfligierenden Aktionskombinationen

13

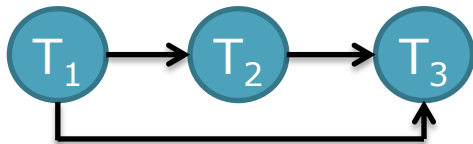
Gegeben sind die folgenden drei Schedules:

1. $r_1(A)$ $r_2(A)$ $r_3(B)$ $w_1(A)$ $r_2(C)$ $r_2(B)$ $w_2(B)$ $w_1(C)$



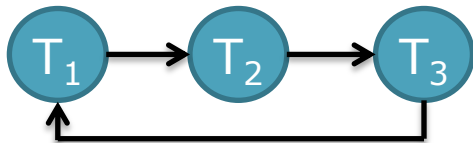
äquivalenter, serieller Schedule: T_3, T_2, T_1

2. $r_1(A)$ $w_1(B)$ $r_2(B)$ $w_2(C)$ $r_3(C)$ $w_3(A)$



äquivalenter, serieller Schedule: T_1, T_2, T_3

3. $w_3(A)$ $r_1(A)$ $w_1(B)$ $r_2(B)$ $w_2(C)$ $r_3(C)$



wegen $w_3(A)$ $r_1(A)$ muss T_3, T_1 gelten
 wegen $w_1(B)$ $r_2(B)$ muss T_1, T_2 gelten
 wegen $w_2(C)$ $r_3(C)$ muss T_2, T_3 gelten



Gegeben die folgenden beiden Transaktionen:

- T_1 : $I_1(A)$ $r_1(A)$ $w_1(A)$ $I_1(B)$ $u_1(A)$ $r_1(B)$ $w_1(B)$ $u_1(B)$
- T_2 : $I_2(B)$ $r_2(B)$ $w_2(B)$ $I_2(A)$ $u_2(B)$ $r_2(A)$ $w_2(A)$ $u_2(A)$
 - Sind T_1 und T_2 konsistent?
 - Sind T_1 und T_2 2PL-konform?

Ja: Vor jedem Zugriff Lock, danach Unlock

Ja: Kein weiterer Lock nach erstem Unlock

Gegeben der folgende Schedule der beiden Transaktionen:

- S: $I_1(A)$ $r_1(A)$ $I_2(B)$ $r_2(B)$ $w_1(A)$ $w_2(B)$ $I_1(B)$ $I_2(A)$ $u_1(A)$ $u_2(B)$
 $r_2(A)$ $r_1(B)$ $w_1(B)$ $w_2(A)$ $u_2(A)$ $u_1(B)$
 - Ist S legal?
 - Ist S (ohne Locks) konfliktserialisierbar?
 - Beschreibe den Ablauf im Scheduler!

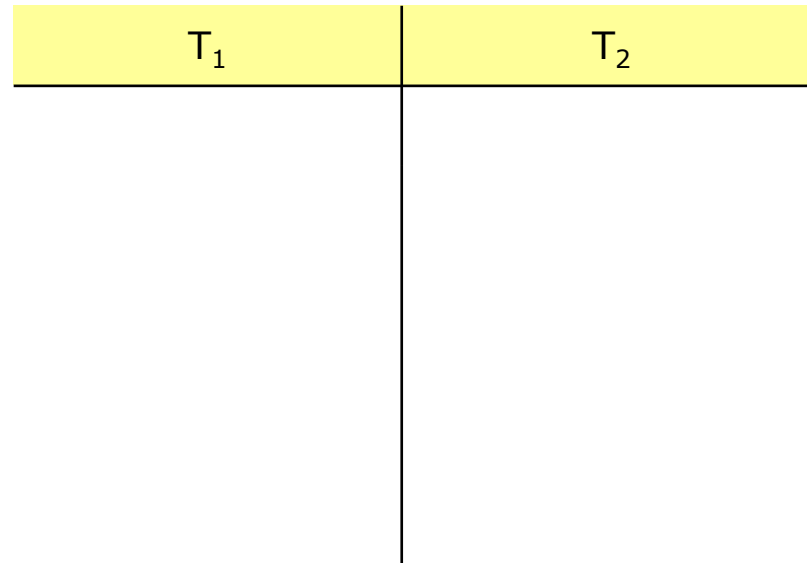
Nein: T_1 und T_2 sperren gleichzeitig A und B

Nein: $T_1 \leftrightarrow T_2$ bilden einen Zyklus

Lösung: Scheduler 1

15

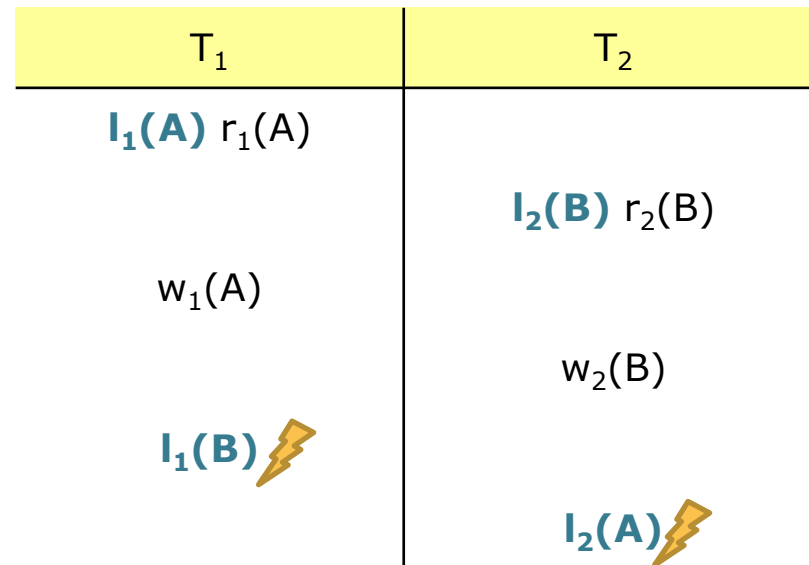
S: **I₁(A)** r₁(A) **I₂(B)** r₂(B) w₁(A) w₂(B) **I₁(B)** **I₂(A)** **u₁(A)** **u₂(B)**
r₂(A) r₁(B) w₁(B) w₂(A) **u₂(A)** **u₁(B)**



Lösung: Scheduler 1

16

S: **I₁(A)** r₁(A) **I₂(B)** r₂(B) w₁(A) w₂(B) **I₁(B)** **I₂(A)** **u₁(A)** **u₂(B)**
r₂(A) r₁(B) w₁(B) w₂(A) **u₂(A)** **u₁(B)**



Deadlock!

Muss vom Scheduler erkannt und aufgelöst werden durch z.B. Rollback von T₂.

Aufgabe: Scheduler 2

17

Gegeben der folgende Schedule:

- S: $r_1(A) r_2(B) r_3(C) r_2(C) r_1(B) w_3(D)$

Aufgaben:

1. Prüfe, ob S konfliktserialisierbar ist.
 - Führe die Prüfung mittels graphbasiertem Test durch.
 - Gib einen seriellen, konfliktequivalenten Schedule bzw. die nicht-serialisierbare, konfligierende Aktionskombination an.

Gegeben der folgende Schedule:

- S: $r_1(A) r_2(B) r_3(C) r_2(C) r_1(B) w_3(D)$

1. Prüfe, ob S konfliktserialisierbar ist.

- Führe die Prüfung mittels graphbasiertem Test durch.



keine Zyklen, daher konflikt-serialisierbar

- Gib einen seriellen, konfliktequivalenten Schedule bzw. die nicht-serialisierbare, konfligierende Aktionskombination an.

T_1, T_2, T_3 ist serieller, konfliktequivalenter Schedule

Aufgabe: Scheduler 2

19

Gegeben der folgende Schedule:

- S: $r_1(A) r_2(B) r_3(C) r_2(C) r_1(B) w_3(D)$

Aufgaben:

1. Prüfe, ob S konfliktserialisierbar ist.
 - Führe die Prüfung mittels graphbasiertem Test durch.
 - Gib einen seriellen, konfliktequivalenten Schedule bzw. die nicht-serialisierbare, konfligierende Aktionskombination an.
2. Ergänze den Schedule um Lock- und Unlock-Operationen, so dass alle enthaltenen Transaktionen konsistent und 2PL-konform sind.

Lösung: Scheduler 2

20

Gegeben der folgende Schedule:

▪ S: $r_1(A) r_2(B) r_3(C) r_2(C) r_1(B) w_3(D)$

2. Ergänze den Schedule um Lock- und Unlock-Operationen, so dass alle enthaltenen Transaktionen konsistent und 2PL-konform sind.

S': $\mathbf{l_1(A)} r_1(A) \mathbf{u_1(A)} \mathbf{l_2(B)} r_2(B) \mathbf{u_2(B)} \mathbf{l_3(C)} r_3(C) \mathbf{u_3(C)} \mathbf{l_2(C)}$
 $r_2(C) \mathbf{u_2(C)} \mathbf{l_1(B)} r_1(B) \mathbf{u_1(B)} \mathbf{l_3(D)} w_3(D) \mathbf{u_3(D)}$

Was ist das Problem dieses Schedules?

Schedule ist nicht 2PL-konform und daher nicht unbedingt konfliktserialisierbar!

Lösung: Scheduler 2

21

Gegeben der folgende Schedule:

▪ S: $r_1(A) r_2(B) r_3(C) r_2(C) r_1(B) w_3(D)$

2. Ergänze den Schedule um Lock- und Unlock-Operationen, so dass alle enthaltenen Transaktionen konsistent und 2PL-konform sind.

S': $\mathbf{l_1(A)} r_1(A) \mathbf{l_2(B)} r_2(B) \mathbf{l_3(C)} r_3(C) \mathbf{l_2(C)} r_2(C) \mathbf{u_2(B)} \mathbf{u_2(C)}$
 $\mathbf{l_1(B)} r_1(B) \mathbf{u_1(A)} \mathbf{u_1(B)} \mathbf{l_3(D)} w_3(D) \mathbf{u_3(C)} \mathbf{u_3(D)}$

2PL verletzt?

Nein, weil die 2PL-Bedingung
nur für Transaktionen gilt!

Aufgabe: Scheduler 2

22

Gegeben der folgende Schedule:

- S: $r_1(A) r_2(B) r_3(C) r_2(C) r_1(B) w_3(D)$

Aufgaben:

1. Prüfe, ob S konfliktserialisierbar ist.
 - Führe die Prüfung mittels graphbasiertem Test durch.
 - Gib einen seriellen, konfliktequivalenten Schedule bzw. die nicht-serialisierbare, konfligierende Aktionskombination an.
2. Ergänze den Schedule um Lock- und Unlock-Operationen, so dass alle enthaltenen Transaktionen konsistent und 2PL-konform sind.
3. Erstelle den tabellarischen Ablaufplan der Transaktionsausführung eines DBMS-Schedulers mit den Locks aus 2).

Lösung: Scheduler 2

23

Gegeben der folgende Schedule:

- S' : $I_1(A)$ $r_1(A)$ $I_2(B)$ $r_2(B)$ $I_3(C)$ $r_3(C)$ $I_2(C)$ $r_2(C)$ $u_2(B)$ $u_2(C)$
 $I_1(B)$ $r_1(B)$ $u_1(A)$ $u_1(B)$ $I_3(D)$ $w_3(D)$ $u_3(C)$ $u_3(D)$

3. Erstelle den tabellarischen Ablaufplan der Transaktionsausführung eines DBMS-Schedulers.

T_1	T_2	T_3

Lösung: Scheduler 2

24

Gegeben der folgende Schedule:

- S' : $I_1(A) r_1(A) I_2(B) r_2(B) I_3(C) r_3(C) I_2(C) r_2(C) u_2(B) u_2(C)$
 $I_1(B) r_1(B) u_1(A) u_1(B) I_3(D) w_3(D) u_3(C) u_3(D)$

3. Erstelle den tabellarischen Ablaufplan der Transaktionsausführung eines DBMS-Schedulers.

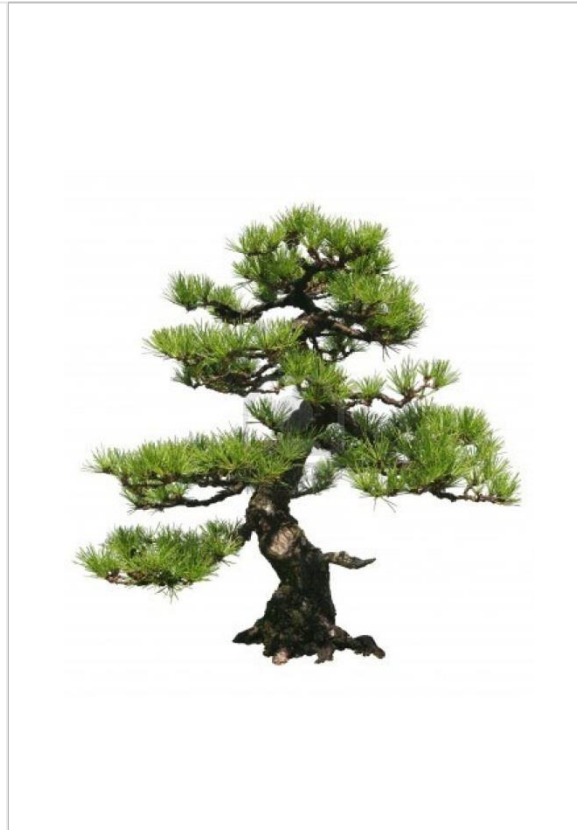
T ₁	T ₂	T ₃
$I_1(A) r_1(A)$	$I_2(B) r_2(B)$	$I_3(C) r_3(C)$
$I_1(B)$ ⚡	$I_2(C)$ ⚡	$I_3(D) w_3(D)$
$I_1(B) r_1(B)$ $u_1(A) u_1(B)$	$I_2(C) r_2(C)$ $u_2(B) u_2(C)$	$u_3(C) u_3(D)$

Übersicht: Übungsthemen

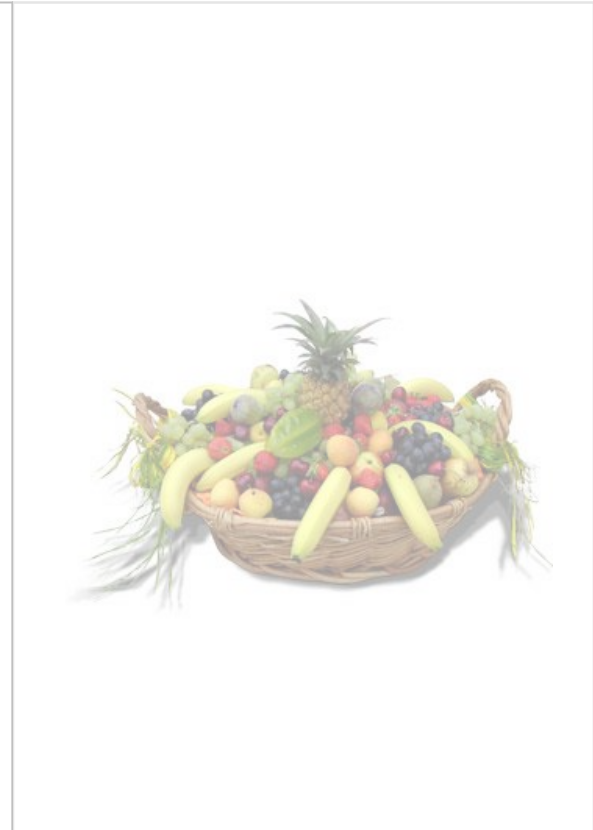
25



Transaktionen



Selektivität



XML

Selektivität: Wiederholung

26

- Selektivität schätzt Anzahl der qualifizierenden Tupel relativ zur Gesamtanzahl der Tupel in einer Relation.

Projektion:

- $sf = |R| / |R| = \mathbf{1}$

Selektion:

- $sf = |\sigma_C(R)| / |R|$

Selektion mit m verschiedenen Werten:

- $sf = (|R| / m) / |R| = \mathbf{1/m}$

Equi-Join zwischen R und S über Fremdschlüssel in S ($S \rightarrow R$):

- $sf = |R \bowtie S| / (|R \times S|) = |S| / (|R| \cdot |S|) = \mathbf{1/|R|}$

Gegeben die folgenden Relationen, Tupel-Verteilungen und Anfrage:

- Zulieferer (zid, name, adresse) 10 Tupel
- Teile (tid, bezeichnung, farbe) 1000 Tupel
- Katalog (zid, tid, kosten) 4000 Tupel

- farbe in {schwarz, rot, blau, weiß} gleichverteilt
- kosten in [1,100] gleichverteilt

- $\sigma_{\text{farbe}=\text{'schwarz'} \vee \text{farbe}=\text{'blau'}} ((\text{Zulieferer} \bowtie \sigma_{\text{kosten} \leq 25}(\text{Katalog})) \bowtie \text{Teile})$

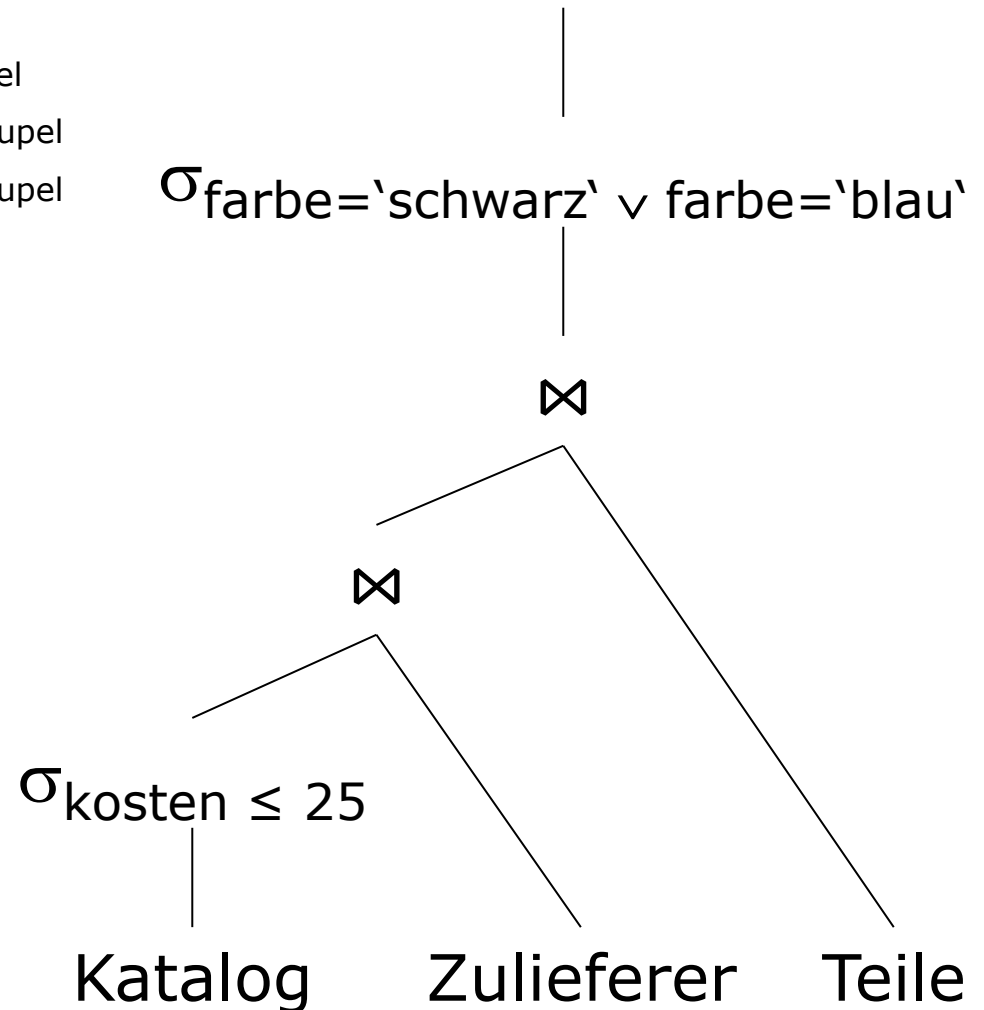
Konstruiere den zugehörigen Parsbaum und annotiere an jeder Kante die zu erwartenden Kardinalitäten!

Lösung: Ergebniskardinalität schätzen

28

- Zulieferer (zid, name, adresse) 10 Tupel
- Teile (tid, bezeichnung, farbe) 1000 Tupel
- Katalog (zid, tid, kosten) 4000 Tupel

- farbe in {schwarz, rot, blau, weiß}
gleichverteilt
- kosten in [1,100]
gleichverteilt

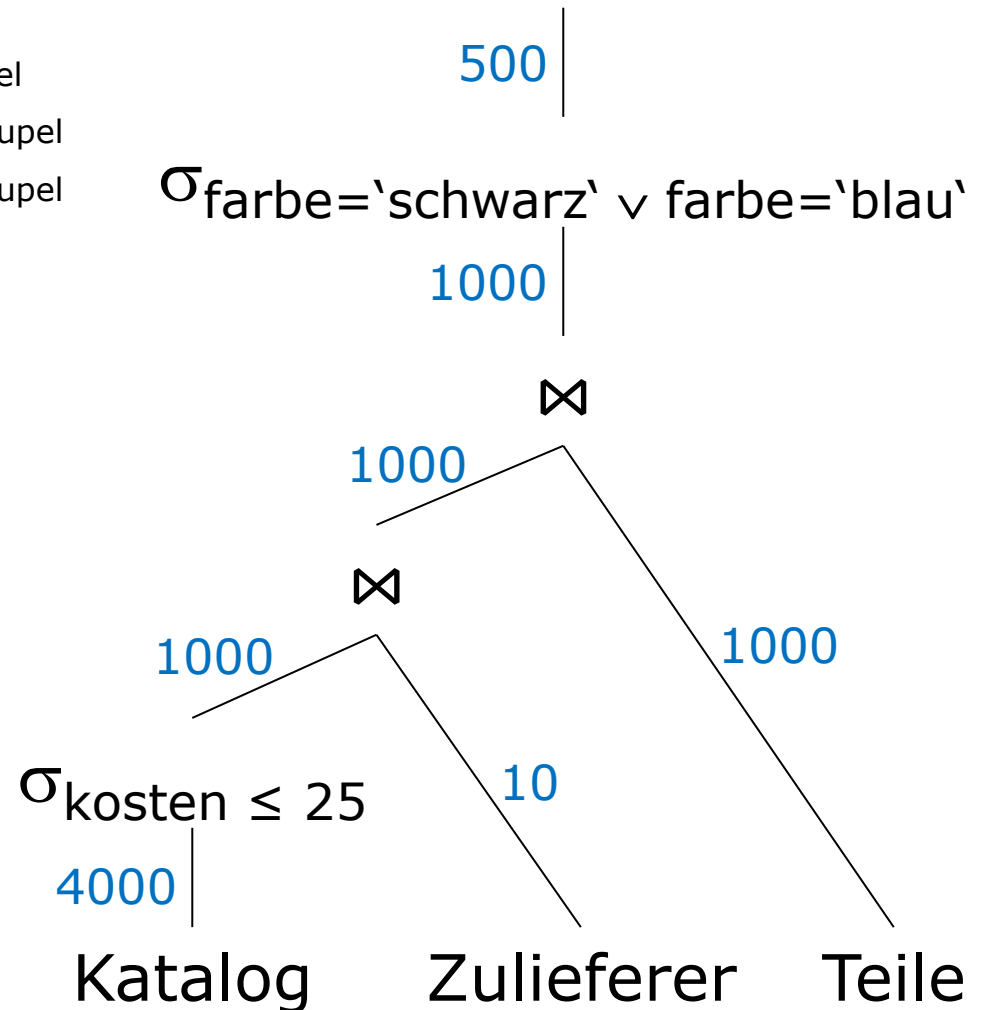


Lösung: Ergebniskardinalität schätzen

29

- Zulieferer (zid, name, adresse) 10 Tupel
- Teile (tid, bezeichnung, farbe) 1000 Tupel
- Katalog (zid, tid, kosten) 4000 Tupel

- farbe in {schwarz, rot, blau, weiß}
gleichverteilt
- kosten in [1,100]
gleichverteilt

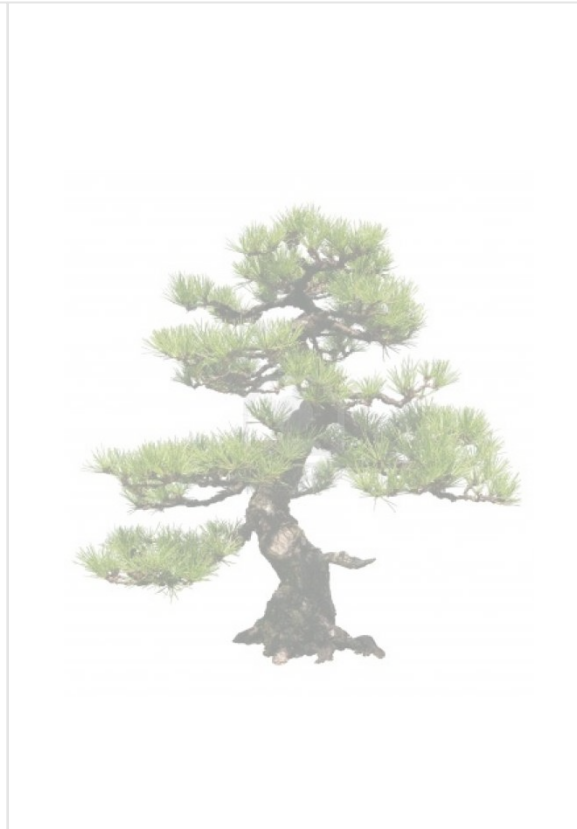


Übersicht: Übungsthemen

30



Transaktionen



Selektivität



XML

XML: CREATE & INSERT

31

```
CREATE TABLE CUSTOMER (  
  cid INTEGER,  
  info XML  
);
```

```
INSERT INTO CUSTOMER (cid, info)  
VALUES (1000,  
  '<customerinfo cid="1000">  
    <name>Kathy Smith</name>  
    <addr country="Canada">  
      <street>5 Rosewood</street>  
      <city>Toronto</city>  
      <prov-state>Ontario</prov-state>  
      <pcode-zip>M6W 1E6</pcode-zip>  
    </addr>  
    <phone type="work">416-555-1358</phone>  
  </customerinfo>'  
);
```

“Lese den Kunden mit ID 1000 aus der Datenbank”

```

SELECT *
FROM CUSTOMER
WHERE XMLEXISTS ('$d/customerinfo[@cid=1000]' passing INFO as "d");
  
```

XML-Inhalt des Attributs "INFO" übergeben als Baum mit dem Wurzelnamen "d"

Ergebnis ist eine Menge von "customerinfo"-Knoten, die die Bedingung erfüllen.
=> {<customerinfo>[...]} | {}

Ergebnis ist eine Menge von Booleschen Werten, die die Bedingung erfüllen.
=> {True} | {False}

```

SELECT *
FROM CUSTOMER
WHERE XMLEXISTS ('$d/customerinfo/@cid=1000' passing INFO as "d");
  
```

$$\mathbf{XMLEXISTS}(X) = \begin{cases} \text{"True", falls } |X| > 0 \\ \text{"False", sonst} \end{cases}$$

XMLEXISTS-Vergleichselement beim Abfragen von XML-Daten
<http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=%2Fcom.ibm.db2.udb.apdv.embed.doc%2Fdoc%2Fc0023906.htm>

“Lese die ID, den Namen und die Nummer von Kathy Smith”

“XMLQUERY” gibt den Wert
des Ausdrucks zurück

```
SELECT Cid,  
    XMLQUERY('$d/customerinfo/name/text()' passing INFO as "d") AS Name,  
    XMLQUERY('$d/customerinfo/phone/text()' passing INFO as "d") AS Phone  
FROM CUSTOMER  
WHERE XMLEXISTS (  
    '$d/customerinfo/name[text()="Kathy Smith"]' passing INFO as "d"  
);
```

“text()” gibt den Text-Inhalt des
Knotens “customerinfo/name” zurück

**XMLQuery-Funktion zur Abfrage
skalärer Wert aus XML-Strukturen**

<http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/index.jsp?topic=%2Fcom.ibm.db2.luw.sql.ref.doc%2Fdoc%2Fr0022193.html>

“Lese alle Kunden aus der Datenbank, deren Nummer 555 enthält”

```

SELECT *
FROM CUSTOMER
WHERE XMLEXISTS (
    '$d/customerinfo/phone[contains(., "555")]' passing INFO as "d"
);
  
```

“contains()” prüft hier, ob 555 enthalten ist.

“Lese alle Kunden aus der Datenbank, die mehr als 2 Nummern haben”

```

SELECT *
FROM CUSTOMER
WHERE XMLEXISTS (
    '$d/customerinfo[count(phone)>2]' passing INFO as "d"
);
  
```

“count()” zählt hier, wie viele “phone”-Kindknoten eine customerinfo hat

Funktion contains zur Prüfung von Teilstrings

<http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/index.jsp?topic=%2Fcom.ibm.db2.luw.admin.ts.doc%2Fdoc%2Fxrqnftcontains.html>

Funktion count zum Zählen von Kindknoten

<http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=%2Fcom.ibm.db2.xquery.doc%2Fdoc%2Fxrqfncnt.htm>

```
SELECT m.mid,  
  xmlelement(name "movie",  
    xmlelement(name "mid", m.mid),  
    xmlelement(name "title", m.title),  
    xmlelement(name "year", m.year),  
    xmlelement(name "actors", xmlagg(  
      xmlelement(name "actor",  
        xmlelement(name "name", a.name))  
    ))  
  ) AS info  
FROM movie m JOIN  
  (SELECT * FROM actor UNION SELECT * FROM actress) a  
  ON m.mid = a.movie_id  
GROUP BY m.mid, m.title, m.year;
```

("Ghosts of the Past (1991) (TV)",

```
"<movie>
  <mid>Ghosts of the Past (1991) (TV)</mid>
  <title>Ghosts of the Past</title>
  <year>1991</year>
  <actors>
    <actor>
      <name>Davis, Carl (IV)</name>
    </actor>
    <actor>
      <name>McCartney, Paul</name>
    </actor>
  </actors>
</movie>"
```

 Ergebnis

