Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

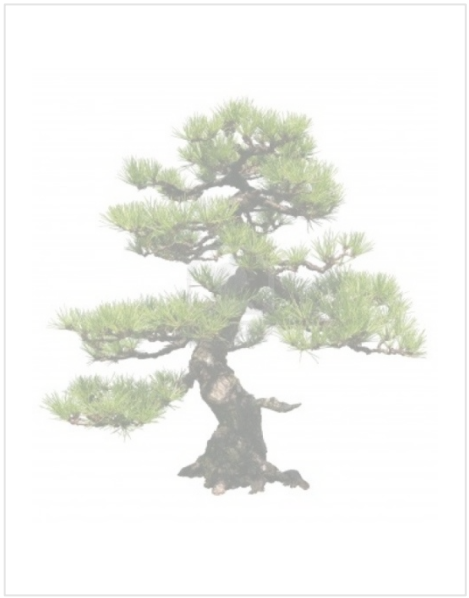
Folien basierend auf Thorsten Papenbrock

Übung Datenbanksysteme I
Transaktionen, Selektivität, XML

Leon Bornemann
G-3.1.14, Campus III / F-E.06 Campus II
Hasso Plattner Institut

Übersicht

Übungsthemen



Transaktionen

Selektivität

XML

DBSI - Übung

Transaktionen,
Selektivität, XML

Leon Bornemann

Chart **2**

Wiederholung

Transaktionen

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.

Wiederholung

Isolationsstufen

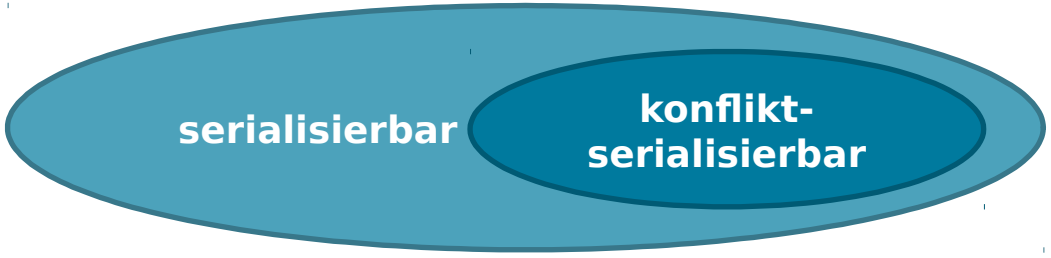
Isolations-Level	Dirty Reads	Non-Repeatable Reads	Phantom Reads
READ_UNCOMMITTED	möglich	möglich	möglich
READ_COMMITTED	verhindert	möglich	möglich
REPEATABLE_READ	verhindert	verhindert	möglich
SERIALIZABLE !	verhindert	verhindert	verhindert

Wenn alle Anomalien verhindert werden sollen,
dann muss Isolationsstufe „serializeable“ sichergestellt werden!

Schedules

- **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



Wiederholung

Locking

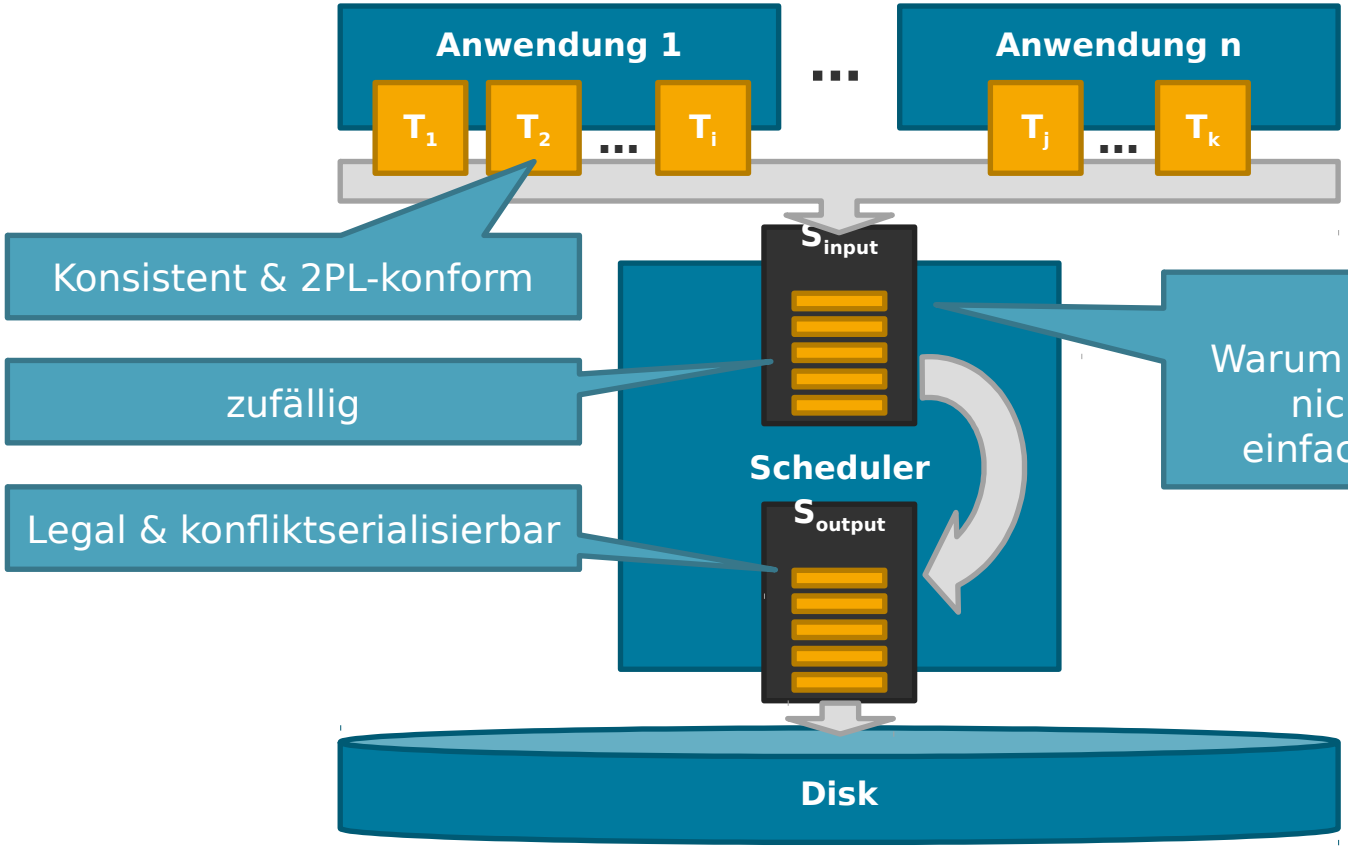
- **Legaler Schedule**: Schedule, der kein gesperrtes Objekt erneut sperrt
- **Konsistente Transaktion**: Transaktion, die Aktionen nur auf korrekt gesperrten Objekten ausführt und gesperrte Objekte nach der Verwendung wieder frei gibt

- **2-Phase-Locking (2PL)**: Alle Sperren einer Transaktion erfolgen vor der ersten Freigabe einer Sperre; ermöglicht Konfliktserialisierbarkeit

- **Zuständigkeiten:**
  - **Transaktion:** 2PL-konform + konsistent
  - **Scheduler:** legal + konflikserialisierbar

Wiederholung

Konzeptionelles Schaubild



Exkurs

Transaktionen in JDBC

```
Connection con = DriverManager.getConnection(URL, name, pw);
con.setAutoCommit(false);
con.setTransactionIsolation(TRANSACTION_SERIALIZABLE);

Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT * FROM [...]"); // read()
stmt.executeUpdate("INSERT INTO [...]"); // write()
stmt.executeUpdate("DELETE FROM [...]"); // write()
con.commit();
```

Transaktionale Ausführung

Isolationsstufe

Commit der Transaktion und Freigabe der Locks

https://docs.oracle.com/javase/tutorial/jdbc/basics/transactions.html
https://dzone.com/articles/jdbc-faq-transactions

Aufgabe

Serialisierbarkeit

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

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

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

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)

Aufgabe

Serialisierbarkeit

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

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)

Aufgabe

Konfliktserialisierbarkeit

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

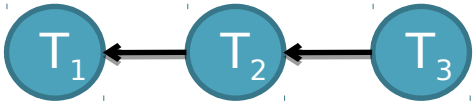
- Erstelle für jeden Schedule den zugehörigen Konfliktgraph
- Bestimme für jeden Schedule, ob dieser konfliktserialisierbar ist
  - falls “ja“, nenne einen konfliktäquivalenten seriellen Schedule
  - falls “nein“, nenne alle nicht-serialisierbaren, konfligierenden Aktionskombinationen

Lösung

Konfliktserialisierbarkeit

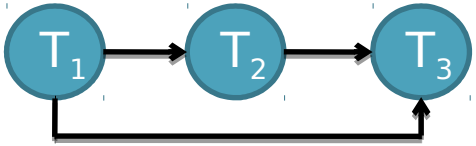
Gegeben sind die folgenden drei Schedules:

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



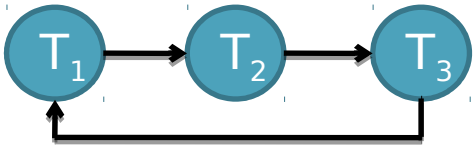
äquivalenter, serieller Schedule: T_3,T_2,T_1

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

Aufgabe

Scheduler 1

Gegeben die folgenden beiden Transaktionen:

- T_1: **$l_1(A)$** $r_1(A)$ $w_1(A)$ **$l_1(B)$** **$u_1(A)$** $r_1(B)$ $w_1(B)$ **$u_1(B)$**
- T_2: **$l_2(B)$** $r_2(B)$ $w_2(B)$ **$l_2(A)$** **$u_2(B)$** $r_2(A)$ $w_2(A)$ **$u_2(A)$**
  - 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: **$l_1(A)$** $r_1(A)$ **$l_2(B)$** $r_2(B)$ $w_1(A)$ $w_2(B)$ **$l_1(B)$** **$l_2(A)$** **$u_1(A)$** **$u_2(B)$** $r_2(A)$ $r_1(B)$ $w_1(B)$ $w_2(A)$ **$u_2(A)$** **$u_1(B)$**
  - 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 und T_2 bilden einen Zyklus

Lösung

Scheduler 1

- S: **$l_1(A)$** $r_1(A)$ **$l_2(B)$** $r_2(B)$ $w_1(A)$ $w_2(B)$ **$l_1(B)$** **$l_2(A)$** **$u_1(A)$** **$u_2(B)$** $r_2(A)$ $r_1(B)$ $w_1(B)$ $w_2(A)$ **$u_2(A)$** **$u_1(B)$**

T_1	T_2

Lösung

Scheduler 1

- S: $l_1(A)$ $r_1(A)$ $l_2(B)$ $r_2(B)$ $w_1(A)$ $w_2(B)$ $l_1(B)$ $l_2(A)$ $u_1(A)$ $u_2(B)$ $r_2(A)$ $r_1(B)$ $w_1(B)$ $w_2(A)$ $u_2(A)$ $u_1(B)$

T_1	T_2
$l_1(A)$ $r_1(A)$	
	$l_2(B)$ $r_2(B)$
$w_1(A)$	
	$w_2(B)$
$l_1(B)$	
	$l_2(A)$

Deadlock!

Muss vom Scheduler erkannt und aufgelöst werden beispielsweise durch Rollback von T_2.

Aufgabe

Scheduler 2

Gegeben der folgende input 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.

Lösung

Scheduler 2

Gegeben der folgende input 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, konfliktäquivalenter Schedule

Aufgabe

Scheduler 2

Gegeben der folgende input 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, konfliktäquivalenten 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.

Scheduler 2

Gegeben der folgende input 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 dieser Locking-Idee?

Transaktionen sind nicht 2PL-konform;
Konfliktserialisierbarkeit kann so nicht sichergestellt werden!

Lösung

Scheduler 2

Gegeben der folgende input 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‘: $\mathbf{l_1(A)}\ r_1(A)\ \mathbf{l_2(B)}\ r_2(B)\ \mathbf{l_3(C)}\ r_3(C)\ \mathbf{l_2(C)}\ r_2(C)\ \mathbf{u_2(B)}\ \mathbf{u_2(C)}$
$\mathbf{l_1(B)}\ r_1(B)\ \mathbf{u_1(A)}\ \mathbf{u_1(B)}\ \mathbf{l_3(D)}\ w_3(D)\ \mathbf{u_3(C)}\ \mathbf{u_3(D)}$

Legalität verletzt? Ja, aber das ist Aufgabe des Schedulers, nicht der Transaktionen!

Einschub

Optimistisches vs. Pessimistisches Locking

Zwei grundsätzliche Locking-Strategien:

- **Optimistisch:**
  - Lock-Operation vor jedem ersten Zugriff eines Objekts
  - Unlock-Operation sofort wenn letzter Zugriff auf das Objekt erfolgt ist **und** keine weiteren Locks in der Transaktion angefragt werden
    - ➢ Bessere Performance bei vielen unabhängigen und langlaufenden Transaktionen
    - ➢ Anomalien können auftreten (z.B. bei Rollbacks)!
- **Pessimistisch:**
  - Lock-Operation vor jedem ersten Zugriff eines Objekts
  - Unlock-Operation(en) nachdem die Transaktion committed ist
    - ➢ Bessere Performance bei vielen abhängigen und kurzlaufenden Transaktionen
    - ➢ Anomalien werden verhindert!

Aufgabe

Scheduler 2

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) und leite so einen konfliktserialisierbaren Schedule für S ab.

Lösung

Scheduler 2

Gegeben der folgende Schedule:

- S‘: **$l_1(A)$** $r_1(A)$ **$l_2(B)$** $r_2(B)$ **$l_3(C)$** $r_3(C)$ **$l_2(C)$** $r_2(C)$ **$u_2(B)$ $u_2(C)$**
  $l_1(B)$ $r_1(B)$ **$u_1(A)$ $u_1(B)$ $l_3(D)$** $w_3(D)$ **$u_3(C)$ $u_3(D)$**

3. Erstelle den tabellarischen Ablaufplan der Transaktionsausführung eines DBMS-Schedulers und leite so einen konfliktserialisierbaren Schedule ab.

T_1	T_2	T_3

Lösung

Scheduler 2

Gegeben der folgende Schedule:

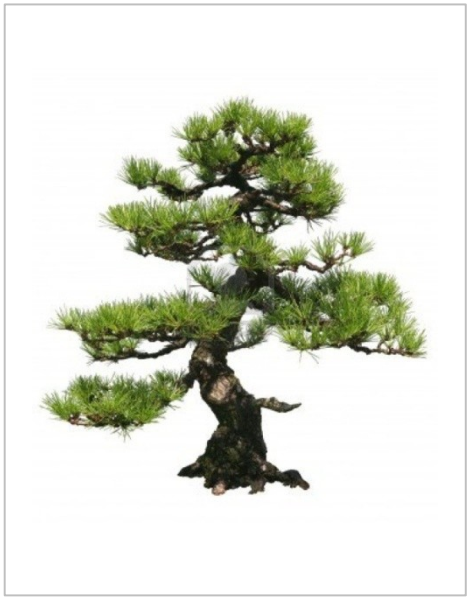
- S‘: **$l_1(A)$** $r_1(A)$ **$l_2(B)$** $r_2(B)$ **$l_3(C)$** $r_3(C)$ **$l_2(C)$** $r_2(C)$ **$u_2(B)$ $u_2(C)$**
  $l_1(B)$ $r_1(B)$ **$u_1(A)$ $u_1(B)$ $l_3(D)$** $w_3(D)$ **$u_3(C)$ $u_3(D)$**

3. Erstelle den tabellarischen Ablaufplan der Transaktionsausführung eines DBMS-Schedulers und leite so einen konfliktserialisierbaren Schedule ab.

T_1	T_2	T_3
$l_1(A)$ $r_1(A)$		
	$l_2(B)$ $r_2(B)$	
		$l_3(C)$ $r_3(C)$
	$l_2(C)$	
$l_1(B)$		
		$l_3(D)$ $w_3(D)$ **$u_3(C)$ $u_3(D)$**
	$l_2(C)$ $r_2(C)$ **$u_2(B)$ $u_2(C)$**	
$l_1(B)$ $r_1(B)$ **$u_1(A)$ $u_1(B)$**		

Übersicht

Übungsthemen



Transaktionen

Selektivität

XML

DBSI - Übung

Transaktionen,
Selektivität, XML

Leon Bornemann

Wiederholung

- **Definition:** Die Selektivität eines Operators schätzt Anzahl der qualifizierenden Tupel relativ zur Gesamtanzahl der Tupel des Operator-Inputs.

- Projektion
  - sf = |R|/|R| = **1**

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

- Selektion (geschätzt als Gleichverteilung mit m verschiedenen Werten)
  - sf = (|R| / m) / |R| = **1/m**

- Equi-Join zwischen R und S über Fremdschlüssel in S (S ⋈ R)
  - sf = |R ⋈ S| / (|R x S|) = |S| / (|R| · |S|) = **1/|R|**

Aufgabe

Ergebniskardinalität schätzen

Gegeben sind 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

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

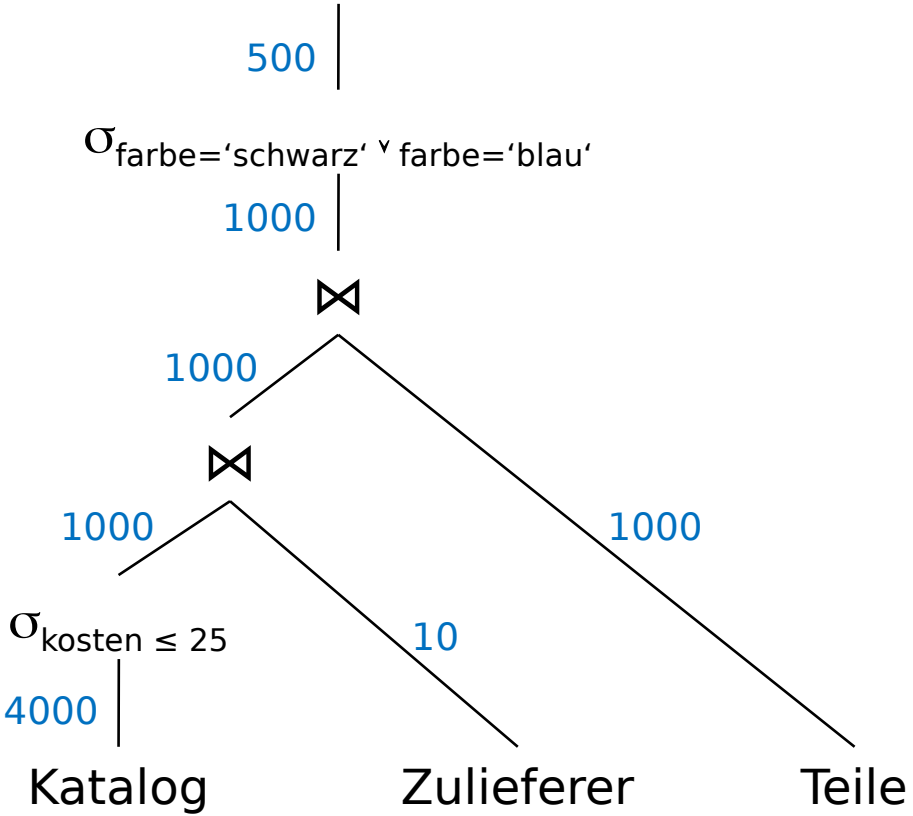
Konstruiere den zugehörigen Parsebaum und annotiere an jeder Kante die zu erwartenden Kardinalitäten!

Lösung

Ergebniskardinalität schätzen

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

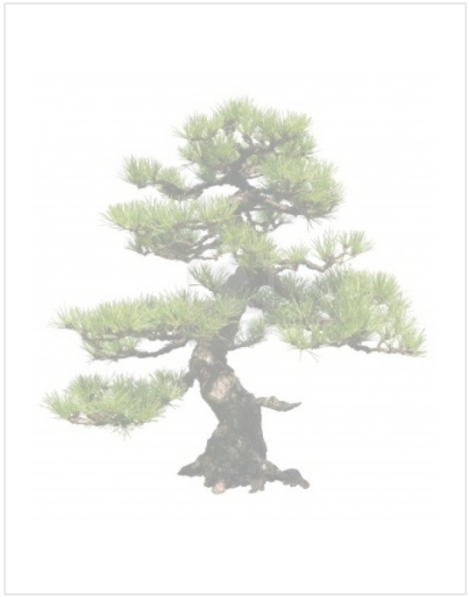
- farbe in {schwarz, rot, blau, weiß}
  gleichverteilt
- kosten in [1,100]
  gleichverteilt



Übersicht

Übungsthemen

Transaktionen



Selektivität

XML

DBSI - Übung

Transaktionen,
Selektivität, XML

Leon Bornemann

XML

CREATE & INSERT

```
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>'
);
```

Relationaler Zugriff (XMLEXISTS)

“Lese den Kunden mit ID 1000 aus der Datenbank”

```
SELECT *
FROM CUSTOMER
WHERE XMLEXISTS ('/customerinfo[@cid=1000]' passing by REF info);
```

XML-Inhalt des Attributs “INFO” als Baum übergeben

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

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

```
SELECT *
FROM CUSTOMER
WHERE XMLEXISTS ('/customerinfo/@cid=1000' passing by REF info);
```

XMLEXISTS(X) = { “True“, falls |X| > 0 ; “False“, sonst }

Relationaler Zugriff (XMLQUERY)

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

“XPATH” gibt den Wert des Ausdrucks zurück

```
SELECT Cid,
   XPATH('/customerinfo/name/text()‘, info) AS Name,
   XPATH('/customerinfo/phone/text()‘, info) AS Phone
FROM CUSTOMER
WHERE XMLEXISTS (
      '/customerinfo/name[text()="Kathy Smith"]' passing by REF info);
```

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

Relationaler Zugriff (CONTAINS & COUNT)

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

```
SELECT *
FROM CUSTOMER
WHERE XMLEXISTS (
      ‘/customerinfo/phone[contains(., “555”)]’ passing by REF info);
```

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

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

```
SELECT *
FROM CUSTOMER
WHERE XMLEXISTS (
      ‘/customerinfo[count(phone)>2]’ passing by REF info);
```

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

Umwandlung: Relational in XML

Anfrage

```
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;
```

Umwandlung: Relational in XML

("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

Übung Datenbanksysteme I
Transaktionen, Selektivität, XML

Leon Bornemann
G-3.1.14, Campus III / F-E.06 Campus II
Hasso Plattner Institut

HPI Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam