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

Übung zur Vorlesung
Einsatz und Realisierung von Datenbanksystemen im SoSe16

Moritz Kaufmann (moritz.kaufmann@tum.de)
http://db.in.tum.de/teaching/ss16/impldb/

Blatt Nr. 02

Hausaufgabe 1

1. Geben Sie alle Eigenschaften an, die von der Historie erfüllt werden.

$$w_1(x), r_2(y), w_3(y), w_2(x), w_3(z), c_3, w_1(z), c_2, c_1$$

richtig	falsch	Aussage
	✔	Serialisierbar (SR)
✔		Rücksetzbar (RC)
✔		Vermeidet kaskadierendes Zurücksetzen (ACA)
	✔	Strikt (ST)

2. Geben Sie alle Eigenschaften an, die von der Historie erfüllt werden.

$$r_1(x), r_1(y), w_2(x), w_3(y), r_3(x), a_1, r_2(x), r_2(y), c_2, c_3$$

richtig	falsch	Aussage
	✔	Serialisierbar (SR)
	✔	Rücksetzbar (RC)
	✔	Vermeidet kaskadierendes Zurücksetzen (ACA)
	✔	Strikt (ST)

3. Gegeben die unvollständige Historie:

$$H = w_1(x), w_1(y), r_2(x), r_2(y)$$

   a) Fügen Sie `commits` in H so ein, dass die Historie RC aber nicht ACA erfüllt:

$$w_1(x), w_1(y), r_2(x), r_2(y), c_1, c_2$$

   b) Fügen Sie `commits` in das ursprüngliche H so ein, dass die Historie ACA erfüllt.

$$w_1(x), w_1(y), c_1, r_2(x), r_2(y), c_2$$

Hausaufgabe 2

a) Erläutern Sie kurz die zwei Phasen des 2PL-Protokolls.

b) Inwiefern unterscheidet sich das *strenge* 2PL?

c) Welche Eigenschaften (SR,RC,ACA,ST) haben Historien, welche vom 2PL und vom strengen 2PL zugelassen werden?

d) Wäre es beim strengen 2PL-Protokoll ausreichend, alle Schreibsperren bis zum EOT (Transaktionsende) zu halten, aber Lesesperren schon früher wieder freizugeben?

Lösung:

a) Jede Transaktion durchläuft zwei Phasen:
   - Eine *Wachstumsphase*, in der sie Sperren anfordern, aber keine freigeben darf und
   - eine *Schrumpfungsphase*, in der sie Sperren freigibt, jedoch keine neuen Sperren anfordern darf.

b) Alle Sperren werden bis zum Ende der Transaktion gehalten und gemeinsam freigegeben. Die Schrumpfungsphase entfällt somit.

c) 2PL garantiert Historien aus SR. Das strenge 2PL garantiert Historien aus SR $\cap$ ST.

d) Es ist ausreichend, beim strengen 2PL-Protokoll nur die Schreibsperren bis zum Ende der Transaktion zu halten. Lesesperren können analog zum normalen 2PL-Protokoll in der Schrumpfungsphase (nach wie vor jedoch nicht in der Wachstumsphase) peu à peu freigegeben werden. Die generierten Schedules bleiben serialisierbar und strikt.

   Begründung
   - Schon das normale 2PL bietet Serialisierbarkeit; diese ist also auch hier gegeben.
   - Das Halten der Schreibsperren bis zum Ende der Transaktion stellt sicher, dass keine Transaktion von einer anderen lesen oder einen von ihr modifizierten Wert überschreiben kann, bevor diese nicht ihr **commit** durchgeführt hat.

     Es gilt:

$$\forall T_i : \forall T_j : (i \neq j)\ \forall A : (w_i(A) <_H r_j(A)) \vee (w_i(A) <_H w_j(A)) \Rightarrow (c_i <_H r_j(A)) \text{ bzw. } (c_i <_H w_j(A))$$

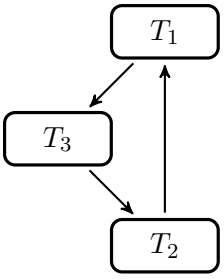
Hausaufgabe 3

Ein inhärentes Problem der sperrbasierten Synchronisationsmethoden ist das auftreten von Verklemmungen (Deadlocks). Zur Erkennung von Verklemmungen wurde der Wartegraph eingeführt. Dabei wird eine Kante $T_i \rightarrow T$ eingefügt, wenn T_i auf die Freigabe einer Sperre durch T wartet.

Skizzieren Sie einen Ablauf von Transaktionen, bei dem ein Deadlock auftritt, der einen Zyklus mit einer Länge von mindestens 3 Kanten im Wartegraphen erzeugt.

Schritt	T_1	T_2	T_3	Bemerkung
1.	BOT			
2.		BOT		
3.			BOT	
4.	lockX(A)			
5.		lockX(B)		
6.			lockX(C)	
7.	write(A)			
8.		write(B)		
9.			write(C)	
10.	lockS(C)			Will C lesen.
11.		lockS(A)		Will A lesen.
12.			lockS(B)	Will B lesen.

Der Wartegraph sieht dann wie folgt aus:



Hausaufgabe 4

Nennen Sie die Vorteile und Nachteile von Deadlock**erkennung** und -**vermeidung** durch:

- Timeouts
- Wartegraphen
- Preclaiming
- Zeitstempel

Sind Kombinationen denkbar/sinnvoll?

Lösung:

- Timeouts
  - \+ Verfahren ist einfach zu implementieren und erfordert sehr geringen Verwaltungsaufwand.
  - − Deadlocks werden erst mit Verzögerung erkannt.
  - − Es gibt false positives.
- Wartegraphen
  - \+ Es werden echte Deadlocks schnell erkannt.
  - − Aufwendig: Pflegen des Graphen, Zyklenerkennung
- Preclaiming
  - \+ Es treten keine Deadlocks auf,
  - − verringert jedoch die Parallelität.
- Deadlockvermeidung durch Zeitstempel
  - \+ Deadlocks treten nicht auf.
  - − Viele Transaktionen müssen zurückgesetzt werden, obwohl nie ein Deadlock auftreten würde – false positives.
  - !!! **Anmerkung:** Nicht zu verwechseln mit *zeitstempelbasierter Synchronisation*. Dort können auch keine Verklemmungen auftreten, aber existieren auch keine Sperren! (vgl. Blatt 13/Aufgabe 5)
    Zur Erinnerung; Zeitstempel treten an verschiedenen Stellen auf:
    - bei der Deadlockvermeidung (wound-wait, wait-die),
    - bei zeitstempelbasierter Synchronisation, und auch
    - bei optimistischer Synchronisation, wo Transaktionen einen Zeitstempel beim Eintritt in die Validierungsphase erhalten. Über die Zeitstempel wird ermittelt, welche Transaktionen parallel gelaufen sind, und mit welchen WriteSets das aktuelle ReadSet verglichen werden muss.

Beispielsweise können Timeout und Wartegraph kombiniert werden. Hierbei wird der Wartegraph erst dann erzeugt, wenn ein Timeout auftritt. Es treten so im Vergleich zum Timeout-Verfahren keine false positives mehr auf und das Verfahren bleibt im Regelfall "billig", da der Wartegraph erst bei Bedarf erzeugt wird. Das Problem der verzögerten Erkennung von Deadlocks bleibt allerdings bestehen.

Gruppenaufgabe 5

Gegeben die Relation „Aerzte", die den Bereitschaftsstatus von Ärtzen modelliert

Name	Vorname	...	Bereit
House	Gregory	...	ja
Green	Mark	...	nein
Brinkmann	Klaus	...	ja

sowie die folgende Transaktion in Pseudocode:

```
dienstende(arzt_name)
```

```
select count(*) into anzahl_bereit from aerzte where bereit='ja'
if anzahl_bereit > 1 then
  update aerzte set bereit='nein' where name=arzt_name
```

Die Transaktion soll dafür sorgen, dass immer mindestens ein Arzt bereit ist.

Betrachten Sie einen Ablauf, bei dem zwei zur Zeit bereite Ärzte zum gleichen Zeitpunkt entscheiden, ihren Status auf „nein “, d.h. nicht bereit zu ändern:

T_1: execute dienstende('House')
T_2: execute dienstende('Brinkmann')

Gehen Sie beispielsweise davon aus, dass das DBMS versucht, die Transaktion jeweils abwechselnd zeilenweise abzuarbeiten.

Diskutieren Sie:

- Was kann bei Snapshot Isolation passieren?
- Warum ist dies bei optimistischer Synchronisation nicht möglich?
- Wie verhält sich strenges 2PL?

- Was kann bei Snapshot Isolation passieren?

  Hier wird defakto die Standardanomalie von Snapshot Isolation gezeigt. Es ist ein Constraint für die Ausprägung der Datenbank gegeben (hier: Es sollte immer mindestens ein Arzt bereit sein, anderes traditionelles Beispiel wäre die Summe des Geldes auf der Welt ist konstant oder ähnliches), jedoch kann dieser bei Snapshot Isolation verletzt werden.

  Im konkreten Fall wird lediglich geprüft, ob sich die Writesets der parallel laufenden Transaktionen überlappen. Dies ist nicht der Fall, weswegen beide Transaktionen bei Snapshot Isolation erfolgreich sind.

- Warum ist dies bei optimistischer Synchronisation nicht möglich?

  Bei optimistischer Synchronisation kann die Anomalie nicht auftreten, da hier geprüft wird, dass sich das Writeset nicht mit dem Readset einer anderen Transaktion überlappt, d.h. das System würde bemerken, dass es versucht etwas zu schreiben, was parallel gelesen wurde und dadurch die Transaktion abbrechen.

- Wie verhält sich strenges 2PL?

  Die Anomalie kann nicht auftreten. Bei abwechselnder zeilenweiser Ausführung würdeni in diesem Fall beide Transaktionen zunächst Shared Locks auf alle Bereitschaftsfelder erwerben. Danach würden beide versuchen, das Lock für ihr Bereitschaftsfeld auf ein Exclusive Lock zu eskalieren. Es entsteht ein Deadlock mit Zykluslänge 2 und eine der Transaktionen wird abgebrochen.