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

Übung zur Vorlesung
Einsatz und Realisierung von Datenbanksystemen im SoSe14

Moritz Kaufmann (moritz.kaufmann@tum.de)
http://www-db.in.tum.de/teaching/ss14/impldb/

Blatt Nr. 2

Aufgabe 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.

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

Aufgabe 2

Wäre es beim strengen 2PL-Protokoll ausreichend, alle Schreibsperren bis zum EOT zu halten, aber Lesesperren schon früher wieder abzutreten? Begründen Sie Ihre Antwort.

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:

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

Aufgabe 3

Weisen Sie (halbwegs) formal nach, dass das 2PL-Protokoll nur serialisierbare Historien zulässt.
Um zu zeigen, dass H serialisierbar ist, weisen wir nach, dass der zugehörige Serialisierbarkeitsgraph $SG(H)$ azyklisch ist:

1. Falls in $SG(H)$ die Kante $T_i \to T_j$ auftritt, so steht eine Operation von T_i mit einer Operation T_j bezüglich eines Datums A in Konflikt. Der Vorüberlegung 1.2 zufolge muss T_i dann die Sperre auf A wieder freigegeben haben, ehe T_j die Sperre darauf erhält.

2. Nehmen wir an, dass der Pfad $T_i \to T_j \to T_k$ in $SG(H)$ auftritt. Aus Schritt 2.1 wissen wir, dass T_i also eine Sperre freigeben musste, ehe T_j eine Sperre erhielt. Analog gab T_j zuvor eine Sperre frei, ehe T_k eine Sperre erlangt. Da wir wissen, dass eine Transaktion keine Sperren mehr anfordert, nachdem sie bereits andere Sperren wieder freigegeben hat, können wir per Transitivität schlussfolgern, dass T_i eine Sperre freigeben muss, ehe T_k eine Sperre erhalten kann. Per Induktion kann man dann folgern, dass, wenn in $SG(H)$ der Pfad $T_1 \to T_2 \to \ldots T_n$ auftritt, T_1 zuerst eine Sperre freigeben muss, ehe T_n eine Sperre erhält.

   Anzumerken ist, dass sich diese Sperren nicht immer auf dasselbe Datum beziehen müssen. T_i und T_j stehen im Allgemeinen bezüglich eines anderen Datums in Abhängigkeit als T_j und T_k. Beispielsweise kann die Historie

$$r_i(A) \to w_j(A) \to w_j(B) \to r_k(B)$$

   zum Pfad $T_i \to T_j \to T_k$ in $SG(H)$ führen.

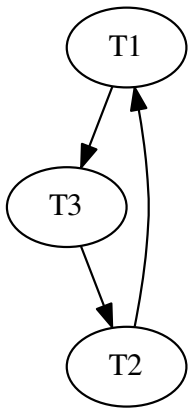
3. Damit ist es ein Leichtes, zu zeigen, dass $SG(H)$ azyklisch ist. Nehmen wir an, in $SG(H)$ tritt der Zyklus $T_1 \to T_2 \to \ldots T_n \to T_1$ auf. Nach Schritt 2.2 muss dann T_1 eine Sperre freigegeben haben, ehe es eine andere Sperre erhält. Dies widerspricht jedoch Eigenschaft (4) des 2PL-Protokolls, bzw. 1.3. Da $SG(H)$ zyklenfrei ist, können wir nach dem Serialisierbarkeitstheorem folgern, dass H serialisierbar ist.

Aufgabe 4

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.

Wartegraph dann



Aufgabe 5

Nennen Sie die Vorteile und Nachteile von Deadlockerkennung / Vermeidung durch:

- Timeouts
- Wartegraphen
- Preclaiming
- Zeitstempel

Sind Kombinationen denkbar/sinnvoll?

- Timeouts
  - + Verfahren ist einfach und billig
  - - Deadlocks werden erst mit Verzögerung erkannt
  - - Es gibt false positives
- Wartegraphen
  - + Es werden echte Deadlocks schnell erkannt
  - - Hohe kosten
- Preclaiming
  - + Deadlocks treten nicht auf
  - - Verringert Parallelität
- Zeitstempel
  - + Deadlocks treten nicht auf
  - - Viele Transaktionen müssen zurückgesetzt werden, obwohl nie ein Deadlock auftreten würde – false positives.

Beispielsweise kann Timeout und Wartegraph kombiniert werden. Hierbei wird der Wartegraph erst 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 „on demand“ erzeugt wird. Das Problem der verzögerten Erkennung von Deadlocks bleibt allerdings bestehen.

Aufgabe 6

Beim „multiple-granularity locking“ (MGL) werden Sperren von oben nach unten (top-down) in der Datenhierarchie erworben. Zeigen Sie mögliche Fehlerzustände, die eintreten könnten, wenn man die Sperren in umgekehrter Reihenfolge (also bottom-up) setzen würde.

Gruppenaufgabe (Muss nicht zuhause vorbereitet werden.)

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.