UNIVERSITÄT DES SAARLANDES
Fachrichtung 6.2 - Informatik
Lehrstuhl Prof. Weikum

Informationssysteme SS 2002

Übung 9 Musterlösung

Aufgabe 1: Serialisierbarkeit I

Gegeben ist der folgende Schedule der Transaktionen T1, T2, T3, T4, T5:

R1(a) R2(b) W1(b) W3(c) W2(c) W2(d) R4(a) W4(c) W5(c) W5(d) W4(a)

Ist der Schedule serialisierbar? Falls ja, geben Sie mindestens einen äquivalenten seriellen Schedule an. Falls nein, warum nicht?

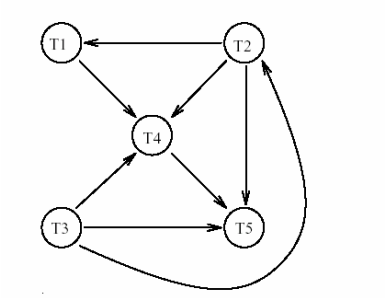
Der Schedule ist serialisierbar. Konfliktpaare des Schedules:

<R2(b), W1(b)>, <R1(a), W4(a)>, <W3(c), W2(c)>,

<W2(c), W4(c)>, <W4(c), W5(c)>,<W2(d), W5(d)>,

<W3(c), W4(c)>, <W3(c), W5(c)>,<W2(c), W5(c)>

Der dazugehörige Abhängigkeitsgraph des Schedules sieht folgendermaßen aus und ist azyklisch.



Durch die topologische Sortierung erhält man den äquivalenten seriellen Schedule :

T3 T2 T1 T4 T5.

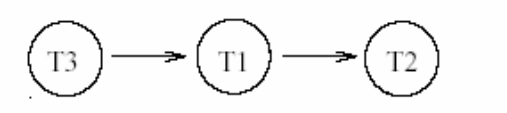
Aufgabe 2: Serialisierbarkeit II

Gegeben ist der folgende Schedule der Transaktionen T1, T2, T3:

R1(a) R2(a) W2(a) R3(b) R2(c) W3(b) R1(c) R1(b)

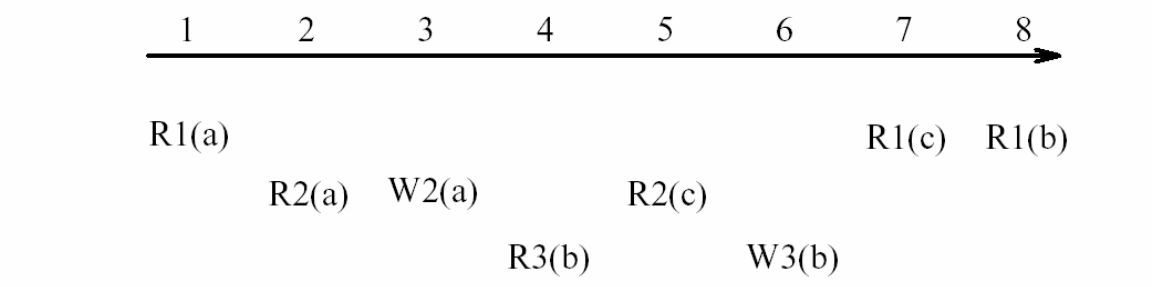
a) Ist dieser Schedule (konflikt-) serialisierbar? Falls ja, zu welchem seriellen Schedule ist er äquivalent? Begründen Sie Ihre Antwort!

Der Schedule ist serialisierbar.
Koniktpaare des Schedules: <R1(a), W2(a)>, <W3(b), R1(b)>
Der dazugehörige Abhängigkeitsgraph des Schedules sieht folgendermaßen aus und ist azyklisch. Durch die topologische Sortierung erhält man den äquivalenten seriellen Schedule : T3 T1 T2.



b) Kann dieser Schedule bei Verwendung des Zweiphasen-Sperrprotokolls entstanden sein? Kann er bei Verwendung des strikten Zweiphasen-Sperrprotokolls entstanden sein? Begründen Sie Ihre Antwort!

Der Schedule kann nicht durch das Zweiphasen-Sperrprotokoll entstanden sein, da:



1. Transaktion T1 muß spätestens zum Zeitpunkt 1 die Seite a sperren (d.h. $L1(a) \leq 1$).
2. Transaktion T2 muß spätestens zum Zeitpunkt 2 die Seite a sperren (d.h. $L2(a) \leq 2 \Rightarrow U1(a) < 2$).
3. Transaktion T1 muß zum Zeitpunkt 8 eine Sperre auf der Seite b halten (d.h. $U1(b) > 8$).
4. Transaktion T3 muß die Sperre auf Seite b mindestens bis zum Zeitpunkt 6 halten (d.h. $U3(b) > 6 \Rightarrow L1(b) \geq 6$).

Aus (1) und (2) folgt, daß die Schrumpfungsphase von Transaktion T1 vor dem Zeitpunkt
2 beginnen muß, während aus (3) und (4) folgt, daß die Wachstumsphase über den
Zeitpunkt 7 hinaus andauern muß.
$\Rightarrow$ Widerspruch

Aufgabe 3: Sperrprotokoll

Betrachten Sie das folgende Sperrprotokoll:

- Für Transaktionen, die mindestens eine Änderungsoperation beinhalten, wird das strikte 2PL verwendet[1].
- Für Transaktionen, die nur Leseoperationen beinhalten, wird ein nicht zweiphasiges Sperrprotokoll wie folgt verwendet:
  - Vor dem Zugriff auf ein Objekt muss dieses gesperrt werden.
  - Alle Sperren einer Transaktion werden jeweils bei Beendigung einer SQL-Anweisung freigegeben. (Bei der nächsten SQL-Anweisung werden wieder neue Sperren angefordert.)

a) Zeigen Sie, dass bei Verwendung dieses Protokolls nichtserialisierbare Schedules entstehen können.

Schedule S: R1(x) W2(x) W2(y) R1(y) ist nicht serialisierbar und kann jedoch bei Verwendung des gegebenen Protokolls entstanden sein:

b) Kann bei Verwendung des angegebenen Sperrprotokolls eine Integritätsbedingung der Datenbank verletzt werden? Falls nein, ist der „Verlust“ der Serialisierbarkeit überhaupt gravierend? Falls ja, geben Sie ein Beispiel an.

Nein, da alle Änderungstransaktionen das strikte 2PL-Protokoll verwenden. Allerdings können Lesetransaktionen scheinbare Verletzungen der Integrität "beobachten" (inkonsistentes Lesen).
Betrachten wir die folgende Situation: In der Datenbank ist eine Integritätsbedingung gegeben, welche verlangt, dass bei Geld-Transfer zwischen Konto x und Konto y die Summe von x und y konstant bleibt, beispielsweise x+y=1200. Seien x=1000 DM und y=200 DM. Schedule S: R1(x) W2(x) W2(y) R1(y) beinhaltet zwei Transaktionen, wobei Transaktion T2 eine bestimmte Menge von Geld, beispielsweise 400 DM, von Konto x nach Konto y transferiert und Transaktion T1 die Summe von Konto x und Konto y prüft. Anhand des folgenden Ablaufs ist leicht zu sehen, daß die Integritätsbedingung für T1 verletzt ist, da die Summe von x und y 1600 anstatt 1200 ist.

T1	T2
x=1000	y=200
R1(x) → x=1000	
	W2(x) → x=x-400 =600
	W2(y) → y=y+400 =600
R1(y) → y=600	
x+y=1600≠ Konstant 1200	
x=600	y=600

[1] Dieses Protokoll entspricht dem TRANSACTION ISOLATION LEVEL READ COMMITED des SQL-Standards.

c) Geben Sie ein Anwendungsbeispiel an, bei dem das angegebene Sperrprotokoll sinnvoll ist.

Das angegebene Sperrprotokoll kann z.B. für statistische Auswertungen sinnvoll sein, die nicht unbedingt eine hohe Anforderung an die Genauigkeit stellen, jedoch sehr lange laufen.
Betrachten wir das folgende Anwendungsbeispiel: In einem großen Lagerhaus wird eine wöchentliche Statistik erstellt. Diese Statistik gibt eine Übersicht, welche und wieviele Artikel in einer Woche verkauft worden sind, um den Markt-Trend festzustellen, weshalb die Angaben über die einzelnen Artikel nicht sehr genau zu sein brauchen. Die Erstellung dieser Statistik dauert sehr lange, u.U. mehrere Stunden. Parallel zur Erstellung dieser Statistik laufen die normalen betrieblichen Buchungsprogramme und sollen möglichst wenig behindert werden.
Für das oben beschriebene Szenario ist das angegebene Protokoll sehr sinnvoll, da ansonsten für die Erstellung der Statistik alle Artikel gesperrt würden und erst nach Beenden der Erstellung freigegeben würden. Alle anderen Buchungsprogramme würden blockiert und müssten sehr lange warten.

Aufgabe 4: Recovery

Gegeben sei der in der in Spalte 1 von Tabelle 1 auf dem Deckblatt aufgeführte chronologische Ablauf von Aktionen. Der Checkpoint, der in diesem Ablauf ausgeführt wird, ist ein asynchroner. Tragen Sie bitte Ihre Lösungen der Teilaufgaben a) bis c) direkt in die vorgegebenen Tabellen ein, und geben Sie die Tabellen mit ab.

a) Geben Sie die vom Log-Manager zu protokollierende Information durch Ausfüllen der Spalten 4 und 5 von Tabelle 1 sowie die Änderungen von Seitenheader-LSNs durch Ausfüllen der Spalten 2 und 3 an. Verwenden Sie die Nummerierung der Aktionen für die LSNs.

   Verwenden Sie bei dieser und allen folgenden Teilaufgaben bitte die folgende Notation: Tragen Sie in den Spalten 2 und 3 jeweils die betroffene(n) Seite(n) mit ihre(n) LSN ein, z.B. `p(17)` bedeutet Seite p mit LSN 17. Tragen Sie in den Spalten 4 die LSN des Logsatzes, die betroffene Seite und die Transaktion, die die Änderung ausgeführt hat, ein, also z.B. `17(p,T1)`. Commit-Logsätze haben keinen Seiteneintrag, z.B. `24(T1)` für das Commit von T1 mit LSN 24. In Spalte 5 genügt es, wenn Sie die LSN's der Logeinträge eintragen, die in die Logdatei geschrieben werden.

Nr	Aktion	Seitenpuffer [Seite (LSN)]	Datenbank [Seite (LSN)]	Logpuffer [LSN (Seite/Trans.)]	Logdatei [LSN's]
1	BOT(T1)		*a(0),b(0),...*		
2	BOT(T2)				
3	W1(a)	*a(3)*		*3(a/T1)*	
4	BOT(T3)				
5	BOT(T4)				
6	W3(b)	b(6)		6(b/T3)	
7	W2(c)	c(7)		7(c/T2)	
8	W1(d)	d(8)		8(d/T1)	
9	EOT(T1)			9(T1)	3,6,7,8,9
10	Flush(d)		d(8)		
11	W3(d)	d(11)		11(d/T3)	
12	BOT(T5)				
13	W5(a)	a(13)		13(a/T5)	
14	Checkpoint	ActiveTrans [Trans(LastLSN)]: T2,T3,T4,T5 DirtyPages [Seite(RedoLSN)]: a(3), b(6), c(7), d(11)			11,13, 14:CP({T2,T3,T4,T5}, {a(3),b(6),c(7),d(11)})
15	EOT(T3)			15(T3)	15
16	Flush(d)		d(11)		
17	W4(d)	d(17)		17(d/T4)	
18	W2(e)	e(18)		18(e/T2)	
19	W5(b)	b(19)		19(b/T5)	
20	Flush(b)		b(19)		17,18,19
21	EOT(T4)			21(T4)	21
22	W5(f)	f(22)		22(f/T5)	
☠ 1. Systemfehler ☠					

b) Nehmen Sie an, der Datenbank-Server falle nach der letzten der angegebenen Aktionen aus. Geben Sie an, welche Aktionen zur Recovery ausgeführt werden müssen, indem Sie Tabelle 2 auf dem Deckblatt ausfüllen. Verwenden Sie dazu die Notation `redo(n)`, um die Aktion mit LSN n zu wiederholen, und `undo(m)`, um die Aktion mit LSN m rückgängig zu machen. Berücksichtigen Sie auch Idempotenztests für Redo- und Undo-Schritte; Schritte, die aufgrund des Idempotenztests unterdrückt werden, werden mit "`consider-redo(<LSN>)`" bzw. "`consider-undo(<LSN>)`" in der Tabelle vermerkt. Zur Verkürzung der Redo-Phase im Falle eines erneuten Absturzes schreibt das Datenbanksystem alle im Cache befindlichen Seiten nach Abschluss der Redo-Phase, aber vor Beginn der Undo-Phase in die Datenbank; nehmen Sie die dazu notwendigen Aktionen in der „Flush-Phase" in Tabelle 2 auf.

Nr	Aktion	Seitenpuffer [Seite (LSN)]	Datenbank [Seite (LSN)]	Logpuffer [LSN (Seite/Trans.)]	Logdatei [LSN's]
☠ 1. Systemfehler ☠					
	Analysephase	Verlierer [Trans(LastLSN)]: T2, T5		DirtyPages [Seite(RedoLSN)]: a(3),b(6),c(7),d(11),e(18)	
	Redo-Phase				
23	redo(3)	a(3)			
24	consider_redo(6)				
25	consider_redo(8)				
26	consider_redo(11)				
27	redo(17)	d(17)			
	Flush der DB				
28	flush(a)		a(3)		
29	flush(d)		d(17)		
	Undo-Phase				
30	undo(19)	b(6)			
	consider_undo(18)				
	consider_undo(13)				
	consider_undo(7)				
☠ 2. Systemfehler ☠					

c) Nehmen Sie an, dass während der Undo-Phase keinerlei Seiten mehr in die Datenbank zurückgeschrieben werden. Nach dem letzten Undo-Schritt falle der Datenbank-Server erneut aus, so dass nach dem Wiederanlauf eine zweite Recovery-Runde benötigt wird. Geben Sie durch Ausfüllen von Tabelle 3 an, welche Schritte bei dieser zweiten Recovery-Runde durchgeführt werden.

Nr	Aktion	Seitenpuffer [Seite (LSN)]	Datenbank [Seite (LSN)]	Logpuffer [LSN (Seite/Trans.)]	Logdatei [LSN's]
	☠ 2. Systemfehler ☠				
	Analysephase	Verlierer [Trans(LastLSN)]: T2, T5		DirtyPages [Seite(RedoLSN)]: a(3),b(6),c(7),d(11),e(18)	
	Redo-Phase				
	consider_redo(3)				
	consider_redo(6)				
	consider_redo(11)				
	consider_redo(17)				
	Flush der DB				
	Undo-Phase				
	undo(19)	b(6)			
	consider_undo(18)				
	consider_undo(13)				
	consider_undo(7)				

d) *Bonusaufgabe:* Nehmen Sie an, der Log-Manager des Datenbank-Servers protokolliere zusätzlich auch das Zurückschreiben von Seiten aus dem Cache in die Datenbank in Form von "`flush(<Seitennummer>)`"-Logsätzen. Nehmen Sie ferner an, dass alle erzeugten flush-Logsätze vor dem ersten bzw. zweiten Crash vom Logpuffer in die Logdatei geschrieben seien. Während der jeweiligen Analysephase der beiden Recovery-Runden soll dann die Dirty-Pages-Liste so weit wie möglich aktualisiert werden. Wie sieht die Dirty-Pages-Liste nach der ersten bzw. zweiten Analysephase aus, und wie ändern sich dann die während der ersten bzw. zweiten Recovery-Runde durchzuführenden Aktionen?

Nach der 1. Analysephase:

a(3),c(7),d(17),e(18)

Nach der 2. Analysephase:

c(7),e(18)

Aktionen in der ersten Recovery-Runde:

keine Änderung bei redo, keine Änderung bei undo

Aktionen in der zweiten Recovery-Runde:

kein Redo mehr erforderlich, keine Änderung bei undo

Aufgabe 5: Recovery & Concurrency Control

Warum kann man bei Verwendung von Tupelsperren als Concurrency–Control–Protokoll die Recovery nicht einfach mittels Before- und After-Images von Seiten realisieren? Könnte man wenigstens nur die Undo-Phase mittels Seiten-Before-Images durchführen oder nur die Redo-Phase mittels Seiten-After-Images?
Welche Vorteile hätten diese eingeschränkten Varianten gegenüber einer Recovery, die sowohl bezüglich Undo als auch Redo auf tupelorientierten Operationen basiert? Geben Sie ggf. Beispiele an, die Ihre Überlegungen veranschaulichen.

Betrachten wir die Seite *X*, die die beiden Tupel *p* und *q* enthält. Vor dem Crash der Datenbank seien u.a. die folgenden Aktionen durchgeführt worden:

T1:	*W1(p)*			|
T2:		*W2(q)*	*EOT*	| **CRASH!**

Während der Redo–Phase wird die Seite *X* so geschrieben, wie sie nach *W2(q)* ausgesehen hat, da *T2* eine Gewinnertransaktion ist. Führen wir jetzt ein seitenorientiertes Undo aus, so wird die Seite *X* wieder überschrieben und zwar mit dem Before–Image von *W1(p)*. Dadurch wird jedoch die Änderung von Transaktion *T2* wieder rückgängig gemacht.
Die Undo–Phase kann bei Tupelsperren nie durch Seiten–Images ausgeführt werden, da entweder bereits erfolgte Änderungen der Redo–Phase wieder rückgängig gemacht werden (Undo–Phase nach Redo–Phase), oder es können nach der Undo–Phase inkonsistente Datenbankzustände existieren, die es unmöglich machen anschließend ein tupelorientiertes Redo zu machen. Betrachten wir z.B. folgende Transaktionen:

T1:	*W1(p)*		*W1(p)* *EOT*	|
T2:		*W2(q)*		| **CRASH!**

Nach dem seitenorientierten Undo sieht die Seite *X* aus wie zwischen dem ersten *W1(p)* und *W2(q)*. Da einzelne Tupel keine LSN besitzen, ist es in der Redo–Phase für Transaktion *T1* unmöglich festzustellen, welche Aktionen wiederholt werden müssen und welche nicht.
Im Gegensatz zum Undo ist es jedoch möglich ein seitenorientiertes Redo für Gewinnertransaktionen mit anschließendem tupelorientierten Undo durchzuführen.