AG Datenbanken und Informationssysteme

Wintersemester 2006 / 2007

Prof. Dr.-Ing. Dr. h. c. Theo Härder
Fachbereich Informatik
Technische Universität Kaiserslautern

http://wwwdvs.informatik.uni-kl.de

10. Übungsblatt

Für die Übung am Donnerstag, **18. Januar 2007**,
von 15:30 bis 17:00 Uhr in 13/222.

Aufgabe 1: Parallele Transaktionen auf unterschiedlichen Konsistenzebenen

In einem DBS werden hierarchische Sperrprotokolle für Datenbank, Segment, Relation und Tupel eingesetzt. Eine Relation R_{11}, die Konten beschreibt, bestehe aus folgenden 5 Tupeln mit den zugehörigen Kontenbeständen:

t_{111} mit 100 Euro
t_{112} mit 200 Euro
t_{113} mit 300 Euro
t_{114} mit 400 Euro
t_{115} mit 500 Euro

Transaktion T_1 überweist 100 Euro von t_{114} nach t_{111}, addiert 100 Euro auf t_{112} und zieht 100 Euro von t_{113} ab. Parallel dazu liest Transaktion T_2 sequentiell alle Kontenstände und bildet die Gesamtsumme.

a) T_1 läuft auf Konsistenzebene 3 und sperrt R_{11} exklusiv. T_2 läuft auf Konsistenzebene 1. Jeder Verarbeitungsschritt beträgt eine Zeiteinheit. Es werden nur die Aktionen mit dem Datenbanksystem als Verarbeitungsschritte berücksichtigt, dabei genügen die Operationen LOCK, UNLOCK, READ, MODIFY.

   (1) Welches Ergebnis erhält T_2, wenn sie zeitgleich mit T_1 startet?

   (2) Welches Ergebnis erhält T_2, wenn sie im 5. Verarbeitungsschritt, bzw. 8. Verarbeitungsschritt von T_1 startet?

b) T_1 und T_2 benutzen zur Erledigung der gleichen Aufgabe Tupelsperren. T_1 läuft wieder auf Konsistenzebene 3, T_2 diesmal auf Konsistenzebene 2. Kurze Lesesperren bedeuten, dass die Operationen LOCK, READ und UNLOCK (wenn möglich) direkt in drei aufeinander folgenden Verarbeitungsschritten durchgeführt werden.

   (1) Beide Transaktionen starten zur gleichen Zeit. Nach wie vielen Zeiteinheiten endet T_2? Welche Gesamtsumme wird ermittelt?

   (2) Was passiert, wenn T_2 ein Sperrprotokoll der Konsistenzebene 3 einhält?

Lösung:

a) T_1 läuft auf Konsistenzebene 3 und sperrt R_{11} exklusiv. T_2 läuft auf Konsistenzebene 1. Jeder Verarbeitungsschritt beträgt eine Zeiteinheit. Es werden nur die Aktionen mit dem Datenbanksystem als Verarbeitungsschritte berücksichtigt, dabei genügen die Operatoren LOCK, UNLOCK, READ, MODIFY.

(1) Welches Ergebnis erhält T_2, wenn sie zeitgleich mit T_1 startet?

(2) Welches Ergebnis erhält T_2, wenn sie im 5. Verarbeitungsschritt, bzw. 8. Verarbeitungsschritt von T_1 startet?

Zeit	T1 in CL3	T2 in CL1	T2 in CL1	T2 in CL1
1	LOCK DB, IX	READ t_{111}, 100		
2	LOCK S_1, IX	READ t_{112}, 200		
3	LOCK R_{11}, X	READ t_{113}, 300		
4	READ t_{114}	READ t_{114}, 400		
5	MODIFY t_{114} (-100)	READ t_{115}, 500	READ t_{111}, 100	
6	READ t_{111}		READ t_{112}, 200	
7	MODIFY t_{111} (+100)		READ t_{113}, 300	
8	READ t_{112}		READ t_{114}, **300**	READ t_{111}, **200**
9	MODIFY t_{112} (+100)		READ t_{115}, 500	READ t_{112}, **300**
10	READ t_{113}			READ t_{113}, 300
11	MODIFY t_{113} (-100)			READ t_{114}, **300**
12	UNLOCK R_{11}			READ t_{115}, 500
13	UNLOCK S_1			
14	UNLOCK DB			
		GS = 1500	GS = 1400	GS = 1600

b) T_1 und T_2 benutzen zur Erledigung der gleichen Aufgabe Tupelsperren. T_1 läuft wieder auf Konsistenzebene 3, T_2 diesmal auf Konsistenzebene 2. Kurze Lesesperren bedeuten, dass die Operationen LOCK, READ und UNLOCK (wenn möglich) direkt in drei aufeinander folgenden Verarbeitungsschritten durchgeführt werden.

(1) Beide Transaktionen starten zur gleichen Zeit. Nach wie vielen Zeiteinheiten endet T_2? Welche Gesamtsumme wird ermittelt?

(2) Was passiert, wenn T_2 ein Sperrprotokoll der Konsistenzebene 3 einhält?

Zeit	T1 in CL3	T2 in CL2
1	LOCK DB, IX	LOCK DB, IR
2	LOCK S_1, IX	LOCK S_1, IR
3	LOCK R_{11}, IX	LOCK R_{11}, IR
4	LOCK t_{114}, X	LOCK t_{111}, R
5	READ t_{114}	READ t_{111}, 100
6	MODIFY t_{114} (-100)	UNLOCK t_{111}
7	LOCK t_{111}, X	LOCK t_{112}, R
8	READ t_{111}	READ t_{112}, 200
9	MODIFY t_{111} (+100)	UNLOCK t_{112}
10	LOCK t_{112}, X	LOCK t_{113}, R
11	READ t_{112}	READ t_{113}, 300
12	MODIFY t_{112} (+100)	UNLOCK t_{113}
13	LOCK t_{113}, X	LOCK t_{114}, R
14	READ t_{113}	warten
15	MODIFY t_{113} (-100)	warten
16	UNLOCK t_{113}	warten
17	UNLOCK t_{112}	warten
18	UNLOCK t_{111}	warten
19	UNLOCK t_{114}	warten
20	UNLOCK R_{11}	READ t_{114}, **300**
21	UNLOCK S_1	UNLOCK t_{114}
22	UNLOCK DB	LOCK t_{115}, R
23		READ t_{115}, 500
24		UNLOCK t_{115}
25		UNLOCK R_{11}
26		UNLOCK S_1
27		UNLOCK DB
		GS=1400

Zeit	T1 in CL3	T2 in CL3
1	LOCK DB, IX	LOCK DB, IR
2	LOCK S_1, IX	LOCK S_1, IR
3	LOCK R_{11}, IR	LOCK R_{11}, IR
4	LOCK t_{114}, X	LOCK t_{111}, R
5	READ t_{114}	READ t_{111}, 100
6	MODIFY t_{114} (-100)	LOCK t_{112}, R
7	LOCK t_{111}, X	READ t_{112}, 200
8	warten	LOCK t_{113}, R
9	warten	READ t_{113}, 300
10	warten	LOCK t_{114}, R
11	DEADLOCK	DEADLOCK

Aufgabe 2: Escrow-Ansatz

Die Anzahl von freien Sitzplätzen in einem Flugzeug soll über ein Escrow-Feld mit LO=0 und HI=165 verwaltet werden. Eine Transaktion kann Plätze reservieren (negativer Wert in Klammern) oder zurückgeben (positiver Wert). Der Anfangszustand (INF, Q, SUP) sei (10, 10, 10), d.h. es sind noch 10 Plätze frei.

a) Bestimmen Sie für folgende Anforderungen/Rückgaben der Transaktionen T_1 bis T_5 das Werteintervall des Escrow-Feldes:
$T_1(-3)$, $T_2(+3)$, $T_3(+2)$, $T_4(-3)$, T_1.commit, T_3.rollback, $T_5(-3)$, T_4.commit, T_2.commit, T_5.rollback

b) Wieviele Sitzplätze sind nach dem Commit von T_1 noch frei? Warum kann die Anzahl nicht exakt angegeben werden?

Lösung:

a) Bestimmen Sie für folgende Anforderungen/Rückgaben der Transaktionen T_1 bis T_5 das Werteintervall des Escrow-Feldes:
$T_1(-3)$, $T_2(+3)$, $T_3(+2)$, $T_4(-3)$, T_1.commit, T_3.rollback, $T_5(-3)$, T_4.commit, T_2.commit, T_5.rollback

T1	T2	T3	T4	T5	INF	Q	SUP
					10	10	10
-3					7	7	10
	+3				7	10	13
		+2			7	12	15
			-3		4	9	15
commit					4	9	12
		rollback			4	7	10
				-3	1	4	10
			commit		1	4	7
	commit				4	4	7
				rollback	7	7	7

b) Wieviele Sitzplätze sind nach dem Commit von T_1 noch frei? Warum kann die Anzahl nicht exakt angegeben werden?

Die Anzahl kann nicht exakt angegeben werden, da die Anzahl der freien Sitzplätze von mehreren Transaktionen verändert wurde, die noch nicht abgeschlossen sind (d.h. deren Änderungen evtl. auch wieder rückgängig gemacht werden müssen). Je nach Ausgang der noch aktiven Transaktionen stehen nach dem Commit von T_1 noch zwischen 4 und 12 Sitzplätzen zur Verfügung.

Aufgabe 3: Präzisionssperren

Gegeben sei folgendes DB-Schema:

Angestellte: PERS (PNR, NAME, GEHALT, BERUF, ANR, MNR, ORT)
Abteilung: ABT (ANR, ANAME, AORT)

PERS.ANR ist Fremdschlüssel auf ABT.ANR, PERS.MNR ist Fremdschlüssel auf PERS.MNR

Führen Sie die Prädikatliste und die Update-Liste, die beide zu Anfang leer sind, für die folgenden SQL-Anweisungen der Transaktionen T1-T5:

```
T1: SELECT * FROM PERS WHERE ORT="Kaiserslautern"
T2: SELECT * FROM PERS WHERE GEHALT >= 0
T3: INSERT INTO PERS VALUES (4711, "Maier", 45000, "Programmierer",
    P11, 0815, "Kaiserslautern")
T4: INSERT INTO ABT VALUES (P12, "Informationsintegration",
    "Karlsruhe")
T5: UPDATE PERS SET ORT="Kaiserslautern" WHERE PNR = 4711
T6: UPDATE PERS SET ORT="Karlsruhe" WHERE PNR = 4712
T7: SELECT * FROM ABT WHERE ANAME="Buchhaltung" OR AORT="Karlsruhe"
```

Welche Operationen werden ausgeführt, welche werden gesperrt?

Lösung:

T1: kein Konflikt

Prädikatsliste: (PERS, ORT = "Kaiserslautern")

Update-Liste: -

T2: kein Konflikt

Prädikatsliste: (PERS, ORT = "Kaiserslautern")
(PERS, GEHALT >= 0)

Update-Liste: -

T3: Konflikt! einzufügendes Tupel erfüllt beide Prädikate

T4: kein Konflikt
Prädikatsliste: (PERS, ORT = "Kaiserslautern")
(PERS, GEHALT >= 0)

Update-Liste: (P12, "Informationsintegration", "Karlsruhe")

T5: Konflikt! Nach Update erfüllt Tupel das erste Prädikat, zudem erfüllt es (mit hoher Wahrscheinlichkeit ;-)) das zweite Prädikat

T6: Konflikt! Tupel erfüllt (mit hoher Wahrscheinlichkeit) das zweite Prädikat
zusätzlich: auch ohne T2 Konflikt möglich, wenn vor dem Update ORT = "Kaiserslautern"!

T7: Konflikt! Tupel in der Update-Liste erfüllt das Prädikat

Aufgabe 4: Synchronisation mit Mehrversionen-Verfahren

In einer Datenbank, die das in der Vorlesung vorgestellte Mehrversionen-Verfahren zur Synchronisation einsetzt, werden vier Schreibe- und fünf Lesetransaktionen nebenläufig ausgeführt, die auf die Datenbankobjekte A und B zugreifen. Alle Sperren werden bis zum Ende der Transaktion gehalten und dort atomar freigegeben (starkes 2PL). Die fünf Lesetransaktionen sind als mehrfache Ausführung des gleichen Transaktionsprogramms aufzufassen.

T_{x1}: BOT (t=0), [1], r(A), [4], w(A), [3], EOT

T_{x2}: BOT (t=2), [2], r(B), [5], w(B), [4], r(A), [2], w(A), [3], EOT

T_{x3}: BOT (t=3), [4], r(A), [2], w(A), [1], EOT

T_{x4}: BOT (t=14), [2], r(A), [2], w(A), [2], r(B), [2], w(B), [1], EOT

T_{r1}-T_{r5}: BOT, [6], r(A), [6], r(B), [6], EOT

Die Transaktionen T_{r1}-T_{r5} starten zu den Zeitpunkten t=0, t=9, t=12, t=19 bzw. t=26.

Hinweis: Die in eckigen Klammern angegebenen Zahlen geben die Verzögerung (durch interne Aktivitäten/Berechnungen der TA, Nutzerinteraktion etc.) zwischen der *tatsächlichen Ausführung* der davor stehenden und der *versuchten Ausführung* der danach stehenden Operation an. Diese Verzögerung ist unabhängig von eventuellen Blockierungen/Wartezeiten der Transaktionen.

Synchronisieren Sie die Transaktionen mit dem in der Vorlesung vorgestellten Mehrversionen-Verfahren, d.h. ermitteln Sie den mit diesem Verfahren erzielten Schedule. Vermerken Sie dazu jede Änderung der vom System verwalteten Versionen der Objekte A und B. Welche Versionen der Objekte werden von den Transaktionen jeweils gelesen oder geschrieben?

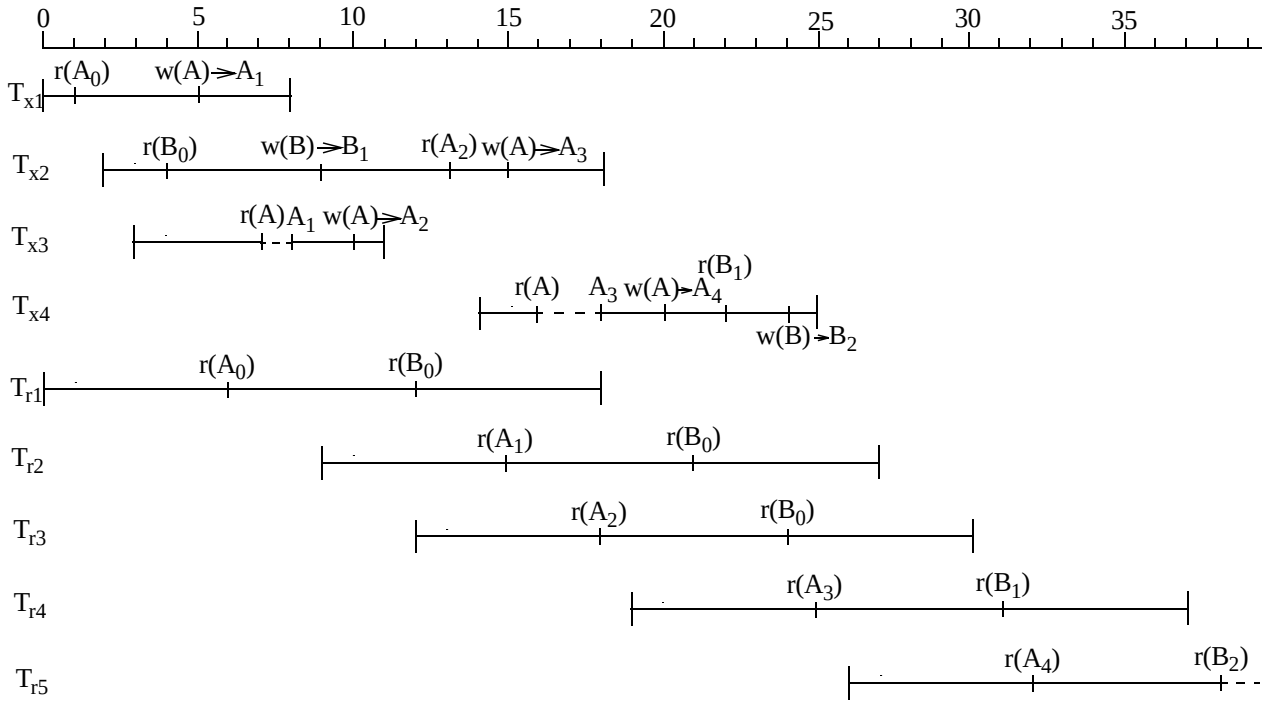
Welche Serialisierbarkeitsreihenfolgen ergeben sich?

Lösung:

Beim Mehrversionen-Verfahren werden Schreibe-Transaktionen mit einem herkömmlichen Sperrverfahren (hier: RX) synchronisiert, da mehrere parallele Schreibtransaktionen auf einem Objekt weiterhin nicht möglich sind. Reine Lesetransaktionen setzen keine Sperren. Für sie wird durch die Versionierung sichergestellt, dass sie jedes Objekt im zum Zeitpunkt ihres BOT gültigen Zustand lesen können. Dies führt dazu, dass von jedem geänderten Objekt alle Versionen aufgehoben werden müssen, die zum Zeitpunkt des BOT eines beliebigen noch aktiven Lesers gültig waren. Dies schließt insbesondere auch die Objekte ein, die von de(r/n) Lesetransaktion(en) (noch) nicht gelesen wurden, aber potentiell noch gelesen werden könnten.

Sind in einem System lange Lesetransaktionen und zahlreiche kurze Schreibtransaktionen aktiv, müssen folglich unter Umständen sehr viele Versionen der geänderten Objekte aufbewahrt werden. Kommt es dadurch zu einem Überlauf des für den Versionenpool vorgesehen Speichers, müssen die langen Leser zurückgesetzt werden, wodurch sehr viel Arbeit verlorengehen kann.

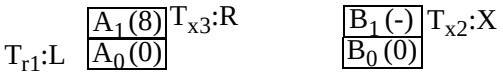
Zeitlicher Ablauf der Transaktionen:



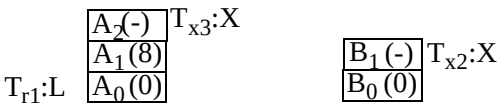
Erklärung und Zustand des Versionenpools:

t=0: BOT von T_{x1} und T_{r1}		$A_0(0)$	$B_0(0)$
t=1: r(A) von T_{x1}, erhält R-Sperre auf A_0		$A_0(0)$ T_{x1}:R	$B_0(0)$
t=2: BOT von T_{x2}		$A_0(0)$ T_{x1}:R	$B_0(0)$
t=3: BOT von T_{x3}		$A_0(0)$ T_{x1}:R	$B_0(0)$
t=4: r(B) von T_{x2}, erhält R-Sperre auf B_0		$A_0(0)$ T_{x1}:R	$B_0(0)$ T_{x2}:R
t=5: w(A) von T_{x1}, Sperrkonversion, Erzeugung einer neuen Version A_1, X-Sperre gesetzt		$A_1(-)$ T_{x1}:X $A_0(0)$	$B_0(0)$ T_{x2}:R
t=6: r(A) von T_{r1}, sieht Version A_0	T_{r1}:L	$A_1(-)$ T_{x1}:X $A_0(0)$	$B_0(0)$ T_{x2}:R
t=7: r(A) von T_{x3}, fordert R an auf A, Konflikt mit X von T_{x1},T_{x3} wird blockiert	T_{r1}:L	(T_{x3}:R) $A_1(-)$ T_{x1}:X $A_0(0)$	$B_0(0)$ T_{x2}:R
t=8: EOT von T_{x1}, Freigabe der X-Sperre auf A_1, T_{x3} erhält R auf A_1 und kann weiterarbeiten	T_{r1}:L	$A_1(8)$ T_{x3}:R $A_0(0)$	$B_0(0)$ T_{x2}:R

t=9: w(B) von T_{x2}, Sperrkonversion, Erzeugung einer neuen Version B_1, X-Sperre gesetzt
BOT von T_{r2}

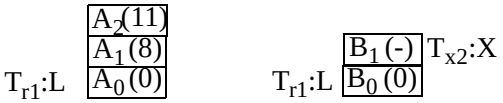


t=10: w(A) von T_{x3}, Sperrkonversion, Erzeugung einer neuen Version A_2, X-Sperre gesetzt

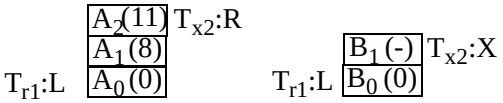


t=11: EOT von T_{x3}, Freigabe der X-Sperre auf A_2

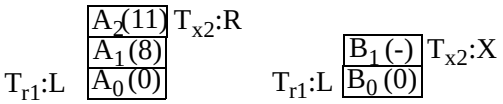
t=12: r(B) von T_{r1}, sieht Version B_0
BOT von T_{r3}



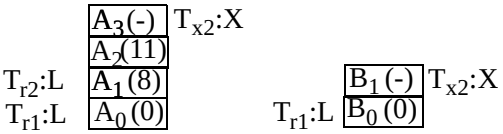
t=13: r(A) von T_{x2}, fordert und erhält R auf A, sieht Version A_2



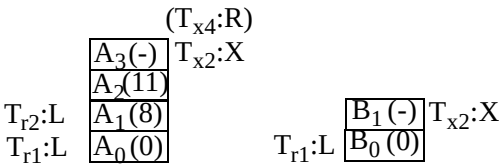
t=14: BOT von T_{x4}



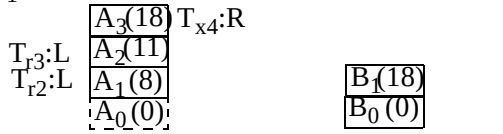
t=15: w(A) von T_{x2}, Sperrkonversion, Erzeugung einer neuen Version A_3
r(A) von T_{r2}, sieht Version A_1



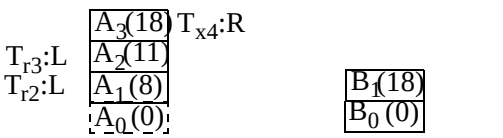
t=16: r(A) von T_{x4}, fordert R an auf A, Konflikt mit X von T_{x2},T_{x4} wird blockiert



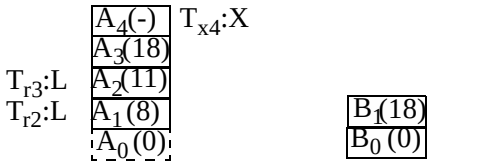
t=18: EOT von T_{x2}, Freigabe der X-Sperre auf A_3 und B_1
T_{x4} erhält R auf A_3 und kann weiterarbeiten
EOT von T_{r1}, Verwerfen der Version A_0
r(A) von T_{r3}, sieht Version A_2



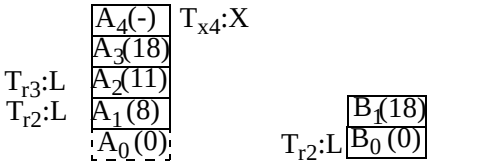
t=19: BOT von T_{r4}



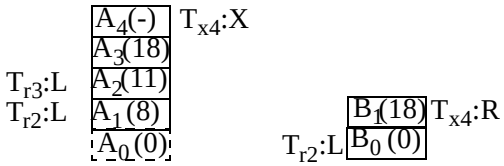
t=20: w(A) von T_{x4}, Sperrkonversion, Erzeugung einer neuen Version A_4, X-Sperre gesetzt



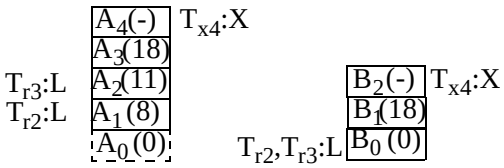
t=21: r(B) von T_{r2}, sieht Version B_0



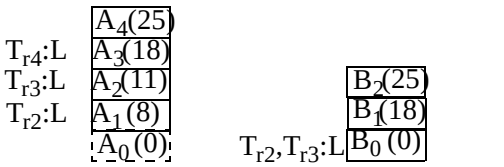
t=22: r(B) von T_{x4}, sieht Version B_1!



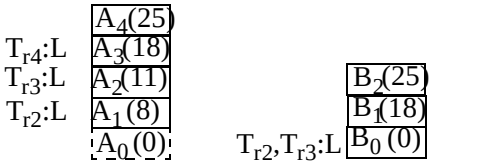
t=24: r(B) von T_{r3}, sieht Version B_0!
w(B) von T_{x4}, Sperrkonversion, Erzeugung einer neuen Version B_2, X-Sperre gesetzt



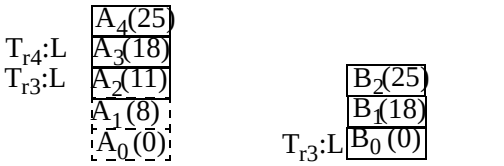
t=25: EOT von T_{x4}, Freigabe der X-Sperre auf A_4 und der X-Sperre auf B_2
r(A) von T_{r4}, sieht Version A_3!



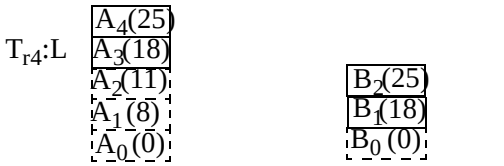
t=26: BOT von T_{r5}



t=27: EOT von T_{r2}, Verwerfen der Version A_1



t=30: EOT von T_{r3}, Verwerfen der Versionen A_2 und B_0



t=31: r(B) von T_{r4}, sieht Version B_1

T_{r4}:L $A_4(25)$ $A_3(18)$ $A_2(11)$ $A_1(8)$ $A_0(0)$

T_{r4}:L $B_2(25)$ $B_1(18)$ $B_0(0)$

t=32: r(A) von T_{r5}, sieht Version A_4

T_{r5}:L T_{r4}:L $A_4(25)$ $A_3(18)$ $A_2(11)$ $A_1(8)$ $A_0(0)$

T_{r4}:L $B_2(25)$ $B_1(18)$ $B_0(0)$

t=37: EOT von T_{r4}, Verwerfen der Versionen A_3 und B_1

T_{r5}:L $A_4(25)$ $A_3(18)$ $A_2(11)$ $A_1(8)$ $A_0(0)$

$B_2(25)$ $B_1(18)$ $B_0(0)$

t=38: r(B) von T_{r5}, sieht Version B_2

T_{r5}:L $A_4(25)$ $A_3(18)$ $A_2(11)$ $A_1(8)$ $A_0(0)$

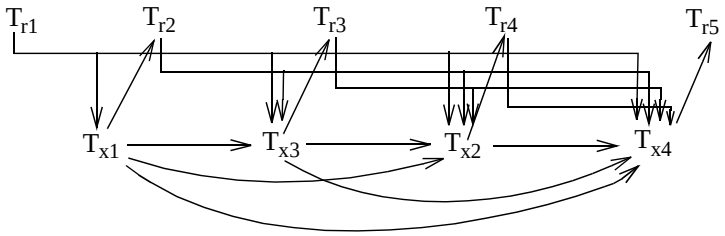
T_{r5}:L $B_2(25)$ $B_1(18)$ $B_0(0)$

t=44: EOT von T_{r5}

$A_4(25)$ $A_3(18)$ $A_2(11)$ $A_1(8)$ $A_0(0)$

$B_2(25)$ $B_1(18)$ $B_0(0)$

Serialisierbarkeitsgraph:



=> äquivalente serielle Ausführungsreihenfolge: T_{r1}, T_{x1}, T_{r2}, T_{x3}, T_{r3}, T_{x2}, T_{r4}, T_{x4}, T_{r5}

Aufgabe 5: Mehrversionen-Synchronisation

Eine Datenbank enthalte die Objekte a, b und c. Außerdem sei eine Integritätsbedingung definiert: a + b + c = konst. = 12 mit den anfänglichen Zahlenwerten a = 5, b = 4, c = 3. Die folgenden fünf Transaktionen sollen mit dem Mehrversionenverfahren synchronisiert werden:

T1: a := a - 1; b := b + 1; x1 := a + b + c;
T2: b := b - 1; c := c + 1; x2 := a + b + c;
T3: c := c - 1; a := a + 1; x3 := a + b + c;
T4: x4 := a + b + c;
T5: x5 := a + b + c;

Im unsynchronisierten Fall entstehe der folgende Ablaufplan für T1-T4. Wird dabei die Leseoperation nicht explizit für ein Datenobjekt angegeben, so erfolgt sie implizit beim ersten Zugriff.

	T1	**T2**	**T3**	**T4**
1	a - 1			
2		lesen(a)		lesen(c)
3			lesen(b)	
4	b + 1			
5		b - 1		
6			c - 1	
7	lesen(c)			lesen(b)
8		c + 1		
9			a + 1	
10	x1 := a + b + c			
11	Commit			
12				lesen(a)
13		x2 := a + b + c		x4 := a + b + c;
14		Commit		
15			x3 := a + b + c	
16			Commit	Commit

a) Welche Werte erhalten Sie für x1, x2 , x3 und x4, wenn die Leseoperationen mit R-Sperren synchronisiert werden?

b) Wie verändern sich diese Werte, wenn Sie die Leseoperationen wie Leser-Transaktionen (Leser) behandeln würden, d. h. die zum jeweiligen Transaktionsbeginn gültige Version der DB-Objekte lesen würden (ohne weitere Synchronisationsmaßnahmen)?

c) Welches Ergebnis erhält Leser T5 für x5 (und a, b, c), wenn T5 nach Commit von T1 bzw. Commit von T2 gestartet wird?

Lösung:

a) Welche Werte erhalten Sie für x1, x2 , x3 und x4, wenn die Leseoperationen mit R-Sperren synchronisiert werden?

	T1	**T2**	**T3**	**T4**
1	lock(a,X), a=a-1=4			
2		lock(a,R), warte(T1)		lock(c,R), lese c=3
3		warte(T1)	lock(b,R), lese b=4	
4	lock(b,X), warte(T3)	warte(T1)		

	T1	T2	T3	T4
5	warte(T3)	warte(T1)		
6	warte(T3)	warte(T1)	lock(c,X), warte(T4)	
7	warte(T3)	warte(T1)	warte(T4)	lock(b,R), warte(T1)
8		DEADLOCK		

b) Wie verändern sich diese Werte, wenn Sie die Leseoperationen wie Leser-Transaktionen (Leser) behandeln würden, d. h. die zum jeweiligen Transaktionsbeginn gültige Version der DB-Objekte lesen würden (ohne weitere Synchronisationsmaßnahmen)?

	T1	T2	T3	T4
1	$a_2=a_1-1=4$			
2		lese $a_1=5$		lese $c_1=3$
3			lesen $b_1=4$	
4	$b_2=b_1+1=5$			
5		warte(T1)		
6		warte(T1)	$c_2=c_1-1=2$	
7	lese $c_1=3$	warte(T1)		lese $b_1=4$
8		warte(T1)		
9		warte(T1)	warte(T1)	
10	$x_1=a_2+b_2+c_1=12$	warte(T1)	warte(T1)	
11	Commit, da $a_2+b_2+c_1=12$			
12		$b_3=b_1-1=3$	$a_3=a_1+1=6$	lese $a_1=5$
13				$x_4=a_1+b_1+c_1=12$
14				
15		warte(T3)		
16		warte(T3)		Commit
17		warte(T3)		
18		warte(T3)	$x_3=a_3+b_1+c_2=12$	
19		warte(T3)	Abort, da $a_3+b_2+c_2=13$	
20		$c_3=c_1+1=4$		
21				
22				
23				
24				
25		$x_2=a_1+b_3+c_3=12$		
26		Abort, da $a_2+b_3+c_3=11$		

c) Welches Ergebnis erhält Leser T5 für x5 (und a, b, c), wenn T5 nach Commit von T1 bzw. Commit von T2 gestartet wird?

(1) T5 nach Commit von T1: $x_5 = a_2 + b_2 + c_1 = 4 + 5 + 3 = 12$

(2) T5 nach Commit von T2: $x5 = a_2 + b_2 + c_1 = 4 + 5 + 3 = 12$

Aufgabe 6: Klassen von Historien (Wiederholung)

Ermitteln Sie für die folgenden Schedules, ob sie rücksetzbar sind (RC), kaskadierendes Rücksetzen vermeiden (ACA) oder strikt (ST) sind. Ermitteln Sie zusätzlich, ob die Historien serialisierbar oder seriell sind.

a) H1: $r_1(x), w_1(x), r_2(x), r_2(y), w_2(x), c_2, r_1(z), c_1$

b) H2: $w_1(x), w_2(x), w_2(y), c_2, r_1(y), w_1(y), c_1$

c) H3: $r_1(x), w_1(y), r_2(y), r_2(z), w_1(x), w_2(y), c_1, w_2(z), c_2$

d) H4: $r_2(z), r_1(x), r_1(y), w_1(z), c_1, w_2(z), c_2$

e) H5: $w_2(x), w_1(x), w_1(z), r_2(z), c_1, w_2(z), c_2$

f) H6: $r_1(x), r_1(y), w_1(y), c_1, r_2(y), r_2(z), c_2$

g) H7: $w_1(y), w_2(y), r_2(z), r_1(x), w_1(x), c_1, r_2(x), w_2(z), c_2$

h) H8: $r_2(y), w_2(y), r_1(x), r_1(z), w_1(z), c_1, r_2(z), w_2(z), c_2$

i) H9: $r_1(x), r_1(y), w_1(z), r_2(z), w_2(x), c_2, w_1(x), c_1$

Lösung:

a) H1: $r_1(x), w_1(x), r_2(x), r_2(y), w_2(x), c_2, r_1(z), c_1$
nicht in RC, da T2 von T1 ($w_1(x) <_H r_2(x)$) liest, aber $c_2 <_H c_1$
folglich weder in ACA noch ST
serialisierbar, da SG (H1) zyklenfrei
offensichtlich nicht seriell

b) H2: $w_1(x), w_2(x), w_2(y), c_2, r_1(y), w_1(y), c_1$
in RC, da T1 von T2 liest und $c_2 <_H c_1$
in ACA, da $c_2 <_H r_1(y)$ (die Änderung von T2 ist also zum Lesezeitpunkt nicht mehr schmutzig)
nicht in ST, da $w_1(x) <_H w_2(x)$ (und vor diversen anderen Operationen von T2), aber nicht $c_1 <_H w_2(x)$
nicht serialisierbar, da die Konfliktoperationen $w_1(x) <_H w_2(x)$ und $w_2(y) <_H r_1(y)$ zu einem zyklischen SG(H) führen, folglich auch nicht seriell

c) H3: $r_1(x), w_1(y), r_2(y), r_2(z), w_1(x), w_2(y), c_1, w_2(z), c_2$
in RC, da T2 von T1 liest ($w_1(y) <_H r_2(y)$) und $c_1 <_H c_2$
nicht in ACA, da NICHT $c_1 <_H r_2(y)$, T2 liest also ein schmutziges Datum und müsste bei einem Abort von T1 auch zurückgesetzt werden.
folglich nicht in ST
serialisierbar, da SG(H3) zyklenfrei, Konfliktoperationen:$w_1(y) <_H r_2(y)$ und $w_1(y) <_H w_2(y)$
offensichtlich nicht seriell

d) H4: $r_2(z), r_1(x), r_1(y), w_1(z), c_1, w_2(z), c_2$
in RC, da keine TA von der anderen liest.
in ACA, dito
in ST, da $w_1(z), <_H w_2(z)$ und gleichzeitig $c_1 <_H w_2(z)$
nicht serialisierbar, da $r_2(z) <_H w_1(z)$ und $w_1(z) <_H w_2(z)$, => zyklischer SG(H4)
folglich und offensichtlich nicht seriell

e) H5: $w_2(x), w_1(x), w_1(z), r_2(z), c_1, w_2(z), c_2$
in RC, da T2 von T1 liest ($w_1(z) <_H r_2(z)$)und $c_1 <_H c_2$
NICHT in ACA, da T2 von T1 liest ($w_1(z) <_H r_2(z)$), aber $r_2(z) <_H c_1$, es erfolgt also ein schmutziges Lesen durch T2, ein Abort von T1 müsste zwangsläufig zum Abort von T2 führen.
folglich nicht in ST,
nicht serialisierbar, da $w_2(x) <_H w_1(x)$ und $w_1(z) <_H r_2(z)$, => zyklischer SG(H5)
folglich und offensichtlich nicht seriell

f) H6: $r_1(x), r_1(y), w_1(y), c_1, r_2(y), r_2(z), c_2$
in RC, da zwar T2 von T1 liest ($w_1(y) <_H r_2(y)$) , aber auch $c_1 <_H c_2$
in ACA, da zwar T2 von T1 liest ($w_1(y) <_H r_2(y)$) , aber $c_1 <_H r_2(y)$
in ST, dito
serialisierbar, (nur eine Konfliktoperation $w_1(y) <_H r_2(y)$), => azyklischer SG(H6)
und offensichtlich seriell

g) H7: $w_1(y), w_2(y), r_2(z), r_1(x), w_1(x), c_1, r_2(x), w_2(z), c_2$
in RC, da zwar T2 von T1 liest ($w_1(x) <_H r_2(x)$) , aber auch $c_1 <_H c_2$
in ACA, da zwar T2 von T1 liest ($w_1(x) <_H r_2(x)$), aber auch $c_1 <_H r_2(x)$
nicht ST, da $w_1(y) <_H w_2(y)$, aber nicht $c_1 <_H w_2(y)$
serialisierbar: Konfliktoperationen: $w_1(y) <_H w_2(y)$ und $w_1(x) <_H r_2(x)$, => azyklischer SG(H7)
offensichtlich nicht seriell

h) H8: $r_2(y), w_2(y), r_1(x), r_1(z), w_1(z), c_1, r_2(z), w_2(z), c_2$
in RC, da zwar T2 von T1 liest ($w_1(z) <_H r_2(z)$) , aber auch $c_1 <_H c_2$
in ACA, da zwar T2 von T1 liest ($w_1(z) <_H r_2(z)$) , aber auch $c_1 <_H r_2(z)$
in ST, da zwar $w_1(z) <_H r_2(z)$ und $w_1(z) <_H w_2(z)$, aber auch $c_1 <_H r_2(z)$ und $c_1 <_H w_2(z)$
serialisierbar, Konfliktoperationen: $w_1(z) <_H r_2(z)$ und $w_1(z) <_H w_2(z)$, => azyklischer SG(H8)
offensichtlich nicht seriell

H9: $r_1(x), r_1(y), w_1(z), r_2(z), w_2(x), c_2, w_1(x), c_1$
nicht in RC, da zwar T2 von T1 liest ($w_1(z) <_H r_2(z)$) , und auch $c_2 <_H c_1$, potentielle, nicht rücksetzbare Weitergabe einer noch nicht festgeschriebenen Änderung, die durch ein Abort von T1 invalidiert werden könnte.
folglich weder in ACA noch ST
nicht serialisierbar, Konfliktoperationen: $w_1(z) <_H r_2(z)$, $r_1(x) <_H w_2(x)$, $w_2(x) <_H w_1(x)$, => zyklischer SG(H9), folglich und offensichtlich nicht seriell