3. Übungsblatt (WS 2018) – Musterlösung

3.0 VU Datenmodellierung 2 / 6.0 VU Datenbanksysteme

Informationen zum Übungsblatt

Allgemeines

In diesem Übungsteil setzen Sie die theoretischen Kenntnisse im Bereich Transaktionsverwaltung, Recovery und Mehrbenutzersynchronisation praktisch um.

Lösen Sie die Beispiele **eigenständig** (auch bei der Prüfung und vermutlich auch in der Praxis sind Sie auf sich alleine gestellt)! Wir weisen Sie darauf hin, dass sämtliche abgeschriebene Lösungen mit 0 Punkten beurteilt werden (sowohl das "Original" als auch die "Kopie").

Geben Sie ein einziges PDF Dokument ab (max. 5MB). Erstellen Sie Ihr Abgabedokument computerunterstützt. Wir akzeptieren keine gescannten handschriftlichen PDF-Dateien.

Das Übungsblatt enthält 8 Aufgaben, auf welche Sie insgesamt 15 Punkte erhalten können.

Deadlines

bis 03.12.	**12:00 Uhr**	Upload der Abgabe über TUWEL
ab 17.12.	13:00 Uhr	Korrektur und Feedback in TUWEL verfügbar

Tutorensprechstunden (freiwillig)

Rund eine Woche vor der Abgabedeadline bieten die TutorInnen Sprechstunden an. Falls Sie Probleme mit oder Fragen zum Stoff des Übungsblattes haben, es Verständnisprobleme mit den Beispielen oder technische Fragen gibt, kommen Sie bitte einfach vorbei. Die TutorInnen beantworten Ihnen gerne Ihre Fragen zum Stoff, oder helfen Ihnen bei Problemen weiter.

Ziel der Sprechstunden ist es, Ihnen beim **Verständnis des Stoffs** zu helfen, nicht, das Übungsblatt für Sie zu rechnen, oder die eigenen Lösungen vorab korrigiert zu bekommen.

Die Teilnahme ist vollkommen freiwillig — Termine und Orte der Tutorensprechstunden finden Sie in TUWEL.

Durchsprache der Übungsbeispiel (freiwillig)

In den Tagen nach Rückgabe der korrigierten Abgaben gibt es die Möglichkeit die Übungsbeispiele in kleineren Gruppen (max. 25 Personen) durchzusprechen. Jede dieser Gruppen wird von einer Assistentin/einem Assistenten geleitet. Der genaue Ablauf in einer Übungsgruppe kann variieren, und hängt auch von Ihren Wünschen und Fragen ab. Die grundsätzliche Idee ist es, die Beispiele durchzurechnen, und speziell auf Ihre Fragen und mögliche Unklarheiten einzugehen. Die (relativ) kleine Gruppengröße soll eine aktive Teilnahme ermöglichen. Daher ist es auch wichtig, dass Sie sich bereits im Vorfeld mit Ihrer korrigierten Abgabe auseiander setzen, und Unklarheiten identifizieren. Trauen Sie sich, entsprechend Fragen zu stellen – keine Frage kann irgendeinen (negativen) Einfluss auf Ihre Note haben.

Die Teilnahme an so einer Gruppe ist absolut freiwillig. Um die Gruppengröße klein zu halten ist eine Anmeldung in TUWEL erforderlich. Termine und Orte finden Sie in TUWEL.

Weitere Fragen – TUWEL Forum

Sie können darüber hinaus das TUWEL Forum verwenden, sollten Sie inhaltliche oder organisatorische Fragen haben.

Aufgaben: Recovery

In diesem Abschnitt wird die Verwendung von Log-Einträgen zur Sicherstellung der Eigenschaften Atomicity und Durability von Transaktionen behandelt. Dabei kommt das in der Vorlesung vorgestellte Format für Log-Einträge zur Verwendung, welches hier noch einmal kurz zusammengefasst ist:
Log-Einträge für von einer Transaktion ausführte Aktionen haben das Format

`[LSN, TA, PageID, Redo, Undo, PrevLSN]`,

wobei `LSN` die LogSequenceNumber des Eintrags angibt, `TA` die Transaktion welche die Aktion ausgeführt hat und `PageID` die veränderte Seite. `Redo` und `Undo` beinhalten die Redo/Undo Information und `PrevLSN` die LSN des vorhergehenden Log-Eintrags der selben Transaktion.

Die *Redo*- und *Undo*-Informationen werden logisch protokolliert, d.h. *relativ* zum Datenbestand mittels Addition bzw. Subtraktion.

Ein Beispiel für einen solchen Log-Eintrag wäre

$[\#i, T_j, P_X, X{+}{=}d_1, X{-}{=}d_2, \#k]$,

welcher besagt dass laut i-tem Logeintrag die Transaktion T_j auf ein Feld X auf der Seite P_X schreibend zugreift, so dass beim *Redo* X um d_1 vergrößert werden müsste und beim *Undo* X um d_2 verkleinert werden müsste. Außerdem hat der vorangegangene Logeintrag dieser Transaktion die Nummer k.

Tür die Log-Einträge für die BOT und commit werden nur die die `LSN`, die `TA`, den Operationsnamen (BOT bzw. commit), sowie die `PrevLSN` angegeben. D.h. die entsprechenden Log-Einträge haben das Format

`[LSN, TA, (BOT|Commit), PrevLSN]`.

Compensation Log Records (CLRs) folgen dem Format

⟨`LSN, TA, PageID, Redo, PrevLSN, UndoNextLSN`⟩,

wobei `UndoNextLSN` die LSN des nächsten Log-Eintrags der Transaktion angibt, welcher rückgängig gemacht werden soll. Äquivalent zu den Standard Log-Einträgen kann auch für BOT-CLRs die verkürzte Schreibweise

⟨`LSN, TA, BOT, PrevLSN`⟩

verwendet werden.

Aufgabe 1 (Logging) *[1 Punkte]*

Betrachten Sie die in Abbildung 1 angegebene Historie der drei Transaktionen T_1, T_2 und T_3. Dabei bezeichnen A, B, C und D Felder in der Datenbank, während a_i, b_i, c_i und d_i lokale Variablen der einzelnen Transaktionen darstellen. Weiters bezeichnet $r_i(\Gamma, \gamma)$ eine Leseoperation (der Wert des Feldes Γ wird aus der Datenbasis in eine lokale Variable γ gelesen) und $w_i(\Gamma, \gamma)$ eine Schreiboperation (der Wert γ wird in das Feld Γ in der Datenbasis geschrieben). Mit `COMMIT` wird der erfolgreiche Abschluss einer Transaktion gekennzeichnet. `ROLLBACK` bezeichnet den Abbruch einer Transaktion. Nehmen Sie dabei an, dass das Zurücksetzen der Transaktion vollständig und erfolgreich abgeschlossen wird bevor die nächste Aktion laut Liste ausgeführt wird (d.h. das `ROLLBACK` in Schritt 21 wird abgeschlossen bevor der Schreibvorgang in Schritt 22 erfolgt).

Nehmen Sie schlussendlich an, dass zu Beginn (Zeile 1) der relevante Datenbestand in der Datenbank aus den Werten $A = 50$, $B = 80$, $C = 30$ und $D = 120$ besteht.

(a) Geben Sie für jede Zeile der Historie, in welcher entweder der Wert eines Feldes, oder einer lokalen Variabel geändert wird, den Wert des entsprechenden Feldes/Variablen *nach* der

	T_1	T_2	T_3
1		BOT	
2	BOT		
3			BOT
4	$r_1(A, a_1)$		
5	$r_1(C, c_1)$		
6			$r_3(A, a_3)$
7		$r_2(D, d_2)$	
8			$w_3(A, a_3 + 100)$
9	$w_1(A, a_1 - c_1)$		
10		$r_2(B, b_2)$	
11			$r_3(A, a_3)$
12	$r_1(D, d_1)$		
13	$w_1(D, d_1 + a_1)$		
14			$r_3(D, d_3)$
15		$w_2(B, b_2 + 3)$	
16		$w_2(D, d_2 - 1)$	
17	$r_1(D, d_1)$		
18			$w_3(D, d_3 + 1)$
19		$r_2(A, a_2)$	
20		$w_2(A, d_2 - a_2)$	
21		ROLLBACK	
22	$w_1(C, d_1 + c_1)$		
23	COMMIT		
24			COMMIT

Abbildung 1: Historie für Aufgabe 1.

Operation an. Geben Sie jeweils die dazugehörige Zeilennummer der Historie an (verwenden Sie für Änderungen auf Grund des ROLLBACKs die Zeilennummer 21).

Lösung:

	Wert
4	$a_1 = 50$
5	$c_1 = 30$
6	$a_3 = 50$
7	$d_2 = 120$
8	$A = 150$
9	$A = 20$
10	$b_2 = 80$
11	$a_3 = 20$
12	$d_1 = 120$
13	$D = 170$

14	$d_3 = 170$
15	$B = 83$
16	$D = 119$
17	$d_1 = 119$
18	$D = 171$
19	$a_2 = 20$
20	$A = 100$
21	$A = 20$
21	$D = 222$
21	$B = 80$
22	$C = 149$

(b) Geben Sie eine Liste der entsprechenden Log-Einträge zu dieser Historie – in der Reihenfolge in welcher diese Einträge angelegt werden – an. Verwenden Sie dabei das am

Beginn dieses Abschnitts beschriebene Format. Denken Sie insbesonders daran, dass die *Redo-* und *Undo* Informationen mittels Addition und Subtraktion angegeben werden sollen. Nehmen Sie an, dass jedes Feld Γ auf der Seite P_Γ gespeichert wird. Zur besseren Lesbarkeit benützen Sie außerdem bitte die Schreibweise $\#i$ für die `LSN` bzw. `PrevLSN`. Sollte es zu einem Eintrag keinen vorangegangenen Eintrag geben so verwenden Sie bitte 0 als Wert. Inkludieren Sie ebenfalls die Log-Einträge für das Zurücksetzen von Transaktion T_2.

Hinweis: Formatieren Sie die Log-Einträge bitte auf eine übersichtliche Art und Weise, z.B. in einer Liste (ein Eintrag pro Zeile) oder einer Tabelle (ein Eintrag pro Zeile). Schreiben Sie die Log-Einträge bitte *nicht* als normalen Fließtext hintereinander. Wir behalten es uns vor für unlesbare Formatierungen 0 Punkte zu vergeben. (Falls Sie die LaTeXVorlage verwenden finden Sie dort bereits einen Vorschlag zur Formatierung.)

Lösung:

	Log: `[LSN, TA, PageID, Redo, Undo, PrevLSN]` bzw. $\langle$`LSN, TA, PageID, Redo, PrevLSN, UndoNextLSN`$\rangle$
1	[#1, T_2, `BOT`, #0]
2	[#2, T_1, `BOT`, #0]
3	[#3, T_3, `BOT`, #0]
8	[#4, T_3, P_A, A+=100, A-=100, #3]
9	[#5, T_1, P_A, A-=130, A+=130, #2]
13	[#6, T_1, P_D, D+=50, D-=50, #5]
15	[#7, T_2, P_B, B+=3, B-=3, #1]
16	[#8, T_2, P_D, D-=51, D+=51, #7]
18	[#9, T_3, P_D, D+=52, D-=52, #4]
20	[#10, T_2, P_A, A+=80, A-=80, #8]
21	$\langle$#11, T_2, P_A, A-=80, #10, #8$\rangle$
21	$\langle$#12, T_2, P_D, D+=51, #11, #7$\rangle$
21	$\langle$#13, T_2, P_B, B-=3, #12, #1$\rangle$
21	$\langle$#14, T_2, `BOT`, #13, #0$\rangle$
22	[#15, T_1, P_C, C+=119, C-=119, #6]
23	[#16, T_1, `COMMIT`, #15]
24	[#17, T_3, `COMMIT`, #9]

Log-Einträge

[#1, T_1, `BOT`, #0]
[#2, T_3, `BOT`, #0]
[#3, T_1, P_C, C+=1, C-=1, #1]
[#4, T_2, `BOT`, #0]
[#5, T_2, P_D, D+=101, D-=101, #4]
[#6, T_3, P_C, C+=20, C-=20, #2]
[#7, T_1, P_A, B-=1, B+=1, #3]
[#8, T_2, P_A, B+=1, B-=1, #5]
[#9, T_3, P_A, A-=10, A+=10, #6]
⟨#10, T_3, P_A, A+=10, #9, #6 ⟩
[#11, T_1, P_C, C+=110, C-=110, #7]
[#12, T_1, P_A, A+=100, A-=100, #11]
⟨#13, T_3, P_C, C-=20, #10, #2⟩
[#14, T_2, P_A, A-=50, A+=50, #8]
[#15, T_2, `COMMIT`, #14]
[#16, T_1, P_C, C-=131, C+=131, #12]

Seiten im Hintergrundspeicher

P_A	LSN: #7
$A = 100$	$B = 99$

P_C	LSN: #13
$C = 201$	

P_D	LSN: #0
$D = 100$	

Abbildung 2: Angabe zur Aufgabe 2: Der Inhalt des Log-Archivs (links) sowie der Datenbankseiten (rechts) nach einem Absturz.

Aufgabe 2 (Recovery) *[1.5 Punkte]*

Nehmen Sie an, nach einem System-Absturz finden Sie die in Abbildung 2 dargestellte Situation vor. Die linke Seite der Abbildung beschreibt den Inhalt der Logdatei, also der gesicherten Log-Einträge. Auf der rechten Seite der Abbildung ist der Inhalt der Seiten P_A, P_C und P_D dargestellt. Führen Sie an Hand dieser Informationen einen Wiederanlauf (Recovery) der Datenbank durch.

(a) Bestimmen Sie die Werte für A, B, C und C nach der *Redo*-Phase.

Lösung:

A: 150; B: 100; C: 70; D: 201

(b) Erzeugen Sie die Compensation Log Records (CLRs), welche während des Wiederanlaufs geschrieben werden.

Lösung:

Log: ⟨`LSN, TA, PageID, Redo, PrevLSN, UndoNextLSN`⟩
⟨#17, T_1, P_C, C+=131, #16, #12⟩
⟨#21, T_1, P_A, A-=100, #17, #11⟩
⟨#22, T_1, P_C, C-=110, #21, #7⟩
⟨#26, T_1, P_A, B+=1, #22, #3⟩
⟨#30, T_1, P_C, C-=1, #26, #1⟩
⟨#31, T_3, `BOT`, #13⟩
⟨#32, T_1, `BOT`, #30⟩

(c) Geben Sie die Werte von A, B, C und D nach dem Wiederanlauf an.

Lösung:

A: 50; B: 101; C: 90; D: 201

Aufgaben: Mehrbenutzersynchronisation

Aufgabe 3 (Eigenschaften von Transaktionen) *[4 Punkte]*

Betrachten Sie die beiden Mengen $\mathcal{T}_1$ und $\mathcal{T}_2$ an Transaktionen sowie die dazugehörigen Historien $\mathcal{H}_1$, $\mathcal{H}_2$, welche durch ihre Folge von Elementaroperationen gegeben sind:

(a) $\mathcal{T}_1 = \{T_1, T_2, T_3, T_4\}$

$\mathcal{H}_1 = b_1 \to b_2 \to b_3 \to b_4 \to r_2(B) \to r_1(B) \to r_3(D) \to r_4(C) \to w_2(B) \to r_4(A) \to w_4(A) \to r_3(A) \to w_4(C) \to w_3(D) \to r_1(A) \to c_4 \to r_2(D) \to w_1(A) \to r_1(D) \to r_2(B) \to w_3(C) \to r_2(C) \to w_2(C) \to c_2 \to c_3 \to c_1$.

(b) $\mathcal{T}_2 = \{T_1, T_2, T_3, T_4, T_5, T_6, T_7\}$

$\mathcal{H}_2 = b_1 \to b_2 \to b_3 \to b_4 \to b_5 \to b_7 \to b_6 \to r_2(C) \to r_1(A) \to w_3(B) \to r_7(E) \to r_4(C) \to w_4(A) \to w_5(B) \to w_4(D) \to w_5(D) \to c_4 \to w_2(C) \to r_2(A) \to w_3(C) \to w_6(A) \to w_1(E) \to c_3 \to r_2(E) \to c_2 \to a_5 \to c_6 \to r_7(A) \to r_1(C) \to r_7(B) \to c_1 \to r_7(D) \to w_7(D) \to c_7$.

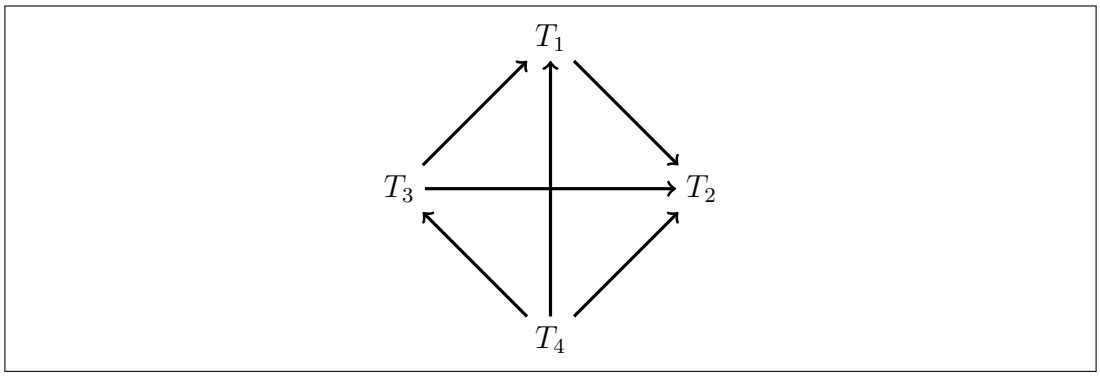
- Zeichnen Sie für beide Mengen $\mathcal{T}_1$ und $\mathcal{T}_2$ jeweils den Serialisierbarkeitsgraphen $\mathrm{SG}(\mathcal{H}_i)$.
- Geben Sie für jede Kante im Serialisierbarkeitsgraphen *sämtliche* Paare $p_i \to p_j$ von Operationen aus welche als Begründung für diese Kante genannt werden können.
- Falls die Historie konfliktserialisierbar ist, geben Sie *eine* mögliche konfliktäquivalente serielle Reihenfolge an. Andernfalls geben Sie eine (möglichst kleine) Menge an Transaktionen an welche man aus der Historie streichen müsste damit diese konfliktserialisierbar wird. Geben Sie anschließend eine mögliche konfliktäquivalente serielle Reihenfolge an.
- Geben Sie für beide Historien $\mathcal{H}_1$ und $\mathcal{H}_2$ alle Paare von Transaktionen (T_i, T_j) an, so dass T_j von T_i liest. Geben Sie zu jedem dieser Paare jeweils sämtliche Paare $(w_i(X), r_j(X))$ von Operationen an auf Grund denen T_j von T_i liest.
- Bestimmen Sie für $\mathcal{H}_1$ und $\mathcal{H}_2$ ob die Historien jeweils die folgenden Eigenschaften besitzen:

- Rücksetzbar
- Ohne kaskadierendes Rücksetzen
- Strikt

Begründen Sie jeweils Ihre Antwort.

Lösung:

(a) **Serialisierbarkeitsgraph:**



"Begründung" für die Kanten:

$T_1 \to T_2$

- $r_1(B) \to w_2(B)$

$T_4 \to T_3$

- $w_4(A) \to r_3(A)$
- $r_4(C) \to w_3(C)$
- $w_4(C) \to w_3(C)$

$T_4 \to T_1$

- $w_4(A) \to r_1(A)$
- $r_4(A) \to w_1(A)$
- $w_4(A) \to w_1(A)$

$T_4 \to T_2$

- $w_4(C) \to r_2(C)$
- $r_4(C) \to w_2(C)$
- $w_4(C) \to w_2(C)$

$T_3 \to T_2$

- $w_3(D) \to r_2(D)$
- $w_3(C) \to r_2(C)$
- $w_3(C) \to w_2(C)$

$T_3 \to T_1$

- $w_3(D) \to r_1(D)$
- $r_3(A) \to w_1(A)$

Konflikt-Serialisierbarkeit:

Ja, die Historie *ist* konfliktserialisierbar. Eine mögliche Reihenfolge ist

$$T_4 \text{ vor } T_3 \text{ vor } T_1 \text{ vor } T_2$$

Lese-Abhängigkeiten:

- T_3 liest von T_4: $(w_4(A), r_3(A))$
- T_1 liest von T_4: $(w_4(A), r_1(A))$
- T_2 liest von T_3: $(w_3(D), r_2(D)), (w_3(C), r_2(C))$

- T_1 liest von T_3: $(w_3(D), r_1(D))$

Rücksetzbar:

Nein, die Historie *ist nicht* rücksetzbar. Eine Historie ist rücksetzbar wenn jede Transaktion von der gelesen wird ihr `COMMIT` vor dem `COMMIT` der jeweils lesenden Transaktionen haben. Dies ist hier nicht der Fall: T_2 liest von T_3, daher müsste das commit von T_3 vor dem commit von T_2 passieren, was jedoch nicht der Fall ist ($c_2 \rightarrow c_3$).

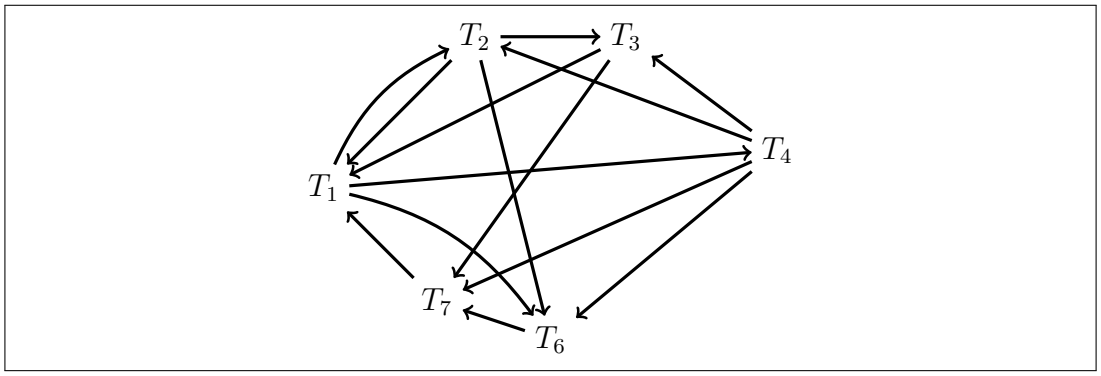
Vermeidet Kaskadierendes Rücksetzen:

Nein. Dies ergibt sich zum Einen sofort daraus dass die Historie nicht rücksetzbar ist. Zum Anderen lässt sich auch ein konkretes Gegenbeispiel angeben. Um kaskadierendes Rücksetzen zu vermeiden ist es notwendig dass bei jeder Leseoperation ein Wert gelesen wird, welcher von einer bereits erfolgreich abgeschlossenen (commited) Transaktion geschrieben wurde. Ein Beispiel wo dies hier verletzt wird ist $w_4(A)$ und $r_1(A)$, da die Transaktion T_4 ihr `COMMIT` c_4 erst nach der Operation $r_1(A)$ hat.

Strikte Historie:

Nein. Dies ergibt sich zum Einen wiederum daraus, dass die Historie kein kaskadierendes Rücksetzen vermeidet. Des Weiteren ist auch das Gegenbeispiel $w_4(A)$ und $r_1(A)$ ein Gegenbeispiel dafür, dass die Historie strikt ist.

(b) **Serialisierbarkeitsgraph:**



"Begründung" für die Kanten:

$T_3 \to T_7$

- $w_3(B) \to r_7(B)$

$T_1 \to T_4$

- $r_1(A) \to w_4(A)$

$T_4 \to T_2$

- $w_4(A) \to r_2(A)$
- $r_4(C) \to w_2(C)$

$T_4 \to T_7$

- $w_4(A) \to r_7(A)$
- $w_4(D) \to r_7(D)$
- $w_4(D) \to w_7(D)$

$T_2 \to T_1$

- $w_2(C) \to r_1(C)$

$T_7 \to T_1$

- $r_7(E) \to w_1(E)$

$T_1 \to T_2$

- $w_1(E) \to r_2(E)$

$T_2 \to T_3$

- $r_2(C) \to w_3(C)$
- $w_2(C) \to w_3(C)$

$T_4 \to T_3$

- $r_4(C) \to w_3(C)$

$T_3 \to T_1$

- $w_3(C) \to r_1(C)$

$T_1 \to T_6$

- $r_1(A) \to w_6(A)$

$T_4 \to T_6$

- $w_4(A) \to w_6(A)$

$T_2 \to T_6$

- $r_2(A) \to w_6(A)$

$T_6 \to T_7$

- $w_6(A) \to r_7(A)$

Konflikt-Serialisierbarkeit:

Nein, die Historie *ist nicht* konfliktserialisierbar.

Es reicht die Transaktion T_1 zu entfernen um eine konfliktserialisierbare Abfolge zu erhalten. Eine solche mögliche Abfolge wäre

$$T_4 \text{ vor } T_2 \text{ vor } T_3 \text{ vor } T_6 \text{ vor } T_7$$

Lese-Abhängigkeiten:

- T_1 liest von T_3: $(w_3(C), r_1(C))$
- T_2 liest von T_1: $(w_1(E), r_2(E))$
- T_2 liest von T_4: $(w_4(A), r_2(A))$
- T_7 liest von T_3: $(w_3(B), r_7(B))$
- T_7 liest von T_4: $(w_4(D), r_7(D))$
- T_7 liest von T_6: $(w_6(A), r_7(A))$

Rücksetzbar:

Nein, die Historie *ist nicht* rücksetzbar. Eine Historie ist rücksetzbar wenn jede Transaktion von der gelesen wird ihr `COMMIT` vor dem `COMMIT` der jeweils lesenden Transaktionen haben. Dies ist hier nicht der Fall: T_2 liest von T_1, daher müsste das commit von T_1 vor dem commit von T_2 passieren, was jedoch nicht der Fall ist ($c_2 \to c_1$).

Vermeidet Kaskadierendes Rücksetzen:

Nein. Dies ergibt sich zum Einen sofort daraus dass die Historie nicht rücksetzbar ist. Zum Anderen lässt sich auch ein konkretes Gegenbeispiel angeben. Um kaskadierendes Rücksetzen zu vermeiden ist es notwendig dass bei jeder Leseoperation ein Wert gelesen wird, welcher von einer bereits erfolgreich abgeschlossenen (commited) Transaktion geschrieben wurde. Ein Beispiel wo dies hier verletzt wird ist $w_1(E)$ und $r_2(E)$, da die Transaktion T_1 ihr `COMMIT` c_1 erst nach der Operation $r_2(E)$ hat.

Strikte Historie:

Nein. Dies ergibt sich zum Einen wiederum daraus, dass die Historie kein kaskadierendes Rücksetzen vermeidet. Des Weiteren ist auch das Gegenbeispiel $w_1(E)$ und $r_2(E)$ ein Gegenbeispiel dafür, dass die Historie strikt ist.

Aufgabe 4 (Eigenschaften von Transaktionen 2) *[0.5 Punkte]*

Geben Sie eine *strikte* Historie von zwei Transaktionen T_1 und T_2 an welche *nicht konfliktserialisierbar* ist. Verwenden Sie dazu die Elementaroperationen b_i (für den Beginn einer Transaktion T_i), c_i (für das Commit einer Transaktion T_i), $r_i(X)$ (für einen Lesezugriff einer Transaktion T_i auf ein Feld X) und $w_i(X)$ (für einen Schreibzugriff einer Transaktion T_i auf ein Feld X). Sie können die Anzahl und Namen der benötigten Datenbankfelder selber wählen.

Lösung:

$$b_1 \rightarrow b_2 \rightarrow r_1(A) \rightarrow w_2(A) \rightarrow c_2 \rightarrow r_1(A)$$

Aufgabe 5 (Sperren und Deadlocks) *[2 Punkte]*

Gegeben ist die untenstehende Folge von Sperranforderungen, wobei „$\texttt{lockS}_i(O)$“ (bzw. „$\texttt{lockX}_i(O)$“) bedeutet, dass eine Transaktion T_i eine Lesesperre (bzw. eine Schreibsperre) auf das Datenobjekt O anfordert, und „$\texttt{rel}_i(O)$“ bedeutet dass eine Transaktion T_i sämtliche Sperren auf das Datenobjekt O aufgibt:

$\texttt{lockX}_2(B) \rightarrow \texttt{lockX}_1(C) \rightarrow \texttt{lockS}_3(A) \rightarrow \texttt{lockS}_1(A) \rightarrow \texttt{lockX}_4(E) \rightarrow \texttt{lockS}_2(A) \rightarrow \texttt{rel}_1(A) \rightarrow \texttt{lockX}_1(E) \rightarrow \texttt{lockS}_3(C) \rightarrow \texttt{lockS}_1(B) \rightarrow \texttt{lockX}_2(D)$.

Nehmen Sie an, dass keine gewährte Sperre wieder freigegeben wurde.

(a) Nehmen Sie an, ein DBMS erhält die angegebene Folge von Sperranforderungen und arbeitet sie in der genannten Reihenfolge ab, wobei Transaktionen, welchen eine gewünschte Sperre nicht gewährt wird, angehalten werden. (D.h. nachfolgende Sperranforderungen der selben Transaktion werden ignoriert und aufgeschoben bis die Transaktion wieder aktiv wird.)

Geben Sie an, in welcher Reihenfolge das DBMS die Sperranforderungen abarbeitet, und geben Sie unmittelbar nach einer Sperranforderung an ob es die gewünschte Sperre gewährt oder die entsprechende Transaktion auf die Sperre warten muss. Verwenden Sie $\texttt{grantS}_i(O)$ bzw. $\texttt{grantX}_i(O)$ um anzugeben, dass eine Lese- bzw. Schreibsperre auf dem Datenobjekt O gewährt wurde, verwenden Sie $\texttt{wait}(i)$ um anzuzeigen, dass eine Transaktion angehalten wurde um auf eine Sperre zu warten, verwenden Sie $\texttt{relS}_i(O)$ bzw. $\texttt{relX}_i(O)$ um anzugeben, dass eine Lese- bzw. Schreibsperre auf dem Datenobjekt O freigegeben wurde (als Reaktion auf ein $\texttt{rel}_i(O)$), und $\texttt{resume}(i)$ um anzuzeigen dass die

Blockierung einer Transaktion wieder aufgehoben wurde, weil das gewünschte Feld nun verfügbar ist.

Nehmen Sie dabei an, dass wenn eine blockierte Transaktion durch die Freigabe einer Sperre wieder aktiv wird diese Transaktion zuerst alle "übersprungenen" Aktionen nachholt, bis sie entweder wieder blockiert oder es keine weiteren ausgelassenen Aktionen dieser Transaktion gibt. Erst danach soll mit dem Abarbeiten der Aktionen bei der ursprünglichen Freigabe fortgefahren werden.

Beispiel: Nehmen Sie die Folge

$\texttt{lockS}_1(A) \rightarrow \texttt{lockS}_2(A) \rightarrow \texttt{lockX}_1(A) \rightarrow \texttt{lockX}_2(B) \rightarrow \texttt{lockS}_1(B)$

von Sperranforderungen zweier Transaktionen T_1, T_2.

Wir erhalten die Liste

1:	$\texttt{lockS}_1(A)$
2:	$\texttt{grantS}_1(A)$
3:	$\texttt{lockS}_2(A)$
4:	$\texttt{grantS}_2(A)$
5:	$\texttt{lockX}_1(A)$
6:	$\texttt{wait}(1)$
7:	$\texttt{lockX}_2(B)$
8:	$\texttt{grantX}_2(B)$

Lösung:

1:	$\texttt{lockX}_2(B)$
2:	$\texttt{grantX}_2(B)$
3:	$\texttt{lockX}_1(C)$
4:	$\texttt{grantX}_1(C)$
5:	$\texttt{lockS}_3(A)$
6:	$\texttt{grantS}_3(A)$
7:	$\texttt{lockS}_1(A)$
8:	$\texttt{grantS}_1(A)$
9:	$\texttt{lockX}_4(E)$
10:	$\texttt{grantX}_4(E)$

11:	$\texttt{lockS}_2(A)$
12:	$\texttt{grantS}_2(A)$
13:	$\texttt{rel}_1(A)$
14:	$\texttt{relS}_1(A)$
15:	$\texttt{lockX}_1(E)$
16:	$\texttt{wait}(1)$
17:	$\texttt{lockS}_3(C)$
18:	$\texttt{wait}(3)$
19:	$\texttt{lockX}_2(D)$
20:	$\texttt{grantX}_2(D)$

(b) Skizzieren Sie bitte die aktuelle Situation der gehaltenen Sperren bzw. angehaltener Transaktionen. Geben Sie dazu eine wie weiter unten dargestellte Tabelle an. Tragen Sie in ein Feld ein X (bzw. ein S) ein, wenn die entsprechende Transaktion eine Schreibsperre (bzw. eine Lesesperre) auf dieses Datenobjekt besitzt. Tragen Sie für jede blockierte Transaktion bitte zusätzlich für jene Sperranforderung auf Grund welcher die Transaktion nun blockiert ist ein WS (*wait shared*) bzw. WX (*wait exclusive*) in das entsprechende Feld ein.

Lösung:

	A	B	C	D	E
T_1			X		WX
T_2	S	X		X	
T_3	S		WS		
T_4					X

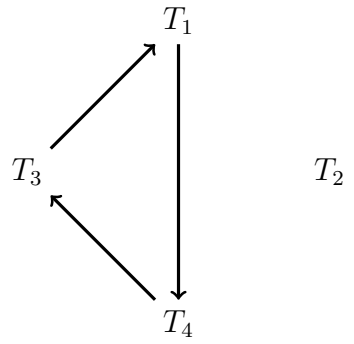
(c) Betrachten Sie die folgende Liste an Sperr- bzw. Freigabeanforderungen, und nehmen Sie an, dass diese im Anschluss an die am Beginn der Aufgabe angegeben Anforderungen folgen (Anforderungen von blockierten Transaktionen sollen weiterhin aufgeschoben werde).

$\texttt{rel}_2(B) \rightarrow \texttt{rel}_1(C) \rightarrow \texttt{lockX}_4(A) \rightarrow \texttt{lockX}_4(B) \rightarrow \texttt{rel}_2(A)$

Zeichnen Sie den Wartegraphen nach Bearbeitung der angegebenen Anforderungen.

Lösung:

Wartegraph:



(d) Geben Sie an, ob zum aktuellen Zeitpunkt ein Deadlock besteht. **Lösung: Ja**

(e) Geben Sie eine mögliche Sequenz an Freigaben an, mit welcher sämtliche gehaltenen Sperren wieder freigegeben werden können. Sollte eine blockierte Transaktion durch die Freigabe einer Sperre weiterlaufen können, so müssen von dieser Transaktion zuerst alle ausstehenden Sperranfragen und Freigaben aus der Angabe abgearbeitet werden, bevor Sie zusätzliche Freigaben definieren können. Sollte derzeit ein Deadlock bestehen, so soll Transaktion T_3 abgebrochen werden (d.h. alle Sperren von T_3 werden sofort freigegeben).

Lösung:
Nachdem derzeit ein Deadlock besteht wird zuerst T_3 abgebrochen. Daraus ergeben sich die Freigaben:

$\texttt{relS}_3(A) \rightarrow \texttt{relS}_3(C)$.

Dadurch kann T_4 die Sperre auf A erhalten, und somit weiterlaufen. Wir erhalten

$\texttt{resume}(4) \rightarrow \texttt{grantX}_4(A) \rightarrow \texttt{lockX}_4(B) \rightarrow \texttt{grantX}_4(B)$.

T_4 hat nun alle angegebenen Anfragen abgearbeitet, T_1 ist nach wie vor blockiert. Eine mögliche Sequenz wäre:

$\texttt{rel}_4(A) \rightarrow \texttt{relX}_4(A) \rightarrow \texttt{rel}_4(B) \rightarrow \texttt{relX}_4(B) \rightarrow \texttt{rel}_4(E) \rightarrow \texttt{relX}_4(E) \rightarrow$

Dadurch kann nun T_1 die Sperre auf E erhalten und weiterlaufen. Es ergibt sich

$\texttt{resume}(1) \rightarrow \texttt{grantX}_1(E) \rightarrow \texttt{lockS}_1(B) \rightarrow \texttt{grantX}_1(B) \rightarrow \texttt{rel}_1(C) \rightarrow \texttt{relX}_1(C) \rightarrow$

Damit hat mit T_1 auch die letzte Transaktion alle vorgegebenen Anforderungen abgearbeitet. Die restlichen Freigaben können zum Beispiel in folgender Reihenfolge aufgegeben werden:

$\texttt{rel}_1(E) \rightarrow \texttt{rel}_1(B) \rightarrow \texttt{rel}_2(D)$.

(f) Erklären Sie (ganz) kurz, warum die für Aufgabe (a) und (b) angegebene Sequenz an Sperranforderungen und Freigaben nicht dem Zwei Phasen Sperrprotokoll entsprechen.

Lösung:
Das 2PL fordert, dass nach der Freigabe einer Sperre keine neue Sperre mehr angefordert werden darf. Dies wird z.B. von Transaktion T_1 verletzt welche die Lesesperre auf A freigibt, und danach eine Schreibsperre auf E anfordert.

Aufgabe 6 (Zwei-Phasen-Sperrprotokoll) *[1 Punkte]*

Betrachten Sie die folgenden drei Transaktionen T_1, T_2 und T_3, für welche jeweils eine Folge von Elementaroperationen gegeben ist ($r_i(O)$ und $w_i(O)$ bezeichnen eine Lese- bzw. Schreiboperation von T_i auf O, und c_i bezeichnet das commit von T_i).

$$\begin{array}{lccccccccccc}
T_1\colon & r_1(A) & \rightarrow & r_1(B) & \rightarrow & w_1(A) & \rightarrow & w_1(C) & \rightarrow & w_1(B) & \rightarrow & c_1 \\
T_2\colon & r_2(A) & \rightarrow & r_2(D) & \rightarrow & w_2(D) & \rightarrow & r_2(A) & \rightarrow & w_2(E) & \rightarrow & c_2 \\
T_3\colon & w_3(C) & \rightarrow & r_3(C) & \rightarrow & r_3(B) & \rightarrow & w_3(D) & \rightarrow & r_3(E) & \rightarrow & c_3
\end{array}$$

Nehmen Sie an, zur Synchronisation dieser Transaktionen wird das *strikte 2-Phasen Sperrprotokoll* verwendet. Geben Sie die dadurch entstehende Historie (bestehend aus Sperranforderungen, Lese- und Schreiboperationen, Freigaben sowie commits) an.

Treffen Sie dabei folgende Annahmen:

- *Notation:* Verwenden Sie bitte die Notation $\texttt{lockS}_i(O)$ und $\texttt{lockX}_i(O)$ um das Anfordern eine Lese- bzw. Schreibsperre von Transaktion T_i auf das Objekt O anzuschreiben. Verwenden Sie bitte ebenso $\texttt{rel}_i(O)$ für die Freigabe sämtlicher Sperren von T_i auf O. (*Hinweis:* Sie brauchen die Information ob eine Sperre gewährt wird oder eine Transaktion blockiert wird nicht explizit angeben. Das ergibt sich daraus, ob auf eine Sperranforderung einer Transaktion eine entsprechende Operation dieser Transaktion folgt, oder nicht).

- *Sperranforderungen:* Geben Sie zu jeder Operation (lesen, schreiben, commit) die benötigten Sperranforderungen an (sofern diese von der Transaktion noch nicht gehalten werden). Nehmen Sie dabei an, dass die Transaktion dabei nicht "in die Zukunft" schaut, sondern nur jene Sperren anfordert, welche für die aktuelle Operation benötigt werden.

- *Freigaben von Sperren:* Transaktionen geben Sperren jeweils zum frühestmöglichen Zeitpunkt wieder her, zu dem es das Sperrprotokoll erlaubt (um diesen zu ermitteln dürfen die noch ausstehenden Operationen der Transaktion in Betracht gezogen werden, d.h. anders als bei den Sperranforderungen darf hier ein "Blick in die Zukunft" geworfen werden).

- *Verzahnung der Transaktionen:* Nehmen Sie an, dass jede Transaktion jeweils eine Operation (lesen, schreiben, commit) abarbeitet, und anschließend die nächste Transaktion ausgeführt wird. D.h. im gegeben Fall wäre die Reihenfolge der Aktionen $r_1(A) \rightarrow r_2(A) \rightarrow w_3(C) \rightarrow r_1(B) \rightarrow \ldots$. Sperranforderungen und Freigaben zählen hierbei nicht als Operationen, d.h. vor und nach jeder Operation (lesen, schreiben, commit) darf die Transaktion eine beliebige Anzahl von Sperranforderungen und Freigaben ausführen, bevor die nächste Transaktion an die Reihe kommt.

  Von dieser Reihenfolge soll nur abgewichen werden, wenn eine Transaktion blockiert ist. Dann wird die Transaktion für diese Runde übersprungen. Die passiert so lange, bis die Blockierung wieder aufgehoben wird, dann nimmt die Transaktion wieder ganz normal am Ablauf Teil – jedoch weiterhin mit nur einer einzigen Operation (lesen, schreiben, commit), bevor wieder die anderen Transaktionen an der Reihe sind.

  Das folgende Beispiel soll den Ablauf an Hand von zwei Transaktionen demonstrieren. Nehmen wir an, T_1 besteht aus Aktionen $\alpha_1, \alpha_2, \ldots$, und T_2 besteht aus Aktionen $\beta_1, \beta_2, \ldots$. Der normale Ablauf wäre $\alpha_1, \beta_1, \alpha_2, \beta_2, \ldots$. Angenommen T_2 bräuchte für die Aktion β_3 eine Sperre, welche T_1 hält. D.h. T_2 blockiert. Dann wäre der weitere Ablauf $\alpha_3, \alpha_4, \alpha_5, \ldots$. Wird die Sperre nach Aktion α_5 wieder aufgegeben, so ist die weitere Abfolge $\alpha_5, \beta_3, \alpha_6, \beta_4, \ldots$.

Lösung:

$\texttt{lockS}_1(A) \rightarrow r_1(A) \rightarrow \texttt{lockS}_2(A) \rightarrow r_2(A) \rightarrow \texttt{lockX}_3(C) \rightarrow w_3(C) \rightarrow \texttt{lockS}_1(B) \rightarrow r_1(B) \rightarrow \texttt{lockS}_2(D) \rightarrow r_2(D) \rightarrow r_3(C) \rightarrow \texttt{lockX}_1(A) \rightarrow \texttt{lockX}_2(D) \rightarrow w_2(D) \rightarrow \texttt{lockS}_3(B) \rightarrow r_3(B) \rightarrow r_2(A) \rightarrow \texttt{lockX}_3(D) \rightarrow \texttt{lockX}_2(E) \rightarrow w_2(E) \rightarrow \texttt{rel}_2(A) \rightarrow \texttt{rel}_2(D) \rightarrow \texttt{rel}_2(E) \rightarrow c_2 \rightarrow w_3(D) \rightarrow w_1(A) \rightarrow \texttt{lockS}_3(E) \rightarrow r_3(E) \rightarrow \texttt{lockX}_1(C) \rightarrow \texttt{rel}_3(B) \rightarrow \texttt{rel}_3(C) \rightarrow \texttt{rel}_3(D) \rightarrow \texttt{rel}_3(E) \rightarrow c_3 \rightarrow w_1(C) \rightarrow \texttt{lockX}_1(B) \rightarrow w_1(B) \rightarrow \texttt{rel}_1(A) \rightarrow \texttt{rel}_1(B) \rightarrow \texttt{rel}_1(C) \rightarrow c_1$

Aufgabe 7 (Multi-Granularity Locking) *[3 Punkte]*

Betrachten Sie die Datenbasis-Hierarchie in Abbildung 3.

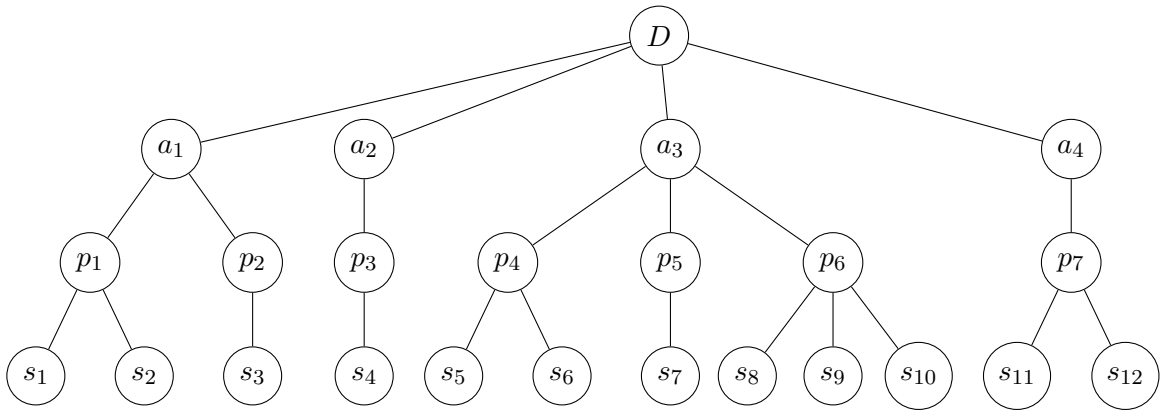


Abbildung 3: Datenbasis-Hierarchie zu Aufgabe 7

Betrachten Sie die folgenden Sequenzen von Sperranforderungen bzw. Freigaben zweier Transaktionen T_1 und T_2 auf die in Abbildung 3 dargestellten Ressourcen.

(a) $\texttt{lockS}_1(s_2) \rightarrow \texttt{lockX}_2(p_5) \rightarrow \texttt{lockX}_2(s_8) \rightarrow \texttt{lockS}_1(s_{11}) \rightarrow \texttt{lockX}_2(s_6) \rightarrow \texttt{lockS}_2(s_{12}) \rightarrow \texttt{lockS}_1(p_7) \rightarrow \texttt{lockX}_2(s_{12}) \rightarrow \texttt{rel}_1(s_2)$

(b) $\texttt{lockX}_1(s_5) \rightarrow \texttt{lockS}_2(s_8) \rightarrow \texttt{lockX}_1(s_1) \rightarrow \texttt{lockX}_2(a_4) \rightarrow \texttt{lockS}_2(p_6) \rightarrow \texttt{lockS}_1(s_9) \rightarrow \texttt{lockX}_1(p_2) \rightarrow \texttt{lockX}_1(s_{10}) \rightarrow \texttt{lockX}_2(s_3) \rightarrow \texttt{lockS}_2(s_2) \rightarrow \texttt{lockX}_1(s_6)$

(c) $\texttt{lockX}_1(s_8) \rightarrow \texttt{lockX}_1(s_{10}) \rightarrow \texttt{lockS}_1(s_9) \rightarrow \texttt{lockX}_2(s_{11}) \rightarrow \texttt{lockS}_2(s_{12}) \rightarrow \texttt{rel}_1(s_8) \rightarrow \texttt{rel}_2(s_{11}) \rightarrow \texttt{rel}_1(s_9) \rightarrow \texttt{lockX}_2(s_8) \rightarrow \texttt{lockS}_1(s_5) \rightarrow \texttt{rel}_1(s_{10}) \rightarrow \texttt{lockX}_2(s_9) \rightarrow \texttt{lockX}_1(p_6)$

Dabei bedeutet $\texttt{lockS}_i(O)$ dass die Transaktion T_i eine Lesesperre (Shared lock) auf das Object O anfordert, $\texttt{lockX}_i(O)$ dass die Transaktion T_i eine Schreibsperre (eXclusive lock) auf das Object O anfordert, und $\texttt{rel}_i(O)$ dass Transaktion T_i alle für das Objet O gehaltene Sperren freigibt.

Bearbeiten Sie folgende Aufgabenstellungen für jede der drei angegeben Sequenzen:

- Geben Sie an, wie vorgegangen werden muss um die Sperranforderungen bzw. Freigaben entsprechend dem Protokoll des Multi Granularity Lockings (MGL) zu verarbeiten: Geben Sie die Sequenz der benötigten Sperranforderungen aus, bzw. im Fall einer Freigabe, geben Sie an welche weiteren Sperren freigegeben werden können. *Hinweis:* Achten Sie sowohl beim Anfordern der Sperren als auch bei der Freigabe auf die richtige Ordnung.

  Verwenden Sie bitte die folgende Notation: $\texttt{lockIS}_i(O)$, $\texttt{lockIX}_i(O)$, $\texttt{lockS}_i(O)$ und $\texttt{lockX}_i(O)$ für das Anfordern einer `IS`-, `IX`-, `S`- bzw. `X`-Sperre von Transaktion T_i auf das Objekt O; $\texttt{relIS}_i(O)$, $\texttt{relIX}_i(O)$, $\texttt{relS}_i(O)$ und $\texttt{relX}_i(O)$ für die Freigabe einer `IS`-, `IS`-, `S`- bzw. `X`-Sperre von Transaktion T_i auf das Objekt O. Achten Sie bitte außerdem darauf, dass nur Sperren angefordert werden, welche die Transaktionen nicht bereits besitzen.

- Markieren Sie Sperren, welche nicht gewährt werden können. Nehmen Sie an, dass die entsprechende Transaktion in so einem Fall angehalten wird, d.h. es werden keine weiteren Aktionen dieser Transaktion durchgeführt bis die benötigte Sperre auf Grund einer Freigabe der anderen Transaktion gewährt werden kann. (Sperranforderungen bzw. Freigaben angehaltener Transaktionen werden bis dahin einfach übersprungen.) Kann eine zuvor angehaltene Transaktion auf Grund einer Freigabe weiterlaufen, so nehmen Sie an dass alle "übersprungenen" Aktionen ausgeführt werden bevor die Abarbeitung der Sequenz nach der Freigabeaktion weitergeführt wird.
- Geben Sie zu jeder nicht gewährten Sperranforderung an, warum diese Sperre verweigert wurde.
- Entsteht während der Abarbeitung der Sequenz ein Deadlock? Falls ja, warum?
- Falls es zu keinem Deadlock kommt, am Ende der Sequenz jedoch eine Transaktion blockiert ist: Geben Sie eine minimale Menge an Freigaben an welche benötigt wird damit die Transaktion weiterlaufen kann. Nehmen Sie dabei an, dass auch von den impliziten IS bzw. IX Sperren nur jene freigegeben werden, die absolut notwendig sind. D.h. es kann sein dass an einem Knoten eine IS oder IX Sperre einer Transaktion erhalten bleibt, obwohl diese Transaktion in keinem Kindknoten eine Sperre besitzt.

  Führen Sie anschließend die blockierte Transaktion weiter. Sollte es dabei zu weiteren Blockierungen kommen, geben Sie jeweils wiederum eine Menge minimale Menge an Freigaben an damit die Transaktion weiterlaufen kann.

Hinweis:

- Sollte der Fall eintreten, dass eine Transaktion eine Sperre auf einem Knoten erhält, welche sie bereits für einen oder mehrere Kindknoten hält, so können Sie davon ausgehen dass die entsprechenden Sperren automatisch in sämtlichen betroffenen Kindknoten entfernt werden (dies brauchen Sie nicht angeben).

Lösung:

(a)

1:	$\texttt{lockIS}_1(D)$
2:	$\texttt{lockIS}_1(a_1)$
3:	$\texttt{lockIS}_1(p_1)$
4:	$\texttt{lockS}_1(s_2)$
5:	$\texttt{lockIX}_2(D)$
6:	$\texttt{lockIX}_2(a_3)$
7:	$\texttt{lockX}_2(p_5)$
8:	$\texttt{lockIX}_2(p_6)$
9:	$\texttt{lockX}_2(s_8)$
10:	$\texttt{lockIS}_1(a_4)$
11:	$\texttt{lockIS}_1(p_7)$
12:	$\texttt{lockS}_1(s_{11})$
13:	$\texttt{lockIX}_2(p_4)$
14:	$\texttt{lockX}_2(s_6)$
15:	$\texttt{lockIS}_2(a_4)$
16:	$\texttt{lockIS}_2(p_7)$
17:	$\texttt{lockS}_2(s_{12})$
18:	$\texttt{lockS}_1(p_7)$
19:	$\texttt{lockIX}_2(a_4)$
20:	$\texttt{lockIX}_2(p_7)$ (1)
21:	$\texttt{relS}_1(s_2)$
22:	$\texttt{relIS}_1(p_1)$
23:	$\texttt{relIS}_1(a_1)$

(b)

1:	$\texttt{lockIX}_1(D)$
2:	$\texttt{lockIX}_1(a_3)$
3:	$\texttt{lockIX}_1(p_4)$
4:	$\texttt{lockX}_1(s_5)$
5:	$\texttt{lockIS}_2(D)$
6:	$\texttt{lockIS}_2(a_3)$
7:	$\texttt{lockIS}_2(p_6)$
8:	$\texttt{lockS}_2(s_8)$
9:	$\texttt{lockIX}_1(a_1)$
10:	$\texttt{lockIX}_1(p_1)$
11:	$\texttt{lockX}_1(s_1)$
12:	$\texttt{lockIX}_2(D)$
13:	$\texttt{lockX}_2(a_4)$
14:	$\texttt{lockS}_2(p_6)$
15:	$\texttt{lockIS}_1(p_6)$
16:	$\texttt{lockS}_1(s_9)$
17:	$\texttt{lockX}_1(p_2)$
18:	$\texttt{lockIX}_1(p_6)$ (1)
19:	$\texttt{lockIX}_2(a_1)$
20:	$\texttt{lockIX}_2(p_2)$ (2)

(c)

1:	$\texttt{lockIX}_1(D)$
2:	$\texttt{lockIX}_1(a_3)$
3:	$\texttt{lockIX}_1(p_6)$
4:	$\texttt{lockX}_1(s_8)$
5:	$\texttt{lockX}_1(s_{10})$
6:	$\texttt{lockS}_1(s_9)$
7:	$\texttt{lockIX}_2(D)$
8:	$\texttt{lockIX}_2(a_4)$
9:	$\texttt{lockIX}_2(p_7)$
10:	$\texttt{lockX}_2(s_{11})$
11:	$\texttt{lockS}_2(s_{12})$
12:	$\texttt{relX}_1(s_8)$
13:	$\texttt{relX}_2(s_{11})$
14:	$\texttt{relS}_1(s_9)$
15:	$\texttt{lockIX}_2(a_3)$
16:	$\texttt{lockIX}_2(p_6)$
17:	$\texttt{lockX}_2(s_8)$
18:	$\texttt{lockIS}_1(p_4)$
19:	$\texttt{lockS}_1(s_5)$
20:	$\texttt{relX}_1(s_{10})$
21:	$\texttt{relIX}_1(p_6)$
22:	$\texttt{lockX}_2(s_9)$
23:	$\texttt{lockX}_1(p_6)$ (1)

(a) (1) kann wegen der Lesesperre von T_1 auf p_7 nicht gewährt werden. Es kommt zu keinem Deadlock, da T_1 nicht blockiert wird, und somit (theoretisch) die Lesesperre auf p_7 aufheben kann.

Die passiert bis zum Ende der Sequenz nicht, d.h. T_1 schließt seine Aktionen ab ohne dass die Sperre auf p_7 aufgegeben wird. Nachdem T_1 in keinem der Kindknoten von p_7 eine Sperre besitzt (die Lesesperre auf s_{11} wurde entsprechend laut Annahme bei Erhalt der Lesesperre auf p_7 automatisch beendet), besteht die minimale Menge an Freigaben einfach aus der Freigabe der Lesesperre auf p_7 durch T_1.

(b) (1) Beim Versuch die Sperre $\texttt{lockX}_1(s_{10})$ zu erhalten wird T_1 bei der Anfrage nach einer `IX`-Sperre auf p_6 blockiert, da T_2 bereits eine Lesesperre auf p_6 hält.

(2) Beim Versuch die Sperre $\texttt{lockX}_2(s_3)$ zu erhalten wird T_2 bei der Anfrage nach einer `IX`-Sperre auf p_2 blockiert, da T_1 bereits eine Schreibsperre auf p_2 besitzt.

Es kommt in dieser Situation zu einem Deadlock, weil T_1 auf die Freigabe der Lesesperre auf p_6 durch T_2 wartet, und T_2 auf die Freigabe der Schreibsperre auf p_2 durch T_1 wartet. Da jedoch beide Transaktionen blockiert sind, wird keine der beiden ausgeführt werden, und somit kann keine der beiden Sperren freigegeben werden.

(c) (1) Beim Versuch eine Schreibsperre auf p_6 zu erhalten wird die Transaktion T_1 blockiert, da T_2 bereits eine `IX`-Sperre auf p_6 besitzt.

Es kommt jedoch zu keinem Deadlock, da T_2 nicht blockiert. Die `IX`-Sperre von T_2 auf p_6

wird bis zum Ende der Sequenz nicht freigegeben, d.h. T_1 ist auch am Ende der Sequenz noch weiter blockiert.

Die minimale Menge an Freigaben durch T_2 welche benötigt wird damit T_1 weiterlaufen kann ist:

$$\texttt{relX}_2(s_8) \rightarrow \texttt{relX}_2(s_9) \rightarrow \texttt{relIX}_2(p_6)$$

Anschließend ist auch T_1 abgeschlossen.

Aufgabe 8 (Zeitstempelbasiertes Sperrverfahren) *[2 Punkte]*

Gegeben ist die folgende Sequenz von Elementaroperationen dreier Transaktionen T_1, T_2 und T_3, welche auf drei Datenobjekte A, B und C zugreifen.

$\texttt{BOT}_1 \rightarrow r_1(C) \rightarrow \texttt{BOT}_4 \rightarrow w_4(A) \rightarrow \texttt{BOT}_2 \rightarrow r_2(C) \rightarrow w_4(C) \rightarrow w_1(C) \rightarrow res? \rightarrow c_4 \rightarrow \texttt{BOT}_3 \rightarrow res? \rightarrow r_3(A) \rightarrow w_3(B) \rightarrow r_2(A) \rightarrow w_3(A) \rightarrow r_3(B) \rightarrow w_1(A) \rightarrow res? \rightarrow c_3 \rightarrow c_2 \rightarrow c_1 \rightarrow res?$

Dabei bezeichnet $\texttt{BOT}_i$ den Beginn der Transaktion i, $r_i(X)$ eine Leseoperation (Transaktion T_i liest Datenobjekt X), $w_i(X)$ eine Schreiboperation (Transaktion T_i schreibt Datenobjekt X) und c_i den erfolgreichen Abschluss (commit) von Transaktion i. Einträge $res?$ bedeuten, dass zu diesem Zeitpunkt eine Transaktion neu gestartet werden soll, falls es zu dieser Zeit eine Transaktion gibt welche zurückgesetzt aber noch nicht neu gestartet wurde (falls es mehrere solcher Transaktionen gibt, starten Sie jene neu, deren Rücksetzen am weitesten zurück liegt). Anschließend an den Neustart führen Sie bitte alle Operationen der Transaktion bis zum aktuellen Zeitpunkt aus.

(a) Wenden Sie die Regeln der Zeitstempel-Synchronisationsmethode auf den gegeben Operationen an, um eine (nach dieser Methode) gültige Historie zu erhalten. Verwenden Sie jene in der Vorlesung vorgestellte Variante, welche nicht notwendigerweise rücksetzbare Historien erzeugt. D.h. Schreiboperationen werden sofort durchgeführt, und der Zugriff auf entsprechende Felder wird für andere Transaktionen ausschließlich über die Lese- und Schreibzeitstempel geregelt (d.h. die Transaktionen werden entweder abgebrochen, oder erhalten Zugriff, Transaktionen werden nicht blockiert).

Sie brauchen sich im Falle eines Zurücksetzens nicht darum kümmern, ob bereits ausgeführte Aktionen andere Transaktionen von dem Zurücksetzen betroffen sind: es wird nur die aktuelle Transaktionen zurückgesetzt, und kein kaskadierendes Rücksetzen durchgeführt.

Im Falle eines Zurücksetzens bleiben sowohl der Lese- als auch der Schreibzeitstempel unverändert.

Geben Sie die daraus entstehende Historie bitte in Form einer Tabelle mit den folgenden Spalten an:

#	Aktion	rTS(A)	wTS(A)	rTS(B)	wTS(B)	rTS(C)	wTS(C)

Die Spalte # soll eine fortlaufende Nummer beinhalten. Für die Spalte Aktion verwenden Sie bitte die weiter oben beschriebene (und in der Angabe verwendete) Schreibweise für BOT-Einträge, COMMIT-Einträge, sowie Lese- und Schreibeinträge. Wird eine Transaktion zurückgesetzt, so verwenden Sie bitte $\texttt{reset}_i$ für den Eintrag. In den restlichen Spalten geben Sie bitte die Werte der Lese- und Schreibzeitstempel jeweils nach der Ausführung der entsprechenden Aktion an.

Lösung:

#	T	rTS(A)	wTS(A)	rTS(B)	wTS(B)	rTS(C)	wTS(C)
1	$\texttt{BOT}_1$	0	0	0	0	0	0
2	$r_1(C)$	0	0	0	0	1	0
3	$\texttt{BOT}_4$	0	0	0	0	1	0
4	$w_4(A)$	0	3	0	0	1	0
5	$\texttt{BOT}_2$	0	3	0	0	1	0
6	$r_2(C)$	0	3	0	0	5	0
7	$\texttt{reset}_4$	0	3	0	0	5	0
8	$\texttt{reset}_1$	0	3	0	0	5	0
9	$\texttt{BOT}_4$	0	3	0	0	5	0
10	$w_4(A)$	0	9	0	0	5	0
11	$w_4(C)$	0	9	0	0	5	9
12	c_4	0	9	0	0	5	9
13	$\texttt{BOT}_3$	0	9	0	0	5	9
14	$\texttt{BOT}_1$	0	9	0	0	5	9
15	$r_1(C)$	0	9	0	0	14	9
16	$w_1(C)$	0	9	0	0	14	14
17	$r_3(A)$	13	9	0	0	14	14
18	$w_3(B)$	13	9	0	13	14	14
19	$\texttt{reset}_2$	13	9	0	13	14	14
20	$w_3(A)$	13	13	0	13	14	14
21	$r_3(B)$	13	13	13	13	14	14
22	$w_1(A)$	13	14	13	13	14	14
23	$\texttt{BOT}_2$	13	14	13	13	14	14
24	$r_2(C)$	13	14	13	13	23	14
25	$r_2(A)$	23	14	13	13	23	14
26	c_3	23	14	13	13	23	14
27	c_2	23	14	13	13	23	14
28	c_1	23	14	13	13	23	14

(b) Verwenden Sie nun die in der Vorlesung vorgestellte Variante der Zeitstempel-Synchronisationsmethode welche *strikte Historien* erzeugt. (Verwenden Sie die Methode mittels *dirty bit.*) Im Falle eines Zurücksetzens sollen diesesmal – falls anwendbar – auch die Schreibzeitstempel zurückgesetzt werden. Lesezeitstempel bleiben wiederum unverändert.

Geben Sie die resultierende Historie wiederum in einer wie in Aufgabe (a) beschriebenen Tabelle aus, aber geben Sie für jedes der Felder A, B und C nun zusätzlich noch aus, ob das dirty bit gesetzt ist oder nicht. D.h. verwenden Sie eine Tabelle mit den Spalten:

#	Aktion	rTS(A)	wTS(A)	d(A)	rTS(B)	wTS(B)	d(B)	rTS(C)	wTS(C)	d(C)

Lösung:

#	T	rTS(A)	wTS(A)	d(A)	rTS(B)	wTS(B)	d(B)	rTS(C)	wTS(C)	d(C)
1	$\texttt{BOT}_1$	0	0	n	0	0	n	0	0	n
2	$r_1(C)$	0	0	n	0	0	n	1	0	n
3	$\texttt{BOT}_4$	0	0	n	0	0	n	1	0	n
4	$w_4(A)$	0	3	j	0	0	n	1	0	n
5	$\texttt{BOT}_2$	0	3	j	0	0	n	1	0	n
6	$r_2(C)$	0	3	j	0	0	n	5	0	n
7	$\texttt{reset}_4$	0	0	n	0	0	n	5	0	n
8	$\texttt{reset}_1$	0	0	n	0	0	n	5	0	n
9	$\texttt{BOT}_4$	0	0	n	0	0	n	5	0	n
10	$w_4(A)$	0	9	j	0	0	n	5	0	n
11	$w_4(C)$	0	9	j	0	0	n	5	9	j
12	c_4	0	9	n	0	0	n	5	9	n
13	$\texttt{BOT}_3$	0	9	n	0	0	n	5	9	n
14	$\texttt{BOT}_1$	0	9	n	0	0	n	5	9	n
15	$r_1(C)$	0	9	n	0	0	n	14	9	n
16	$w_1(C)$	0	9	n	0	0	n	14	14	j
17	$r_3(A)$	13	9	n	0	0	n	14	14	j
18	$w_3(B)$	13	9	n	0	13	j	14	14	j
19	$\texttt{reset}_2$	13	9	n	0	13	j	14	14	j
20	$w_3(A)$	13	13	j	0	13	j	14	14	j
21	$r_3(B)$	13	13	j	13	13	j	14	14	j
22	$\texttt{BOT}_2$	13	13	j	13	13	j	14	14	j
23	c_3	13	13	n	13	13	n	14	14	j
24	$w_1(A)$	13	14	j	13	13	n	14	14	j
25	c_1	13	14	n	13	13	n	14	14	n
26	$r_2(C)$	13	14	n	13	13	n	22	14	n
27	$r_2(A)$	22	14	n	13	13	n	22	14	n
28	c_2	22	14	n	13	13	n	22	14	n

Verwenden Sie zum Lösen der Aufgabe bitte die folgenden Annahmen bzw. Konventionen:

- Nehmen Sie an, dass die Anfangswerte für die Werte `readTS` und `writeTS` aller drei Felder A, B und C jeweils 0 sind.
- Als Zeitstempel für die Transaktionen nehmen Sie bitte die # ihres des jeweiligen `BOT`-Eintrags.
- Wenn Sie am Ende der Historie angekommen sind, und nicht alle rückgestzten Transaktionen neu gestartet wurden (kein passender *res*? Eintrag), dann wird diese Transaktion einfach nicht erneut durchgeführt.