

# 3. Übungsblatt (WS 2020)

## 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 PDF-Dateien mit handschriftlichen Inhalten**.

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

#### Deadlines

**bis 11.12. 12:00 Uhr:** Upload der Abgabe über TUWEL  
**ab 12.01. 13:00 Uhr:** Korrektur und Feedback in TUWEL verfügbar

#### Weitere Fragen – TUWEL Forum

Sie können darüber hinaus das TUWEL Forum verwenden, sollten Sie inhaltliche oder organisatorische Fragen haben. **Posten Sie auf keinen Fall (partielle) Lösungen im Forum!**

#### Änderungen im Ablauf bzgl. COVID-19

Wegen der andauernden Sondersituation werden heuer keine Sprechstunden zum Übungsblatt stattfinden. Bitte wenden Sie sich stattdessen verstärkt an das TUWEL Forum wenn Sie Probleme damit haben die Angaben zu verstehen oder technische Schwierigkeiten haben.

Wenn möglich empfehlen wir Ihnen auch das Forum zur Diskussion mit Ihren Kommilitonen zu Nutzen. Ein gemeinsames Analysieren von Problemen hilft erfahrungsgemäß allen Beteiligten dabei den Stoff besser zu verstehen.

## 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 ausgeführte 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  $k$ .

Für die Log-Einträge für den Beginn (BOT – Begin of Transaction) und das Commit von Transaktionen werden nur die  $LSN$ , die  $TA$ , der 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

$\langle LSN, TA, PageID, Redo, PrevLSN, UndoNextLSN \rangle$ ,

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

$\langle LSN, TA, BOT, PrevLSN \rangle$

verwendet werden.

### Aufgabe 1 (Logging)

[1 Punkt]

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  $\Gamma$  wird aus der Datenbasis in eine lokale Variable  $\gamma$  gelesen) und  $w_i(\Gamma, \gamma)$  eine Schreiboperation (der Wert  $\gamma$  wird in das Feld  $\Gamma$  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 20 wird abgeschlossen bevor der Schreibvorgang in Schritt 21 erfolgt).

Nehmen Sie schlussendlich an, dass zu Beginn (Zeile 1) der relevante Datenbestand in der Datenbank wie folgt aussieht:

$A: 22 \quad B: 56 \quad C: 0 \quad D: -3$

	$T_1$	$T_2$	$T_3$
1	BOT		
2	$r_1(B, b_2)$		
3		BOT	
4	$r_1(A, a_1)$		
5			BOT
6			$w_3(C, 0 + 13)$
7		$r_2(A, a_3)$	
8	$w_1(B, a_1 + 14)$		
9		$w_2(C, 0 - 11)$	
10			$r_3(A, a_2)$
11	$w_1(D, a_1 + b_2)$		
12		$w_2(A, a_3 + 47)$	
13			$w_3(A, a_2 - 24)$
14			$r_3(D, d_1)$
15	$r_1(B, b_7)$		
16		$r_2(B, b_3)$	
17			COMMIT
18		$w_2(D, b_3 + a_3)$	
19	$w_1(B, b_2 + b_7)$		
20		ROLLBACK	
21	$r_1(D, d_3)$		
22	$w_1(B, d_3 + 17)$		
23	COMMIT		

Abbildung 1: Historie für Aufgabe 1.

- (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 Operation an. Geben Sie jeweils die dazugehörige Zeilennummer der Historie an (verwenden Sie für Änderungen auf Grund des ROLLBACKs die Zeilennummer 20).
- (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 insbesondere daran, dass die *Redo*- und *Undo* Informationen mittels Addition und Subtraktion angegeben werden sollen. Nehmen Sie an, dass jedes Feld  $\Gamma$  auf der Seite  $P_\Gamma$  gespeichert wird. Zur besseren Lesbarkeit benutzen 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  $T_2$ .

*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 L<sup>A</sup>T<sub>E</sub>X Vorlage verwenden finden Sie dort bereits einen Vorschlag zur Formatierung.)

**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.

**Log-Einträge**

[#1,  $T_1$ , BOT, #0]  
 [#2,  $T_3$ , BOT, #0]  
 [#3,  $T_3$ ,  $P_A$ ,  $A+=12$ ,  $A-=12$ , #2]  
 [#4,  $T_2$ , BOT, #0]  
 [#5,  $T_1$ ,  $P_D$ ,  $D+=21$ ,  $D-=21$ , #1]  
 [#6,  $T_2$ ,  $P_A$ ,  $A+=34$ ,  $A-=34$ , #4]  
 [#7,  $T_3$ ,  $P_A$ ,  $B-=22$ ,  $B+=22$ , #3]  
 <#8,  $T_3$ ,  $P_A$ ,  $B+=22$ , #7, #3>  
 [#9,  $T_1$ ,  $P_C$ ,  $C+=50$ ,  $C-=50$ , #5]  
 [#10,  $T_5$ , BOT, #0]  
 [#11,  $T_1$ ,  $P_D$ ,  $D+=100$ ,  $D-=100$ , #9]  
 <#12,  $T_1$ ,  $P_D$ ,  $D-=100$ , #11, #9>  
 [#13,  $T_5$ ,  $P_C$ ,  $C-=11$ ,  $C+=11$ , #10]  
 <#14,  $T_3$ ,  $P_A$ ,  $A-=12$ , #8, #2>  
 <#15,  $T_1$ ,  $P_C$ ,  $C-=50$ , #12, #5>  
 [#16,  $T_2$ ,  $P_D$ ,  $D-=23$ ,  $D+=23$ , #6]  
 <#17,  $T_3$ , BOT, #14>  
 [#18,  $T_4$ , BOT, #0]  
 [#19,  $T_5$ ,  $P_A$ ,  $A+=76$ ,  $A-=76$ , #13]  
 [#20,  $T_4$ ,  $P_A$ ,  $B+=15$ ,  $B-=15$ , #18]  
 [#21,  $T_4$ , COMMIT, #20]

**Seiten im Hintergrundspeicher**

$P_A$	LSN: #6
$A = 55$	$B = 7$

$P_C$	LSN: #13
$C = 50$	

$P_D$	LSN: #11
$D = 121$	

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

Führen Sie an Hand dieser Informationen einen Wiederanlauf (Recovery) der Datenbank durch.

- Bestimmen Sie die Werte für  $A$ ,  $B$ ,  $C$  und  $D$  nach der *Redo*-Phase.
- Erzeugen Sie die Compensation Log Records (CLRs), welche während des Wiederanlaufs geschrieben werden.
- Geben Sie die Werte von  $A$ ,  $B$ ,  $C$  und  $D$  nach dem Wiederanlauf an.

## Aufgaben: Mehrbenutzersynchronisation

### Aufgabe 3 (Eigenschaften von Transaktionen)

[2.6 Punkte]

Gegeben ist die Menge  $\mathcal{T}$  an Transaktionen sowie die dazugehörige Historie  $\mathcal{H}$ , welche durch ihre Folge von Elementaroperationen gegeben ist:

$$\mathcal{T} = \{T_1, T_2, T_3, T_4\}$$

$$\mathcal{H} = b_1 \rightarrow r_1(A) \rightarrow b_3 \rightarrow r_1(B) \rightarrow b_2 \rightarrow w_1(B) \rightarrow r_2(C) \rightarrow b_4 \rightarrow w_2(C) \rightarrow r_4(C) \rightarrow w_3(D) \rightarrow r_3(C) \rightarrow r_2(B) \rightarrow w_1(D) \rightarrow a_3 \rightarrow r_1(D) \rightarrow c_1 \rightarrow w_2(A) \rightarrow c_2 \rightarrow r_4(A) \rightarrow w_4(A) \rightarrow c_4.$$

- (a) Zeichnen Sie den Serialisierbarkeitsgraphen  $\text{SG}(\mathcal{H})$ .
- (b) Geben Sie für jede Kante im Serialisierbarkeitsgraphen mindestens ein Paar  $p_i \rightarrow p_j$  von Operationen an, welches begründet warum diese Kante Teil des Graphen ist.
- Geben Sie für die Kanten  $T_1 \rightarrow T_2$ ,  $T_2 \rightarrow T_3$  und  $T_2 \rightarrow T_4$  (sofern sie tatsächlich Teil des Graphen sind) *sämtliche* Konfliktoperationen an, welche verlangen dass diese Kanten Teil des Graph sind.
- (c) 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 abbrechen müsste damit die Historie konfliktserialisierbar wird. Geben Sie anschließend eine mögliche konfliktäquivalente serielle Reihenfolge an.
- (d) Geben Sie für die Historie  $\mathcal{H}$  die Leseabhängigkeiten zwischen den Transaktionen an (d.h., geben Sie an welche Transaktionen von welchen Transaktionen lesen). Geben Sie zu jeder Leseabhängigkeit mindestens ein Paar  $(w_i(X), r_j(X))$  von Operationen an, welches die Leseabhängigkeit belegt.
- (e) Bestimmen Sie, welche der folgenden Eigenschaften die Historie  $\mathcal{H}$  besitzt:
- Rücksetzbar
  - Vermeidet kaskadierendes Rücksetzen
  - Strikt

Begründen Sie jeweils Ihre Antwort.

- (f) Bestimmen Sie, ob die folgende Historie konfliktserialisierbar ist und welche der drei Eigenschaften aus Aufgabe (e) (Rücksetzbar, vermeidet kaskadierendes Rücksetzen und strikt) die Historie erfüllt.

$$b_1 \rightarrow r_1(A) \rightarrow b_3 \rightarrow b_2 \rightarrow w_1(A) \rightarrow w_2(A) \rightarrow b_4 \rightarrow r_4(D) \rightarrow r_2(B) \rightarrow w_4(D) \rightarrow w_3(B) \rightarrow r_3(D) \rightarrow w_3(D) \rightarrow r_3(C) \rightarrow w_4(C) \rightarrow r_1(C) \rightarrow r_2(D) \rightarrow w_2(D) \rightarrow c_4 \rightarrow c_1 \rightarrow c_3 \rightarrow c_2$$

**Aufgabe 4 (Sperranforderungen und Deadlocks)**

[2.5 Punkte]

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

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

- (a) Nehmen Sie an, ein DBMS erhält die angegebene Folge von Sperranforderungen für die Transaktionen  $T_1$ ,  $T_2$ ,  $T_3$  und  $T_4$ , 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  $\text{grantS}_i(O)$  bzw.  $\text{grantX}_i(O)$  um anzugeben, dass eine Lese- bzw. Schreibsperre auf dem Datenobjekt  $O$  gewährt wurde, verwenden Sie  $\text{wait}(i)$  um anzuzeigen, dass eine Transaktion angehalten wurde um auf eine Sperre zu warten, verwenden Sie  $\text{relS}_i(O)$  bzw.  $\text{relX}_i(O)$  um anzugeben, dass eine Lese- bzw. Schreibsperre auf dem Datenobjekt  $O$  freigegeben wurde (als Reaktion auf ein  $\text{rel}_i(O)$ ), und  $\text{resume}(i)$  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

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

von Sperranforderungen zweier Transaktionen  $T_1$ ,  $T_2$ .

Wir erhalten die Liste

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

- (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 Schreibsperr (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.

	$A$	$B$	$C$	$D$
$T_1$				
$T_2$				
$T_3$				
$T_4$				

- (c) Geben Sie den der aktuellen Situation entsprechenden Wartegraphen an.
- (d) Geben Sie an, ob zum aktuellen Zeitpunkt ein Deadlock besteht.
- (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).
- (f) Betrachten Sie noch einmal die gegebene Sequenz an Sperranforderungen und Freigaben. Widerspricht diese dem Zwei Phasen Sperrprotokoll? Wie sieht es bei der von Ihnen in Aufgabe (e) erstellten Sequenz aus?

**Aufgabe 5 (Zwei-Phasen-Sperrprotokoll)**

[1.5 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  $T_i$ ).

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

Nehmen Sie an, zur Synchronisation dieser Transaktionen wird das (“normale”) *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  $\text{lockS}_i(O)$  und  $\text{lockX}_i(O)$  um das Anfordern einer Lese- bzw. Schreibsperre von Transaktion  $T_i$  auf das Objekt  $O$  anzuschreiben. Verwenden Sie bitte ebenso  $\text{rel}_i(O)$  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 und Freigaben:* 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 Sperren so sparsam wie möglich beantragt werden. D.h.
  - Es werden nur Sperren beantragt die auch tatsächlich benötigt werden.
  - Sperren werden so kurz wie möglich gehalten, d.h. sie werden spät wie möglich beantragt, und so früh wie möglich wieder freigegeben.
- *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 gegebenen Fall wäre die Reihenfolge der Aktionen  $w_1(C) \rightarrow w_2(A) \rightarrow r_3(C) \rightarrow r_1(B) \rightarrow \dots$  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  $\alpha_1, \alpha_2, \dots$ , und  $T_2$  besteht aus Aktionen  $\beta_1, \beta_2, \dots$ . Der normale Ablauf wäre  $\alpha_1, \beta_1, \alpha_2, \beta_2, \dots$ . Angenommen  $T_2$  bräuchte für die Aktion  $\beta_3$  eine Sperre, welche  $T_1$  hält. D.h.  $T_2$  blockiert. Dann wäre der weitere Ablauf  $\alpha_3, \alpha_4, \alpha_5, \dots$ . Wird die Sperre nach Aktion  $\alpha_5$  wieder aufgegeben, so ist die weitere Abfolge  $\alpha_5, \beta_3, \alpha_6, \beta_4, \dots$ .



**Aufgabe 6 (Multi-Granularity Locking)**

[2.4 Punkte]

Betrachten Sie die Datenbasis-Hierarchie in Abbildung 3.

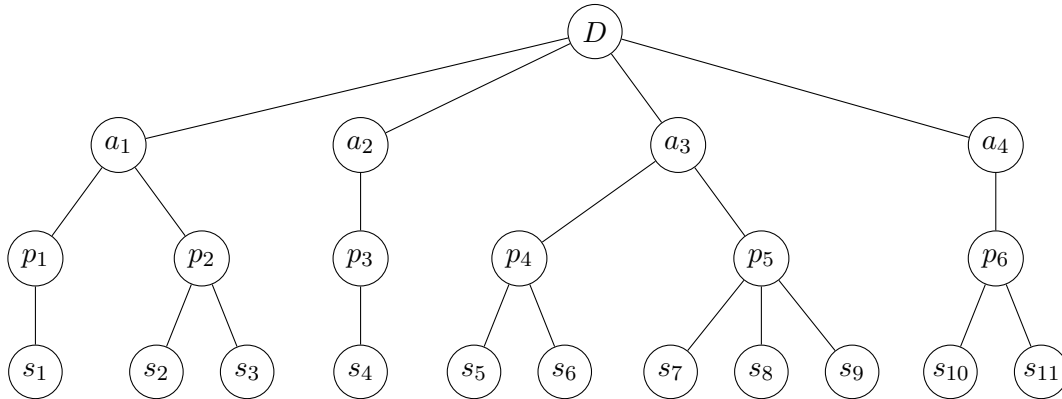


Abbildung 3: Datenbasis-Hierarchie zu Aufgabe 6

Betrachten Sie die folgenden Sequenzen von Sperranforderungen bzw. Freigaben der vier Transaktionen  $T_1$ ,  $T_2$ ,  $T_3$  und  $T_4$  auf die in Abbildung 3 dargestellten Ressourcen.

- (a)  $\text{lockS}_2(p_2) \rightarrow \text{lockX}_1(s_8) \rightarrow \text{lockS}_1(p_3) \rightarrow \text{lockS}_4(s_{11}) \rightarrow \text{lockX}_3(s_{10}) \rightarrow \text{lockX}_4(s_4) \rightarrow \text{lockS}_2(s_6) \rightarrow \text{rel}_3(s_{10})$
- (b)  $\text{lockS}_4(p_1) \rightarrow \text{lockS}_3(p_5) \rightarrow \text{lockS}_1(s_4) \rightarrow \text{lockS}_2(p_6) \rightarrow \text{lockX}_4(s_5) \rightarrow \text{lockX}_3(p_2) \rightarrow \text{lockX}_2(p_1) \rightarrow \text{lockS}_1(s_1) \rightarrow \text{lockS}_4(s_7) \rightarrow \text{lockS}_1(p_4) \rightarrow \text{rel}_3(p_5) \rightarrow \text{lockX}_4(s_{10}) \rightarrow \text{lockS}_3(a_3) \rightarrow \text{rel}_4(s_5) \rightarrow \text{rel}_2(p_6) \rightarrow \text{rel}_4(p_1) \rightarrow \text{lockX}_1(s_6)$

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

Bearbeiten Sie folgende Aufgabenstellungen für jede der beiden angegebenen 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:  $\text{lockIS}_i(O)$ ,  $\text{lockIX}_i(O)$ ,  $\text{lockS}_i(O)$  und  $\text{lockX}_i(O)$  für das Anfordern einer IS-, IX-, S- bzw. X-Sperre von Transaktion  $T_i$  auf das Objekt  $O$ ;  $\text{relIS}_i(O)$ ,  $\text{relIX}_i(O)$ ,  $\text{relS}_i(O)$  und  $\text{relX}_i(O)$  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 Sperren an, welche Transaktionen freigeben müssen, damit die blockierten Transaktionen weiterlaufen können. Beachten Sie dabei, dass Transaktionen nur Schreib- und Lesesperren explizit freigeben können; geben Sie aber auch an welche IX- und IS- Sperren dadurch implizit freigegeben werden können. (Achten Sie dabei auf die richtige Reihenfolge.)

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).

#### Aufgabe 7 (Zeitstempelbasiertes Sperrverfahren)

[3 Punkte]

Gegeben ist die folgende Sequenz von Elementaroperationen von vier Transaktionen  $T_1$ ,  $T_2$ ,  $T_3$  und  $T_4$ , welche auf drei Datenobjekte  $A$ ,  $B$  und  $C$  zugreifen.

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

Dabei bezeichnet  $\text{BOT}_i$  den Beginn der Transaktion  $T_i$ ,  $r_i(X)$  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  $T_i$ . Einträge  $\text{res?}$  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 gegebenen 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 anderer 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 **reset<sub>i</sub>** für den Eintrag. In den restlichen Spalten geben Sie bitte die Werte der Lese- und Schreibzeitstempel jeweils nach der Ausführung der entsprechenden Aktion an.

- (b) Ist die erzeugte Historie rücksetzbar?
- (c) 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 diesmal – falls anwendbar – auch die Schreibzeitstempel zurückgesetzt werden. Lesezeitstempel bleiben wiederum unverändert.

Um in der Tat alle Deadlocks auszuschließen ist eine leichte Erweiterung des Algorithmus gegenüber der in der Vorlesung vorgestellten Version nötig: Wann immer die Zeitstempel eines Datensatzes geändert werden, so soll die dazugehörige Warteschlange überprüft werden, ob dort Transaktionen auf den Zugriff warten, welche auf Grund der neuen Zeitstempel nicht auf den Datensatz zugreifen dürfen. Ist dies der Fall sollen die entsprechenden wartenden Transaktionen abgebrochen werden.

Geben Sie die resultierende Historie wiederum in einer wie in Aufgabe (a) beschriebenen Tabelle aus, jedoch mit den folgenden beiden zusätzlichen Informationen:

- 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)
---	--------	--------	--------	------	--------	--------	------	--------	--------	------

- Wenn eine Transaktion  $T_i$  blockiert, ergänzen Sie die Historie mit einem Eintrag **block<sub>i</sub>**.

- (d) Ist die erzeugte Historie rücksetzbar?

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.

**Aufgabe 8 (Transaktionen in SQL)**

[0.5 Punkte]

Betrachten Sie das folgende Relationenschema eines Unternehmens, in welchem durchgeführte Arbeiten und deren Verrechnung gespeichert werden:

(Primärschlüssel sind unterstrichen, Fremdschlüssel sind kursiv geschrieben).

```

Deliveries      ( package_id, sender, address, weight )
distributed     ( package_id: Deliveries.package_id )
customs_fees    ( package_id: Deliveries.package_id, amount )

```

Bestimmen Sie für jedes der unten aufgeführten Szenarien das jeweils *niedrigste* Isolation Level welches das geforderte Grad an Isolation bietet. Beschreiben Sie außerdem, ob die Transaktionen bei der angegebenen Verzahnung in dem jeweiligen Isolation Level wie gewünscht ablaufen können, oder ob es zu Problemen kommt. Aus Platzgründen wird unten nur eine Skizze der Abfragen angezeigt. Die vollständigen Abfragen finden Sie in den zur Verfügung gestellten SQL Dateien.

(a) *Beschreibung:*

Es ist verlangt, dass eine Lieferung erst in Verteilung gehen darf, nachdem die erwartbaren Zölle für diese Lieferung abgeschätzt wurden. Diese Vorabschätzung erfolgt durch eine n Vorarbeiter\_in und wird durch eigene Transaktionen in die Datenbank eingetragen. Es ist erlaubt, dass andere Transaktionen parallel auf die Datenbank zugreifen können, solange nur abgeschlossene Änderungen der Transaktionen der Vorarbeiter\_in nach außen sichtbar sind.

*Historie:*

Supervisor	Parallele Anfragen
BEGIN;	
SET TRANSACTION ISOLATION LEVEL ____	
SELECT * FROM Deliveries WHERE ¬fee_determined	
	INSERT INTO Deliveries VALUES (12, ...);
	INSERT INTO Deliveries VALUES (13, ...);
SELECT * FROM Deliveries NATURAL JOIN distributed	
SELECT * FROM Deliveries NATURAL JOIN distributed WHERE ¬distributed	
INSERT INTO customs_fees VALUES (5, ...);	
SELECT * FROM Deliveries WHERE ¬distributed	
	INSERT INTO Deliveries VALUES (10, ...);
INSERT INTO customs_fees VALUES (12, ...);	INSERT INTO Deliveries VALUES (11, ...);
SELECT * FROM Deliveries WHERE ¬distributed	
COMMIT;	

wobei  $\neg$ fee\_determined und  $\neg$ distributed für die Bedingungen  
 package\_id NOT IN (SELECT package\_id FROM customs\_fees) bzw.  
 package\_id NOT IN (SELECT package\_id FROM distributed) stehen.

(b) *Beschreibung:*

Ein Manager möchte eine Übersicht über alle Lieferungen für die bereits die Zölle abgeschätzt wurden, die jedoch noch nicht in Verteilung sind. Es ist entscheidend, dass während diese Übersicht erstellt wird eine konsistente Datenbasis besteht, welche nicht durch parallele Schreibzugriffe verändert werden darf.

*Historie:*

Supervisor	Parallele Anfragen
BEGIN;	
SET TRANSACTION ISOLATION LEVEL _____	
SELECT * FROM undistributed	
	INSERT INTO distributed VALUES ( ...);
SELECT sum(weight) FROM undistributed	
	INSERT INTO Deliveries VALUES ( ...);
	INSERT INTO Deliveries VALUES ( ...);
SELECT count(package_id) FROM undistributed	
	INSERT INTO customs_fees VALUES ( ...);
COMMIT;	

wobei undistributed für den Ausdruck  
 Deliveries NATURAL JOIN customs\_fee WHERE package\_id NOT IN (SELECT package\_id  
 FROM distributed) steht.

(c) *Beschreibung:*

Theresa und Boris haben sich Zugang zur Datenbank des Transportunternehmens verschafft, und wollen alle Pakete die vom jeweils anderen verschickt worden sind abfangen, und an ihre eigene Adresse schicken (natürlich nur jene Lieferungen, die nicht bereits in Verteilung sind). Das System des Transportunternehmens verlangt allerdings (aus irgendeinem gutem, aber uns unbekannten Grund), dass neue Lieferungen nicht parallel angelegt werden dürfen, sondern immer eine klare Reihenfolge der Aktionen nachvollziehbar sein muss.

*Historie:*

Boris	Theresa
	BEGIN; SET TRANSACTION ISOLATION LEVEL ____
BEGIN; SET TRANSACTION ISOLATION LEVEL ____	
SELECT sum(weight) FROM Deliveries('Theresa');	
INSERT INTO Deliveries VALUES (8, 'Boris', 'Downing Street 10', 211.10);	
	SELECT sum(weight) FROM Deliveries('Boris');
	INSERT INTO Deliveries VALUES (9, 'Theresa', 'Posh Street 20', 99.99);
COMMIT;	
	COMMIT;

wobei Deliveries(a) für folgende Zeichenkette steht:

Deliveries WHERE sender=a AND package\_id NOT IN (SELECT package\_id FROM distributed)

*Hinweis:* Sie können die Szenarien mit Hilfe der SQL Dateien auch einfach auf einem DBMS ausprobieren (die Dateien sind für Postgres geschrieben, können aber, evtl. mit kleineren Änderungen, auch auf anderen DBMS ausprobiert werden). Auf Postgres, gehen Sie dabei wie folgt vor:

1. Öffnen Sie mittels **psql** eine Datenbankkonsole.
2. Öffnen Sie in einem anderen Terminal mittels **psql** eine weitere Datenbankkonsole.
3. Laden Sie einem Terminal das Szenario mittels **\i Szenario1-a.sql** (wobei Sie "1" durch die gewünschte Zahl ersetzen)
4. Laden Sie im anderen Terminal das parallele Szenario mittels **1Szenario1-b.sql** (wobei Sie "1" durch die gewünschte Zahl ersetzen)
5. Die Ausführung der SQL-Befehle sollte nun in beiden Konsolen mit der Meldung **Press Enter to continue (i)** unterbrochen sein.
6. Drücken Sie jeweils in der Konsole in der **i** den kleineren Wert hat [Enter].

Sollte sich das Ergebnis bei einem gewählten Isolation Level von Ihrer Erwartung unterscheiden, so können Sie dies gerne dokumentieren. In dem Fall geben Sie bitte an, auf welchem DBMS Sie die Transaktionen ausgeführt haben.

Sie können das Beispiel entweder auf einer Datenbank bei sich laufen lassen, es steht Ihnen aber auch unser Übungsserver **bordo** zur Verfügung. Zu diesem können Sie sich per ssh verbinden und haben anschließend mittels **psql** Zugriff auf eine PostgreSQL Datenbank. Weitere Informationen darüber, wie Sie sich am Server und der Datenbank einloggen können finden Sie in TUWEL.