3. Übungsblatt (WS 2017) – 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. Zur Vorbereitung auf das 4. Übungsblatt wiederholen Sie das Überführen eines EER-Diagramms in ein Relationenschema.

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 6 Aufgaben, auf welche Sie insgesamt 15 Punkte erhalten können.

Deadlines

bis 01.12.	**12:00 Uhr**	Upload der Abgabe über TUWEL
ab 18.12.	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

Aufgabe 1 (Logging & Recovery) *[2 Punkte]*

Gegeben ist die folgende Historie von Transaktionen:

	T_1	T_2	T_3
1	BOT		
2		BOT	
3			BOT
4	$r(C, c_1)$		
5		$r(C, c_2)$	
6			$r(A, a_3)$
7		$w(A, c_2 * 2)$	
8	$w(A, c_1 + 200)$		
9			$r(B, b_3)$
10			$w(B, a_3 + b_3)$
11		$w(C, c_2 + 200)$	
12		$r(B, b_2)$	
13		$w(B, b_2 + c_2)$	
14		commit	

Dabei bezeichnen A, B, und C Felder in der Datenbank, während a_i, b_i und c_i lokale Variablen darstellen. Weiters bezeichnet $r(\Gamma, \gamma)$ eine Leseoperation (der Wert des Feldes Γ wird aus der Datenbasis in eine lokale Variable γ gelesen) und $w(\Gamma, \gamma)$ eine Schreiboperation (der Wert γ wird in das Feld Γ in der Datenbasis geschrieben). Mit commit wird der erfolgreiche Abschluss einer Transaktion gekennzeichnet.

Nehmen Sie an, dass zu Beginn (Zeile 1) der relevante Datenbestand in der Datenbank aus den Werten $A = 100$, $B = 200$ und $C = 100$ 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 Operation an. Geben Sie jeweils die dazugehörige Zeilennummer der Historie an.

Lösung:

	Wert
4	$c_1 = 100$
5	$c_2 = 100$
6	$a_3 = 100$
7	$A = 200$
8	$A = 300$
9	$b_3 = 200$
10	$B = 300$
11	$C = 300$
12	$b_2 = 300$
13	$B = 400$

(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 Format

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

wobei Sie für die Log-Einträge für die BOT und commit nur die die `LSN`, die `TA`, den Operationsnamen (BOT bzw. commit), sowie die `PrevLSN` angeben müssen. D.h. die entsprechenden Log-Einträge haben das Format

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

Geben Sie bitte zu jedem Log-Eintrag die Nummer der dazugehörigen Zeile der Historie an.

Die *Redo*- und *Undo*-Informationen sollen logisch protokolliert werden, d.h. *relativ* zum Datenbestand mittels Addition bzw. Subtraktion. Zur besseren Lesbarkeit benützen Sie außerdem bitte die Schreibweise $\#i$ für die `LSN` bzw. `PrevLSN`. Nehmen Sie darüber hinaus bitte an, dass jedes Feld Γ auf der Seite P_Γ gespeichert wird.

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. Sollte es zu einem Eintrag keinen vorangegangenen Eintrag geben verwenden Sie bitte die Zahl 0 (ohne vorangestelltes #) als Wert.

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]`
1	$[\#1, T_1, \text{BOT}, 0]$
2	$[\#2, T_2, \text{BOT}, 0]$
3	$[\#3, T_3, \text{BOT}, 0]$
7	$[\#4, T_2, P_A, \text{A}{+}{=}100, \text{A}{-}{=}100, \#2]$
8	$[\#5, T_1, P_A, \text{A}{+}{=}100, \text{A}{-}{=}100, \#1]$
10	$[\#6, T_3, P_B, \text{B}{+}{=}100, \text{B}{-}{=}100, \#3]$
11	$[\#7, T_2, P_C, \text{C}{+}{=}200, \text{C}{-}{=}200, \#4]$
13	$[\#8, T_2, P_B, \text{B}{+}{=}100, \text{B}{-}{=}100, \#7]$
14	$[\#9, T_2, \text{commit}, \#8]$

(c) Nehmen Sie an, dass unmittelbar nach Zeile 14 ein System-Absturz mit anschließendem Wiederanlauf passiert. Geben Sie die Liste der beim Wiederanlauf erzeugten CRL-Einträge an. Verwenden Sie für CRL-Einträge das Format

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

wobei alle in Aufgabe (a) genannten Anforderungen weiter verwendet werden sollen (d.h. geben Sie insbesonders die *Redo*-Informationen in relativer Schreibweise mittels Addition bzw. Subtraktion an).

Lösung:

	Log: ⟨`LSN, TA, PageID, Redo, PrevLSN, UndoNextLSN`⟩
15	⟨#10, T_3, P_B, B-=100, #6, #3⟩
16	⟨#11, T_1, P_A, A-=100, #5, #1⟩
17	⟨#12, T_3, (BOT), #10, 0⟩
18	⟨#13, T_1, (BOT), #11, 0⟩

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

Lösung:

A :200; B :300; C :300

Aufgaben: Mehrbenutzersynchronisation

Aufgabe 2 (Serialisierbarkeit) *[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) $\mathcal{T}_1 = \{T_1, T_2, T_3, T_4\}$
$\mathcal{H}_1 = r_3(D) \to w_3(C) \to r_2(C) \to r_3(A) \to r_4(B) \to w_2(D) \to w_4(A) \to w_2(B) \to r_1(B) \to w_2(A) \to w_1(C) \to c_1 \to c_2 \to c_3 \to c_4$.

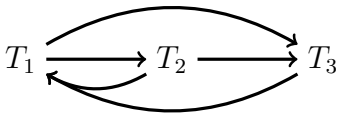
(b) $\mathcal{T}_2 = \{T_1, T_2, T_3, T_4, T_5, T_6\}$
$\mathcal{H}_2 = r_4(B) \to r_6(E) \to w_1(C) \to r_1(A) \to w_5(B) \to r_3(D) \to w_3(D) \to w_3(E) \to w_6(B) \to r_1(B) \to w_4(A) \to w_4(C) \to w_2(C) \to r_6(D) \to r_1(E) \to w_5(A) \to r_4(D) \to c_1 \to c_2 \to c_3 \to c_4 \to c_5 \to c_6$.

- 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 könnten.
- Falls die Historie serialisierbar ist, geben Sie *eine* mögliche Reihenfolge an. Andernfalls geben Sie eine (möglichst kleine) Menge an Transaktionen an welche man aus der Historie streichen müsste damit diese serialisierbar wird. Geben Sie anschließend eine mögliche serielle Reihenfolge an.

Beispiel: Für drei Transaktionen T_1, T_2, T_3 mit der Historie

$$w_1(A) \to r_2(A) \to w_3(A) \to w_1(B) \to w_2(B) \to w_1(A)$$

ergibt sich folgender Serialisierbarkeitsgraph



Für die Kanten ergeben sich folgende "Begründungen":

$T_1 \to T_2$:

- $w_1(A) \to r_2(A)$
- $w_1(B) \to w_2(B)$

$T_2 \to T_3$:

- $r_2(A) \to w_3(A)$

$T_1 \to T_3$:

- $w_1(A) \to w_3(A)$

$T_2 \to T_1$:

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

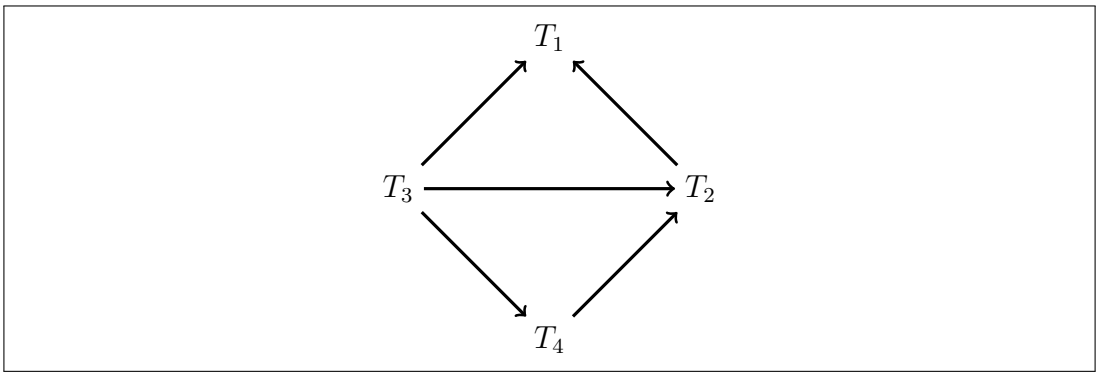
$T_3 \to T_1$:

- $w_3(A) \to w_1(A)$

Die Historie ist *nicht* serialisierbar. Nach Löschen der Transaktion T_1 wird sie jedoch serialisierbar. Eine gültige serielle Reihenfolge wäre dann T_2 vor T_3.

Lösung:

(a) **Serialisierbarkeitsgraph:**



"Begründung" für die Kanten:

$T_3 \to T_1$

- $w_3(C) \to w_1(C)$

$T_3 \to T_2$

- $r_3(D) \to w_2(D)$
- $w_3(C) \to r_2(C)$
- $r_3(A) \to w_2(A)$

$T_3 \to T_4$

- $r_3(A) \to w_4(A)$

$T_2 \to T_1$

- $r_2(C) \to w_1(C)$
- $w_2(B) \to r_1(B)$

$T_4 \to T_2$

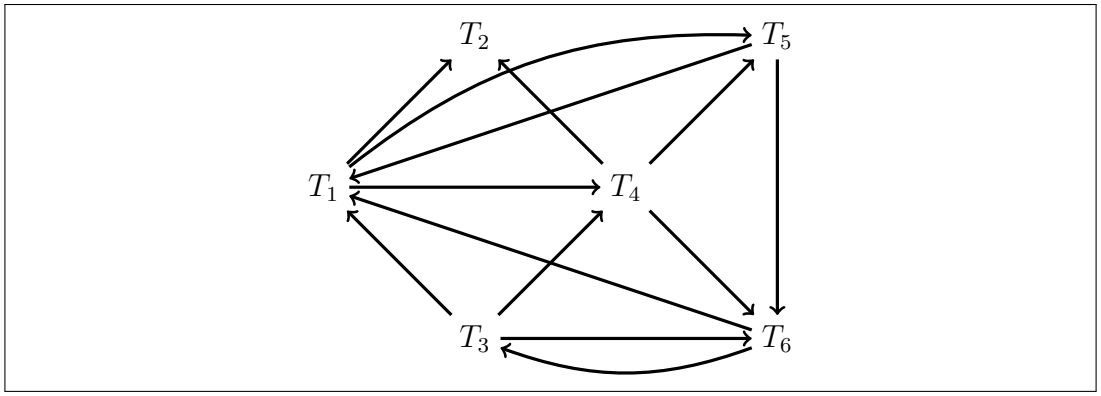
- $w_4(A) \to w_2(A)$
- $r_4(B) \to w_2(B)$

Serialisierbarkeit:

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

T_3 vor T_4 vor T_2 vor T_1

(b) **Serialisierbarkeitsgraph:**



"Begründung" für die Kanten:

$T_1 \to T_2$
- $w_1(C) \to w_2(C)$

$T_1 \to T_4$
- $w_1(C) \to w_4(C)$
- $r_1(A) \to w_4(A)$

$T_1 \to T_5$
- $r_1(A) \to w_5(A)$

$T_3 \to T_1$
- $w_3(E) \to r_1(E)$

$T_3 \to T_4$
- $w_3(D) \to r_4(D)$

$T_3 \to T_6$
- $w_3(D) \to r_6(D)$

$T_4 \to T_2$
- $w_4(C) \to w_2(C)$

$T_4 \to T_5$
- $w_4(A) \to w_5(A)$
- $r_4(B) \to w_5(B)$

$T_4 \to T_6$
- $r_4(B) \to w_6(B)$

$T_5 \to T_1$
- $w_5(B) \to r_1(B)$

$T_5 \to T_6$
- $w_5(B) \to w_6(B)$

$T_6 \to T_1$
- $w_6(B) \to r_1(B)$

$T_6 \to T_3$
- $r_6(E) \to w_3(E)$

Serialisierbarkeit:

Nein, die Historie ist *nicht* serialisierbar.

Es müssen immer mindestens zwei Transaktionen entfernt werden um eine serialisierbare Abfolge zu erhalten. Möglich wären $\{T_1, T_3\}, \{T_1, T_6\}, \{T_5, T_6\}$.

Entfernt man z.B. $\{T_1, T_3\}$ wäre eine gültige serielle Abfolge

$$T_4 \text{ vor } T_2 \text{ vor } T_5 \text{ vor } T_6$$

Aufgabe 3 (Zwei-Phasen-Sperrprotokoll) *[3 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:

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

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, und verwenden Sie $\texttt{wait}(i)$ um anzuzeigen, dass eine Transaktion angehalten wurde um auf eine Sperre zu warten.

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 zweiter 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:	$\texttt{grantX}_2(B)$

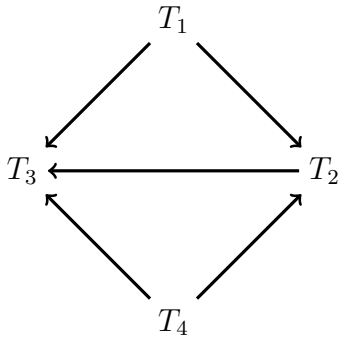
Lösung:

1:	$\texttt{lockS}_4(B)$
2:	$\texttt{grantS}_4(B)$
3:	$\texttt{lockS}_2(A)$
4:	$\texttt{grantS}_2(A)$
5:	$\texttt{lockS}_3(A)$
6:	$\texttt{grantS}_3(A)$
7:	$\texttt{lockX}_1(A)$
8:	$\texttt{wait}(1)$
9:	$\texttt{lockS}_2(B)$
10:	$\texttt{grantS}_2(B)$
11:	$\texttt{lockS}_3(C)$
12:	$\texttt{grantS}_3(C)$
13:	$\texttt{lockX}_2(C)$
14:	$\texttt{wait}(2)$
15:	$\texttt{lockX}_4(A)$
16:	$\texttt{wait}(4)$
17:	$\texttt{lockX}_3(C)$
18:	$\texttt{grantX}_3(C)$

(b) Zeichnen Sie den Wartegraphen am Ende der in Aufgabe (a) erstellten Liste an Schritten (d.h. nehmen Sie weiterhin an, dass keine Sperre wieder freigegeben wurde, d.h. nachfolgende Sperranforderungen von angehaltenen Transaktionen wurden nach wie vor noch nicht bearbeitet).

Lösung:

Wartegraph:



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

(d) Erklären Sie in ein paar kurzen Sätzen die Auswirkungen auf jede einzelne Transaktion falls T_3 alle Sperren freigibt. Sollten angehaltene Transaktionen dadurch weiterlaufen können, nehmen Sie an, dass ihre Sperranforderungen in der Reihenfolge abgearbeitet werden in denen sie ursprünglich gestellt wurden.

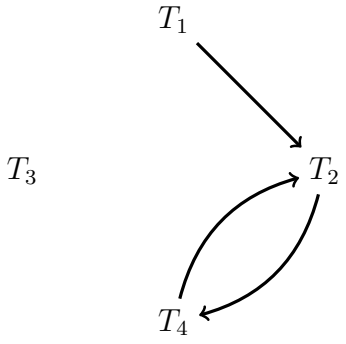
Lösung:

- T_1: T_1 wartet noch immer auf die Sperranforderung $\text{lockX}_1(A)$, da T_2 die Lesesperre auf A noch nicht freigegeben hat.
- T_2: T_2 kann weiterlaufen, und erhält die Sperranforderung $\texttt{lockX}_2(C)$. Bei der Sperranforderung $\texttt{lockX}_2(B)$ wird T_2 wiederum angehalten, da es auf die Freigabe der Lesesperre der Transaktion T_4 auf B warten muss.
- T_4: T_4 wartet noch immer auf die Sperranforderung $\texttt{lockX}_4(A)$, da Transaktion T_2 die Lesesperre auf A noch nicht freigegeben hat.

(e) Zeichnen Sie erneut den Wartegraphen.

Lösung:

Wartegraph:



(f) Besteht jetzt ein Deadlock? **Lösung: Ja**

Aufgabe 4 (Multi-Granularity Locking) *[2 Punkte]*

Betrachten Sie die Datenbasis-Hierarchie in Abbildung 1.

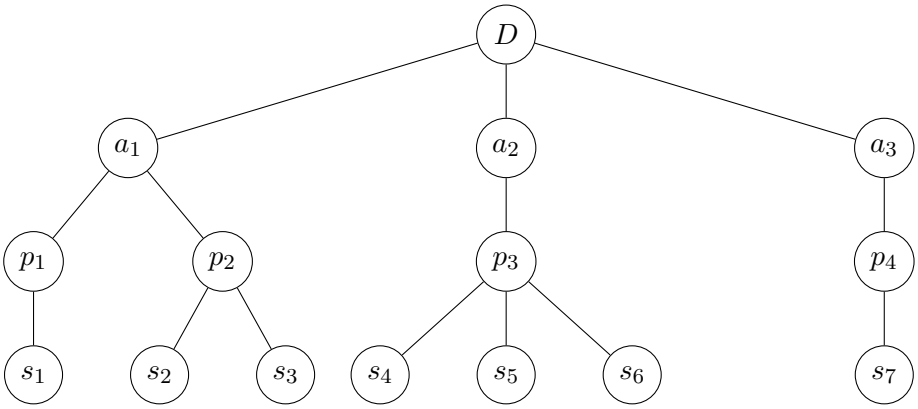


Abbildung 1: Datenbasis-Hierarchie zu Aufgabe 4

Beantworten Sie, welche der folgenden geplanten Sequenzen von Sperr-Anforderungen (bei zwei Transaktionen T_1 und T_2) zu Blockierungen bzw. Deadlocks führen. (Hier bedeutet (T_i, x, L), dass Transaktion T_i versucht, Knoten x in der Hierarchie mit einer Sperre vom Typ L zu belegen.)

Falls eine Transaktion blockiert wird:

- Geben Sie an welche Transaktion bei welcher Sperranforderung blockiert wird.
- Geben Sie außerdem an, warum die Sperranforderung nicht erfüllt werden kann (d.h. geben Sie an, welche von der anderen Transaktion gehaltene Sperre an welchem Knoten dazu führt dass sie Sperranforderung nicht erfüllt werden kann).
- Geben Sie die minimale Folge von Freigaben an, welche die andere Transaktion durchführen müsste damit die Sperre gewährt werden kann.

Hinweis: Unter Umständen werden nicht alle Sperren dieser Sequenzen auch tatsächlich angefordert, d.h.: Im Falle einer Blockierung einer Transaktion werden die weiter hinten liegenden Sperr-Anforderungen dieser Transaktion gar nicht mehr durchgeführt.

(a) (T_1, D, IS), (T_2, D, IX), (T_1, a_1, IS), (T_1, p_1, IS), (T_2, a_1, IX), (T_2, p_2, X), (T_1, s_1, S).

Lösung: Keine Blockierung, kein Deadlock

(b) (T_1, D, IX), (T_2, D, IX), (T_1, a_3, IX), (T_2, a_2, IX), (T_1, p_4, IX), (T_2, p_3, IX), (T_1, s_7, X), (T_2, s_5, X).

Lösung: Keine Blockierung, kein Deadlock

(c) (T_1, D, IX), (T_2, D, IX), (T_1, a_1, IX), (T_2, a_2, X), (T_1, p_2, X), (T_2, a_1, IX), (T_2, p_2, IX), (T_2, s_2, X):

Lösung: Es kommt zu einer Blockierung, kein Deadlock

- Transaktion T_2 wird bei Sperranforderung (T_2, p_2, IX) blockiert.
- Transaktion T_1 besitzt am Knoten p_2 bereits eine X-Sperre.
- Es reicht, wenn T_1 die X-Sperre auf Knoten p_2 aufgibt.

(d) $(T_1, D, IS), (T_2, D, IX), (T_1, a_1, S), (T_2, a_2, IX), (T_2, p_3, X), (T_1, a_2, S), (T_2, a_1, IX), (T_2, p_1, IX), (T_2, s_1, X)$:

Lösung: Es kommt zu einer Blockierung und zu einem Deadlock

- Transaktion T_1 wird bei Sperranforderung (T_1, a_2, S) blockiert.
- Dies geschieht da T_2 auf a_2 bereits eine IX-Sperre besitzt.
- T_2 müsste zuerst auf p_3 die X-Sperre aufgeben, und anschließend auf a_2 die IX-Sperre.
- Transaktion T_2 wird bei Sperranforderung (T_2, a_1, IX) blockiert.
- Dies geschieht da T_1 auf a_1 bereits eine S-Sperre besitzt.
- T_1 müsste zuerst auf a_1 die S-Sperre aufgeben.

Aufgabe 5 (Zeitstempelbasiertes Sperrverfahren) *[2 Punkte]*

Gegeben ist die unten angeführte Historie dreier Transaktionen T_1, T_2, T_3, welche auf drei Datenobjektion A, B und C zugreifen.

Nehmen Sie an dass in einem zeitstempelbasierten Sperrverfahren die Anfangswerte für `readTS` und `writeTS` für alle drei Werte 0 sind.

Verwenden Sie die Regeln Zeitstempel-basierende Synchronisation wie in der Vorlesung besprochen.

- Geben Sie die tatsächliche Historie der drei Transaktionen aus wie sie auf Grund der Synchronisation mittels Zeitstempel abgearbeitet werden.

  Nehmen Sie an, dass abgebrochene Transaktionen in der Reihenfolge wiederholt werden in der sie abgebrochen wurden, und dass eine Wiederholung beginnt sobald alle anderen Transaktionen abgeschlossen wurden.

- Geben Sie zu jeder Lese- oder Schreiboperation die Werte von `readTS`(A), `readTS`(b), `readTS`(C), `writeTS`(A), `writeTS`(b), `writeTS`(C) an. Als Zeitstempel für die Transaktionen nehmen Sie bitte die # ihres `BOT`.

#	T_1	T_2	T_3
1	BOT		
2		BOT	
3			BOT
4	$r_1(A)$		
5		$r_2(C)$	
6		$w_2(A)$	
7			$r_3(C)$
8	$w_1(C)$		
9	commit		
10		$r_2(B)$	
11			$w_3(C)$
12			commit
13		$w_2(C)$	
14		$w_2(A)$	
15		commit	

Lösung:

#	T_1	T_2	T_3	rTS(A)	wTS(A)	rTS(B)	wTS(B)	rTS(C)	wTS(C)
1	BOT			0	0	0	0	0	0
2		BOT		0	0	0	0	0	0
3			BOT	0	0	0	0	0	0
4	$r_1(A)$			1	0	0	0	0	0
5		$r_2(C)$		1	0	0	0	2	0
6		$w_2(A)$		1	2	0	0	2	0
7			$r_3(C)$	1	2	0	0	3	0
8	reset			1	2	0	0	3	0
10		$r_2(B)$		1	2	2	0	3	0
11			$w_3(C)$	1	2	2	0	3	3
12			commit	1	2	2	0	3	3
13		reset		1	0	2	0	3	3
13	BOT			1	0	2	0	3	3
14	$r_1(A)$			13	0	2	0	3	3
15	$w_1(C)$			13	0	2	0	3	13
16	commit			13	0	2	0	3	13
17		BOT		13	0	2	0	3	13
18		$r_2(C)$		13	0	2	0	17	13
19		$w_2(A)$		13	17	2	0	17	13
20		$r_2(B)$		13	17	17	0	17	13
21		$w_2(C)$		13	17	17	0	17	17
22		$w_2(A)$		13	17	17	0	17	17
23		commit		13	17	17	0	17	17

Aufgaben: EER-Diagramme

Aufgabe 6 (Überführung ins Relationenschema) *[2 Punkte]*

Überführen Sie das EER-Diagramm aus Abbildung 2 in ein Relationenschema. Beachten Sie, dass das EER-Diagramm auf der Musterlösung von Übungsblatt 1 basiert, jedoch nicht exakt gleich ist. Nullwerte sind nicht erlaubt (Sie können dabei annehmen, dass alle für einen Entitätstyp angegebenen Attribute für die Entitäten dieses Typs existieren). Verwenden Sie möglichst wenig Relationen. Unterstreichen Sie sämtliche Primärschlüssel, schreiben Sie die Fremdschlüssel kursiv und stellen Sie sicher, dass ein Fremdschlüssel eindeutig der passenden Relation zugeordnet werden kann.

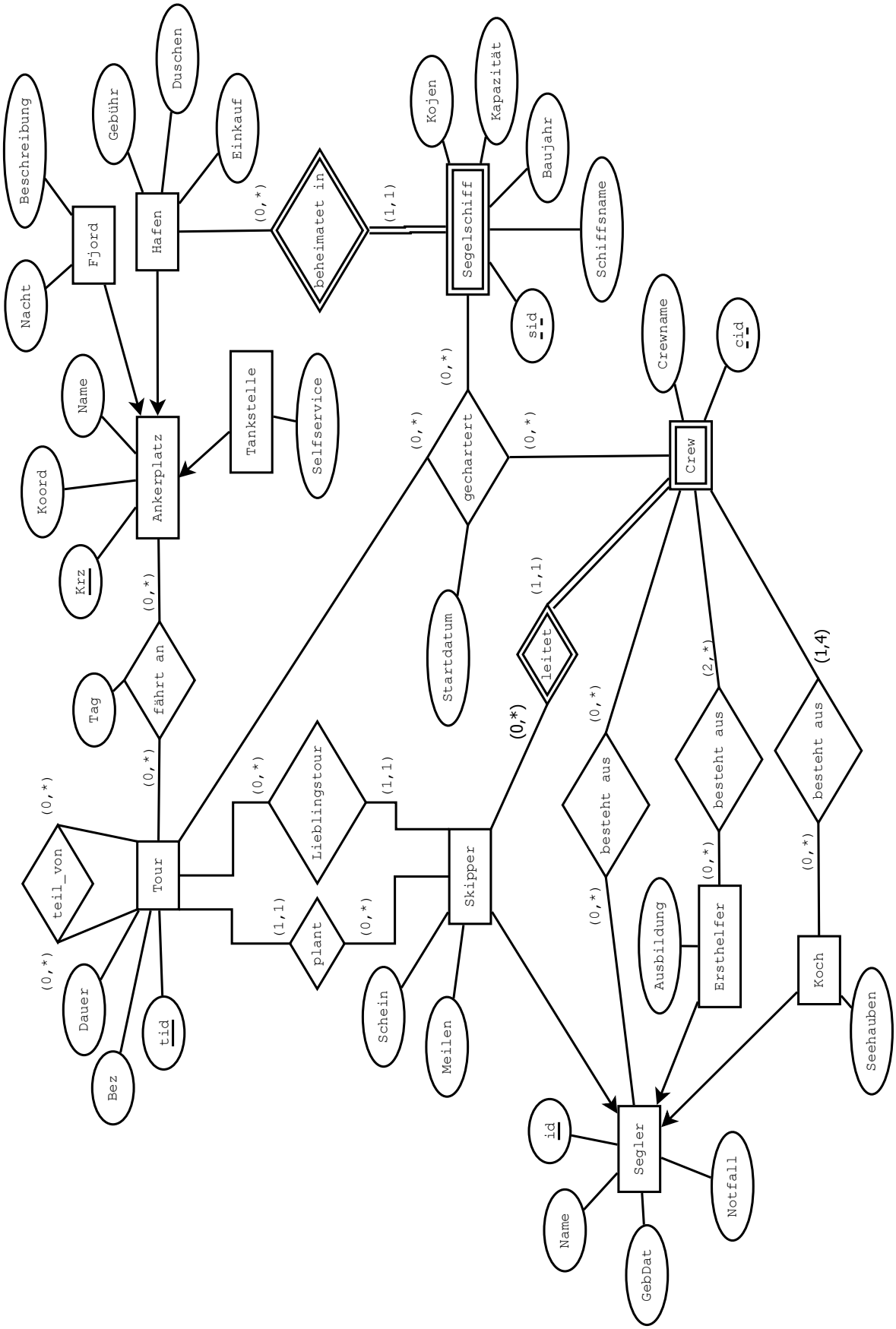


Abbildung 2: EER-Diagramm zu Aufgabe 6

Lösung:

```
Segler          (id, name, gebdatum, notfallkontakt)
Koch            (id:Segler.id, seehauben)
Ersthelfer      (id:Segler.id, ausbildung)
Skipper         (id:Segler.id, schein, seemeilen, lieblingstour:Tour.tid)
Crew            (leiter:Skipper.id, cid, crewname)
CrewMitglieder  (segler:Segler.id, cleiter:Crew.leiter, cid:Crew.cid)
CrewErsthelfer  (ersthelfer:Ersthelfer.id, cleiter:Crew.leiter,
                 cid:Crew.cid)
CrewKoch        (koch:Koch.id, cleiter:Crew.leiter, cid:Crew.cid)

Ankerplatz      (krz, koordinaten, name)
Tankstelle      (krz:Ankerplatz.krz,selfservice)
Fjord           (krz:Ankerplatz.krz,nacht,beschreibung)
Hafen           (krz:Ankerplatz.krz,gebuehr,duschen,einkauf)
Segelschiff     (hafen:Hafen.krz,sid,name,baujahr,kojen,kapazitaet)

Tour            (tid, bez, dauer, geplant_von:Skipper.id)
Teiltour        (tid:Tour.tid,teil_von:Tour.tid)
TourAnkerplatz  (tid:Tour.tid,krz:Ankerplatz.krz,tag)
Gechartert      (cleiter:Crew.leiter,cid:Crew.cid,hafen:Segelschiff.hafen,
                 sid:Segelschiff.sid,tid:Tour.tid,startdatum)
```