TU München, Fakultät für Informatik
Lehrstuhl III: Datenbanksysteme
Prof. Alfons Kemper, Ph.D.

Übung zur Vorlesung *Grundlagen: Datenbanken* im WS23/24

Christoph Anneser, Michael Jungmair, Stefan Lehner, Moritz Sichert, Lukas Vogel
(gdb@in.tum.de)
https://db.in.tum.de/teaching/ws2324/grundlagen/

Blatt Nr. 13

Bitte beachten Sie, dass die Gruppe 32 am Donnerstag, den 01.02.2024, ausfällt.

Hausaufgabe 1

Wofür stehen die vier Buchstaben ACID? Erklären Sie für jeden der vier Konzepte, warum es für eine Datenbank wichtig ist. Geben Sie dazu jeweils ein Beispiel an, was passieren könnte, wenn dieses Konzept nicht gelten würde.

Lösung:

- **A**tomicity
- **C**onsistency
- **I**solation
- **D**urability

Wenn eine Datenbank keine Atomarität garantieren kann, kann es zu inkonsistenten Daten kommen (unabhängig von der Konsistenzeigenschaft/Consistency). Hierzu dient eine Überweisung in einer Bank als Beispiel: Wenn eine Transaktion daraus besteht, einen Kontostand zu verringern und einen andern zu erhöhen, entsteht ein inkonsistenter Zustand, wenn nur eine der beiden Operationen tatsächlich gespeichert wird.

Eine Datenbank ist immer in einem konsistenten Zustand. Eine Transaktion überführt die Datenbank immer von einem konsistenten Zustand in einen anderen konsistenten Zustand. Somit ist sichergestellt, dass alle Integritätsbedingungen (z.B. Fremdschlüsselbedingungen) immer eingehalten werden. Wäre das nicht der Fall, könnte z.B. die referentielle Integrität verletzt werden, also z.B. Studenten nicht existierende Vorlesungen hören.

Bei dem Beispiel einer Überweisung kann es auch zu Problemen kommen, wenn die Datenbank Transaktionen nicht korrekt voneinander isoliert. Zwei parallele Überweisungen können gleichzeitig Kontostände ändern, was zu inkorrekten Kontoständen nach der Ausführung beider Transaktionen führen kann.

Wenn eine Datenbank eingesetzt wird, die keine Dauerhaftigkeit garantieren kann, kann nie darauf vertraut werden, dass ein commit eine Transaktion tatsächlich festschreibt. Das ist aber z.B. bei einem Geldautomaten notwendig, der erst Geld ausgeben sollte, sobald die Datenbank garantieren kann, dass die Abhebetransaktion verbucht ist.

Hausaufgabe 2

In Abbildung 1 ist die verzahnte Ausführung der beiden Transaktionen T_1 und T_2 und das zugehörige *Log* auf der Basis logischer Protokollierung gezeigt. Wie sähe das Log bei physischer Protokollierung aus, wenn die Datenobjekte A, B und C die Initialwerte 1000, 2000 und 3000 hätten?

Schritt	T_1	T_2	Log
			[LSN,TA,PageID,Redo,Undo,PrevLSN]
1.	**BOT**		[#1, T_1, **BOT**, 0]
2.	$r(A, a_1)$		
3.		**BOT**	[#2, T_2, **BOT**, 0]
4.		$r(C, c_2)$	
5.	$a_1 := a_1 - 50$		
6.	$w(A, a_1)$		[#3, T_1, P_A, A–=50, A+=50, #1]
7.		$c_2 := c_2 + 100$	
8.		$w(C, c_2)$	[#4, T_2, P_C, C+=100, C–=100, #2]
9.	$r(B, b_1)$		
10.	$b_1 := b_1 + 50$		
11.	$w(B, b_1)$		[#5, T_1, P_B, B+=50, B–=50, #3]
12.	**commit**		[#6, T_1, **commit**, #5]
13.		$r(A, a_2)$	
14.		$a_2 := a_2 - 100$	
15.		$w(A, a_2)$	[#7, T_2, P_A, A=100, A+=100, #4]
16.		**commit**	[#8, T_2, **commit**, #7]

Abbildung 1: Verzahnte Ausführung zweier Transaktionen und das erstellte Log

Lösung: Vgl. Übungsbuch

[#1, T_1, **BOT**, 0]
[#2, T_2, **BOT**, 0]
[#3, T_1, P_A, A=950, A=1000, #1]
[#4, T_2, P_C, C=3100, C=3000, #2]
[#5, T_1, P_B, B=2050, B=2000, #3]
[#6, T_1, **commit**, #5]
[#7, T_2, P_A, A=850, A=950, #4]
[#8, T_2, **commit**, #7]

Hausaufgabe 3

Sie verwenden ein Datenbanksystem mit Write-Ahead-Logging und der Strategie ¬*force* und *steal*. Die Datenbank verwaltet lediglich zwei Datenobjekte, X mit dem Anfangswert 10 und Y mit dem Anfangswert 100.

Sie starten die 3 Transaktionen T_1, T_2 und T_3 zum gleichen Zeitpunkt:

T_1	T_2	T_3
BOT	**BOT**	**BOT**
$r(X, x_1)$	$r(Y, y_2)$	$r(X, x_3)$
$x_1 := x_1 + 1$	$r(X, x_2)$	$x_3 := x_3 \cdot 10$
$w(X, x_1)$	$y_2 := y_2 \cdot 2$	$w(X, x_3)$
COMMIT	$x_2 := x_2 + 5$	**COMMIT**
	$w(Y, y_2)$	
	$w(X, x_2)$	
	COMMIT	

Während der Ausführung stürzt Ihre Datenbank ab. Sie wissen nicht, ob - und wenn ja, welche - Transaktionen festgeschrieben wurden. Sie wissen nur, dass die Datenbank ausschließlich *serielle Historien* erzeugt, also dass Transaktionen immer atomar ausgeführt werden und somit keine Verzahnung möglich ist. Bevor Sie die Datenbank neu starten, durchsuchen Sie die Festplatte und stellen fest, dass Y dort den Wert 200 hat. Nachdem die Datenbank neu gestartet wurde und der Recovery-Prozess abgeschlossen ist, liefert sie für X den Wert 110.

Sie wollen nun dem Fehler auf den Grund gehen:

a) Finden Sie zunächst anhand der Zwischenwerte für X und Y heraus, welche Transaktionen *winner* sind, und welche *loser*.

b) Geben Sie das Log an, wie es zum Zeitpunkt des Absturzes auf der Platte stand (verwenden Sie logische Protokollierung).

c) Geben Sie das Log nach Beendigung des Recovery-Prozesses an.

Lösung:

a) Zum Zeitpunkt des Absturzes hatte Y auf der Festplatte den Wert 200. Da Y zu Beginn den Wert 100 hatte, muss Transaktion T_2 zu diesem Zeipunkt schon gestartet und mindestens bis zur Aktion $w(Y, y_2)$ ausgeführt worden sein.

   Betrachten wir nun den Wert von X nach Abschluss der Recovery. Aus $X = 110$ folgt, dass zuerst T_1 (X hat nun den Zwischenwert 11) und dann T_3 (X hat nun den Endwert 110) ausgeführt wurde, T_2 aber nicht ausgeführt wurde! T_2 muss also eine Losertransaktion sein, welche während des Recovery-Prozesses wieder rückgängig gemacht wurde.

   Das Datenbanksystem hat die Transaktionen also in der logischen Reihenfolge T_1, T_3, T_2 ausgeführt, wobei T_1 und T_3 *winner* sind und T_2 *loser* ist.

b) Hier ein mögliches Log. Bitte beachten: auf die Festplatte wird nur das Log selbst (also die „Log“-Spalte) geschrieben.

Schritt	T_1	T_2	T_3	Log
1.	**BOT**			$[\#1, T_1, \textbf{BOT}, 0]$
2.	$r(X, x_1)$			
3.	$x_1 := x_1 + 1$			
4.	$w(X, x_1)$			$[\#2, T_1, P_X, X \mathrel{+}= 1, X \mathrel{-}= 1, \#1]$
5.	**commit**			$[\#3, T_1, \textbf{commit}, \#2]$
6.			**BOT**	$[\#4, T_3, \textbf{BOT}, 0]$
7.			$r(X, x_3)$	
8.			$x_3 := x_3 \cdot 10$	
9.			$w(x, x_3)$	$[\#5, T_3, P_X, X \cdot= 10, X \mathrel{/}= 10, \#4]$
10.			**commit**	$[\#6, T_3, \textbf{commit}, \#5]$
11.		**BOT**		$[\#7, T_2, \textbf{BOT}, 0]$
12.		$r(Y, y_2)$		
13.		$r(X, x_2)$		
14.		$y_2 := y_2 \cdot 2$		
15.		$x_2 := x_2 + 5$		
16.		$w(Y, y_2)$		$[\#8, T_2, P_Y, Y \cdot= 2, Y \mathrel{/}= 2, \#7]$
17.		$w(X, x_2)$		$[\#9, T_2, P_X, X \mathrel{+}= 5, X \mathrel{-}= 5, \#8]$

   Ob Schritt 17 tatsächlich ausgeführt wurde, können wir nicht wissen. Der Vollständigkeit zuliebe gehen wir hier vom *worst case* aus. Die Datenbank stürzt direkt vor dem

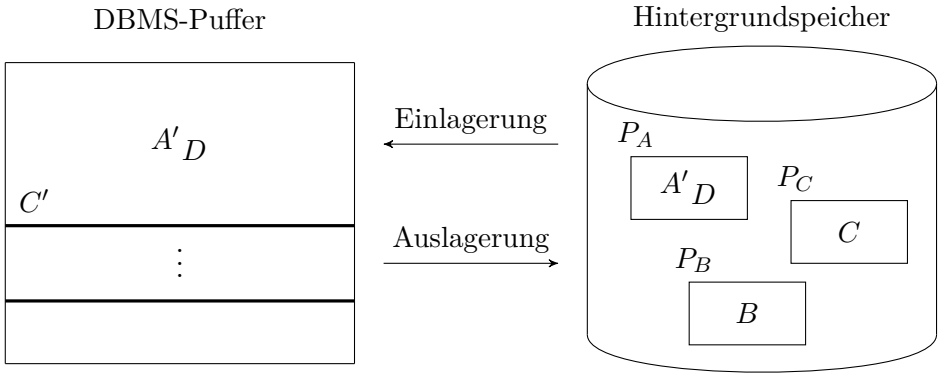
commit von T_2 ab.

c) Da T_2 eine *loser*-Transaktion ist, wird die Datenbank während der Recovery in der *Undo-Phase Compensation Log Records* schreiben, um die partiell durchgeführte Transaktion rückgängig zu machen. Das endgültige Log sieht dann wie folgt aus:

$$[\#1, T_1, \textbf{BOT}, 0]$$
$$[\#2, T_1, P_X, X \mathrel{+}= 1, X \mathrel{-}= 1, \#1]$$
$$[\#3, T_1, \textbf{commit}, \#2]$$
$$[\#4, T_3, \textbf{BOT}, 0]$$
$$[\#5, T_3, P_X, X \mathrel{\cdot}= 10, X \mathrel{/}= 10, \#4]$$
$$[\#6, T_3, \textbf{commit}, \#5]$$
$$[\#7, T_2, \textbf{BOT}, 0]$$
$$[\#8, T_2, P_Y, Y \mathrel{\cdot}= 2, Y \mathrel{/}= 2, \#7]$$
$$[\#9, T_2, P_X, X \mathrel{+}= 5, X \mathrel{-}= 5, \#8]$$
$$\langle \#9', T_2, P_X, X \mathrel{-}= 5, \#9, \#8 \rangle$$
$$\langle \#8', T_2, P_Y, Y \mathrel{/}= 2, \#9', \#7 \rangle$$
$$\langle \#7', T_2, -, -, \#8', 0 \rangle$$

Hausaufgabe 4

Demonstrieren Sie anhand eines Beispiels, dass man die Strategien *force* und ¬*steal* nicht kombinieren kann, wenn parallele Transaktionen gleichzeitig Änderungen an Datenobjekten innerhalb einer Seite durchführen. Betrachten Sie dazu z.B. die unten dargestellte Seitenbelegung, bei der die Seite P_A die beiden Datensätze A und D enthält. Entwerfen Sie eine verzahnte Ausführung zweier Transaktionen, bei der eine Kombination aus *force* und ¬*steal* ausgeschlossen ist.



Lösung:

Folgendes Beispiel zeigt, warum man *force* und ¬*steal* nicht kombinieren kann:

Schritt	T_1	T_2
1.	**BOT**	
2.		**BOT**
3.	read(A)	
4.		read(D)
5.		write(D)
6.	write(A)	
7.	**commit**	

In Schritt 7 führt T_1 einen commit aus. Aufgrund der *force*-Strategie müssen nun alle von dieser Transaktion geänderten Seiten ausgelagert werden. Im Beispiel hat T_1 nur P_A geändert, also muss diese ausgelagert werden. Gleichzeitig existiert aber noch eine laufende Transaktion T_2, die ebenfalls die Seite P_A verändert hat. Wegen der $\neg$*steal*-Strategie dürfen keine Seiten ausgelagert werden, die von noch nicht beendeten Transaktionen bearbeitet wurden. Im Beispiel muss also P_A zwingend ausgelagert werden, da T_1 einen commit ausführt, aber P_A darf nicht ausgelagert werden, da sie von der noch laufenden Transaktion T_2 verändert wurde, was einen Widerspruch darstellt.