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

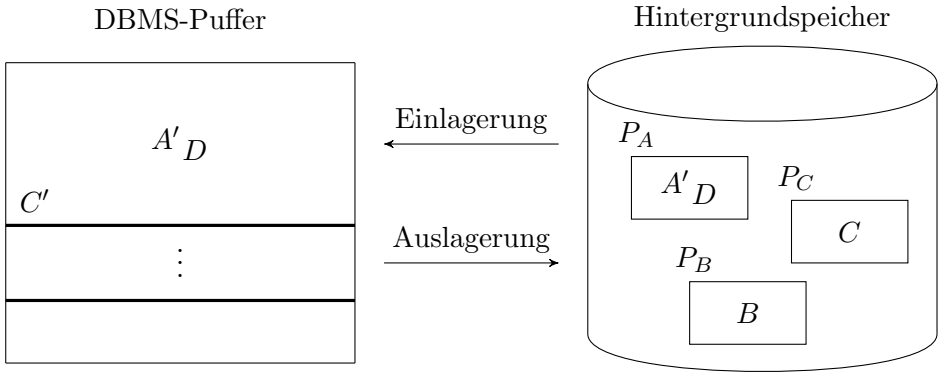
Übung zur Vorlesung *Einsatz und Realisierung von Datenbanken* im SoSe20

Maximilian {Bandle, Schüle}, Josef Schmeißer (i3erdb@in.tum.de)
http://db.in.tum.de/teaching/ss20/impldb/

Blatt Nr. 01

Hausaufgabe 1

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 ¬*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.

Hausaufgabe 2

Überlegen Sie sich, bei welcher Seitenersetzungsstrategie bei einem Wiederanlauf eine *Redo-* bzw. eine *Undo*-Phase notwendig ist. Verwenden Sie in diesem Zusammenhang den Begriff *dirty*. Welche der beiden Phasen entfällt bei einer Hauptspeicherdatenbank?

Lösung:

- $\neg force$: erfordert eine *Redo*-Phase, da nach einem erfolgreichen *commit* die Änderungen einer somit abgeschlossenen Transaktion (*Winner*) nicht in der Datenbasis materialisiert sein müssen.
- *steal*: erfordert eine *Undo*-Phase, da Änderungen einer noch nicht abgeschlossenen Transaktion (*Loser*) bereits auf der Datenbasis (im Hintergrundspeicher) materialisiert sein können, wenn die entsprechende Seite ausgelagert worden ist, da modifizierte Seiten (*dirty*) von einer Auslagerung nicht ausgeschlossen sind. Eine Seite ist *dirty*, sobald ihr Inhalt geändert worden ist und nicht mehr mit der materialisierten Datenbasis übereinstimmt.
- Bei einer Hauptspeicherdatenbank befindet sich die materialisierte Datenbasis ausschließlich im Hauptspeicher ($\neg force$ und $\neg steal$), ein Verlust des Hauptspeichers erfordert ein *Redo* der vollständigen Datenbasis, dafür ist eine *Undo*-Phase nicht nötig.

Hausaufgabe 3

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

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

Hausaufgabe 4

Leider erhalten wir einen Fehler mit Hauptspeicherverlust der in Abbildung 1 gezeigten Ausführung nach Schritt 13. Welche Transaktion ist ein *Winner*, welche ein *Loser*? Geben Sie alle nötigen Kompensations-Rekorde (CLR) an.

Lösung: T_1 konnte erfolgreich abschließen und ist somit ein *Winner*, T_2 konnte kein *commit* ausführen und ist ein *Loser*. Daher müssen wir für alle Änderungen von T_2 einen CLR anlegen. Es erfolgte eine Änderung von T_2 in Schritt 8, diese müssen wir, nach einem erfolgten Wiederanlauf aller bis Schritt 13 geschriebenen Log-Dateien, zurücksetzen.

$$<\#4', T_2, P_C, C{-}{=}100, \#4, \#2>$$
$$<\#2', T_2, -, -, \#4', 0>$$

Hausaufgabe 5

Warum ist es für die Erzielung der Idempotenz der Redo-Phase notwendig, die – und nur die – LSN einer tatsächlich durchgeführten Redo-Operation in der betreffenden Seite zu vermerken? Zeigen Sie für die folgenden Szenarien anhand von Beispielen mit logischer Protokollierung, dass die Idempotenz nicht sichergestellt werden kann.

a) LSN-Eintäge werden in der Redo-Phase nicht auf Datenseiten geschrieben.

b) LSN-Einträge von Log-Records, für die die Redo-Operation nicht ausgeführt wird, werden trotzdem in die Datenseiten übertragen.

Beantworten Sie außerdem folgende Frage:

c) Wie wird die Idempotenz der Undo-Phase sichergestellt, wenn ein Kompensationseintrag geschrieben wurde und dann noch vor der Ausführung des Undo das Datenbanksystem abstürzt?

Lösung:

a) Als Beispiel seien folgende Logeinträge und die Seite P_1 gegeben:

$$[\#2, T_1, P_1, A \mathrel{+}= 1, A \mathrel{-}= 1, \#1]$$
$$[\#3, T_1, P_1, B \mathrel{+}= 2, B \mathrel{-}= 2, \#2]$$
$$[\#4, T_1, \textbf{commit}, \#3]$$

#1	A=10 B=20

Nun stürzt die Datenbank ab nachdem die Daten von Logeintrag #2 auf die Seite P_1 geschrieben wurden. Diese Seite sieht nach dem Absturz folgendermaßen aus:

#2	A=11 B=20

Beim Wiederanlauf muss die Transaktion T_1 wiederhergestellt werden, da ihr commit im Log enthalten ist und sie somit eine Winner-Transaktion ist. Dazu werden die Logeinträge wieder von vorne bearbeitet. Für den ersten Logeintrag wird an der Seite nichts geändert, da die LSN der Seite (#2) nicht kleiner ist als die des Logeintrags (#2). Der zweite Logeintrag hingegen muss bearbeitet werden. Dazu wird die Redo-Operation ausgeführt. Im Szenario dieser Aufgabe wird verlangt, dass in der Redo-Phase keine LSNs verändert werden. Also wird die Redo-Operation des zweiten Logeintrags ausgeführt (also $B \mathrel{+}= 2$) ohne die LSN zu ändern. Danach sieht die Seite folgendermaßen aus:

#2	A=11 B=22

Wenn die Datenbank nun wieder abstürzt, kommt es zu einer Verletzung der Idempotenz. Wie auch beim ersten Absturz, muss die Winner-Transaktion T_1 wiederhergestellt werden. Dazu werden die Logeinträge wieder nacheinander abgearbeitet wobei der erste übersprungen werden kann. Die LSN des zweiten Logeintrags (#3) ist aber größer als die, die auf der Seite gespeichert ist (#2), weswegen die Redo-Operation $B \mathrel{+}= 2$ wieder ausgeführt wird. Danach enthält die Seite folgende Daten:

#2	A=11 B=24

Man kann sehen, dass die Redo-Operation des zweiten Logeintrags fälschlicherweise mehrmals ausgeführt wurde, wenn die LSN-Einträge in der Redo-Phase nicht auf die Seite geschrieben werden.

b) Als Beispiel seien folgende Logeinträge (identisch zur letzten Teilaufgabe) und die Seite P_1 gegeben:

$$[\#2, T_1, P_1, A \mathrel{+}= 1, A \mathrel{-}= 1, \#1]$$
$$[\#3, T_1, P_1, B \mathrel{+}= 2, B \mathrel{-}= 2, \#2]$$
$$[\#4, T_1, \textbf{commit}, \#3]$$

#3	A=11 B=22

Nun stürzt die Datenbank ab. Die Transaktion T_1 muss wiederhergestellt werden, da sie eine Winner-Transaktion ist. Dazu werden die Logeinträge von vorne abgearbeitet. Die Redo-Operation des ersten Logeintrags muss nicht ausgeführt werden, da die LSN auf der Seite (#3) größer ist als die des Logeintrags (#2). In dem Szenario dieser Teilaufgabe ist vorgegeben, dass die LSN für nicht ausgeführte Redo-Operationen aber trotzdem auf die Seite geschrieben werden. Dementsprechend sieht die Seite nach der Bearbeitung des ersten Logeintrags folgendermaßen aus:

#2	A=11 B=22

Wenn die Datenbank jetzt wieder abstürzt, wird die Idempotenz verletzt. Wenn die Logeinträge durchlaufen werden, kann der erste Eintrag übersprungen werden. Beim zweiten Logeintrag ist allerdings die LSN (#3) größer als die LSN auf der Seite (#2) weswegen die Redo-Operation ausgeführt werden muss. Dies folgt zu folgendem Inhalt der Seite:

#3	A=11 B=24

Man kann sehen, dass die Redo-Operation des zweiten Logeintrags wieder fälschlicherweise doppelt ausgeführt wurde, da während des ersten Redos die LSN einer übersprungenen Redo-Operation auf die Seite geschrieben wurde.

Anhand der beiden Szenarien kann man sehen, dass nur die LSN einer tatsächlich durchgeführten Redo-Operation auf die Seite geschrieben werden darf, damit die Idempotenz sichergestellt werden kann.

c) Wie alle anderen Logeinträge auch erhalten CLRs eine LSN. Diese kann bei einem Wiederanlauf mit der LSN auf einer Seite verglichen werden, um zu entscheiden, ob die Undo-Operation, die durch den CLR repräsentiert wird, ausgeführt werden muss. Werden die LSNs für CLRs nur dann auf die Seite geschrieben, wenn das Undo ausgeführt wurde, ist somit auch die Idempotenz der Undo-Phase garantiert.