



Übung zur Vorlesung *Einsatz und Realisierung von Datenbanksystemen im SoSe17*

Maximilian E. Schüle (schuele@in.tum.de)
<http://db.in.tum.de/teaching/ss17/impldb/>

Blatt Nr. 01

Hausaufgabe 1

Demonstrieren Sie anhand eines Beispiels, dass man die Strategien *force* und \neg *steal* nicht kombinieren kann, wenn parallele Transaktionen gleichzeitig Änderungen an Datenobjekten innerhalb einer Seite durchführen. Betrachten Sie dazu z.B. die in Abbildung 1 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 \neg *steal* ausgeschlossen ist.

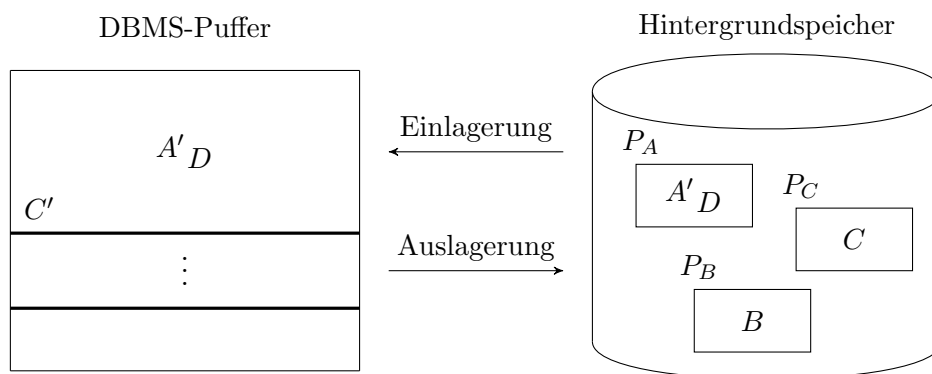


Abbildung 1: Schematische Darstellung der (zweistufigen) Speicherhierarchie

Lösung: Vgl. Übungsbuch

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

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 (*Looser*) 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, ein Verlust des Hauptspeichers erfordert ein *Redo* der vollständigen Datenbasis, dafür ist eine *Undo*-Phase nicht nötig.

Hausaufgabe 3

In Abbildung 2 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 2: 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 4

Leider erhalten wir einen Fehler mit Hauptspeicherverlust der in Abbildung 2 gezeigten Ausführung nach Schritt 13. Welche Transaktion ist ein *Winner*, welche ein *Looser*? 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 *Looser*. 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.

$\langle \#4', T_2, P_C, C-=100, \#4, \#2 \rangle$
 $\langle \#2', T_2, -, -, \#4', 0 \rangle$

Hausaufgabe 5

Zeigen Sie, dass es für die Erzielung der Idempotenz der *Redo*-Phase notwendig ist, die – und nur die – LSN einer tatsächlich durchgeführten *Redo*-Operation in der betreffenden Seite zu vermerken.

Was würde passieren, wenn man in der *Redo*-Phase gar keine LSN-Einträge in die Daten-seiten schriebe?

Was wäre, wenn man auch LSN-Einträge von Log-Records, für die die *Redo*-Operation nicht ausgeführt wird, in die Datenseiten übertragen würde?

Was passiert, wenn der Kompensationseintrag geschrieben wurde, und dann noch vor der Ausführung des *Undo* das Datenbanksystem abstürzt?

Lösung: Vgl. Übungsbuch:

In dieser Aufgabe beweisen wir die Idempotenz der Redo-Phase anhand von zwei Beispielen. Dazu zeigen wir, dass die LSN dann und nur genau dann auf die Seite übertragen wird, wenn die zugehörige Redo-Operation ausgeführt worden ist.

Folglich sind zwei Schritte zu zeigen:

Teilaufgabe 1:

Was passiert, wenn keine LSN geschrieben werden: keine Idempotenz, alle Änderungen des Logs werden bei einem Wiederanlauf vollzogen, eventuell werden Redo-Operationen auf dem *After-Image*. Beim *After-Image* handelt es sich um die modifizierten aus dem Hintergrundspeicher geladenen und somit materialisierten Seiten. Diese enthalten (ohne die LSN) keinerlei Informationen, ob Redo-Operationen bereits auf dieser Seite ausgeführt worden sind.

Teilaufgabe 2: Wann schreibe ich LSN raus?

Beispiel: Absturz der materialisierten Datenbasis, danach Wiederanlauf. Redo erfolgt nur für die Log-Records, die noch nicht materialisiert sind (sowohl für Looser als auch für Winner). Dazu vergleichen wir die LSN im Log-Record mit der der Seite. Ist die LSN des Log-Records größer der der Seite, muss eine Redo-Operation durchgeführt werden. Jetzt ist die Idee, was passierte, wenn die LSN jedes Log-Records in die Seite geschrieben wird, unabhängig davon ob der zugehörige Redo-Eintrag ausgeführt worden ist. Das ginge zwar beim ersten Mal gut, da kein Redo notwendig ist. Beim nächsten Mal ist nun die kleinere LSN vom vorherigen Log-Record eingetragen und fälschlicherweise wird ein Redo ausgeführt.

Kurz gefasst: Redo nur, wenn LSN vom Log-Record größer als das der Seite der materialisierten Datenbasis ist, und dann auch die LSN schreiben.

Teilaufgabe 3: CLR auf Platte geschrieben, konnte aber nicht ausgeführt werden. Dann wird der CLR beim Wiederanlauf ausgeführt.

CLRs erhalten ebenfalls eine fortlaufende LSN, echt größer als alle vorherigen Log-Records zuvor. Damit genügt wieder ein Vergleich der LSN um zu wissen, welche CLR bereits ausgeführt worden sind und welche nicht. Somit werden CLR nie fälschlicherweise wiederholt. Wegen Write-Ahead-Logging darf ich eine Seite erst auslagern, wenn alle zugehörigen Log-Einträge bereits herausgeschrieben sind. Somit ist sichergestellt, dass zu einer Änderung auf Platte auch der entsprechende CLR geschrieben ist.