Grundlagen Datenbanken: Übung 13

Gruppe 20 & 21

Tanmay Deshpande

ge94vem@mytum.de

QR-Code für die Folien

Wiederholung

Woche 13

Transaktionsverwaltung

- Transkationen = Menge von Operationen, die als eine Einheit ausgeführt werden
- Operationen = Änderungen/Zugriffe auf die Datenbasis
- Wir wollen Fehler in Transaktionen beheben können (Recovery) und Transaktionen synchronisieren
- Eigenschaften von Transaktionen:

  - Atomicity („ganz oder gar nicht“)
  - Consistency (Datenbasis soll vor und nach der Transaktion im konsistenten Zustand bleiben)
  - Isolation (parallel ausgeführte Transaktionen laufen unabhängig voneinander)
  - Durability (Änderungen zu der Datenbasis durch Transkationen sollen erhalten bleiben)

Operationen

- **begin of transaction (BOT)**
  Beginn von Befehlsfolge einer Transaktion
- **commit**
  Änderung zu der Datenbasis werden übernommen und Transaktion wird beendet
- **abort**
  Transkation beenden und Änderungen zur Datenbasis durch Transaktion zurücksetzen
- **define savepoint**
  Sicherungspunkte für aktive Transaktionen definieren, damit beim rollback nicht die ganze Transaktion neu ausgeführt werden muss
- **backup transaction**
  rollback zu letzter Sicherungspunkt

Abschluss einer Transaktion

Für den Abschluss einer Transaktion gibt es zwei Möglichkeiten:

1. Den erfolgreichen Abschluss durch ein **commit**.

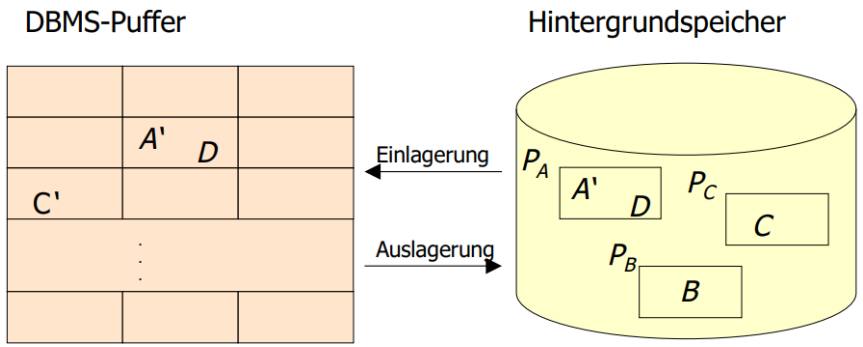
2. Den erfolglosen Abschluss durch ein **abort**.

Fehlerklassifikation

1. Lokaler Fehler in einer noch nicht festgeschriebenen (committed) Transaktion
   - Wirkung muss zurückgesetzt werden
   - **R1-Recovery**
2. Fehler mit Hauptspeicherverlust
   - Abgeschlossene TAs müssen erhalten bleiben
     - **R2-Recovery → redo**
   - Noch nicht abgeschlossene TAs müssen zurückgesetzt werden
     - **R3-Recovery → undo**
3. Fehler mit Hintergrundspeicherverlust
   - **R4-Recovery**

Zweistufige Speicherhierarchie

- Daten im Hintergrundspeicher gespeichert aber im Puffer bearbeitet
- Daten sollen nicht zu früh commited werden (z.B., falls es einen abort gibt)
- Verschiedene Strategien, wie man Seiten im Puffer ersetzen könnte:
  - **¬*steal***: Seiten, die von einer noch aktiven Transkation modifiziert wurden, werden nicht ersetzt
  - ***steal***: Jede nicht fixierte Seite kann ersetzt werden
- Verschiedene Strategien, wie man Änderungen abgeschlossener Transaktionen einbringen könnte:
  - ***force***: Nach Transaktionsende werden Änderungen auf dem Hintergrundspeicher geschrieben
  - **¬*force***: Geänderte Seiten dürfen im Puffer bleiben



	force	¬force
¬steal	kein Undo kein Redo	Redo kein Undo
steal	kein Redo Undo	Redo Undo

Logging

- **Struktur eines Log-Eintrags:** [LSN, TA, PageID, Redo, Undo, PrevLSN]
- **Logische Protokollierung:** Wie wird Before-Image zu After-Image geändert
- **Physische Protokollierung:** Wie sieht Before-Image und After-Image aus

- **LSN** = Log Sequence Number. Eindeutige Kennung für einen Eintrag. Bsp: #3
- **TA** = Transaktionskennung. Bsp: T_A
- **PageID** = Seite, auf die eine Transaktion zugreift. Bsp: P_A
- **Redo** = Wie wurden die Daten verändert (logische Protokollierung), bsp: A += 10/ Zustand der Daten nach der Änderung (physische Protokollierung), bsp: A = 10
- **Undo** = Wie können Änderungen rückgängig gemacht werden (logische Protokollierung), bsp: A -= 10 / Zustand der Daten vor der Änderung (physische Protokollierung), bsp: A = 0
- **PrevLSN** = LSN der vorherigen Transaktion. Bsp: #1

- Bei **BOT**, **commit**, **abort** sieht der Log-Eintrag so aus:

  [LSN, TA, **BOT/commit/abort** , PrevLSN]

Logging

Beispiel einer Log-Datei

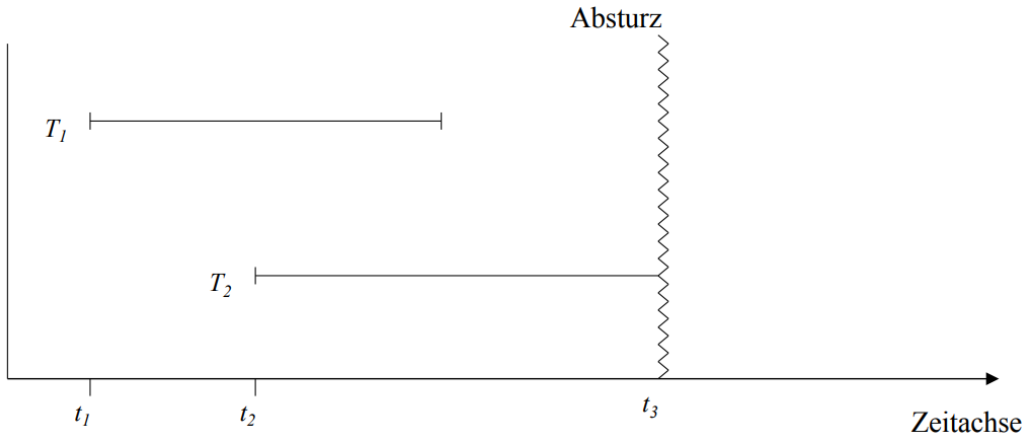
Schritt	T_1	T_2	Log
			[LSN, TA, PageID, Redo, Undo, PrevLSN]
1.	**BOT**		[#1,T_1,**BOT**,0]
2.	$r(A,a_1)$		
3.		**BOT**	[#2,T_2,**BOT**,0]
4.		$r(C,c_2)$	
5.	$a_1 := a_1 - 50$		
6.	$w(A,a_1)$		[#3,T_1,P_A,A-=50,A+=50,#1]
7.		$c_2 := c_2 + 100$	
8.		$w(C,c_2)$	[#4,T_2,P_C,C+=100,C-=100,#2]
9.	$r(B,b_1)$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

Write-Ahead Logging (WAL)

- Log-Einträge werden doppelt in Log-Datei (R1, R2 und R3 Recovery) und Log-Archiv gespeichert (R4 Recovery)
- WAL Prinzip:
  - Bevor eine Transaktion commited wird, müssen seine Log-Einträge in Log-Datei und Log-Archiv gespeichert werden, um Redo zu ermöglichen
  - Bevor eine modifizierte Seite ausgelagert wird, müssen alle Log-Einträge, die diese Seite modifizieren, in Log-Datei und Log-Archiv gespeichert werden, um Undo zu ermöglichen

Systemabsturz

- Winner = Transkationen mit erfolgreichem commit. Diese Änderungen sollen erhalten bleiben
- Loser = Transkationen, die unterbrochen wurden. Müssen rückgängig gemacht werden
- Nach dem Systemabsturz:
  - Analysiere Log-Datei um Winner und Loser zu ermitteln
  - Redo von Winner und Loser
  - Undo von Loser



- Kompensationseinträge für Undos einführen:
  <LSN‘,TA,PageID,Redo,PrevLSN,UndoNextLSN>

  Bsp: Undo zu [#7,T2,P_A, A+=100, A-=100, #6]
  <#7‘,T2,P_A,A-=100,#7,#6>

Aufgaben

Woche 13

Aufgabe 01

- 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

Lösungsvorschlag 01

- **Atomicity:**
  Bei einer Banküberweisung, muss der Kontostand von dem Absender verringert und der Kontostand von dem Empfänger erhöht werden
  Wenn es nicht atomar ausgeführt wurde, könnte es sein, dass der Absender geld verliert, ohne dass der Empfänger es bekommt
- **Konsistenz:**
  Verletzung referenzieller Integrität wie z.B. Studenten die nicht existierende Vorlesungen hören
- **Isolation:**
  Bei einer Banküberweisung versuchen zwei parallele Transkationen den Kontostand gleichzeitig zu ändern, was zu einem unerwarteten Ergebnis führt
- **Dauerhaftigkeit:**
  Wenn man nicht garantieren kann, dass alle commits tatsächlich dauerhaft festgeschrieben werden, kann nie auf eine Datenbank vertraut werden. Eine ATM kann nur dann Geld ausgeben, wenn die Abhebetransaktion commited wurde

Aufgabe 02

- Initialwerte: A = 1000, B = 2000, C = 3000
- Schreibe die Log-Einträge mit physischer Protokollierung anstatt logischer Protokollierung

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

Lösungsvorschlag 02

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

Aufgabe 03

- WAL, $\neg force, steal$
- Anfangswerte: X = 10, Y = 100
- Während der Ausführung stürzt Ihre Datenbank ab. Sie wissen nicht welche Transaktionen commited wurden
- keine Verzahnung möglich.
- Sie stellen fest, dass gleich nach dem Systemabsturz Y den Wert 200 hat.
- Nachdem die Datenbank neu gestartet wurde und der Recovery-Prozess abgeschlossen ist, liefert sie für X den Wert 110.

- a) Finden Sie die Winner und Loser

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	

Lösungsvorschlag 03a

- Zum Zeitpunkt des Absturzes war Y = 200 auf die Festplatte
- Da zu Beginn Y = 100 war, muss Transaktion T2 zu diesem Zeipunkt schon gestartet und mindestens bis zur Aktion w(Y, y2) ausgeführt worden sein.
- Aus X = 110 folgt, dass zuerst T1 (X hat nun den Zwischenwert 11) und dann T3 (X hat nun den Endwert 110) ausgeführt wurde, T2 aber nicht ausgeführt wurde!
- T2 muss also eine Losertransaktion sein, welche während des Recovery-Prozesses wieder rückgängig gemacht wurde.
- Das Datenbanksystem hat die Transaktionen also in der logischen Reihenfolge T1, T3, T2 ausgeführt, wobei T1 und T3 winner sind und T2 loser ist.

Aufgabe 03

- WAL, $\neg force, steal$
- Anfangswerte: X = 10, Y = 100
- Während der Ausführung stürzt Ihre Datenbank ab. Sie wissen nicht welche Transaktionen commited wurden
- keine Verzahnung möglich.
- Sie stellen fest, dass nach dem Systemabsturz Y den Wert 200 hat.
- Nachdem die Datenbank neu gestartet wurde und der Recovery-Prozess abgeschlossen ist, liefert sie für X den Wert 110.
- b) Geben Sie das Log an, wie es zum Zeitpunkt des Absturzes auf der Platte stand (verwenden Sie logische Protokollierung).

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	

Lösungsvorschlag 03b

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.			$w(x, x_3)$	$[\#5, T_3, P_X, X \mathrel{\cdot}= 10, X \mathrel{/}= 10, \#4]$
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.		$w(Y, y_2)$		$[\#8, T_2, P_Y, Y \mathrel{\cdot}= 2, Y \mathrel{/}= 2, \#7]$
17.		$w(X, x_2)$		$[\#9, T_2, P_X, X \mathrel{+}= 5, X \mathrel{-}= 5, \#8]$

Aufgabe 03

- WAL, $\neg force, steal$
- Anfangswerte: X = 10, Y = 100
- Während der Ausführung stürzt Ihre Datenbank ab. Sie wissen nicht welche Transaktionen commited wurden
- keine Verzahnung möglich.
- Sie stellen fest, dass nach dem Systemabsturz Y den Wert 200 hat.
- Nachdem die Datenbank neu gestartet wurde und der Recovery-Prozess abgeschlossen ist, liefert sie für X den Wert 110.
- c) Geben Sie das Log nach Beendigung des Recovery-Prozesses an.

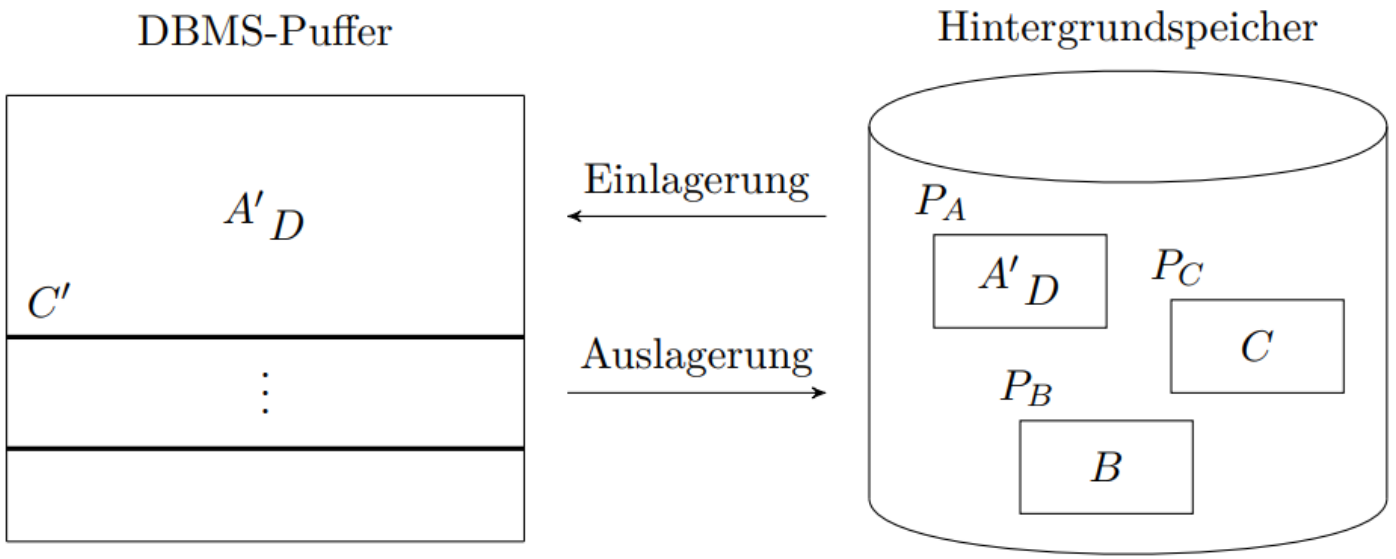
T_1	T_2	T_3
BOT	**BOT**	**BOT**
$r(X, x_1)$	$r(Y, y_2)$	$r(X, x_3)$
$x_1 := x_1 + 1$	$r(X, x_2)$	$x_3 := x_3 \cdot 10$
$w(X, x_1)$	$y_2 := y_2 \cdot 2$	$w(X, x_3)$
COMMIT	$x_2 := x_2 + 5$	**COMMIT**
	$w(Y, y_2)$	
	$w(X, x_2)$	
	COMMIT	

Lösungsvorschlag 03c

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

Aufgabe 04

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.



Lösungsvorschlag 04

- In Schritt 7 führt T_1 einen commit aus. Aufgrund der force-Strategie muss P_A 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.
- Das führt zu einem Widerspruch

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