AG Datenbanken und Informationssysteme

Wintersemester 2006 / 2007

Prof. Dr.-Ing. Dr. h. c. Theo Härder
Fachbereich Informatik
Technische Universität Kaiserslautern

http://wwwdvs.informatik.uni-kl.de

12. Übungsblatt

Für die Übung am Donnerstag, **01. Februar 2007**,
von 15:30 bis 17:00 Uhr in 13/222.

Aufgabe 1: Redo Lauf

Weisen Sie nach, daß beim ARIES-Ansatz (Non-Atomic, Steal, NoForce, Fuzzy Checkpoint) die Idempotenz des Wiederanlaufs das REDO aller protokollierten Änderungen – also auch der von Verlierertransaktionen durchgeführten Änderungen – verlangt. Betrachten Sie dazu zwei Transaktionen T_1 und T_2, die parallel ablaufen und die Datensätze $A(T_1)$ resp. $B(T_2)$ der Seite P modifizieren. Hierbei soll T_1 die Verlierer- und T_2 die Gewinnertransaktion sein. Diskutieren Sie die Zustände der Seite P auf der Platte:

- Vor der Modifikation von A.
- Nach der Modifikation von A, aber vor der Modifikation von B.
- Nach der Modifikation von B.

Was passiert bei Wiederanlauf in Bezug auf diese drei möglichen Zustände der Seite P?

Lösung:

a) Die Seite P_x hat sich auf der Platte noch nicht geändert.

Zeit	Aktion	Änderung im DB-Puffer (Seite, LSN)	Änderung in der DB (Seite, LSN)	Log-Eintrag im Log-Puffer (LSN, TAID, Log-Info, PrevLSN, UndoNxtLSN)	Zur Log-Datei hinzugefügte Log-Einträge (LSNs)
z_1	b_1			1, T_1, BOT, 0, 0	
z_2	b_2			2, T_2, BOT, 0, 0	
z_3	$w_1(P_A)$	P_p, 3		3, T_1, U/R(P_A), 1, 0	
z_4	$w_2(P_B)$	P_p, 4		4, T_2, U/R(P_B), 2, 0	
z_5	c_2			5, T_2, EOT, 4, 0	1, 2, 3, 4, 5
z_6	crash				

Redo-Phase: Redo von $w_1(P_A)$ und Redo von $w_2(P_B)$ muss durchgeführt werden.
Undo-Phase: Comensation von $w_1(P_A)$.
Würden wir in diesem Fall auf das Redo der Verlierertransaktion verzichten, so würden auf der Platte keine Fehler entstehen. Jedoch wäre es auch notwendig zu erkennen, dass keine Comensation notwendig ist.

b) T_1 hat den Datensatz A der Seite P_x bereits geändert:

Zeit	Aktion	Änderung im DB-Puffer (Seite, LSN)	Änderung in der DB (Seite, LSN)	Log-Eintrag im Log-Puffer (LSN, TAID, Log-Info, PrevLSN, UndoNxtLSN)	Zur Log-Datei hinzugefügte Log-Einträge (LSNs)
z_1	b_1			1, T_1, BOT, 0, 0	
z_2	b_2			2, T_2, BOT, 0, 0	
z_3	$w_1(P_A)$	P_p, 3		3, T_1, U/R(P_A), 1, 0	
z_4	flush(P_p)		P_p, 3		1, 2, 3
z_5	$w_2(P_B)$	P_p, 4		4, T_2, U/R(P_B), 2, 0	
z_6	c_2			5, T_2, EOT, 4, 0	4, 5
z_7	crash				

Redo-Phase: Redo von $w_2(P_B)$ muss/wird durchgeführt werden.
Undo-Phase: Comensation von $w_1(P_A)$.

Nun, da die Änderung von T_1 die Platte bereits erreicht hat ist es zwingend notwendig in der Undo-Phase die Comensation durchzuführen.

c) T_2 hat den Datensatz Bder Seite P_x bereits geändert:

Zeit	Aktion	Änderung im DB-Puffer (Seite, LSN)	Änderung in der DB (Seite, LSN)	Log-Eintrag im Log-Puffer (LSN, TAID, Log-Info, PrevLSN, UndoNxtLSN)	Zur Log-Datei hinzugefügte Log-Einträge (LSNs)
z_1	b_1			1, T_1, BOT, 0, 0	
z_2	b_2			2, T_2, BOT, 0, 0	
z_3	$w_1(P_A)$	P_p, 3		3, T_1, U/R(P_A), 1, 0	
z_4	$w_2(P_B)$	P_p, 4		4, T_2, U/R(P_B), 2, 0	
z_5	flush(P_p)		P_p, 4		1, 2, 3, 4
z_6	c_2			5, T_2, EOT, 4, 0	5
z_7	crash				

Redo-Phase: nichts mehr.
Undo-Phase: Comensation von $w_1(P_A)$.

Nur noch die Comensation wird durchgeführt.

Aufgabe 2: Logging und Recovery mit Rollback und Sicherungspunkten

In der folgenden Tabelle sei eine Historie gegeben, in der ein Fuzzy Checkpoint erzeugt wurde. Sie enthält außerdem zwei Rollback-Operationen, die mit a_i begonnen wurden, wobei die eine Rollback-Operation vor dem Systemabsturz erfolgreich ausgeführt wurde und die andere nicht. Das DBMS stürzt nach der Ausführung der letzten Operation ab und verwendet vollständiges Redo bei der Recovery.

a) Vervollständigen Sie die folgende Tabelle: b) Restart durchführen Tabelle aus a) ergänzen

Zeit	Aktion	Änderung im DB-Puffer (Seite, LSN)	Änderung in der DB (Seite, LSN)	Log-Eintrag im Log-Puffer (LSN, TAID, Log-Info, PrevLSN, UndoNxtLSN)	Zur Log-Datei hinzugefügte Log-Einträge
z_1	b_1				
z_2	$w_1(A)$				
z_3	$w_1(B)$				
z_4	c_1				
z_5	flush(P_A)				
z_6	b_2				
z_7	$w_2(A)$				
z_8	$w_2(C)$				
z_9	cp				
z_{10}	a_2				
z_{11}	$comp_2(C)$				
z_{12}	$comp_2(A)$				
z_{13}	r_2				
z_{14}	b_3				
z_{15}	flush(P_A)				
z_{16}	$w_3(A)$				
z_{17}	$w_3(B)$				
z_{18}	flush(P_B)				
z_{19}	$w_3(C)$				
z_{20}	a_3				
z_{21}	$comp_3(C)$				
z_{22}	b_4				
z_{23}	$comp_3(B)$				
z_{24}	$w_4(D)$				
z_{25}	flush(P_B)				
z_{26}	c_4				

Lösung:

a)

Zeit	Aktion	Änderung im DB-Puffer (Seite, LSN)	Änderung in der DB (Seite, LSN)	Log-Eintrag im Log-Puffer (LSN, TAID, Log-Info, PrevLSN, UndoNxtLSN)	Zur Log-Datei hinzugefügte Log-Einträge (LSNs)
z_1	b_1			1, T_1, BOT, 0, 0	
z_2	$w_1(A)$	P_A, 2		2, T_1, U/R(A), 1, 0	
z_3	$w_1(B)$	P_B, 3		3, T_1, U/R(B), 2, 0	
z_4	c_1			4, T_1, EOT, 3, 0	1, 2, 3, 4
z_5	flush(P_A)		P_A, 2		
z_6	b_2			5, T_2, BOT, 0, 0	
z_7	$w_2(A)$	P_A, 6		6, T_2, U/R(A), 5, 0	
z_8	$w_2(C)$	P_C, 7		7, T_2, U/R(C), 6, 0	
z_9	cp			8, -, CP({(P_B, 3), (P_A, 6), (P_C, 7)}, {T_2}), 0, 0	5, 6, 7, 8
z_{10}	a_2				
z_{11}	$comp_2(C)$	P_C, 9		9, T_2, -/U(C), 7, 6	
z_{12}	$comp_2(A)$	P_A, 10		10, T_2, -/U(A), 9, 5	
z_{13}	r_2			11, T_2, Rollback, 10, 0	
z_{14}	b_3			12, T_3, BOT, 0, 0	
z_{15}	flush(P_A)		P_A, 10		9, 10, 11, 12
z_{16}	$w_3(A)$	P_A, 13		13, T_3, U/R(A), 12, 0	
z_{17}	$w_3(B)$	P_B, 14		14, T_3, U/R(B), 13, 0	
z_{18}	flush(P_B)		P_B, 14		13, 14
z_{19}	$w_3(C)$	P_C, 15		15, T_3, U/R(C), 14, 0	
z_{20}	a_3				
z_{21}	$comp_3(C)$	P_C, 16		16, T_3, -/U(C), 15, 14	
z_{22}	b_4			17, T_4, BOT, 0, 0	
z_{23}	$comp_3(B)$	P_B, 18		18, T_3, -/U(B), 16, 13	
z_{24}	$w_4(D)$	P_D, 19		19, T_4, U/R(D), 17, 0	
z_{25}	flush(P_B)		P_B, 18		15, 16, 17, 18, 19
z_{26}	c_4			20, T_4, EOT, 19, 0	20

b) Restart bei vollständigem Redo:

(1) Analyse-Phase:

(i) Gewinner: T_1, T_2 und T_4 (alle TAs, die zum Zeitpunkt des Systemabsturzes nicht mehr aktiv waren)

(ii) Verlierer: T_3

(iii) Relevante Seiten: P_A, P_B, P_C und P_D

(2) Vollständige Redo-Phase:
Prüfe alle Log-Sätze aller TAs (T_1, T_2 , T_3 und T_4) inkl. CLRs ab MIN(Start-LSN) vorwärts.
MIN(Start-LSN) kann aus der SP-Zustandsinformation ermittelt werden => (P_B, 3)

Log-Satz LSN	TA	Seite	Seiten- vs. Log-Satz-LSN	Aktion
3	T_1	P_B	18 !< 3	Kein Redo
6	T_2	P_A	10 !< 6	Kein Redo
7	T_2	P_C	0 < 7	Redo $w_2(C)$, LSN(P_C)=7
9	T_2	P_C	7 < 9	Redo $comp_2(C)$, LSN(P_C)=9
10	T_2	P_A	10 !< 10	Kein Redo
13	T_3	P_A	10 < 13	Redo $w_3(A)$, LSN(P_A)=13
14	T_3	P_B	18 !< 14	Kein Redo
15	T_3	P_C	9 < 15	Redo $w_3(C)$, LSN(P_C)=15
16	T_3	P_C	15 < 16	Redo $comp_3(C)$, LSN(P_C)=16
18	T_3	P_B	18 !< 18	Kein Redo
19	T_4	P_D	0 < 19	Redo $W_4(D)$, LSN(P_D)=19

(3) Undo-Phase:
Prüfe Log-Sätze der Verlierer-Transaktionen, also TAs, die zum Zeitpunkt des Systemabsturzes noch aktiv waren, also T_3, (inkl. CLRs) rückwärts.
Für jeden Log-Satz wird die zugehörige Undo-Operation ausgeführt und dann entsprechend ein CLR in der Log-Datei angelegt.
Bei der Ausführung eines CLR springt man zum UndoNxtLSN und setzt die Undo-Operation fort.
Hier enthält die Redo-Info die Undo-Aktion.

Log-Satz-LSN	TA	Aktion
18	T_3	UndoNxtLSN = 13, weiter mit 13. Log-Satz.
13	T_3	UndoNxtLsn = 0 -> $comp_3(A)$, LSN(P_A)=21 lege CLR [21, T_3, -/U(A), 18, 12] an, weiter mit 12. Log-Satz
12	T_3	Lege Rollback-Log-Satz [22, T_3, Rollback, 21, 0] an

Ergänzung zu Tabelle aus (a):

Zeit	Aktion	Änderung im DB-Puffer (Seite, LSN)	Änderung in der DB (Seite, LSN)	Log-Eintrag im Log-Puffer (LSN, TAID, Log-Info, PrevLSN, UndoNxtLSN)	Zur Log-Datei hinzugefügte Log-Einträge (LSNs)
z_{27}	$comp_3(A)$	P_A, 21		21, T_3, -/U(A), 18, 12	
z_{28}	r_3			22, T_3, Rollback, 21, 0	

Aufgabe 3: Integritätsbedingungen in SQL am Beispiel „Lieferung“

Gegeben seien folgende Relationen:

```
LIEFERANT (LNR, LNAME, STATUS, ORT)

PROJEKT   (PNR, PNAME, ORT)

LIEFERUNG (LNR, TNR, PNR, MENGE)

TEIL      (TNR, TNAME, FARBE, GEWICHT)
```

Formulieren Sie folgende Integritätsbedingungen in SQL:

a) Jedes Teil hat einen positiven Gewichtswert.

b) Alle roten Teile wiegen weniger als 50 (Gewichtseinheiten).

c) Keine zwei Projekte dürfen sich an demselben Ort befinden.

d) Jedes Projekt muss sich an einem Ort befinden, an dem es mindestens einen Lieferanten gibt.

e) Die Lieferanten aus ’MZ’ liefern insgesamt mehr Teile als die aus ’SB’.

Lösung:

a) Jedes Teil hat einen positiven Gewichtswert.

```
CREATE ASSERTION IB_A
  CHECK  (NOT EXISTS
         (SELECT * FROM TEILE
          WHERE GEWICHT <= 0))
```

b) Alle roten Teile wiegen weniger als 50 (Gewichtseinheiten).

```
CREATE ASSERTION IB_B
  CHECK (NOT EXISTS
       (SELECT * FROM TEILE
        WHERE FARBE = 'Rot' AND GEWICHT >= 50))
```

c) Keine zwei Projekte dürfen sich an demselben Ort befinden.

```
CREATE ASSERTION IB_C
  CHECK (NOT EXISTS
       (SELECT * FROM PROJEKT P1, PROJEKT P2
        WHERE P1.ORT = P2.ORT
             AND   P1.PNR <> P2.PNR))
```

d) Jedes Projekt muss sich an einem Ort befinden, an dem es mindestens einen Lieferanten gibt.

```
CREATE ASSERTION IB_D
  CHECK (NOT EXISTS
       (SELECT * FROM PROJEKT P
        WHERE NOT EXISTS
            (SELECT * FROM LIEFERANT L
             WHERE L.ORT = P.ORT)))
```

e) Die Lieferanten in 'MZ' liefern insgesamt mehr Teile als die in 'SB'.

```
CREATE ASSERTION IB_E
  CHECK (( SELECT SUM (LF.MENGE)
           FROM LIEFERUNG LF, LIEFERANT L
           WHERE  LF.LNR = L.LNR
           AND    L.ORT = 'MZ')
         >
         ( SELECT SUM (LF.MENGE)
           FROM LIEFERUNG LF, LIEFERANT L
           WHERE  LF.LNR = L.LNR
           AND    L.ORT = 'SB'))
```