

**Übung zur Vorlesung
„Einsatz und Realisierung von Datenbanksystemen“
im Sommersemester 2007**

Richard Kuntschke (richard.kuntschke@in.tum.de)

Lösungen zu Blatt 2

Aufgabe 1

Klassifizierung von Historien

SR Menge der serialisierbaren Historien. Eine serialisierbare Historie ist äquivalent zu einer seriellen Historie. Das Serialisierbarkeitstheorem besagt, dass eine Historie genau dann serialisierbar ist, wenn der zugehörige Serialisierbarkeitsgraph $SG(H)$ azyklisch ist.

RC Menge der rücksetzbaren Historien. Eine Historie H gehört in die Klasse *RC*, wenn für alle Transaktionen T_i, T_j in der Historie mit $i \neq j$ und T_i liest von T_j gilt:

$$c_j <_H c_i$$

Eine Transaktion kann also zurückgesetzt werden, ohne dadurch Änderungen einer anderen Transaktion plötzlich inkonsistent werden zu lassen.

ACA Menge der Historien ohne kaskadierendes Rücksetzen. Eine Historie H gehört in die Klasse *ACA*, wenn für alle Transaktionen T_i, T_j in der Historie mit $i \neq j$ und T_i liest Datenobjekt A von T_j gilt:

$$c_j <_H r_i(A)$$

Historien in dieser Klasse haben die Eigenschaft, dass Transaktionen zurückgesetzt werden können, ohne dadurch das Zurücksetzen einer anderen Transaktion notwendig zu machen.

ST Menge der strikten Historien. Eine Historie H ist in der Klasse *ST*, wenn für alle Transaktionen T_i, T_j in der Historie mit $i \neq j$ und T_i liest Datenobjekt A von T_j oder T_i überschreibt eine Änderung auf A von T_j gilt:

$$c_j <_H r_i(A) \vee a_j <_H r_i(A) \text{ beziehungsweise } c_j <_H w_i(A) \vee a_j <_H w_i(A)$$

Um die Diskussion interessanter zu gestalten, haben wir drei Historien H_1, H_2 und H_3 (siehe Abbildungen 2 bis 4) gewählt, die bewusst nicht minimale Beispiele sind. Wir überlassen es den Lesern, einfachere aber auch ggf. komplexere Beispiele selbst zu erstellen.

Behauptung 1 H_1 aus Abbildung 2 ist in $(RC \cap SR) - ACA$

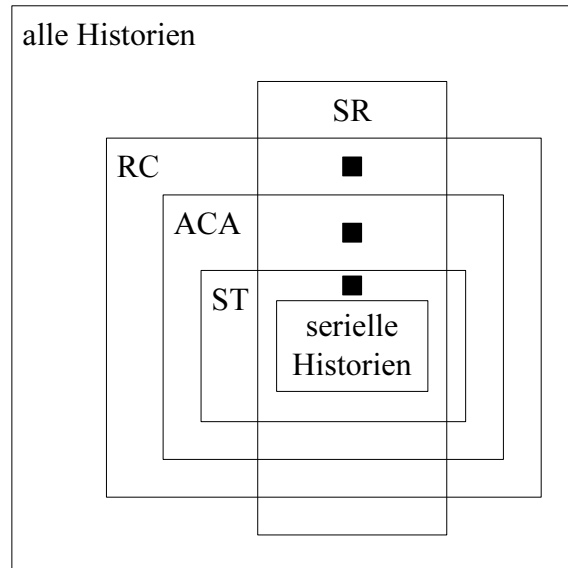


Abbildung 1: Beziehungen der Historienklassen zueinander (gesucht sind Historien aus den durch ■ hervorgehobenen Bereichen)

Beweis

- $H_1 \in RC$: T_{i+1} liest von T_i und $c_i <_{H_1} c_{i+1}$
- $H_1 \in SR$: Die Historie ist äquivalent zu $T_1 \mid T_2 \mid T_3 \mid T_4$
- $H_1 \notin ACA$: T_{i+1} liest von T_i und $\neg(c_i <_{H_1} r_{i+1})$

Behauptung 2 H_2 aus Abbildung 3 ist in $(ACA \cap SR) - ST$

Beweis

- $H_2 \in ACA$: Die einzige Lesebeziehung in der Historie besteht zwischen T_2 und T_3 . T_3 liest von T_2 das Datenobjekt B und es gilt: $c_2 <_{H_2} r_3(B)$

Schritt	T_1	T_2	T_3	T_4
1.	write(A)			
2.		read(A)		
3.		write(B)		
4.			read(B)	
5.			write(C)	
6.				read(C)
7.	commit			
8.		commit		
9.			commit	
10.				commit

Abbildung 2: Beispielhistorie H_1

Schritt	T ₁	T ₂	T ₃	T ₄
1.	write(A)			
2.				read(B)
3.		write(A)		
4.				write(B)
5.	commit			
6.				commit
7.		write(B)		
8.		commit		
9.			read(B)	
10.			write(C)	
11.			commit	

Abbildung 3: Beispielhistorie H_2

Schritt	T ₁	T ₂	T ₃	T ₄
1.	write(A)			
2.				read(B)
3.	commit			
4.		write(A)		
5.				write(B)
6.				commit
7.		write(B)		
8.		commit		
9.			read(B)	
10.			write(C)	
11.			commit	

Abbildung 4: Beispielhistorie H_3

- $H_2 \in SR$: H_2 ist äquivalent zu $T_1 \mid T_4 \mid T_2 \mid T_3$ oder auch zu $T_4 \mid T_1 \mid T_2 \mid T_3$
- $H_2 \notin ST$: Welche Transaktionen sind hier dafür verantwortlich, dass die Historie nicht strikt ist?

Die Transaktionen T_1 und T_2 verletzen die Eigenschaften einer strikten Historie, indem T_2 ein von T_1 geschriebenes Datenobjekt vor dem **commit** von T_1 überschreibt. Somit ist die Bedingung $c_1 <_{H_2} w_2(A)$ verletzt.

Behauptung 3 H_3 aus Abbildung 4 ist in $(ST \cap SR) - \{H' \mid H' \text{ ist serielle Historie}\}$

Beweis

- $H_3 \in ST$:

In dieser Historie gilt folgende Lesebeziehung:

- T_3 liest von T_2 .

In dieser Historie gelten folgende Schreibbeziehungen auf (im Laufe der Historie) modifizierten Datenobjekten:

- T_2 überschreibt eine Modifikation von T_1 ,
- T_2 überschreibt eine Modifikation von T_4 .

Die Transaktionen, von denen die Modifikationen kamen, sind zu den jeweiligen Zeitpunkten der Lese- oder Änderungsoperationen der abhängigen Transaktionen schon per **commit** beendet gewesen. $\Rightarrow H_3 \in ST$

- $H_3 \in SR$: H_3 ist äquivalent zu $T_1 \mid T_4 \mid T_2 \mid T_3$ oder auch zu $T_4 \mid T_1 \mid T_2 \mid T_3$
- H_3 ist keine serielle Historie.

Aufgabe 2

Wie in Abbildung 1 dargestellt, sind strikte Historien auch in den Klassen RC und ACA . Die Besonderheit bei Historien in $(ACA \cap SR) - ST$ ist, dass so genannte *blind writes* noch erlaubt sind. Das bedeutet, die von einer noch laufenden Transaktion verursachten Änderungen können durch andere Transaktionen überschrieben werden.

Für die Recovery-Komponente ist dabei problematisch, dass Änderungsoperationen auf einem Datenobjekt von mehr als einer aktiven Transaktion stammen können. Für Historien aus ST ist dies nicht der Fall.

Wir betrachten nun folgende Historie in $(ACA \cap SR) - ST$:

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

Wie sollte nun das lokale Rücksetzen von Transaktion T_1 aussehen? Führt man das lokale Rücksetzen nur bezogen auf T_1 durch, so gehen Änderungen von T_2 verloren. Bei logischer Logprotokollierung kann man dies ggf. ausschließen:

$$\begin{aligned}
 & [\#1, T_1, \mathbf{BOT}, 0] \\
 & [\#2, T_2, \mathbf{BOT}, 0] \\
 & [\#3, T_1, P_A, A+ = 50, A- = 50, \#1] \\
 & [\#4, T_2, P_A, A+ = 20, A- = 20, \#2] \\
 & \vdots
 \end{aligned}$$

Bei Schritt 5 kann die Änderung durch T_1 durch eine Subtraktion von 50 von Datum A rückgängig gemacht werden, ohne dass dadurch ein inkonsistenter Datenzustand auftreten würde. Anders verhält es sich bei physischer Protokollierung:

$$\begin{aligned}
 & [\#1, T_1, \mathbf{BOT}, 0] \\
 & [\#2, T_2, \mathbf{BOT}, 0] \\
 & [\#3, T_1, P_A, A = 150, A = 100, \#1] \\
 & [\#4, T_2, P_A, A = 170, A = 150, \#2] \\
 & \vdots
 \end{aligned}$$

In diesem Fall kann der Eintrag mit LSN #3 nicht einfach zurückgesetzt werden, da dessen Before-Image-Wert die Änderung von T_2 auf A nicht enthält.

Wenn die Konfiguration des Datenbanksystems *blind writes* erlaubt, reicht es folglich nicht, nur die Log-Einträge der Transaktion zu betrachten, die lokal zurückgesetzt wird (im Beispiel ist dies T_1). Es ist vielmehr erforderlich, im gesamten Log rückwärts nach überschreibenden Aktionen anderer Transaktionen (hier T_2) zu suchen, da diese nicht verloren gehen dürfen. Durch die Einschränkung auf strikte Historien wird dieser zusätzliche Analyseschritt vermieden.

Aufgabe 3

Wir betrachten folgende Historie H :

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

Die Historie ist äquivalent zur seriellen Abarbeitung $T_1 \mid T_2$. Die einzigen in Konflikt stehenden Operationen sind $r_1(A)$ und $w_2(A)$ mit $r_1(A) <_H w_2(A)$. Folgende Abbildung vervollständigt die gezeigte Historie um die benötigten Sperranforderungen:

Schritt	T_1	T_2	
1.	lockS (A)		
2.	read(A)		
3.		lockX (A)	← Sperre kann nicht
4.		read(A)	gewährt werden
5.		write(A)	
6.		unlock (A)	
7.		commit	
8.	lockS (B)		
9.	read(B)		
10.	lockS (C)		
11.	read(C)		
12.	unlock (A,B,C)		
13.	commit		

Die Sperranforderung von T_2 in Schritt 3 kann nicht erfüllt werden, da T_1 eine Lesesperre auf A hält. Diese Sperre darf jedoch noch nicht freigegeben werden, da sich die Transaktion noch in der Wachstumsphase befindet.

Durch das Zwei-Phasen-Sperrprotokoll wird die Schreiboperation $w_2(A)$ verzögert. Die Sperre auf A kann erst nach der Sperrfreigabe von T_1 gewährt werden:

Schritt	T_1	T_2	
1.	lockS (A)		
2.	read(A)		
3.		lockX (A)	← Warten bis Sperre gewährt werden kann
4.	lockS (B)		
5.	read(B)		
6.	lockS (C)		
7.	read(C)		
8.	unlock (A)		
9.		read(A)	
10.	⋮	⋮	

Aufgabe 4

Gegeben sei folgende Historie H , die das 2PL-Verfahren zulässt:

Schritt	T_1	T_2	
1.	lockX (A)		
2.	read(A)		
3.	write(A)		
4.	lockX (B)		
5.	read(B)		
6.	unlock (A)		
7.		lockX (A)	← dirty read
8.		read(A)	
9.		write(A)	
10.		unlock (A)	
11.		commit	
12.	write(B)		
13.	⋮		

Diese Historie ist äquivalent zur seriellen Ausführung $T_1 \mid T_2$, d.h. $H \in SR$. Sie ist aber nicht rücksetzbar, da T_2 das Datum A liest, welches von T_1 in Schritt 3 geschrieben wird, jedoch $c_2 <_H c_1$ gilt. Damit ist $H \notin RC$, und insgesamt $H \in SR - RC$.

Aufgabe 5

Es ist ausreichend, beim strengen 2PL-Protokoll nur die Schreibsperrern bis zum Ende der Transaktion zu halten. Lesesperren können analog zum normalen 2PL-Protokoll in der Schrumpfungsphase (nach wie vor jedoch nicht in der Wachstumsphase) peu à peu freigegeben werden. Die generierten Schedules bleiben serialisierbar und strikt.

Begründung

- Schon das normale 2PL bietet Serialisierbarkeit; diese ist also auch hier gegeben.
- Das Halten der Schreibsperrern bis zum Ende der Transaktion stellt sicher, dass keine Transaktion von einer anderen lesen oder einen von ihr modifizierten Wert überschreiben kann, bevor diese nicht ihr **commit** durchgeführt hat.

Es gilt:

$$\forall T_i : \forall T_j : (i \neq j) \forall A : (w_i(A) <_H r_j(A)) \vee (w_i(A) <_H w_j(A)) \Rightarrow (c_i <_H r_j(A)) \text{ bzw. } (c_i <_H w_j(A))$$