

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

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

Lösungen zu Blatt 3

Aufgabe 1

Kanten im Serialisierbarkeitsgraphen $SG(H)$ einer Historie H geben an, in welcher Reihenfolge Konfliktoperationen, also kritische Operationen auf denselben Datensätzen, ausgeführt werden. Eine Kante $T_i \rightarrow T_j$ besagt, dass T_i eine Konfliktoperation vor T_j hat – z.B. schreibt T_i das Datum A , bevor T_j A liest.

Demgegenüber wird eine Kante $T_i \rightarrow T_j$ in den Wartegraphen dann eingefügt, wenn T_i auf eine Sperrfreigabe durch T_j wartet. Während der Wartegraph somit eine momentane Aufnahme von Abhängigkeitsbeziehungen darstellt, gibt der Serialisierbarkeitsgraph Abhängigkeiten über den gesamten Zeitablauf der Historie wieder.

Eine Kante $T_i \rightarrow T_j$ im Wartegraphen taucht als Kante $T_j \rightarrow T_i$ im Serialisierbarkeitsgraphen auf.

Behauptung Wird das normale oder das strenge 2PL-Protokoll eingesetzt und tritt im Serialisierbarkeitsgraphen ein Zyklus auf (d.h. die Historie ist nicht serialisierbar), so existiert ein Zeitpunkt, zu dem auch im Wartegraphen ein Zyklus auftritt. Das heißt, es ergibt sich eine Verklemmungssituation.

Beweisidee Man kann zeigen, dass durch den Zyklus im Serialisierbarkeitsgraphen eine zyklische Wartesituation auftritt. Sei also der Zyklus $T_1 \rightarrow T_2 \rightarrow \dots \rightarrow T_n \rightarrow T_1$ in $SG(H)$ gegeben. Wegen des Pfads $T_1 \rightarrow T_2 \rightarrow \dots \rightarrow T_n$ muss T_1 Sperren freigeben ehe T_n Sperren erlangen kann (dabei müssen T_1 und T_n nicht notwendigerweise Konfliktoperationen auf denselben Daten ausführen). Wegen der Kante $T_n \rightarrow T_1$ in $SG(H)$ muss T_n aber Sperren vor T_1 freigeben. Die Transaktion T_1 befindet sich also noch nicht in ihrer Schrumpfungsphase, kann also selbst noch keine Sperren freigeben. Aufgrund der Abhängigkeiten im Zyklus gilt dies auch für die Transaktionen T_2, \dots, T_n . Damit tritt ein Zyklus im Wartegraphen auf, an dem die Transaktionen T_1, \dots, T_n beteiligt sind.

Obige Behauptung lässt sich im Umkehrschluss auch wie folgt formulieren: Ist der Wartegraph zu jedem Zeitpunkt zyklensfrei, so gilt dies auch für den Serialisierbarkeitsgraphen. Wenn also während der Ausführung kein Deadlock eintritt, dann ist die Historie serialisierbar. Es gilt jedoch keine Äquivalenz, wie das in Abbildung 1 gezeigte Beispiel zweier nebenläufiger Transaktionen zeigt.

Die vorgegebene Historie ist äquivalent zur seriellen Ausführung $T_1 | T_2$. Im Wartegraphen tritt ebenfalls kein Zyklus auf. Die Situation ist jedoch anders, wenn die Sperren nicht wie in Abbildung 1 dargestellt, sondern zeitlich versetzt angefordert werden, wie nachfolgende Historie mit Sperranforderungen zeigt:

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

Abbildung 1: Beispiel einer serialisierbaren Historie

Schritt	T_1	T_2	
1.	lockX(A)		
2.		lockX(B)	
3.	write(A)		
4.	lockS(B)		← T_1 muss warten
5.		lockS(A)	← T_2 muss warten
	⋮	⋮	Deadlocksituation

Dies ändert nichts an der eigentlichen Ausführung. Auch der Serialisierbarkeitsgraph wird dadurch nicht modifiziert, da dieser zwar die in Konflikt stehenden Operationen, wie *read* und *write*, nicht aber Sperranforderungen und -freigaben berücksichtigt. Mit anderen Worten, da die Ausführungsreihenfolge der *read*- und *write*-Operationen gleich bleibt, ändert sich die Historie nicht und $SG(H)$ bleibt zyklensfrei. Dies trifft nicht auf den Wartegraphen zu. Zum Zeitpunkt 5 enthält dieser die Kanten $T_1 \rightarrow T_2$ und $T_2 \rightarrow T_1$. Es liegt also eine Verklemmung $T_1 \rightleftarrows T_2$ vor, die durch eine Deadlockbehandlung behoben werden muss.

Fazit: Der Wartegraph gibt eine Momentaufnahme der Sperranforderungen, die gegenwärtig nicht gewährt werden können, wieder. Der Serialisierbarkeitsgraph berücksichtigt keine Sperren und beschreibt die Abfolge der Konfliktoperationen, bietet also eine „globale“ Sicht auf die Transaktionsabfolge.

Aufgabe 2

Wir betrachten zwei Transaktionen. T_1 beziehe sich auf folgendes SQL-Statement:

```
update Studenten set Semester = Semester + 1;
```

Die zweite Transaktion T_2 beziehe sich auf folgende einfache Anfrage:

```
select * from Studenten;
```

A , B und C seien Datensätze der Tabelle *Studenten*. Damit gilt:

$$T_1 : w_1(A) \rightarrow w_1(B) \rightarrow w_1(C)$$

$$T_2 : r_2(A) \rightarrow r_2(B) \rightarrow r_2(C)$$

Nehmen wir an, dass T_1 eine längere Ausführungszeit benötigt als T_2 und auch vor T_2 startet. Ferner sei angenommen, dass T_1 vor der ersten Schreiboperation alle benötigten Schreibsperrern anfordert.

Schritt	T_1	T_2	
1.	lockX (A)		
2.	lockX (B)		
3.	lockX (C)		
4.	write(A)		
5.		lockS (A)	← T_2 wird verzögert
6.	unlock (A)		
7.		read(A)	
8.	write(B)		
9.		lockS (B)	← T_2 wird verzögert
10.	unlock (B)		
11.		read(B)	
	⋮	⋮	

Betrachtet man den Wartegraphen zum Zeitpunkt 5, so wird die Kante $T_2 \rightarrow T_1$ eingefügt, da T_2 auf die Freigabe der Sperre auf A durch T_1 wartet. Wird das normale 2PL eingesetzt, so kann T_1 bereits zum Zeitpunkt 6 diese Sperre freigeben und T_2 kann mit der Bearbeitung fortfahren. Dieser Wartezustand wiederholt sich entsprechend für die Datensätze B und C . T_1 kann jeweils unmittelbar nachdem sie B und C abgearbeitet hat die Sperren wieder freigeben. Die Kante $T_2 \rightarrow T_1$ wird also im Beispiel dreimal eingefügt – zu jeweils anderen Zeitpunkten. Verwendet man das strenge 2PL, so wird die Kante nur einmal eingefügt, da Transaktion T_1 die Sperren alle auf einmal wieder freigeben muss.

Sowohl beim normalen als auch beim strengen 2PL kann es nicht vorkommen, dass dieselbe Kante zur selben Zeit mehrfach im Wartegraph auftaucht. Der Grund dafür ist, dass, wenn eine Wartekante eingefügt wird, die abhängige Transaktion verzögert wird, d.h. sie kann in ihrer Ausführung erst fortfahren, wenn sie die entsprechende Sperre erhalten hat. Damit gleiche Kanten mehrfach zur selben Zeit eingefügt werden können, müsste die Sperranforderung parallel ausgeführt werden, bzw. eine Transaktion parallel abgearbeitet werden. Dies entspricht einer Art *Intra-Transaktions-Parallelität*, im Gegensatz zur *Inter-Transaktions-Parallelität*, die die nebenläufige Ausführung unterschiedlicher Transaktionen beschreibt.

Aufgabe 3

Seien T_2, \dots, T_n weitere Transaktionen, von denen jede eine Lesesperre auf dem Datum A hält. Nun fordere T_1 eine exklusive Schreibsperre auf A an.

Behandlung bei wound-wait Gemäß *wound-wait* setzen sich ältere Transaktionen durch. Wir unterscheiden zwei Fälle.

- T_1 ist die älteste Transaktion, d.h. $\forall 2 \leq i \leq n : TS(T_i) > TS(T_1)$. In diesem Fall werden alle Transaktionen T_i abgebrochen und T_1 erhält die Schreibsperre auf A .

- Es existiert mindestens eine Transaktion, die älter ist als T_1 . Sei T_m die jüngste dieser Transaktionen, d.h. $TS(T_m) < TS(T_1)$ und es existiert keine weitere Transaktion T_i ($2 \leq i \leq n, i \neq m$) mit $TS(T_m) < TS(T_i) < TS(T_1)$. Dann wartet T_1 auf die Sperrfreigabe von T_m . Jüngere Transaktionen (als T_1), die dann noch eine Lesesperre auf A halten, werden, sobald T_1 die Sperre auf A erhält, abgebrochen (vgl. obigen Fall).

Behandlung bei wait-die Bei *wait-die* müssen ältere Transaktionen auf die Sperrfreigabe jüngerer Transaktionen warten. Fordert eine jüngere Transaktion eine Sperre an, die von einer älteren gehalten wird, so wird sie abgebrochen und zurückgesetzt. Folgende Fälle können auftreten:

- $\exists 2 \leq i \leq n : TS(T_i) < TS(T_1)$, d.h. es existiert eine Transaktion, die älter ist als T_1 und die eine Lesesperre auf A hält. Dann wird T_1 zurückgesetzt.
- Sonst, also in dem Fall, dass alle anderen Transaktionen jünger sind als T_1 ($\forall 2 \leq i \leq n : TS(T_1) < TS(T_i)$), wartet T_1 auf die Sperrfreigaben von T_i mit $2 \leq i \leq n$.

Aufgabe 4

Erweiterte Kompatibilitätsmatrix

	<i>NL</i>	<i>S</i>	<i>X</i>	<i>IS</i>	<i>IX</i>	<i>SIX</i>
<i>S</i>	✓	✓	-	✓	-	-
<i>X</i>	✓	-	-	-	-	-
<i>IS</i>	✓	✓	-	✓	✓	✓
<i>IX</i>	✓	-	-	✓	✓	-
<i>SIX</i>	✓	-	-	✓	-	-

Bezüglich dem höchsten Gruppenmodus gilt für die *SIX* Sperre:

$$SIX > IS$$

Anwendungsbeispiel Eine Transaktion T , die viele Daten liest, aber nur wenige ändert, ermöglicht durch Setzen einer *SIX*-Sperre eine erhöhte Parallelität. Parallel dazu können nämlich Transaktionen lesend auf diese Teilbereiche des per *SIX* gesperrten Teilbaums zugreifen, die von T nicht mit exklusiver Sperre belegt sind.

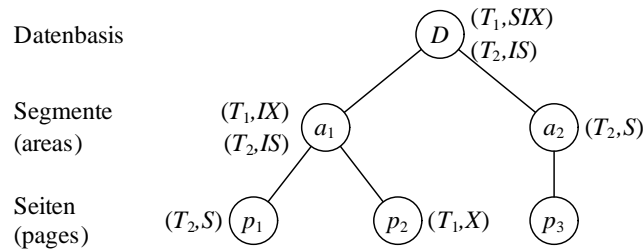
Als Beispiel wird folgende Abfrage betrachtet, die jedem „gut verdienenden“ Angestellten, d.h. Beschäftigten mit einem Einkommen von mehr als 1 Mio. Euro jährlich, das Gehalt um 100.000 Euro kürzt. In der Regel sollte dies nur eine sehr elitäre, eingeschränkte Gruppe von Beschäftigten betreffen.

```
update Angestellte
set Gehalt = Gehalt - 100000
where Gehalt > 1000000;
```

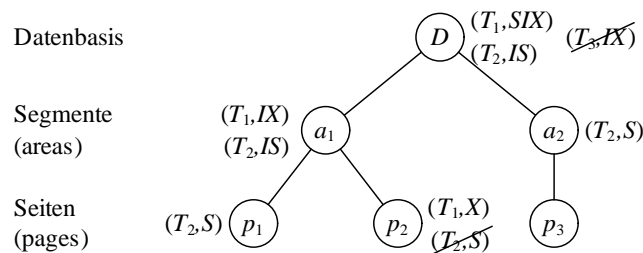
Parallel dazu kann z.B. eine Kontrollabfrage ausgeführt werden, die nur lesend auf die Datenbasis zugreift:

```
select count(*)
from Angestellte
where Gehalt < 500000;
```

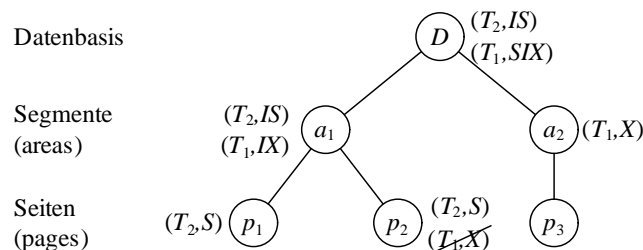
Die erste Abfrage wird von Transaktion T_1 , die zweite von Transaktion T_2 bearbeitet. Folgende Abbildung verdeutlicht die Sperren auf der Datenbasis-Hierarchie (Seiten können dabei noch weiter in Sätze unterteilt sein):



Parallele Schreiboperationen werden nicht geduldet. Leseoperationen werden ebenfalls geblockt, wenn sie mit Schreiboperationen der „ SIX -Transaktion“ T_1 kollidieren.



Folgende Abbildung zeigt den Fall, dass T_1 nach T_2 Sperren anfordert. In diesem Fall kann T_1 keine Sperren auf p_2 erhalten, solange T_2 eine Lesesperre darauf hält. Mit anderen Worten, eine „ SIX -Transaktion“ muss bei Anforderung einer Schreibsperre warten, falls eine andere Transaktion bereits eine Sperre auf den betroffenen Datensätzen hält.



Aufgabe 5

Behauptung 1 Die Zeitstempel-basierende Synchronisationsmethode erlaubt nur serialisierbare Schedules.

Beweis Um nachzuweisen, dass jede Historie H , die durch die Zeitstempel-basierende Synchronisationsmethode erzeugt wird, serialisierbar ist, zeigen wir, dass der zugehörige Serialisierbarkeitsgraph $SG(H)$ azyklisch sein muss. Sei also eine Historie H gegeben, die von der Zeitstempel-basierenden Synchronisationsmethode zugelassen ist. Falls $T_i \rightarrow T_j$ eine Kante in $SG(H)$ ist, dann gibt es bezüglich eines Datums A Konfliktoperationen $o_i(A)$ und $p_j(A)$, derart dass $o_i(A) <_H p_j(A)$. Gemäß den Regeln der Zeitstempel-basierenden Synchronisationsmethode

gilt dann $TS(T_i) < TS(T_j)$. Liegt ein Zyklus $T_1 \rightarrow T_2 \rightarrow \dots \rightarrow T_n \rightarrow T_1$ vor, so folgt per Induktion, dass $TS(T_1) < TS(T_1)$ gilt, was ein Widerspruch ist. Folglich ist $SG(H)$ azyklisch. Nach dem Serialisierbarkeitstheorem ist H damit serialisierbar.

Behauptung 2 Die Zeitstempel-basierende Synchronisationsmethode verhindert das Auftreten von Verklemmungen.

Beweis Die Synchronisation einer Menge von Transaktionen wird so durchgeführt, dass immer ein Schedule entsteht, der einer seriellen Abarbeitung der Transaktionen in Zeitstempel-Reihenfolge entspricht. Damit ein Deadlock entsteht, müssen Transaktionen gegenseitig auf Sperrfreigaben warten. Der Zeitstempel-basierten Synchronisationsmethode zufolge werden in Konfliktfällen ältere Transaktionen zurückgesetzt. Eine Wartebeziehung kann nur dann auftreten, wenn eine Transaktion T_i auf ein Datum zugreift, das eine ältere noch aktive Transaktion T_j geschrieben hat – andernfalls würde ein *dirty read* auftreten. Um *dirty reads* zu verhindern werden sog. *dirty* Bits verwendet, die solange gesetzt bleiben, bis die Datenobjekte festgeschrieben sind.

Im Allgemeinen kann eine Verklemmung mehr als zwei Transaktionen einschließen. Wie zuvor beschrieben, muss, tritt die Kante $T_i \rightarrow T_j$ im Wartegraphen auf, $TS(T_i) > TS(T_j)$ gelten. Tritt nun zu einem Zeitpunkt ein Zyklus $T_1 \rightarrow T_2 \rightarrow \dots \rightarrow T_n \rightarrow T_1$ im Wartegraphen auf, so kann man analog per Induktion nachweisen, dass $TS(T_1) < TS(T_1)$ gelten muss, was ein Widerspruch ist. Damit ist der Wartegraph zu jeder Zeit azyklisch. Die Zeitstempel-basierende Synchronisationsmethode verhindert also das Auftreten von Verklemmungen.