TU München, Fakultät für Informatik
Lehrstuhl III: Datenbanksysteme
Prof. Alfons Kemper, Ph.D.

Übung zur Vorlesung *Grundlagen: Datenbanken* im WS20/21

Christoph Anneser, Josef Schmeißer, Moritz Sichert, Lukas Vogel (gdb@in.tum.de)
https://db.in.tum.de/teaching/ws2021/grundlagen/

Blatt Nr. 13

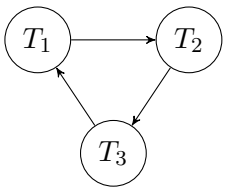
Hausaufgabe 1

a) Geben Sie alle Eigenschaften an, die von der Historie erfüllt werden.

$$H_1 = w_1(x), r_2(y), w_3(y), w_2(x), w_3(z), c_3, w_1(z), c_2, c_1$$

richtig	falsch	Aussage
	✔	Serialisierbar (SR)
✔		Rücksetzbar (RC)
✔		Vermeidet kaskadierendes Zurücksetzen (ACA)
	✔	Strikt (ST)

Die Historie H_1 ist nicht serialisierbar, da ihr Serialisierbarkeitsgraph (SG) einen Kreis enthält. Der SG sieht wie folgt aus:



Die Kanten im SG enstehen wegen folgenden Operationen:

$$\text{Kante von } T_1 \text{ zu } T_2 \Leftarrow w_1(x) <_{H_1} w_2(x)$$
$$\text{Kante von } T_2 \text{ zu } T_3 \Leftarrow r_2(y) <_{H_1} w_3(y)$$
$$\text{Kante von } T_3 \text{ zu } T_1 \Leftarrow w_3(z) <_{H_1} w_1(z)$$

Um die Rücksetzbarkeit zu überprüfen, muss bestimmt werden, welche Transaktionen voneinander lesen. In H_1 liest keine Transaktion von einer anderen, daher ist sie also rücksetzbar.

Da keine Transaktion von einer anderen liest, vermeidet die Historie auch kaskadierendes Zurücksetzen.

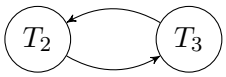
Um die Striktheit zu überprüfen, muss nicht nur bestimmt werden, ob eine Transaktion von einer anderen liest, sonder auch, ob eine Transaktion den Wert, den vorher eine andere Transaktion geschrieben hat, überschreibt. In H_1 überschreibt T_2 den Wert x von T_1, da $w_1(x) <_{H_1} w_2(x)$. Analog überschreibt T_1 den Wert z von T_3, da $w_3(z) <_{H_1} w_1(z)$. Eine Historie ist genau dann strikt, wenn diese Konfliktoperationen erst nach dem commit (oder abort) der gelesenen oder überschriebenen Transaktion ausgeführt werden. In H_1 ist aber $w_2(x) <_{H_1} c_1$ womit die Striktheit verletzt ist.

b) Geben Sie alle Eigenschaften an, die von der Historie erfüllt werden.

$$H_2 = r_1(x), r_1(y), w_2(x), w_3(y), r_3(x), a_1, r_2(x), r_2(y), c_2, c_3$$

richtig	falsch	Aussage
	✔	Serialisierbar (SR)
	✔	Rücksetzbar (RC)
	✔	Vermeidet kaskadierendes Zurücksetzen (ACA)
	✔	Strikt (ST)

Um die Serialisierbarkeit zu überprüfen, erstellen wir wieder den SG. T_1 ist nicht im SG enthalten, da sie nicht erfolgreich ist.



Die Kanten im SG enstehen wegen folgenden Operationen:

$$\text{Kante von } T_2 \text{ zu } T_3 \Leftarrow w_2(x) <_{H_2} r_3(x)$$
$$\text{Kante von } T_3 \text{ zu } T_2 \Leftarrow w_3(y) <_{H_2} r_2(y)$$

Der SG enthält einen Kreis, also ist H_2 nicht serialisierbar.

T_3 liest von T_2, da $w_2(x) <_{H_1} r_3(x)$. Außerdem liest T_2 von T_3, da $w_3(y) <_{H_1} r_2(y)$. Damit H_2 also rücksetzbar ist, müsste $c_2 <_{H_1} c_3$ und $c_3 <_{H_1} c_2$ gelten. Das ist bei H_2 nicht der Fall und im Allgemeinen natürlich auch nicht möglich, deswegen ist H_2 nicht rücksetzbar.

Da alle Historien, die kaskadierendes Zurücksetzen vermeiden oder strikt sind, auch rücksetzbar sind, kann man hier direkt folgern, dass H_2 weder ACA noch ST ist.

Hausaufgabe 2

In der Vorlesung haben Sie Serialisierbarkeitsgraphen und den Wartegraphen des (strikten) 2PL kennen gelernt.

a) Was bedeutet eine Kante $T_1 \to T_2$ im Serialisierbarkeitsgraphen einer Historie H?

b) Gehen Sie davon aus, dass die Datenbank die 2PL-Strategie verwendet. Was bedeutet eine Kante $T_1 \to T_2$ in einem Wartegraphen? Worin besteht der Unterschied zu Aufgabe a)?

c) Was bedeutet ein Kreis im Serialisierbarkeitsgraphen einer Historie H? Was im Wartegraphen? Wo liegt der Unterschied?

d) Wie viele neue Kanten werden dem Wartegraphen maximal hinzugefügt, wenn eine Transaktion eine S-Sperre anfordert? Wie viele bei einer X-Sperre?

Lösung

a) In jeder zu H äquivalenten seriellen Historie wird T_1 vor T_2 ausgeführt, da es in der Historie H Konfliktoperationen zwischen T_1 und T_2 gibt, für die H die Reihenfolge T_1 vor T_2 festlegt.

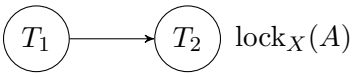
b) Die Transaktion T_1 fordert eine Sperre auf mindestens ein Datenobjekt an, auf welches T_2 bereits eine Sperre hat und muss daher warten. Die Kante verläuft hier also anders herum als in Teilaufgabe a), da sie als *warten auf* und nicht *geschieht vor* definiert ist.

c) Ein Kreis im Serialisierbarkeitsgraphen bedeutet, dass H nicht serialisierbar ist, es also keine äquivalente serielle Historie gibt. Ein Datenbanksystem kann Sie also nicht ausführen ohne das Prinzip der Isolation zu verletzen.

Ein Kreis im Wartegraphen hingegen bedeutet, dass es unter der strikten 2PL-Stratgie zu einem Deadlock gekommen ist und eine der im Kreis enthaltenen Transaktionen zurückgesetzt werden muss. Es bedeutet **nicht**, dass es keine äquivalente serielle Historie gibt. Es kann also sein, dass der Scheduler sie lediglich nicht gefunden hat. Striktes 2PL kann also eigentlich serialisierbare Historien ablehnen. Es garantiert aber, dass es bei allen **nicht serialisierbaren** Historien irgendwann während der Ausführung zum Deadlock kommt.

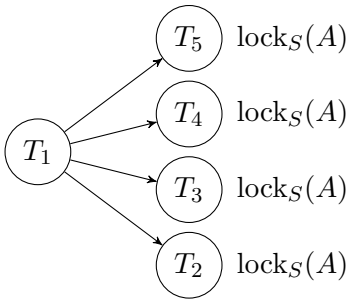
d) **S-Sperre:** Wenn eine Transaktion T eine S-Sperre anfordert und warten muss, dann muss eine andere Transaktion bereits eine X-Sperre auf das entsprechende Datenobjekt haben, denn: Hätten alle anderen Transaktionen keine Sperren, so bekäme unsere Transaktion T die Sperre sofort. Hätten andere Transaktionen S-Sperren, so müsste T ebenfalls nicht warten, da jedes Objekt mehrere Leser haben kann.

Hat jedoch eine andere Transaktion bereits eine X-Sperre, so ist diese exklusiv, da X-Sperren weder mit S-Sperren noch mit anderen X-Sperren kompatibel sind. Unsere Transaktion T würde also ausschließlich auf diese Transaktion warten und würde dem Wartegraphen höchstens eine Kante hinzufügen.



In dieser Darstellung hat T_2 bereits eine X-Sperre auf A. Die Transaktion T_1 will eine S-Sperre und fügt dem Wartegraphen so maximal eine Kante hinzu.

X-Sperre: Wenn eine Transaktion T nun eine X-Sperre auf ein Datenobjekt anfordert, dann kann es im schlimmsten Fall sein, dass alle anderen in der Datenbank aktiven Transaktionen dieses Datenobjekt bereits lesen (denn S-Sperren sind zueinander kompatibel). Bei n Transaktionen könnte es also bis zu $n-1$ S-Sperren geben. Wenn nun T eine X-Sperre anfordert, muss sie warten, bis jede einzelne dieser S-Sperren aufgehoben worden ist und damit dem Wartegraphen $n-1$ Kanten hinzufügen.



In dieser Darstellung haben T_2 bis T_5 bereits S-Sperren auf A. Die Transaktion T_1 will eine X-Sperre und fügt dem Wartegraphen so vier Kanten hinzu.

Hausaufgabe 3

Dies ist eine alte (und sehr verzwickte) Klausuraufgabe von Michael Stonebraker (Miterfinder von Ingres und Postgres) am MIT. Wer diese Aufgabe lösen kann, hat alle Fallstricke, die es bzgl. Recovery gibt, verstanden!

You are using a database management system that implements the ARIES protocol for logging and recovery (Chapter 10 of the book). The system uses strict two-phase locking, and the *no-force* and *steal* strategies. The database has just two items in it, X with starting value 10, and Y with starting value 100. You start three transactions at the same time (TA, TB, and TC):

```
TA:
  BEGIN TRANSACTION
  X = X + 1
  Y = Y * 3
  COMMIT TRANSACTION

TB:
  BEGIN TRANSACTION
  Y = Y * 2
  X = X + 5
  COMMIT TRANSACTION

TC:
  BEGIN TRANSACTION
  X = X * 10
  COMMIT TRANSACTION
```

These three transactions are the only activity in the system. The system crashes due to a power failure soon after you start the transactions. You are not sure whether or not any of them completed. You look at the disk while the system is down and see that, in the (due to the failure Tx-inconsistent) disk file, Y has the value 200. You restart the system and let the database recovery procedure complete. You query the database for the value of X, and it returns the value 110.

- What is the value of Y after the recovery?
- Show the log file that is consistent with the above "story" after recovery (i.e., at failover time)

Lösung: Siehe Vorlesungsvideo. Prof. Kemper erklärt das Problem und Lösung im letzten Teil der dieswöchigen Vorlesung.