Datenbanksysteme 2, 9. Übung
Transaktionsmanagement

Aufgabe 9.1: Serialisierbarkeit und Konfliktserialisierbarkeit

Betrachten Sie die folgenden Ausführungen S_1, S_2 und S_3 der beiden Transaktionen T_1 und T_2:

S_1		S_2		S_3	
T_1	T_2	T_1	T_2	T_1	T_2
R(a)		R(a)		R(a)	
a:=a-10			R(b)	a:=a-10	
W(a)		a:=a-10			R(b)
R(b)			b:=b*1.2	W(a)	
b:=b+10		W(a)			b:=b*1.2
W(b)			W(b)	R(b)	
	R(b)	R(b)			W(b)
	b:=b*1.2		R(c)	b:=b+10	
	W(b)	b:=b+10			R(c)
	R(c)		c:=c+20	W(b)	
	c:=c+20	W(b)			c:=c+20
	W(c)		W(c)		W(c)

Für welche der 3 Schedules treffen die Begriffe seriell, serialisierbar oder nicht serialisierbar zu? Gehen Sie davon aus, dass alle Datenwerte mit 0 initialisiert sind. Welche sind konfliktserialisierbar? Begründen Sie Ihre Entscheidung.

Berechnung der Werte:
T1, T2: Nach T1 ist a=-10, b=10; Nach T2 ist b=12, c=20
T2, T1: Nach T2 ist b=0, c=20; Nach T1 ist b=10, a=-10.
→ Gültige Ergebnisse sind a=-10, b=12, c=20 oder a=-10, b=10, c=20

S2: a=-10, c=20 (jeweils unkritisch), b=10, also ok, d.h. serialisierbar.
S3: a=-10, c=20 (jeweils unkritisch), b=10 (trotz lost update, da das update 0*1.2 war!!!), also ebenfalls serialisierbar.

S1: seriell (klar), damit auch serialisierbar und konfliktserialisierbar
S2: konfliktserialisierbar: T_2 -> T_1, (z.B. weil R(b) in T_2 vor W(b) in T_1), aber nicht T_1 -> T_2 (alle Operationen auf b von T_1 sind nach den Operationen auf b von T_2) (ergibt sich nicht unbedingt aus der Serialisierbarkeit und muss getrennt geprüft werden!)
S3: nicht konfliktserialisierbar: T_2 -> T_1 (Konfliktpaar R_2(b), W_1(b)), und T_1 -> T_2 (Konfliktpaar R_1(b), W_2(b)), das ist also ein Beispiel für ein Schedule, welches serialisierbar, aber nicht konfliktserialisierbar ist.

Aufgabe 9.2: Serialisierbarkeit

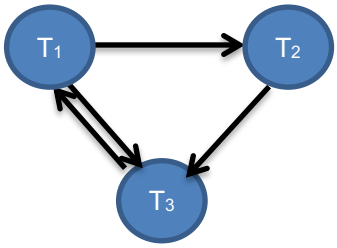
Betrachten Sie die beiden folgenden Schedules:

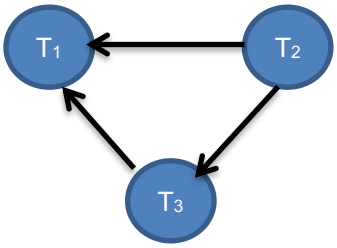
- $R_1(X); R_3(X); W_1(X); R_2(X); W_3(X)$
- $R_3(X); R_2(X); W_3(X); R_1(X); W_1(X)$

a) Schreiben Sie zu beiden Schedules die Konfliktpaare auf.

- Erstes Schedule:
  - $R_1(X)$ und $W_3(X)$
  - $R_3(X)$ und $W_1(X)$
  - $W_1(X)$ und $R_2(X)$
  - $W_1(X)$ und $W_3(X)$
  - $R_2(X)$ und $W_3(X)$
- Zweites Schedule:
  - $R_3(X)$ und $W_1(X)$
  - $R_2(X)$ und $W_3(X)$
  - $R_2(X)$ und $W_1(X)$
  - $W_3(X)$ und $R_1(X)$
  - $W_3(X)$ und $W_1(X)$

b) Zeichnen Sie zu beiden Schedules den Abhängigkeitsgraphen.





c) Falls ein Schedule konfliktserialisierbar ist, überführen Sie ihn durch konfliktäquivalente Umformungen in einen seriellen Schedule.

Es ist nur der zweite Schedule konfliktserialisierbar.
Umformungsschritte:

- $R_3(X); R_2(X); W_3(X); R_1(X); W_1(X)$
- $R_2(X); R_3(X); W_3(X); R_1(X); W_1(X)$

Aufgabe 9.3: Rücksetzbarkeit von Schedules

Betrachten Sie die Schedules aus Aufgabe 9.1 in Bezug auf Rücksetzbarkeit und kaskadierende Abbrüche. Untersuchen Sie dabei auch die Varianten, dass T_1 einen Commit direkt nach der letzten Operation der Transaktion absetzt, T_2 aber nicht, und umgekehrt.

Untersuchen Sie für alle 9 Schedules, ob sie rücksetzbar sind und ob kaskadierende Abbrüche auftreten könnten. Begründen Sie Ihre Antworten jeweils.

S1: T2 liest von T1.

S1 ohne Commit:
Der Schedule ist rücksetzbar. Wenn T2 abgebrochen wird, passiert nichts, wenn T1 abgebrochen wird, muss T2 ebenfalls abgebrochen werden da dann der Wert von b ungültig wird. Es könnten also kaskadierende Abbrüche auftreten.
S1 mit Commit am Ende von T1:
Der Schedule ist rücksetzbar, und er vermeidet kaskadierende Abbrüche.
S1 mit Commit am Ende von T2:
Der Schedule ist nicht rücksetzbar, da ein Abbruch von T1 nicht mehr verarbeitet werden kann.

S2: T1 liest von T2

S2 ohne Commit:
Der Schedule ist rücksetzbar. Wenn T1 abgebrochen wird, passiert nichts, wenn T2 abgebrochen wird, muss T1 ebenfalls abgebrochen werden. Es könnten also kaskadierende Abbrüche auftreten.
S2 mit Commit am Ende von T1:
Der Schedule ist nicht rücksetzbar, da ein Abbruch von T2 nicht mehr verarbeitet werden kann.
S2 mit Commit am Ende von T2:
Der Schedule ist rücksetzbar, und kaskadierende Abbrüche können nicht mehr auftreten.

S3: keine Transaktion liest von der anderen

Daher ist die Commit-Reihenfolge egal, alle Schedules sind rücksetzbar und vermeiden kaskadierende Abbrüche.

Anmerkung:
Insgesamt zeigt dieses Beispiel, dass Serialisierbarkeit und Rücksetzbarkeit getrennt betrachtet werden müssen. S3 ist z.B. nicht konfliktserialisierbar, vermeidet aber kaskadierende Abbrüche und ist rücksetzbar.