

o	DES3UEG1: Niklas	Name	Angelos Angelis	Aufwand in h	5
✗	DES3UEG2: Niklas				
o	DES3UEG3: Traxler	Punkte		Kurzzeichen Tutor	

## 1. Fehler bei Mehrbenutzerbetrieb

(4 Punkte)

Kreuzen Sie die richtige(n) Antwort(en) zu den gegebenen Fragen bezüglich Fehler im Mehrbenutzerbetrieb und zum Umgang von Datenbanken damit an.

a) Welche Aussagen treffen auf den Umgang bei Mehrbenutzerbetrieb zu?

- ☐ Datenbanksysteme bewerten im Mehrbenutzerbetrieb die Fehlersicherheit, um eine Isolationsstufe zu wählen.
- ☒ Serielle Schedules gewährleisten, dass keine Konflikte auftreten, allerdings sinkt die Performance.
- ☐ Wenn ein Schedule strikt ist, können Sie davon ausgehen, dass er auch konfliktserialisierbar ist.
- ☒ Datenbanksysteme verwenden verschiedene Strategien und kombinieren diese auch, um im Mehrbenutzerbetrieb die Performance bestmöglich bereitzustellen und Nebeneffekte zu reduzieren.

b) Welche Aussagen treffen auf die angegebene Ausführung zu?

T1	T2
R(A)	
	R(A)
	$A = A * 2$
$A = A + 50$	
W(A)	
	W(A)
COMMIT	
	COMMIT

- ☐ Dieses Problem wird als *Non-Repeatable Read* beschrieben, da sich A bei einem erneuten Lesen von T1 verändert hat.
- ☒ Mit dem Schreiben von T2 gehen die Änderungen von T1 verloren, dies wird auch als *Lost Update* bezeichnet.
- ☐ Würde T1 nach dem Schreiben abgebrochen, so ist der vorangegangene Lesevorgang ungültig (*Dirty Read*).
- ☐ Hier erfolgt ein *Lesen von inkonsistenten Zuständen*, da A nie größer als 10 sein darf.

c) Welche Aussagen treffen auf die angegebene Ausführung zu?

T3	T4
	R(X)
R(X)	
	R(Y)
	$X = Y * 2$
R(Y)	
	W(X)
R(X)	
	$Y = Y + 1$
COMMIT	
	COMMIT

- ☐ Da T3 zum ersten Mal X liest, nachdem dieses bereits vorher von T4 gelesen wurde, entsteht ein *Dirty Read*-Problem.
- ☐ T4 verändert Y, dieses wird allerdings nicht geschrieben – es liegt somit ein *Lost Update* vor.
- ☒ Es handelt sich um ein *Non-Repeatable Read* Problem, da T3 beim erneuten Lesen von X andere Werte erhält.
- ☐ Da Y erst nach der Neuberechnung von X verändert wird, entsteht ein sogenanntes *Phantom*-Problem.

d) Welche Aussagen treffen auf die unten angegebene Ausführung zu?

T5	T6
R(S = SUM(X))	
	R(X)
	INS(DATA,X)
	COMMIT
R(C = COUNT(X))	
A = S / C	
W(A)	
COMMIT	

- ☐ Die Berechnung von A enthält keine in sich stimmigen Daten, dies wird als *Lost Update* bezeichnet.
- ☒ Dies ist ein *Phantom-Problem*, da Datensätze während einer Berechnung eingefügt werden und in T5 nicht sichtbar sind.
- ☐ Beim *Lesen inkonsistenter Zustände* werden nicht zusammenhängende Daten bearbeitet.
- ☐ Da sich die Daten zwischen den Leseoperationen von T5 verändert haben, spricht man von einem *Non-Repeatable Read*.

## 2. Fehlersicherheit von Transaktionen

(9 Punkte)

Betrachten Sie die folgenden Ausführungspläne und stellen Sie fest, welcher Ausführungsplan strikt (ST), kaskadenlos (ACA) und/oder rücksetzbar (RC) ist. **Begründen** Sie Ihre Antwort für jeden Ausführungsplan.

$S_1 = w_1(x) w_1(y) r_2(z) c_1 w_2(x) r_2(y) w_2(y) c_2$   
 $S_2 = r_1(x) r_2(z) r_1(z) r_3(x) r_3(y) w_1(x) w_3(y) r_2(y) w_2(z) w_2(y) c_1 c_2 c_3$   
 $S_3 = r_1(y) r_1(x) w_1(x) w_2(x) r_1(x) w_1(y) a_1 a_2$

	RC	ACA	ST
S <sub>1</sub>	yes	yes	yes
S <sub>2</sub>	no	no	no
S <sub>3</sub>	yes	no	no

Hinweis:

**ST:** Schedule s heißt strikt (engl. strict), falls folgende Bedingung gilt:

$$(w_j(x) \rightarrow_s p_i(x) \wedge j \neq i) \Rightarrow (a_j <_s p_i(x) \vee c_j <_s p_i(x), (p \in \{r, w\}))$$

D.h., es darf kein „geschriebenes“ Objekt einer noch nicht beendeten Transaktion gelesen oder überschrieben werden.

**RC:** s heißt rücksetzbar (engl. recoverable), falls folgende Bedingung erfüllt ist:

$$(T_i \text{ liest von } T_j \text{ in } s) \wedge (c_i \in s) \Rightarrow (c_j \rightarrow_s c_i)$$

Eine Transaktion darf erst dann ihr COMMIT durchführen, wenn alle Transaktionen, von denen sie gelesen hat, beendet sind.

**ACA:** Schedule s vermeidet kaskadierende Abbrüche (engl. avoiding cascading aborts ACA), falls folgende Bedingung erfüllt ist:

$$(T_i \text{ liest } x \text{ von } T_j \text{ in } s) \Rightarrow (c_j \rightarrow_s r_i(x))$$

D.h., eine Transaktion darf nur Daten lesen, die zuletzt von einer bereits abgeschlossenen Transaktion geschrieben wurden.

## 3. Konfliktgraph

(8 Punkte)

Betrachten Sie die folgenden Transaktionen und Ausführungspläne. Geben Sie die Konfliktrelationen an und zeichnen Sie die Serialisierbarkeits- bzw. Konfliktgraphen. Identifizieren Sie, ob jeder Ausführungsplan serialisierbar ist. Je nachdem notieren Sie den bzw. die äquivalenten seriellen Ausführungspläne oder die Konflikte.

$$\begin{aligned} \text{a) } T_1 &= w_1(x) \ r_1(y) \ w_1(y) \\ T_2 &= r_2(x) \ w_2(y) \\ T_3 &= w_3(x) \ w_3(y) \end{aligned}$$

$$\begin{aligned} S_1 &= w_3(x) \ w_1(x) \ r_2(x) \ w_3(y) \ r_1(y) \ w_1(y) \ w_2(y) \\ S_2 &= w_1(x) \ r_2(x) \ w_2(y) \ r_1(y) \ w_1(y) \ w_3(x) \ w_3(y) \end{aligned}$$

$$\begin{aligned} \text{b) } T_1 &= r_1(y) \ r_1(x) \ w_1(y) \ w_1(x) \\ T_2 &= r_2(z) \ w_2(x) \ w_2(y) \ w_2(z) \\ T_3 &= r_3(y) \ w_3(y) \end{aligned}$$

$$\begin{aligned} S_1 &= r_1(y) \ r_2(z) \ r_1(x) \ w_1(y) \ r_3(y) \ w_3(y) \ w_1(x) \ w_2(x) \ w_2(y) \ w_2(z) \\ S_2 &= r_1(y) \ r_1(x) \ w_1(y) \ r_3(y) \ r_2(z) \ w_2(x) \ w_1(x) \ w_2(y) \ w_3(y) \ w_2(z) \end{aligned}$$

$$\begin{aligned} \text{c) } T_1 &= r_1(x) \ r_1(z) \ w_1(x) \\ T_2 &= r_2(z) \ r_2(y) \ w_2(z) \ w_2(y) \\ T_3 &= r_3(x) \ r_3(y) \ w_3(y) \end{aligned}$$

$$\begin{aligned} S_1 &= r_1(x) \ r_2(z) \ r_3(x) \ r_1(z) \ r_2(y) \ r_3(y) \ w_1(x) \ w_2(z) \ w_3(y) \ w_2(y) \\ S_2 &= r_1(x) \ r_2(z) \ r_1(z) \ r_3(x) \ r_3(y) \ w_1(x) \ w_3(y) \ r_2(y) \ w_2(z) \ w_2(y) \end{aligned}$$

#### 4. Zwei-Phasen-Sperrprotokoll

(3 Punkte)

Gegeben ist der Ausführungsplan  $S_1$  mit folgender Parallelausführung von Transaktionen:

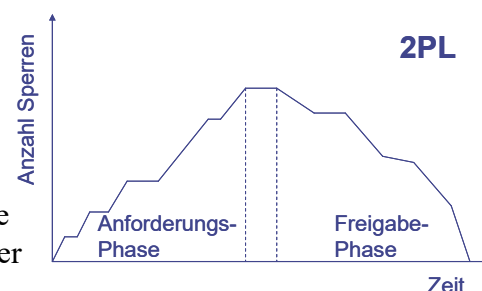
$$S_1 = r_1(x) \ r_2(y) \ w_1(x) \ r_3(z) \ r_2(x) \ r_3(x) \ w_3(z) \ w_2(z) \ C_1 \ C_2 \ C_3$$

Sie haben zwei Sperrbefehle  $rl_i(x)$  und  $wl_i(x)$  sowie einen **Unlock-Befehl**  $u_i(x)$  zur Verfügung. Wie würde das einfache Zwei-Phasen-Sperrprotokoll bei diesem Ausführungsplan vorgehen? Geben Sie für  $s_1$  einen möglichen **Ausführungsplan für das Zwei-Phasen-Sperrprotokoll** an.

Hinweis:

**Der Zugriff auf gemeinsam benutzte Daten wird durch Sperren synchronisiert:**

- Schreibzugriff  $w(x)$  nur nach Setzen einer Schreibsperre  $wl(x)$  möglich
- Lesezugriffe  $r(x)$  nur nach  $rl(x)$  oder  $wl(x)$  erlaubt
- Eine Schreibsperre  $wl(x)$  darf nur auf Objekte erfolgen, die nicht bereits von einer anderen Transaktion zum Schreiben gesperrt sind
- Ein  $rl(x)$  kann zu  $wl(x)$  verschärft werden, wenn keine andere Transaktion ein  $rl(x)$  hält; Sperren derselben Art auf ein Objekt werden innerhalb einer Transaktion maximal einmal gesetzt.
- Wenn die Sperränderung zulässig ist, muss die Verschärfung von Sperren, z.B. von  $rl(x)$  zu  $wl(x)$ , während der Wachstumsphase erfolgen.
- Nach  $u(x)$  durch  $T_i$  darf  $T_i$  kein erneutes  $rl(x)$  oder  $wl(x)$  ausführen
- Vor einem commit müssen alle Sperren aufgehoben werden
- Wenn die Sperränderung zulässig ist, darf die Abschwächung von Sperren, z.B. eine  $rl(x)$ -Operation, die eine bereits gehaltene Schreibsperre  $wl(x)$  abschwächt, nur in der Schrumpfungsphase erfolgen.
- Eine Transaktion folgt dem Zwei-Phasen-Sperrprotokoll, wenn alle Sperroperationen ( $rl$ ,  $wl$ ) vor der ersten Entsperr-Operation einer Transaktion ausgeführt werden.




# Aufgabe 2

$$S_1 = w_1(x) w_1(y) r_2(z) c_1 w_2(x) r_2(y) w_2(y) c_2$$

$w_2(x) / r_2(y) / w_2(y)$  wird erst durchgeführt  
nachdem  $T_1$  committed  $\Rightarrow ST$

$T_2$  liest nur nach dem Commit von  $T_1$   
 $\Rightarrow RC$


$T_2$  liest nur committed Attribute  
 $\Rightarrow ACA$

$$S_2 = r_1(x) r_2(z) r_1(z) r_3(x) r_3(y) w_1(x) \underline{w_3(y)} \underline{r_2(y)} w_2(z) w_2(y) c_1 c_2 c_3$$


$T_2$  liest  $y$  bevor  $c_3 \Rightarrow !ST$

$T_3$  schreibt vor  $T_2$  aber committed nach  $c_2 \Rightarrow !RC$

$T_2$  liest  $y$  bevor  $T_3$  committed  $\Rightarrow !ACA$

$$S_3 = r_1(y) r_1(x) w_1(x) \underline{w_2(x)} \underline{r_1(x)} w_1(y) a_1 a_2$$


$T_2$  überschreibt  $x$  bevor  $T_1$  abgeschlossen ist  $\Rightarrow !ST$

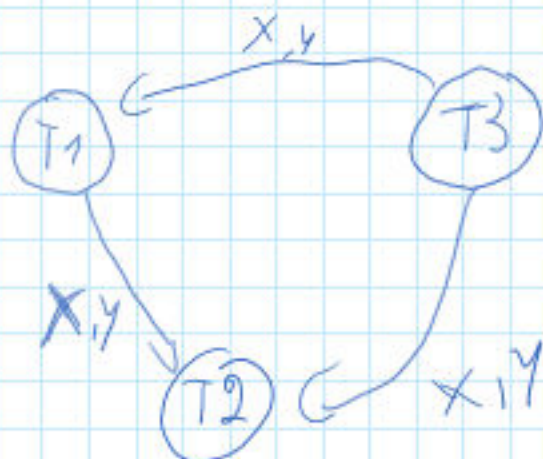
Es findet kein Commit statt

$T_1$  liest  $x$  bevor  $T_2$  abgeschlossen ist  
 $\Rightarrow !ACA$



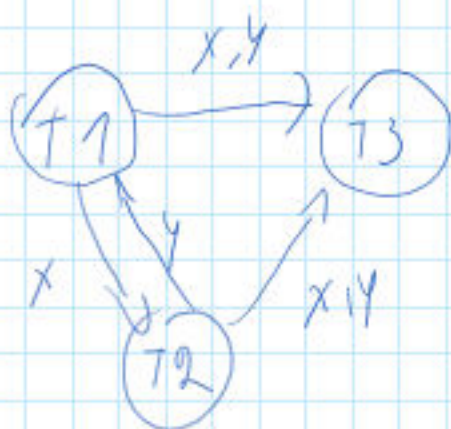
# a) Aufgabe 3

$$CCS1 = \{ (w3(x), w1(x)), (w3(x), r2(x)), (w1(x), r2(x)), (w3(y), r1(y)), (w3(y), w1(y)), (w3(y), w2(y)), (r1(y), w2(y)), (w1(y), w2(y)) \}$$



=> Konfliktserialisierbar  
 $T3 \rightarrow T1 \rightarrow T2$

$$CS2 = \{ (w1(x), r2(x)), (w1(x), w3(x)), (r2(x), w3(x)), (w2(y), r1(y)), (w2(y), w1(y)), (w2(y), w3(y)), (r1(y), w3(y)), (w1(y), w3(y)) \}$$

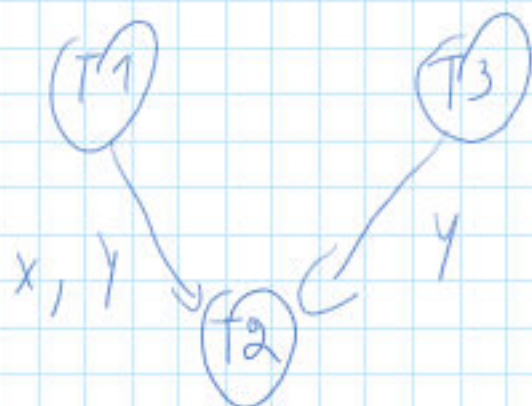


Wäre Konfliktserialisierbar

# Aufgabe 3

b)  $c(s1) = \{$

$$\begin{aligned} & (r1(y), w3(y)), (r1(y), w2(y)), \\ & (r3(y), w2(y)), (w3(y), w2(y)), \\ & (r1(x), w2(x)), (w1(y), r3(y)), (w1(y), w3(y)), \\ & (w1(y), w2(y)), (w1(x), w2(x)) \} \end{aligned}$$

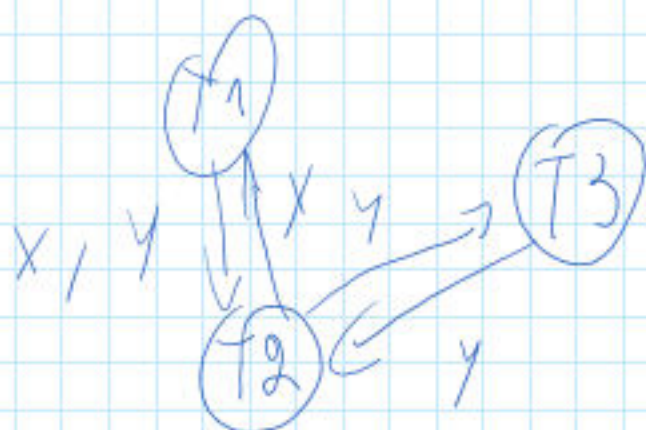


Konfliktserialisierbar

$T1 \rightarrow T3 \rightarrow T2$

$c(s2) = \{$

$$\begin{aligned} & (r1(y), w2(y)), \\ & (r1(y), w3(y)), (r3(y), w2(y)), \\ & (w2(y), w3(y)), (r1(x), w2(x)), \\ & (w2(x), w1(x)) \} \end{aligned}$$



nicht Konfliktserialisierbar

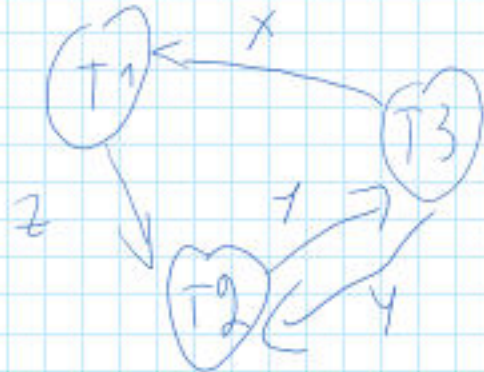


# Aufgabe 3

$S_1 = r_1(x) r_2(z) r_3(x) r_1(z) r_2(y) r_3(y) w_1(x) w_2(z) w_3(y) w_2(y)$

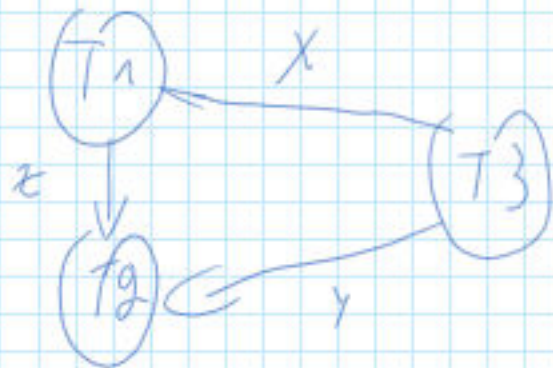
$S_2 = r_1(x) r_2(z) r_1(z) r_3(x) r_3(y) w_1(x) w_3(y) r_2(y) w_2(z) w_2(y)$

$$C(S_1) = \{ (r_3(x), w_1(x)), (r_1(z), w_2(z)), (r_2(y), w_3(y)), (r_3(y), w_2(y)), (w_3(y), w_2(y)) \}$$



nicht konfliktserialisierbar

$$C(S_2) = \{ (r_3(x), w_1(x)), (r_1(z), w_2(z)), (r_3(x), w_2(y)), (w_3(y), r_2(y)), (w_3(y), w_2(y)) \}$$



konfliktserialisierbar:

$T_3 \rightarrow T_1 \rightarrow T_2$

# Aufgabe 4

T1	T2	T3
$w_1(x)$		
$r_1(x)$	$r_1(y)$	
$w_1(x)$	$r_2(y)$	
	$r_1(x)$	$w_1(z)$
	$r_2(x)$	$r_2(z)$
		$r_1(x)$
		$r_3(x)$
		$w_3(z)$
	$w_1(z)$	
	$w_2(z)$	

$w_1(x), r_1(x), r_2(y), w_1(x), u_1(x), r_2(y), u_2(y), r_2(x), w_3(z),$   
 $r_2(x), u_2(x), r_3(z), r_3(x), r_3(x), u_3(x), w_3(z), u_3(z),$   
 $w_2(z); w_2(z); u_2(z)$