

# *Datenbanken:* Tutorium 14

Marvin Jahn

05.02.2020

# Mehrbenutzersynchronisation einer Datenbank

- Situation: mehrere verschiedene Nutzer greifen gleichzeitig auf dieselbe Datenbank zu

# Mehrbenutzersynchronisation einer Datenbank

- Situation: mehrere verschiedene Nutzer greifen gleichzeitig auf dieselbe Datenbank zu
- nach dem ACID-Prinzip sollen die Nutzer möglichst isoliert von einander sein

# Mehrbenutzersynchronisation einer Datenbank

- Situation: mehrere verschiedene Nutzer greifen gleichzeitig auf dieselbe Datenbank zu
- nach dem ACID-Prinzip sollen die Nutzer möglichst isoliert von einander sein
- die Transaktionen strikt nacheinander auszuführen ist sicher, aber ineffizient

# Transaktionen

Die möglichen Operationen einer Transaktion sind:

- $r(A)$ : Lesen (*read*) des Datenobjekts  $A$
- $w(A)$ : Schreiben (*write*) des Datenobjekts  $A$
- $a$ : Abbrechen (*abort*) der Aktion
- $c$ : Durchführen (*commit*) der Aktion

# Transaktionen

Die möglichen Operationen einer Transaktion sind:

- $r(A)$ : Lesen (*read*) des Datenobjekts  $A$
- $w(A)$ : Schreiben (*write*) des Datenobjekts  $A$
- $a$ : Abbrechen (*abort*) der Aktion
- $c$ : Durchführen (*commit*) der Aktion

Eine **Transaktion (TA)**  $T$  ist eine Menge bestehend aus Operationen der obigen Art und einer partiellen Ordnung “ $<$ ”, sodass:

- Entweder *abort* oder *commit* ist in  $T$  enthalten, d.h.  
 $a \in T \iff c \notin T$ .

# Transaktionen

Die möglichen Operationen einer Transaktion sind:

- $r(A)$ : Lesen (*read*) des Datenobjekts  $A$
- $w(A)$ : Schreiben (*write*) des Datenobjekts  $A$
- $a$ : Abbrechen (*abort*) der Aktion
- $c$ : Durchführen (*commit*) der Aktion

Eine **Transaktion (TA)**  $T$  ist eine Menge bestehend aus Operationen der obigen Art und einer partiellen Ordnung “ $<$ ”, sodass:

- Entweder *abort* oder *commit* ist in  $T$  enthalten, d.h.  
 $a \in T \iff c \notin T$ .
- In beiden Fällen werden alle anderen Operationen vor dem *abort* bzw. *commit* ausgeführt, d.h. für  $t = a$  bzw.  $t = c$ :  $p < t \ \forall \ p \in T \setminus \{t\}$ .

# Transaktionen

Die möglichen Operationen einer Transaktion sind:

- $r(A)$ : Lesen (*read*) des Datenobjekts  $A$
- $w(A)$ : Schreiben (*write*) des Datenobjekts  $A$
- $a$ : Abbrechen (*abort*) der Aktion
- $c$ : Durchführen (*commit*) der Aktion

Eine **Transaktion (TA)**  $T$  ist eine Menge bestehend aus Operationen der obigen Art und einer partiellen Ordnung “ $<$ ”, sodass:

- Entweder *abort* oder *commit* ist in  $T$  enthalten, d.h.  
 $a \in T \iff c \notin T$ .
- In beiden Fällen werden alle anderen Operationen vor dem *abort* bzw. *commit* ausgeführt, d.h. für  $t = a$  bzw.  $t = c$ :  $p < t \forall p \in T \setminus \{t\}$ .
- Die Reihenfolge von Schreibe- und Lesezugriffen auf dasselbe Objekt ist festgelegt, d.h. wenn  $r(x) \in T$  und  $w(x) \in T$ , dann  $r(x) < w(x)$  oder  $w(x) < r(x)$ .



# Historien und Konfliktoperationen

- Mehrere Transaktionen können nebenläufig ausgeführt werden.

# Historien und Konfliktoperationen

- Mehrere Transaktionen können nebenläufig ausgeführt werden.
- Eine **Historie** gibt an, wie Operationen aus verschiedenen TAs relativ zueinander ausgeführt werden (siehe VL für eine formale Definition).

# Historien und Konfliktoperationen

- Mehrere Transaktionen können nebenläufig ausgeführt werden.
- Eine **Historie** gibt an, wie Operationen aus verschiedenen TAs relativ zueinander ausgeführt werden (siehe VL für eine formale Definition).
- Zwei Operationen aus verschiedenen Transaktionen sind **Konfliktoperationen** (“stehen in Konflikt zueinander”), falls beide auf dasselbe Datenobjekt zugreifen und mindestens eine der beiden Operationen eine Schreibeoperation ist.

# Historien und Konfliktoperationen

- Mehrere Transaktionen können nebenläufig ausgeführt werden.
- Eine **Historie** gibt an, wie Operationen aus verschiedenen TAs relativ zueinander ausgeführt werden (siehe VL für eine formale Definition).
- Zwei Operationen aus verschiedenen Transaktionen sind **Konfliktoperationen** (“stehen in Konflikt zueinander”), falls beide auf dasselbe Datenobjekt zugreifen und mindestens eine der beiden Operationen eine Schreibeoperation ist.
- Zwei Konfliktoperationen dürfen nicht gleichzeitig ausgeführt werden.

# Mögliche Fehler bei unkontrolliertem Mehrbenutzerbetrieb

*Lost update* (verlorengegangene Änderungen)

Schritt	$T_1$	$T_2$
1.	read( $A, a_1$ )	
2.	$a_1 := a_1 - 300$	
3.		read( $A, a_2$ )
4.		$a_2 := a_2 * 1.03$
5.		write( $A, a_2$ )
6.	write( $A, a_1$ )	
7.	read( $B, b_1$ )	
8.	$b_1 := b_1 + 300$	
9.	write( $B, b_1$ )	

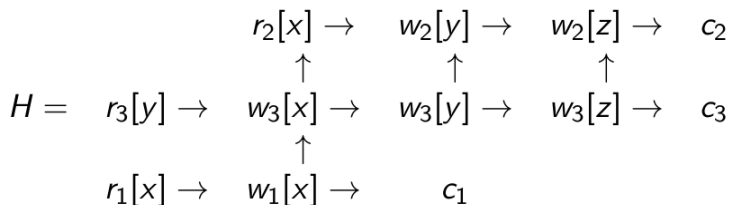
# Mögliche Fehler bei unkontrolliertem Mehrbenutzerbetrieb

*Dirty read* (Abhängigkeit von nicht freigegebenen Änderungen)

Schritt	$T_1$	$T_2$
1.	read(A,a <sub>1</sub> )	
2.	a <sub>1</sub> := a <sub>1</sub> - 300	
3.	write(A,a <sub>1</sub> )	
4.		read(A,a <sub>2</sub> )
5.		a <sub>2</sub> := a <sub>2</sub> * 1.03
6.		write(A,a <sub>2</sub> )
7.	read(B,b <sub>1</sub> )	
8.	...	
9.	abort	

Zum *phantom problem* und *non-repeatable read* siehe VL.

# Beispiel einer Historie



Ein Pfeil  $A \rightarrow B$  im Diagramm bedeutet, dass  $A$  vor  $B$  ausgeführt werden muss.

Z.B. muss im Beispiel  $r_3[y]$  vor  $w_3[x]$  ausgeführt werden, die Reihenfolge von  $r_3[y]$  und  $r_1[x]$  ist aber irrelevant.

# Äquivalenz zweier Historien

Zwei Historien  $H$  und  $H'$  sind **(konflikt-) äquivalent**, falls sie die gleiche Menge von Transaktionen (samt aller zugehörigen Operationen) enthalten und die Konfliktoperationen der nicht abgebrochenen Transaktionen in der gleichen Weise anordnen.

Zwei äquivalente Historien liefern das gleiche Endergebnis.



# Serialisierbarkeitsgraph

Der **Serialisierbarkeitsgraph**  $SG(H)$  einer Historie  $H = \{T_1, \dots, T_n\}$  ist ein gerichteter Graph, der folgendermaßen entsteht:

- Die Knoten sind die erfolgreich abgeschlossenen Transaktionen aus  $H$ .
- Eine Kante von  $T_i$  nach  $T_j$  existiert genau dann, wenn es zwei Konfliktoperationen  $p_i \in T_i$ ,  $p_j \in T_j$  gibt mit  $p_i < p_j$ .

# Serialisierbarkeitsgraph

Der **Serialisierbarkeitsgraph**  $SG(H)$  einer Historie  $H = \{T_1, \dots, T_n\}$  ist ein gerichteter Graph, der folgendermaßen entsteht:

- Die Knoten sind die erfolgreich abgeschlossenen Transaktionen aus  $H$ .
- Eine Kante von  $T_i$  nach  $T_j$  existiert genau dann, wenn es zwei Konfliktoperationen  $p_i \in T_i$ ,  $p_j \in T_j$  gibt mit  $p_i < p_j$ .

Eine Historie  $H$  heißt **seriell**, wenn alle beteiligten Transaktionen strikt nacheinander ausgeführt werden.

# Serialisierbarkeitsgraph

Der **Serialisierbarkeitsgraph**  $SG(H)$  einer Historie  $H = \{T_1, \dots, T_n\}$  ist ein gerichteter Graph, der folgendermaßen entsteht:

- Die Knoten sind die erfolgreich abgeschlossenen Transaktionen aus  $H$ .
- Eine Kante von  $T_i$  nach  $T_j$  existiert genau dann, wenn es zwei Konfliktoperationen  $p_i \in T_i$ ,  $p_j \in T_j$  gibt mit  $p_i < p_j$ .

Eine Historie  $H$  heißt **seriell**, wenn alle beteiligten Transaktionen strikt nacheinander ausgeführt werden.

Eine Historie  $H$  heißt **serialisierbar**, wenn  $H$  konflikt-äquivalent zu einer seriellen Historie ist.

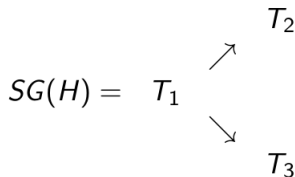
Äquivalente Charakterisierung:  $SG(H)$  ist azyklisch.

# Serialisierbarkeitsgraph: Beispiel

- Historie  $H$

$H = w_1[x] \rightarrow w_1[y] \rightarrow c_1 \rightarrow r_2[x] \rightarrow r_3[y] \rightarrow w_2[x] \rightarrow c_2 \rightarrow w_3[y] \rightarrow c_3$

- $SG(H)$



# Weitere Definitionen

In einer Historie  $H$  **liest** eine Transaktion  $T_i$  von einer Transaktion  $T_j$ , wenn die folgenden Bedingungen erfüllt sind:

- $T_j$  schreibt mindestens einen Datensatz  $A$ , den  $T_i$  anschließend liest.
- $T_j$  wird nicht vor dem Lesevorgang von  $T_i$  abgebrochen.
- Falls es zwischenzeitliche Schreibzugriffe auf  $A$  durch andere Transaktionen gibt, so werden diese vor dem Lesen von  $T_i$  zurückgesetzt.

# Rücksetzbarkeit

Eine Historie heißt **rücksetzbar**, wenn gilt:

Immer wenn eine Transaktion  $T_i$  von einer anderen Transaktion  $T_j$  liest ( $i \neq j$ ) und  $c_i \in H$ , dann gilt  $c_j < c_i$ .

*Bsp.:* Die Historie

$$H = w_1[x], r_2[x], w_2[y], c_2, a_1$$

ist nicht rücksetzbar.

# Vermeidung von kaskadierendem Rücksetzen

Schritt	$T_1$	$T_2$	$T_3$	$T_4$	$T_5$
0.	...				
1.	$w_1(A)$				
2.		$r_2(A)$			
3.		$w_2(B)$			
4.			$r_3(B)$		
5.			$w_3(C)$		
6.				$r_4(C)$	
7.				$w_4(D)$	
8.					$r_5(D)$
9.	$a_1(\text{abort})$				

Eine Historie  $H$  **vermeidet kaskadierendes Rücksetzen**, wenn gilt:  
Immer wenn eine Transaktion  $T_i$  von einer anderen Transaktion  $T_j$  liest  
( $i \neq j$ ), dann gilt  $c_j < r_i[A]$ .

# Strikttheit

Eine Historie  $H$  heißt **strikt**, falls für zwei beliebige Operationen  $w_j[A] < o_i[A]$  ( $o_i[A] = r_i[A]$  oder  $o_i[A] = w_i[A]$ ) gilt, dass  $a_j < o_i[A]$  oder  $c_j < o_i[A]$ .