Mehrbenutzerkontrolle (Concurrency Control)

Übersicht

1. TRANSAKTIONEN

führen einen konsistenten Datenbankzustand in einen anderen konsistenten Datenbankzustand über.

Eigenschaften (ACID-Eigenschaften)

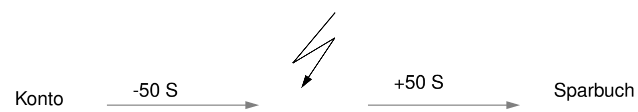
1. Atomarität
2. Concurrency (Parallelität)
3. Isolation
4. Dauerhaftigkeit

Eigenschaft 1: Atomarität

eine Transaktion wird entweder zur Gänze ausgeführt oder gar nicht

- BEGIN transaction
- COMMIT transaction
- ABORT transaction

Beispiel: Umbuchung: Konto - Sparbuch



Eigenschaft 2: Concurrency (Parallelität)

Transaktionen sollen parallel ausgeführt werden können

Traditionelles Korrektheitskriterium für Parallelausführung von Transaktionen ist die *Serialisierbarkeit.*

Serialisierbarkeit garaniert: Das Ergebnis einer verzahnten Ausführung mehrerer Transaktionen entspricht dem Ergebnis irgendeiner Hintereinanderausführung, d.h. seriellen Ausführung, dieser Transaktionen.

Verfahren:

- Sperrverfahren
- Zeitstempelverfahren

Beispiel: Serielle Ausführung

T_1	T_2
READ A	
A: = A - 10	
WRITE A	
READ B	
B: = B + 10	
WRITE B	
	READ B
	B: = B -20
	WRITE B
	READ C
	C:= C + 20
	WRITE C

Beispiel: Serialisierbare Ausführung

T_1	T_2
READ A	
	READ B
A: = A - 10	
	B: = B -20
WRITE A	
	WRITE B
READ B	
	READ C
B: = B + 10	
	C: = C + 20
WRITE B	
	WRITE C

Beispiel: nicht serialisierbare Ausführung

Transaktion A	*Transaktion B*
READ Konto	
	READ Konto
Konto:=Konto+10	
	Konto:= Konto+20
WRITE Konto	
	WRITE Konto

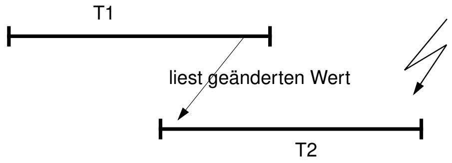
t

→ 10 öS “gehen verloren”

Eigenschaft 3: Isolation

Effekte einer Transaktion werden erst nach ihrem erfolgreichen Abschluß für andere Transaktionen sichtbar.

Dient zur Vermeidung des '*cascading rollback*'. Sieht eine Transaktion Teilergebnisse einer anderen Transaktion, die abgebrochen wird, zieht das auch den Abbruch der ersteren mit sich.



Eigenschaft 4: Dauerhaftigkeit

Die Effekte einer einmal erfolgreich abgeschlossenen Transaktion bleiben auch bei etwaigen Hard- und Softwarefehlern erhalte, die die Datenbank bzw. den zugreifenden oder die Datenbank verwaltenden Rechner betreffen.

Wird durch die Wiederanlauffähigkeit (Recovery) des Datenbanksystems gewährleistet.

Standardverfahren:

Logging (Protokollierung der Änderungen)

2. Mehrbenutzerbetrieb

- Allgemeines
- Ausführung von Transaktionen
- Konflikte zwischen Ausführungsschritten
- Konfliktserialisierbarkeit
- Sperrverfahren

Allgemeines

Daten werden von mehreren Benutzern gemeinsam verwendet.

Jeder Benutzer kann das System benutzen, als ob er der einzige wäre.

- Spezielle Techniken sind notwendig um zu verhindern, daß sich mehrere Benutzer gegenseitig stören(*Concurrency Control)*.
- Traditionelles Korrektheitskriterium: *Serialisierbarkeit* von Transaktionen.

Ausführung von Transaktionen

Menge von Transaktionen: **T** = { T_1, T_2, ...,T_n}

Ausführung von **T:**

enthält für jede Transaktion *T* in **T** alle ihre Operationen in derselben Reihenfolge wie in *T*.

Serielle Ausführung:

Ausführung, in der für jede Transaktion alle ihre Operationen unmittelbar hintereinander liegen.

Konflikte zwischen Ausführungsschritten

2 Operationen in einer Ausführung von Transaktionen stehen in Konflikt, wenn

1. sie auf dasselbe Objekt zugreifen,
2. sie zu verschiedenen Transaktionen gehören,
3. mindestens eine von ihnen eine Schreiboperation ist.

Konfliktserialisierbarkeit

Eine Ausführung einer Transaktionsmenge **T** ist *konflikt-serialisierbar*, wenn es eine serielle Ausführung von *T* gibt, sodaß jedes Paar in Konflikt stehender Operationen in beiden Ausführungen in derselben Reihenfolge vorkommt.

Hinweis:

Auch zu einer nicht konflikt-serialisierbaren Ausführung kann es eine serielle Ausführung mit demselben Ergebnis geben.

Beispiel: Konfliktserialisierbarkeit

I1

T_1:	T_2:
WRITE a	
WRITE b	
	WRITE a
	WRITE b

I2

T_1:	T_2:
WRITE a	
	WRITE a
WRITE b	
	WRITE b

I_2 ist konflikt-serialisierbar

Beispiel: Konfliktserialisierbarkeit

I1

T_1:	T_2:
WRITE a	
WRITE b	
	WRITE a
	WRITE b

I3

T_1:	T_2:
WRITE a	
	WRITE a
	WRITE b
WRITE b	

I_3 ist nicht konflikt-serialisierbar

Test auf Konfliktserialisierbarkeit

Zeichne einen Präzedenzgraph wie folgt:

1. Zeichne einen Knoten für jede Transaktion.
1. Zeichne eine Kante zwischen T_i und T_j ($i \neq j$) falls
   - T_j ein Objekt liest, das zuletzt von T_i geschrieben wurde, oder
   - T_j ein Objekt schreibt auf das zuletzt von T_i lesend oder schreibend zugegriffen wurde.

Die Ausführung ist konflikt-serialisierbar, falls der Präzedenzgraph azyklisch ist. Durch topologisches Sortieren erhält man eine äquivalente serielle Ausführung.

Beispiel 1

T*1*:	T*2*:	T*3*:	T*4*:
	READ a		
		READ a	
	UPDATE b		
		UPDATE a	
READ b			
			READ b
UPDATE c			
			UPDATE a
UPDATE b			

Sperrtypen

		Bestehende Sperre	
		S-Lock	X-Lock
Sperranforderung	S-Lock	+	−
	X-Lock	−	−

$+ \ldots$ Verträglich

$- \ldots$ Konflikt

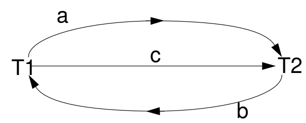
Wohlgeformte Transaktionen

... erfüllen folgende Bedingungen:

1. eine Transaktion *T* greift auf ein Objekt *O* nur zu, wenn sie dieses vorher gesperrt hat
2. *T* sperrt kein Objekt, das sie bereits gesperrt hat
3. *T* gibt kein Objekt frei, das sie nicht vorher gesperrt hat
4. Vor *EOT* werden alle Sperren, die in *T* gesetzt wurden, wieder freigegeben.

Sperren allein ist zuwenig

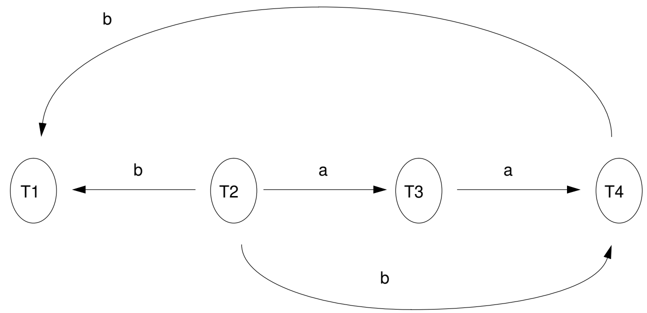
T1:	T2:
LOCK *a*	
LOCK *c*	
WRITE *a*	
WRITE *c*	
UNLOCK *a*	
	LOCK *a*
	WRITE *a*
	LOCK *b*
	UNLOCK *a*
	WRITE *b*
	UNLOCK *b*
UNLOCK *c*	
LOCK *b*	
WRITE *b*	
UNLOCK *b*	
	LOCK *c*
	WRITE *c*
	UNLOCK *c*



Sperrverfahren

- 2-Phasen-Sperrverfahren
- Baumprotokoll
- Hierarchisches Sperrprotokoll

Lösung zu Beispiel 1



Serialisierbar, da Abhängigkeitsbeziehungen **zyklenfrei** sind. Äquivalente serielle Ausführung: *T2, T3, T4, T1*.

Sperrverfahren

Ziel: sperre mögliche “Konkurrenten” aus

- Lesesperre (auch andere dürfen lesen, aber niemand darf schreiben)
- Schreibsperre (niemand sonst darf lesen o. schreiben)

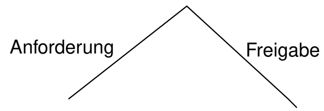
2-Phasen Sperrverfahren

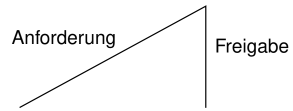
Standardverfahren:

Eine Transaktion entspricht dem 2-Phasensperrverfahren, wenn sie nach der ersten Sperrenfreigabe keine neue Sperre mehr anfordert.

Eine Menge von Transaktionen, die dem 2-Phasensperrverfahren gehorcht, ist serialisierbar.

Zyklisches Warten auf Sperrenfreigabe (Deadlock) kann auftreten.





2-Phasen-Sperrprotokoll

T_1	1. LOCK	A	T_2	1. LOCK	A
	2. WRITE	A		2. WRITE	A
	3. LOCK	B		3. LOCK	B
	4. UNLOCK	A		4. WRITE	B
	5. WRITE	B		5. WRITE	A
	6. UNLOCK	B		6. UNLOCK	A
				7. WRITE	B
				8. UNLOCK	B

Deadlock

T_1:	T_2:	
LOCK a		
	LOCK b	
(LOCK b)		← T1 wartet auf b
	(LOCK a)	← T2 wartet auf a

Hierarchisches Sperrprotokoll

2 Arten von Sperren:

- e**X**clusive (X-Sperre)
- **I**ntention (I-Sperre)

  “Absichtserklärung”, daß später ein Subobjekt (Teilbaum) gesperrt wird

Kompatibilitätsrelation:

T_2 \ T_1	X-Lock	I-Lock
X-Lock	Konflikt	Konflikt
I-Lock	Konflikt	kompatibel

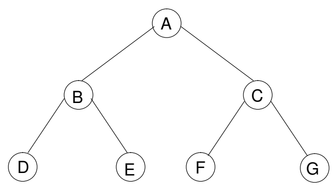
Hierarchische Sperrprotokolle (Fortsetzung)

Eine wohlgeformte 2-Phasen-Transaktion T gehorcht dem Hierarchischen Sperrprotokoll, wenn gilt:

- T greift auf die Knoten des Baumes in hierarchischer Reihenfolge von der Wurzel ausgehend zu.
- T sperrt einen Knoten K nur dann, wenn sie eine **I**-Sperre auf alle Vorgänger von K besitzt.
- Eine X-Sperre auf einem Knoten impliziert eine X-Sperre an allen Nachfolgern im Baum.
- T gibt keinen Knoten frei, bevor sie nicht alle Nachfolger von T freigegeben hat.

Beipiel: Hierarchisches Sperrprotokoll

	T_1		T_2		T_3	
1.	I-LOCK	A				
2.			I-LOCK	A		
3.			I-LOCK	B		
4.					I-LOCK	A
5.	I-LOCK	B				
6.			X-LOCK	E		
7.	X-LOCK	D				
8.			UNLOCK	E		
9.	UNLOCK	D				
10.			UNLOCK	B		
11.			UNLOCK	A		
12.	UNLOCK	B				
13.					X-LOCK	B
14					I-LOCK	C
15.					X-LOCK	F
16.	UNLOCK	A				
17.					UNLOCK	B
18.					UNLOCK	F
19.					UNLOCK	C
20.					UNLOCK	A



Zeitstempelverfahren (Idee)

Ziel: Transaktionen greifen in der Reihenfolge ihrer Startzeitpunkte auf Objekte zu.

- Transaktionen werden aufgrund ihrer Startzeitpunkte gereiht.
- Lese- und Schreibzugriffe auf Objekte werden bei diesen vermerkt.
- Ein Zugriff auf ein Objekt wird vom System verhindert, wenn dieser die Serialisierungsreihenfolge verletzten würde, d.h. wenn dieser nicht der Reihenfolge der Startzeitpunkte der Transaktionen entsprechen würde.

Zeitstempelverfahren (Realisierung)

- markiere Transaktionen mit ihren Startzeitpunkten
- jedes Objekt erhält einen *Lesestempel* (Startzeitpunkt der jüngsten lesendenTransaktion) und einen *Schreibstempel* (Startzeitpunkt der jüngsten schreibenden Transaktion)
- eine Transaktion darf ein Objekt lesen, wenn ihr Startzeitpunkt nach dem Schreibstempel des Objektes liegt; ansonsten wird sie abgebrochen
- eine Transaktion darf ein Objekt schreiben, wenn ihr Startzeitpunkt nach dem Lese- oder Schreibstempel des Objektes liegt, ansonsten wird sie abgebrochen und zurückgesetzt

Beispiel: Zeitstempelverfahren

T*1*	T2	T*3*	A	B	C
200	130	175	RT = 0 WT = 0	RT = 0 WT = 0	RT = 0 WT = 0
(1) READ B				RT = 200	
(2)	READ A		RT = 130		
(3)		READ C			RT = 175
(4) WRITE B				WT = 200	
(5) WRITE A			WT = 200		
(6)	WRITE C				
(7)		WRITE A	WT = 200		

Livelock

T*1*	T2	T*1*	T2	A	B
100	110	120	130	RT = 0 WT = 0	RT = 0 WT = 0
(1) WRITE B					WT = 100
(2)	WRITE A			WT = 110	
(3) READ A					
(4)		WRITE B			WT = 120
(5)	READ B				
(6)			WRITE A	WT = 130	
(7)		READ A			

Beispiel

Parallele Transaktionen werden mit Hilfe des Zeitstempelverfahrens synchronisiert. Gegeben sind folgende Transaktionen mit ihren Zeitstempeln:

T1	*T2*	*T3*	*a*	*b*	*c*
45	65	90	*RT = WT = 0*	*RT = WT = 0*	*RT = WT = 0*
READ *a*			*RT = 45*		
	WRITE *b*			*WT = 65*	
	WRITE *c*				*WT = 65*
		WRITE *a*	*WT = ...*		
WRITE *c*					*WT = ...*
		READ *c*			*RT = ...*
READ *b*				*RT = ...*	

Beispiel (Forts.)

1. Tragen Sie in der obigen Darstellung die fehlenden Werte für die Lese- und Schreibstempel der Objekte a, b und c ein. Kennzeichnen Sie den Abbruch einer Transaktion.

2. Welche “Serialisierungsreihenfolge” von T1, T2 und T3 wird in der obigen Ausführung durch die Anwendung des Zeitstempelverfahrens erzwungen?

Lösung

1.

T1	*T2*	*T3*	*a*	*b*	*c*
45	65	90	*RT = WT = 0*	*RT = WT = 0*	*RT = WT = 0*
READ *a*			*RT = 45*		
	WRITE *b*			*WT = 65*	
	WRITE *c*				*WT = 65*
		WRITE *a*	*WT = 90*		
WRITE *c*					*WT = 65*
		READ *C*			*RT = 90*
READ *b*				*RT = ...*	

2.

T2, T3, T1

Baumprotokoll

Annahmen:

1. Ansammlung von Objekten, die Knoten eines Baumes sind z.B.:
   - B*-Baum (siehe Kapitel “Physische Datenorganisation”)
   - Datenbank, Anwendung (Menge von Relationen), Relation, Tupel.
2. nur exklusive Sperren (wegen der Kompatibilitätsrelation kann jedes Objekt nur einmal mit einer Sperre belegt werden).

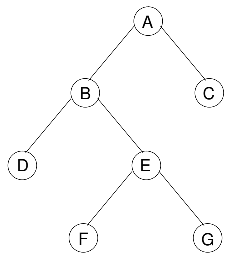
Baumprotokoll (Fortsetzung)

Eine Transaktion *T* erfüllt das Baumprotokoll, wenn gilt:

- Ein Objekt *O* wird nur dann von *T* gesperrt, wenn *T* bereits eine Sperre auf den Vater von *O* hält.
- Die obige Regel gilt nicht für das erste Objekt im Baum, das von *T* gesperrt wird.
- Sperren können jederzeit freigegeben werden.
- Ein Objekt *O*, das von einer Transaktion *T* gesperrt und wieder freigegeben wurde, kann von *T* anschließend nicht mehr gesperrt werden.

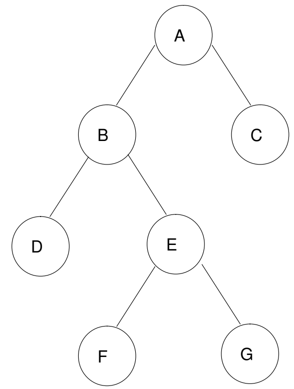
Beispiel: Baumprotokoll

	T_1	T_2	T_3
1.	LOCK A		
2.	LOCK B		
3.	LOCK D		
4.	UNLOCK B		
5.		LOCK B	
6.	LOCK C		
7.			LOCK E
8.	UNLOCK D		
9.			LOCK F
10.	UNLOCK A		
11.			LOCK G
12.	UNLOCK C		
13.			UNLOCK E
14.		LOCK E	
15.			UNLOCK F
16.		UNLOCK B	
17.			UNLOCK G
18.		UNLOCK E	



2. Beispiel

Gegeben ist folgende Ansammlung von Objekten, die Knoten eines Baumes sind:



Beispiel (Forts.)

Geben Sie für die Transaktionen *T1, T2* und *T3* an, ob sie das Baumprotokoll erfüllen. Falls eine Transaktion das Baumprotokoll verletzt, geben Sie das Objekt an, das falsch gesperrt wird.

T1: LOCK *A*, LOCK *B*, UNLOCK *A*, LOCK *C*, UNLOCK *C*, UNLOCK *B*

T2: LOCK *B*, LOCK *E*, UNLOCK *E*, LOCK *D*, UNLOCK *B*, UNLOCK *D*

T3: LOCK *A*, LOCK *E*, LOCK *F*, UNLOCK *F*, UNLOCK *E*, UNLOCK *A*

Lösung

T1: verletzt durch LOCK *C*

T2: erfüllt

T3: verletzt durch LOCK *E*

3. Transaktionen in SQL

- Allgemeines
- SET TRANSACTION-Option
- Phänomene im Mehrbenutzerbetrieb

Allgemeines

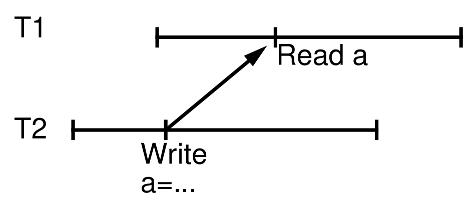
- kein besonderer Befehl zum Starten einer Transaktion
- Die meisten SQL-Statements können nur innerhalb einer Transaktion ablaufen (SELECT, INSERT, ...)
- Soll ein solches Statement bearbeitet werden und es ist noch keine Transaktion aktiv, wird vom System eine Transaktion gestartet.
- Abschließen mit COMMIT: Alle durchgeführten Änderungen werden dauerhaft gespeichert.
- Abbrechen mit ROLLBACK: Alle durchgeführten Änderungen werden ungültig.
- Setzen verschiedener Attribute: SET TRANSACTION

SET TRANSACTION

- SET TRANSACTION READ ONLY:
  nur lesender Zugriff innerhalb der Transaktion
- SET TRANSACTION READ WRITE:
  Lese- u. Schreibzugriff, sofern Benutzer berechtigt

Dirty Read

T1 liest einen von T2 veränderten Wert, bevor T2 mit COMMIT abgeschlossen wurde.

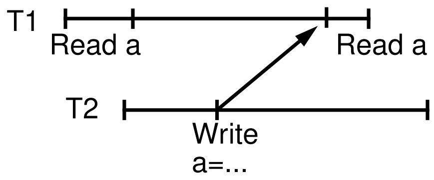


Nonrepeatable Read

T1 liest einen Satz.

Zu einem späteren Zeitpunkt wird derselbe Satz von T1 noch einmal gelesen.

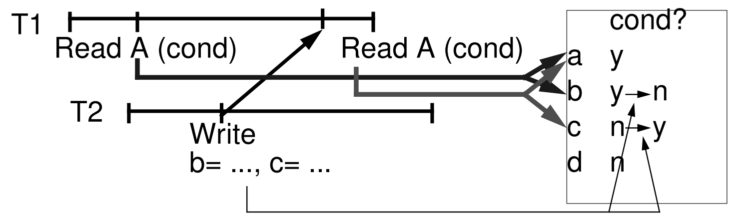
Zwischen den beiden Leseoperationen von T1 hat T2 den Wert dieses Satzes verändert. T1 bekommt also zwei verschiedene Werte zurückgeliefert.



Phantom Read

T1 liest eine Menge von Sätzen aufgrund bestimmter Kriterien. Dieser Lesevorgang wird später von T1 wiederholt.

Zwischen den beiden Leseoperationen ändert T2 einige Werte, sodaß beim 2. Lesevorgang eine andere Menge von Sätzen die Kriterien erfüllt und an T1 zurückgeliefert wird.



Isolation levels

Vermeidung der Phänomene durch *Isolation levels*

Level	Dirty Read	Nonrep. Read	Phantom Read
READ UNCOMMITTED	möglich	möglich	möglich
READ COMMITTED	nicht möglich	möglich	möglich
REPEATABLE READ	nicht möglich	nicht möglich	möglich
SERIALIZABLE	nicht möglich	nicht möglich	nicht möglich

Beispiel:

SET TRANSACTION READ WRITE

ISOLATION LEVEL SERIALIZABLE