

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zürich

Architektur und Implementierung von Datenbanksystemen WS 05/06

Dr. Jens-Peter Dittrich
jens.dittrich@inf
www.inf.ethz.ch/~jensdi
Institut für Informationssysteme



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zürich

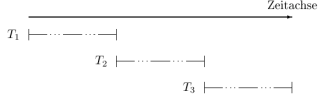
Mehrbenutzersynchronisation



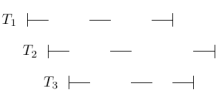
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zürich

Mehrbenutzersynchronisation (ACID: Wie wird Isolation realisiert?)

- Ausführen der drei Transaktionen T_1 , T_2 , T_3
 - im Einbenutzerbetrieb



- im Mehrbenutzerbetrieb



26. Januar 2006 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zürich

Verlorengegangene Änderungen (lost updates)

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)	

- Effekt von T_2 geht verloren

26. Januar 2006 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zürich

Abhängigkeit von nicht freigegebenen Änderungen

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

- wird auch als "dirty read" bezeichnet
- Transaktion T_2 wurde auf Basis inkonsistenter Daten durchgeführt

26. Januar 2006 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zürich

Phantomproblem

T_1	T_2
insert into Konten values (C, 1000, ...)	select sum(KontoStand) from Konten
	select sum(KontoStand) from Konten

- T_2 führt dasselbe SQL statement zweimal aus mit unterschiedlichem Ergebnis
- der neueingefügte Wert von T_1 wird als "Phantom" bezeichnet

26. Januar 2006 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf

ETH
Hörsaal für Informatiker
ETH Zürich

Serielle Ausführung von T_1 und T_2 : $T_1|T_2$

Schritt	T_1	T_2
1.	BOT	
2.	read(A)	
3.	write(A)	
4.	read(B)	
5.	write(B)	
6.	commit	
7.		BOT
8.		read(C)
9.		write(C)
10.		read(A)
11.		write(A)
12.		commit

26. Januar 2006 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 7

ETH
Hörsaal für Informatiker
ETH Zürich

Serialisierbarkeit

- Serialisierbare Historie von T_1 und T_2
 - Historie ist äquivalent zu einer seriellen Historie $T_1|T_2$
 - dennoch parallele (verzahnte) Ausführung möglich
 - Datenbasis ist in demselben Zustand wie bei einer seriellen Historie

Schritt	T_1	T_2
1.	BOT	
2.	read(A)	
3.		BOT
4.		read(C)
5.	write(A)	
6.		write(C)
7.	read(B)	
8.	write(B)	
9.	commit	
10.		read(A)
11.		write(A)
12.		commit

26. Januar 2006 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 8

ETH
Hörsaal für Informatiker
ETH Zürich

Nicht serialisierbare Historie

- Bezogen auf das Datenobjekt A kommt T_1 vor T_3
- Bezogen auf das Datenobjekt B kommt T_3 vor T_1
- Diese Historie ist **nicht** äquivalent zu einer der beiden seriellen Ausführungen $T_1|T_3$ oder $T_3|T_1$

Schritt	T_1	T_3
1.	BOT	
2.	read(A)	
3.	write(A)	
4.		BOT
5.		read(A)
6.		write(A)
7.		read(B)
8.		write(B)
9.		commit
10.	read(B)	
11.	write(B)	
12.	commit	

26. Januar 2006 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 9

ETH
Hörsaal für Informatiker
ETH Zürich

Zwei verzahnte Überweisungstransaktionen

- Achtung: diese Historie ist **nicht** serialisierbar
- Trotzdem wäre die Datenbasis in diesem speziellen Fall konsistent!
- Das ist aber Zufall und liegt an der in diesem Fall verwendeten Anwendungssemantik!

Schritt	T_1	T_3
1.	BOT	
2.	read(A, a ₁)	
3.	a ₁ := a ₁ - 50	
4.	write(A, a ₁)	
5.		BOT
6.		read(A, a ₂)
7.		a ₂ := a ₂ - 100
8.		write(A, a ₂)
9.		read(B, b ₂)
10.		b ₂ := b ₂ + 100
11.		write(B, b ₂)
12.		commit
13.	read(B, b ₁)	
14.	b ₁ := b ₁ + 50	
15.	write(B, b ₁)	
16.	commit	

26. Januar 2006 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 10

ETH
Hörsaal für Informatiker
ETH Zürich

Zwei verzahnte Überweisungstransaktionen

- Achtung: diese Historie ist **nicht** serialisierbar
- In diesem Fall wäre die Datenbasis inkonsistent!
- Hier sorgt die Anwendungssemantik (+) für eine inkonsistente Datenbasis!
- Aus Sicht des DBMS ist diese Semantik aber nicht erkennbar und darf nicht berücksichtigt werden!

Schritt	T_1	T_3
1.	BOT	
2.	read(A, a ₁)	
3.	a ₁ := a ₁ - 50	
4.	write(A, a ₁)	
5.		BOT
6.		read(A, a ₂)
7.		a ₂ := a ₂ + 1.03
8.		write(A, a ₂)
9.		read(B, b ₂)
10.		b ₂ := b ₂ + 1.03
11.		write(B, b ₂)
12.		commit
13.	read(B, b ₁)	
14.	b ₁ := b ₁ + 50	
15.	write(B, b ₁)	
16.	commit	

26. Januar 2006 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 11

ETH
Hörsaal für Informatiker
ETH Zürich

Rücksetzbare Historien (recoverable schedules)

- Zusätzliche Anforderung (neben Serialisierbarkeit): Jede Transaktion muss vor Ausführung eines commit lokal zurückgesetzt werden können.
- Eine Historie heisst rücksetzbar, falls immer die schreibende Transaktion $T_{j\#i}$ vor der lesenden Transaktion T_i ihr commit durchführt.
- Anders ausgedrückt: eine Transaktion T_i darf erst dann ihr commit durchführen, wenn alle Transaktionen $T_{j\#i}$, von denen sie **gelesen** hat, beendet sind.

26. Januar 2006 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 12

ETH
 Eidgenössische Technische Hochschule Zürich
 Swiss Federal Institute of Technology Zürich

Kaskadierendes 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 (abort)				

- T_2 ist abhängig von T_1 , T_3 ist abhängig von T_2 , usw.
- In Schritt 9 bricht T_1 ab.
- Folge: es müssen alle Transaktionen T_2 bis T_5 zurückgesetzt werden! (Schneeballeffekt)
- Ablhilfe: Änderungen erst nach dem commit **lesen**.

26. Januar 2006 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf 13

ETH
 Eidgenössische Technische Hochschule Zürich
 Swiss Federal Institute of Technology Zürich

Strikte Historien (strict schedules)

- Zusätzliche Anforderung (neben Vermeidung von Kaskadierendem Rücksetzen): veränderte Daten einer laufenden Transaktion dürfen nicht **überschrieben** werden
- D.h. ändert eine Transaktion T_1 einen Wert A, darf dieser Wert von keiner anderen Transaktion gelesen oder überschrieben werden bis T_1 abgeschlossen wurde durch commit
- siehe striktes Zwei-Phasen-Sperrprotokoll, exklusives Sperren

26. Januar 2006 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf 14

ETH
 Eidgenössische Technische Hochschule Zürich
 Swiss Federal Institute of Technology Zürich

Beziehungen zwischen den Historien

26. Januar 2006 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf 15

ETH
 Eidgenössische Technische Hochschule Zürich
 Swiss Federal Institute of Technology Zürich

Der Datenbank-Scheduler

- Muss Einzeloperationen so ausführen, dass die resultierende Historie serialisierbar ist
- Meist wird zusätzlich verlangt: Historie ohne kaskadierendes Rücksetzen
- im folgenden: Techniken für die Realisierung eines DB-Schedulers

26. Januar 2006 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf 16

ETH
 Eidgenössische Technische Hochschule Zürich
 Swiss Federal Institute of Technology Zürich

Beziehungen zwischen den Historien

Scheduler verlangt diese Historien D.h. der Schnitt von serialisierbaren Historien und denjenigen ohne kaskadierendes Rücksetzen

26. Januar 2006 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf 17

ETH
 Eidgenössische Technische Hochschule Zürich
 Swiss Federal Institute of Technology Zürich

Implementierung des Datenbank-Scheduler

- Pessimistische Verfahren
 - sperrbasiert
 - zeitstempelbasiert
- Optimistische Verfahren
 - optimistische Synchronisation

26. Januar 2006 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf 18

ETH
Hörsaal für Informatiker
ETH Zürich

Sperrbasierte Synchronisation

- Zwei Sperrmodi
 - S (Shared lock, read, Lesesperre)
 - X (eXclusive lock, write, Schreibsperre)
- Verträglichkeitsmatrix (Kompatibilitätsmatrix)

	NL	S	X
S	✓	✓	-
X	✓	-	-

26. Januar 2006 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 19

ETH
Hörsaal für Informatiker
ETH Zürich

Zwei-Phasen-Sperrprotokoll: Definition

- Jedes Objekt, das von einer TA benutzt werden soll, muss vorher entsprechend gesperrt werden
- Eine TA fordert eine Sperre, die sie bereits besitzt, nicht erneut an.
- Eine TA muss andere Sperren gemäss Verträglichkeitsmatrix beachten. Kann die Sperre nicht gewährt werden, wird die A in eine Warteschlange eingereiht.
- Jede TA durchläuft zwei Phasen
 - Eine **Wachstumsphase**, in der sie Sperren anfordern kann aber keine freigeben darf
 - Eine **Schrumpfungsphase**, in der sie ihre bisher erworbenen Sperren freigibt aber keine neuen anfordern darf
- Bei EOT muss die TA alle ihre erworbenen Sperren zurückgeben

26. Januar 2006 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 20

ETH
Hörsaal für Informatiker
ETH Zürich

2PL

- 2PL garantiert Serialisierbarkeit

26. Januar 2006 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 21

ETH
Hörsaal für Informatiker
ETH Zürich

Verzahnung zweier TAs gemäss 2PL

- T₁ modifiziert nacheinander die Datenobjekte A und B (z.B. eine Überweisung)
- T₂ liest nacheinander dieselben Datenobjekte A und B (z.B. zur Aufsummierung der beiden Kontostände)

Schritt	T ₁	T ₂	Bemerkung
1.	BOT		
2.	lockX(A)		
3.	read(A)		
4.	write(A)		
5.			
6.		BOT	
7.	lockX(B)	lockS(A)	T ₂ muß warten
8.	read(B)		
9.	unlockX(A)		T ₂ wecken
10.		read(A)	
11.		lockS(B)	T ₂ muß warten
12.	write(B)		
13.	unlockX(B)		T ₂ wecken
14.		read(B)	
15.	commit		
16.		unlockS(A)	
17.		unlockS(B)	
18.		commit	

26. Januar 2006 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 22

ETH
Hörsaal für Informatiker
ETH Zürich

Kaskadierendes Rücksetzen

- Was passiert, wenn T₁ direkt vor Schritt 15 scheitert?
- Dann muss T₂ zurückgesetzt werden, da T₂ 'dreckige' Daten von T₁ gelesen hat!
- 2PL vermeidet dies also nicht!

Schritt	T ₁	T ₂	Bemerkung
1.	BOT		
2.	lockX(A)		
3.	read(A)		
4.	write(A)		
5.			
6.		BOT	
7.	lockX(B)	lockS(A)	T ₂ muß warten
8.	read(B)		
9.	unlockX(A)		T ₂ wecken
10.		read(A)	
11.		lockS(B)	T ₂ muß warten
12.	write(B)		
13.	unlockX(B)		T ₂ wecken
14.		read(B)	
15.	commit		
16.		unlockS(A)	
17.		unlockS(B)	
18.		commit	

26. Januar 2006 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 23

ETH
Hörsaal für Informatiker
ETH Zürich

Strenges Zwei-Phasen Sperrprotokoll

- Problem: 2PL schliesst kaskadierendes Rücksetzen nicht aus
- Erweiterungen zum **strengen** 2PL
 - alle Sperren werden bis EOT gehalten
 - es gibt keine Schrumpfungsphase mehr
 - damit ist kaskadierendes Rücksetzen (Schneeballeffekt) ausgeschlossen

26. Januar 2006 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 24

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zürich

Verklemmungen (Deadlocks)

- Ein verklemmter Schedule

Schritt	T_1	T_2	Bemerkung
1.	BOT		
2.	lockX(A)		
3.		BOT	
4.		lockS(B)	
5.		read(B)	
6.	read(A)		
7.	write(A)		
8.	lockX(B)		T_1 muß warten auf T_2
9.	lockS(A)		T_2 muß warten auf T_1
10.	\Rightarrow Deadlock

26. Januar 2006 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 25

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zürich

Erkennung von Verklemmungen: Timeout

- Timeout-Verfahren
 - Falls die TA nach einem gewissen Zeitintervall keinen Fortschritt erzielt hat wird die TA abgebrochen
- Problem: Wahl des Zeitintervalls
 - zu klein: zu viele TAs unnötig abgebrochen
 - zu gross: Sperren werden zulange geduldet

26. Januar 2006 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 26

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zürich

Erkennung von Verklemmungen: Wartegraph

- Wartegraph mit zwei Zyklen
 - $T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_4 \rightarrow T_1$
 - $T_2 \rightarrow T_3 \rightarrow T_5 \rightarrow T_2$

```

graph TD
  T1((T1)) --> T2((T2))
  T2 --> T3((T3))
  T3 --> T4((T4))
  T4 --> T1
  T2 --> T5((T5))
  T3 --> T5
  T5 --> T2
  
```

- beide Zyklen können durch Rücksetzen von T_3 "gelöst" werden
- Zyklenerkennung durch Tiefensuche im Wartegraphen

26. Januar 2006 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 27

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zürich

Vermeidung von Verklemmungen: Preclaiming

Wie weiss die TA vorher, welche Datenobjekte sie benötigt?

Deswegen in der Praxis nicht realisierbar

(Analogie: Preplanning im DB-Puffer, Kap. 2, Folie 38)

26. Januar 2006 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 28

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zürich

Verklemmungsvermeidung durch Zeitstempel

- Jeder TA wird ein eindeutiger Zeitstempel (TS) zugeordnet
- TAs dürfen nicht mehr "bedingungslos" auf eine Sperre warten
- wound-wait Strategie
 - T_1 will Sperre erwerben, die von T_2 gehalten wird
 - Falls $TS(T_1) < TS(T_2)$: // "T₁ älter T₂?"
 - T_2 wird abgebrochen und zurückgesetzt **wound**
 - T_1 erhält die Sperre
 - Sonst
 - T_1 wartet auf die Freigabe der Sperre durch T_2 **wait**
- wait-die Strategie
 - T_1 will Sperre erwerben, die von T_2 gehalten wird
 - Falls $TS(T_1) < TS(T_2)$: // "T₁ älter T₂?"
 - T_1 wartet auf die Freigabe der Sperre durch T_2 **wait**
 - Sonst
 - T_1 wird abgebrochen und zurückgesetzt **die**
 - T_2 erhält die Sperre

26. Januar 2006 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 29

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zürich

Verklemmungsvermeidung durch Zeitstempel

- Zeitstempelmethode ist garantiert verklemmungsfrei
- Nachteile
 - zu viele TAs werden unnötig zurückgesetzt
 - wound-wait: junge TAs haben das Nachsehen
 - wait-die: alte TAs haben das Nachsehen

26. Januar 2006 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 30

ETH
Hörsaal für Informatiker
ETH Zurich

Multi-Granularity Locking

```

    graph TD
      DB[Datenbasis] --> S1[ ]
      DB --> S2[ ]
      S1 --> S11[ ]
      S1 --> S12[ ]
      S2 --> S21[ ]
      S2 --> S22[ ]
      S11 --> S111[Sätze]
      S11 --> S112[Sätze]
      S12 --> S121[Sätze]
      S12 --> S122[Sätze]
      S21 --> S211[Sätze]
      S21 --> S212[Sätze]
      S22 --> S221[Sätze]
      S22 --> S222[Sätze]
  
```

- Hierarchische Anordnung möglicher Sperrgranulate
- Analogie: Reservierung von Plätzen in der Mensa

26. Januar 2006 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 31

ETH
Hörsaal für Informatiker
ETH Zurich

Erweiterte Sperrmodi für MGL

- NL: keine Sperrung (no lock)
- S: Sperrung durch Leser
- X: Sperrung durch Schreiber
- IS (intention share): weiter unten in der Hierarchie ist eine Lesesperre beabsichtigt
- IX (intention exclusive): weiter unten in der Hierarchie ist eine Schreib Sperre beabsichtigt

26. Januar 2006 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 32

ETH
Hörsaal für Informatiker
ETH Zurich

MGL: Sperrprotokoll

- Bevor ein Knoten mit S oder IS gesperrt wird, müssen alle Vorgänger in der Hierarchie vom Sperrer (der TA, die die Sperre anfordert) im IX- oder IS-Modus gehalten werden.
- Bevor ein Knoten mit X oder IX gesperrt wird, müssen alle Vorgänger im IX-Modus gehalten werden.
- Die Sperren werden von unten nach oben (bottom up) freigegeben, so dass bei keinem Knoten die Sperre freigegeben wird, wenn die betreffende TA noch Nachfolger dieses Knotens gesperrt hat.
- Kompatibilitätsmatrix

	NL	S	X	IS	IX
S	✓	✓	-	✓	-
X	✓	-	-	-	-
IS	✓	✓	-	✓	✓
IX	✓	-	-	✓	✓

26. Januar 2006 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 33

ETH
Hörsaal für Informatiker
ETH Zurich

Datenbasis-Hierarchie mit Sperren

```

    graph TD
      D((D)) --- a1((a1))
      D --- a2((a2))
      a1 --- p1((p1))
      a1 --- p2((p2))
      a2 --- p3((p3))
      p1 --- s1((s1))
      p1 --- s2((s2))
      p2 --- s3((s3))
      p2 --- s4((s4))
      p3 --- s5((s5))
      p3 --- s6((s6))
  
```

- T₁ will Seite p₁ exklusiv sperren
- T₂ will Seite p₂ mit S-Sperre belegen
- T₃ will Segment a₂ mit X sperren

26. Januar 2006 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 34

ETH
Hörsaal für Informatiker
ETH Zurich

Datenbasis-Hierarchie mit Sperren

```

    graph TD
      D((D)) --- a1((a1))
      D --- a2((a2))
      a1 --- p1((p1))
      a1 --- p2((p2))
      a2 --- p3((p3))
      p1 --- s1((s1))
      p1 --- s2((s2))
      p2 --- s3((s3))
      p2 --- s4((s4))
      p3 --- s5((s5))
      p3 --- s6((s6))
  
```

- T₄ und T₅ sind blockiert (warten auf Freigabe von Sperren)
- T₄ und T₅ sind blockiert aber nicht verklemt
- Achtung: auch bei MGL können Verklemmungen auftreten!

26. Januar 2006 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 35

ETH
Hörsaal für Informatiker
ETH Zurich

Einfüge- und Löschoperationen, Phantome

- Naheliegende Methode
 - Vor dem Löschen eines Objektes muss die TA eine X-Sperre für dieses Objekt erwerben.
 - Man beachte aber, dass eine andere TA, die für dieses Objekt ebenfalls eine Sperre erwerben will, diese nicht mehr erhalten kann, falls die Löschransaktion erfolgreich (mit commit) abschliesst.
 - Beim Einfügen eines neuen Objektes erwirbt die einfügende TA eine X-Sperre.
- Funktioniert aber leider nicht

26. Januar 2006 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 36

ETH
Hörsaal für Informatiker
ETH Zürich

Phantomprobleme

T_1	T_2
<pre>select count(*) from prüfen where Note between 1 and 2;</pre>	<pre>insert into prüfen values(29555, 5001, 2137, 1);</pre>
<pre>select count(*) from prüfen where Note between 1 and 2;</pre>	

- Problem lässt sich dadurch lösen, dass man zusätzlich zu den Tupeln auch den Zugriffsweg, auf den man zu den Objekten gelangt, sperrt.
- Wenn also ein Sekundärindex auf Note existiert, würde der Indexbereich [1;2] für T_1 mit einer S-Sperre belegt werden.
- Wenn jetzt T_2 versucht, das Tupel (29555, 5001, 2137, 1) in Tabelle *prüfen* einzufügen, wird die T_2 blockiert.
- Zusätzlich müssen die Tupel gesperrt werden, da der Zugriff möglicherweise nicht über den Index erfolgt...

26. Januar 2006 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 37

ETH
Hörsaal für Informatiker
ETH Zürich

Optimistische Synchronisation

Grundidee: Erst beim Commit wird entschieden, ob es einen Konflikt gegeben hat

1. Lese-phase

- Alle Operationen der TA einschliesslich Änderungsoperationen werden ausgeführt
- Geänderte Werte werden noch nicht in die Datenbasis eingebracht sondern TA-lokal als Kopie gespeichert
- alle Änderungsoperationen werden zunächst auf diesen lokalen Variablen durchgeführt

2. Validierungsphase

- In dieser Phase wird entschieden, ob die TA möglicherweise in Konflikt mit anderen TA geraten ist.
- Dies wird mit Hilfe von Zeitstempeln entschieden, die den TAs in der Reihenfolge zugewiesen werden, in der sie in die Validierungsphase eintreten.

26. Januar 2006 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 38

ETH
Hörsaal für Informatiker
ETH Zürich

Optimistische Synchronisation

3. Schreibphase

- Die Änderungen derjenigen TAs, bei denen die Validierung positiv verlaufen ist, werden in die Datenbasis eingebracht. Sonst wird die TA zurückgesetzt.

Bemerkungen

- TAs, die zurückgesetzt werden, haben keinen Effekt auf die Datenbasis, deswegen gibt es auch kein kaskadierendes Zurücksetzen.
- In der Datenbasis gibt es also nur Änderungen erfolgreicher TAs.
- Optimistische Synchronisation eignet sich insbesondere für DB-Anwendungen mit überwiegend Lese-TAs.

26. Januar 2006 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 39

ETH
Hörsaal für Informatiker
ETH Zürich

Validierung bei optimistischer Synchronisation

- Vereinfachende Annahme: es ist immer nur eine TA gleichzeitig in der Validierungsphase
- Wir wollen eine Transaktion T_j validieren.
- Die Validierung ist erfolgreich, falls für alle älteren Transaktionen T_a - also solche die bereits ihre Validierung abgeschlossen haben - eine der beiden folgenden Bedingungen gilt:
 - T_a war zum Beginn von T_j bereits abgeschlossen - einschliesslich der Schreibphase.
 - Die Menge der von T_a geänderten Datenelemente, genannt *WriteSet* (T_a), enthält keine Elemente der Menge der gelesenen Datenelemente von T_j , genannt *ReadSet*(T_j). Es muss also gelten:

$$WriteSet(T_a) \cap ReadSet(T_j) = \emptyset$$

26. Januar 2006 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 40

ETH
Hörsaal für Informatiker
ETH Zürich

Synchronisation von Indexstrukturen

- Strenges 2PL würde kompletten Pfad auf dem Weg zum Tupel sperren
- Effekt: zu grosse Teile des B*-Baums unnötigerweise gesperrt
- Idee: füge rechts-Verweise auf den (Nicht-Blatt-) Knoten ein
- Schreiboperation: nur Knoten sperren, nicht den Pfad darüber

26. Januar 2006 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 41

ETH
Hörsaal für Informatiker
ETH Zürich

Synchronisation von Indexstrukturen

- Zwei parallele Operationen: *probe*(15) & *insert*(14)

- probe* erreicht Blattknoten [7;15] und wird dann angehalten
- insert* spaltet Blattknoten [7;15] in [7;11] und [14;15]
- probe* rechnet weiter: Schlüssel 15 ist nicht mehr auf dem Blatt [7;11]!
- probe* muss nach rechts laufen! (kann auch Nicht-Blatt-Knoten betreffen)

26. Januar 2006 Dr. Jens-Peter Dittrich / Institut für Informationssysteme / jens.dittrich@inf.ethz.ch 42

Transaktionsverwaltung in SQL

```

set transaction
[read only, | read write,]
[isolation level
  read uncommitted, |
  read committed, |
  repeatable read, |
  serializable,]
[diagnostics size . . . .]
    
```

- **read uncommitted**
 - liest beliebig inkonsistente Datenbestände
 - darf nur für read-only TAs spezifiziert werden
 - für browsing geeignet
 - TA behindert parallele Ausführung anderer Transaktionen nicht, da TA keine Sperren benötigt

Transaktionsverwaltung in SQL

- **read committed**
 - liest nur festgeschriebene Werte
 - TA kann innerhalb der Transaktion möglicherweise unterschiedliche Zustände von Werten sehen:
non-repeatable read-Problematik
- **repeatable read**
 - non-repeatable read ist ausgeschlossen
 - Phantomproblem möglich, beispielsweise durch eine parallele Änderungstransaktion, die dazu führt, dass ein Tupel jetzt ein Selektionsprädikat erfüllt, das es vorher nicht erfüllt hat
- **serializable**
 - Serialisierbarkeit gefordert
 - Dies ist der default.

T_1	T_2
read(A)	
	write(A)
	write(B)
	commit
	read(B)
	read(A)
	...

Nächste Woche

Ausblick

DataSpace Management Systems

Semester- und Masterarbeiten in der DBIS-Gruppe

