Einführung in Datenbanksysteme

Tutorium: Transaktionsverwaltung

Tutoren

Mit Folienmaterial aus der Vorlesung und …

Fachgebiet Datenbanksysteme und Informationsmanagement
Technische Universität Berlin

http://www.dima.tu-berlin.de/

Überblick

- Heute
  - Transaktionen und Ausgabe der Probeklausur
- Nächste Woche
  - Wiederholung und Fragen besprechen

Bekanntgabe

- Probeklausur
  - Zur Selbstkontrolle
  - Dauer 75 Min
  - Keine Musterlösung
  - Wird von uns nicht bewertet
  - Fragen können aber diskutiert werden

Transaktionen

- Eine Transaktion ist
  - eine **Folge von Operationen** (Aktionen)
  - die Datenbank bleibt **konsistent**
  - wobei das **ACID-Prinzip** eingehalten werden muss.

- ACID
  - Atomicity
    - Alles oder nichts Prinzip
  - Consistency
    - Datenbank ist vor und nach der Transaktion in einem konsistenten Zustand
  - Isolation
    - Transaktionen laufen ungestört von anderen Transaktionen ab
  - Durability
    - Mit Commit abgeschlossene Änderungen sind persistent

Schedule

- „Ablaufplan“ für Transaktion, bestehend aus Abfolge von Transaktionsoperationen

- Transaktionen hier T1, T2
- Zwei grundlegende Operationen
  - R(A): Lesen von A
  - W(A):Schreiben in A

- Weitere Begriffe
  - BOT(Begin of Transaction)
  - EOT(End of Transaction)
    - Auch **commit**
  - abort – Transaction abbrechen

- Serieller Schedule S
  - Transaktionen nacheinander ausgeführt

Schedule S

Schritt	T_1	T_2
1	BOT	
2	R(A)	
3	R(B)	
4	W(B)	
5	commit	
6		BOT
7		R(A)
8		W(A)
9		W(B)
10		commit

Schedule

- Was sind die Vor-/Nachteile von folgenden Schedules S1 und S2?

Schedule S1

Schritt	T_1	T_2
1	BOT	
2	R(A)	
3		BOT
4		R(A)
5	R(B)	
6	W(B)	
7	commit	
8		W(A)
9		W(B)
10		commit

VS.

Schedule S2

Schritt	T_1	T_2
1	BOT	
2	R(A)	
3	R(B)	
4	W(B)	
5	commit	
6		BOT
7		R(A)
8		W(A)
9		W(B)
10		commit

Vor- und Nachteile von Schedule S2

- Vorteile
  - ideale Isolation bei serieller Ausführung:
    - jede Transaktion hat Datenbank für sich alleine
- Nachteile
  - Transaktionen müssen aufeinander warten
  - keine Nebenläufigkeit
  - geringer Durchsatz an Transaktionen
- S2 ist ein serieller Schedule

Schedule S2

Schritt	T_1	T_2
1	BOT	
2	R(A)	
3	R(B)	
4	W(B)	
5	commit	
6		BOT
7		R(A)
8		W(A)
9		W(B)
10		commit

Vor- und Nachteile von Schedule S1

- Vorteil
  - Transaktionen sind verzahnt
  - Ressourcen effizienter
  - keine Transaktion muss warten
- Nachteile
  - Transaktionen teilen Datenbank
- Das „parallele" Nutzen von Datenbank kann kompliziert werden.

Schedule S1

Schritt	T_1	T_2
1	BOT	
2	R(A)	
3		BOT
4		R(A)
5	R(B)	
6	W(B)	
7	commit	
8		W(A)
9		W(B)
10		commit

Probleme im Mehrbenutzerbetrieb

T_1	T_2
`UPDATE Manager SET Gehalt=Gehalt+10 WHERE ID = X;`	
	`UPDATE Manager SET Gehalt=Gehalt + 30 WHERE ID = X;`
	Commit
Abort	

Problem: T2 liest den nicht nicht-committeten Zustand T1

DIRTY READ

Probleme im Mehrbenutzerbetrieb

T_1	T_2
Select Name From Mitarbeiter Where ID=100 ----> **‚Marcus‘**	
	UPDATE Mitarbeiter SET Name=‚Martin‘ Where ID=100
	commit
Select…… ---> **‚Martin‘**	

Problem: Mehrmaliges lesen der selben Tupel durch T_1 führt zu unterschiedlichen Ergebnissen

Non-repeatable Read

Probleme im Mehrbenutzerbetrieb

T_1	T_2	A
`read(A,x)`		10
	`read(A,x)`	10
`x:=x+1`		10
	`x:=x+1`	10
`write(x,A)`		11
	`write(x,A)`	11

Problem: Die Erhöhung von T1 wird nicht berücksichtigt

Lost Update

Folie nach Kai-Uwe Sattler (TU Ilmenau)

Probleme im Mehrbenutzerbetrieb

T_1	T_2
`X = SELECT COUNT(*)` `FROM Mitarbeiter`	
	`INSERT INTO Mitarbeiter` `VALUES ('Meier', 50000, …)`
	`commit`
`UPDATE Mitarbeiter` `SET Gehalt=Gehalt +10000/X`	
`commit`	

Problem: Meier geht nicht in die Gehaltsabrechnung ein. Meier ist das PHANTOM

Das Phantom-Problem

Beispiel nach Kai-Uwe Sattler (TU Ilmenau)

Zusammenfassung: Probleme im Mehrbenutzerbetrieb

- Abhängigkeiten von nicht freigegebenen Daten: *Dirty Read*
- Inkonsistentes Lesen: *Nonrepeatable Read*
- Verlorengegangenes Ändern: *Lost Update*
- Berechnungen auf unvollständigen Daten: *Phantom-Problem*

Übung

- Welche der bereits besprochenen Anomalien treten bzw. treten nicht auf bei den folgenden Schedules?

S1

Sch	T_1	T_2
1	R(X)	
2	W(X)	
3		R(X)
4		W(Y)
5		W(Z)
6	abort	
7		commit

Dirty read
T2 liest einen ungültigen Wert von X

S2

Sch	T_1	T_2
1	R(X)	
2		R(X)
3		W(Y)
4		W(X)
5		commit
6	R(X)	
7	W(Z)	
8	commit	

Nonrepeatable read
T1 liest zwei Mal X,
T2 aber liest und schreibt X in der Zwischenzeit

S3

Sch	T_1	T_2
1	R(X)	
2		R(X)
3	W(X)	
4		W(Y)
5		W(X)
6	commit	
7		commit

Lost update
T1 liest X,
T2 liest X bevor T1 wieder X schreibt

S4

Sch	T_1	T_2
1	R(X)	
2		R(Y)
3	W(Z)	
4	W(Y)	
5	commit	
6		commit

keine der Anomalie tritt auf

Serialisierbar

- Idee
  - Serielle Ausführung nicht nötig
    - Für jede Transaktion „nur so aussehen“ als wäre sie isoliert
    - dazu reicht die Existenz eines äquivalenten seriellen Ausführungsplans
  - Äquivalent heißt:
    - Die Transaktionen bezüglich S1 und S2 jeweils dieselben Werte lesen und
    - **S1 und S2 dieselben Endzustände erzeugen**

- Solch ein Ausführungsplan heißt **serialisierbar**.

Folie nach Prof. Dr. Christoph Dalitz

Übung

- Ist S2 serialisierbar?
- T1->T2 oder T2-T1

- Falls ja, gebt ein äquivalente Umformung in einen seriellen Schedule an!

Schedule S2

Schritt	T_1	T_2
1	R(A)	
2	A:=A-10	
3	W(A)	
4		R(A)
5		A:=A-20
6		W(A)
7	R(B)	
8	B:=B+10	
9	W(B)	
10		R(B)
11		B:=B+20
12		W(B)

Übung

serieller Schedule von S2

Schritt	T_1	T_2
1	R(A)	
2	A:=A-10	
3	W(A)	
4	R(B)	
5	B:=B+10	
6	W(B)	
7		R(A)
8		A:=A-20
9		W(A)
10		R(B)
11		B:=B+20
12		W(B)

Schedule S2

Schritt	T_1	T_2
1	R(A)	
2	A:=A-10	
3	W(A)	
4		R(A)
5		A:=A-20
6		W(A)
7	R(B)	
8	B:=B+10	
9	W(B)	
10		R(B)
11		B:=B+20
12		W(B)

(Konflikt-) Serialisierbar

- Frage: welche Operationen dürfen **gefahrlos**, d.h. ohne Auswirkung auf Endergebnis vertauscht werden?

- Ein Schedule ist (Konflikt-) Serialisierbar, wenn er ohne Vertauschung von Konflikt Operationen in einen seriellen Schedule umgeformt werden kann.

- Welche sind Konflikt Operationen?

- Gegeben Transaktionen T_i und T_k
  - $r_i(X)$ und $r_k(X)$ stehen nicht in Konflikt
  - $r_i(X)$ und $w_k(X)$ stehen in Konflikt
  - $w_i(X)$ und $w_k(X)$ stehen in Konflikt

Schedule S1

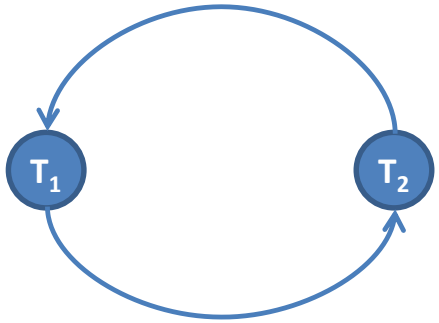
Schritt	T_1	T_2
1	R(A)	
2		R(B)
3	W(B)	
4		W(B)
5		W(C)
6	R(C)	

Folie nach Prof. Dr. Christoph Dalitz

Graph basierter Test auf (Konflikt-) Serialisierbarkeit

- Idee
  - Stelle Reihenfolge der **Konflikt**operationen durch Kanten in gerichtetem Graphen dar

- Aufbau von Konfliktgraphen
  - für jede Transaktion erstelle einen Knoten
  - für jeden Konflikt zwischen T füge eine gerichtete Kante



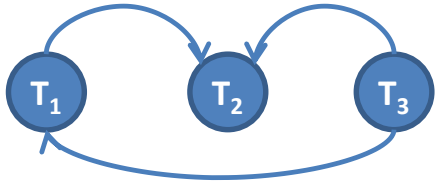
=> nicht Konfliktserialisierbar

Schedule S1

Schritt	T_1	T_2
1	R(A)	
2		R(B)
3	W(B)	
4		W(B)
5		W(C)
6	R(C)	

Bedeutung von Präzedenzgraph

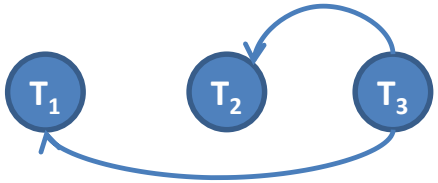
- Konfliktoperationen dürfen nicht vertauscht werden
  - T1 -> T2 bedeutet, dass T1 im äquivalenten Seriellen Schedule vor T2 kommen muss
  - Graph gibt also Reihenfolge der Transaktionen im äquivalenten seriellen Schedule an.
  - Ein Schedule ist (konflikt-) serialisierbar gdw. sein Präzedenzgraph **keine Zyklen** hat



Konfliktserialisierbar

Äquivalent-serieller Schedule

T3 -> T1 -> T2

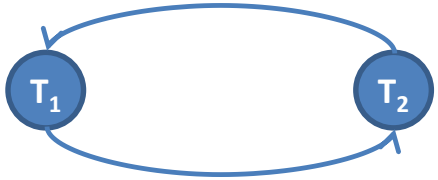


Konfliktserialisierbar

Äquivalente-serieller Schedule

T3 -> T1 -> T2

T3 -> T2 -> T1



nicht Konfliktserialisierbar

Aufgabe 1

- Gegeben seien die drei Schedules für die beiden Transaktionen T1 und T2 (BOT und COMMIT wurde zur Vereinfachung weggelassen).

- Schedule S1: r1(a), w1(a), r1(b), w1(b), r2(b), w2(b), r2(c), w2(c)
- Schedule S2: r1(a), r2(b), w1(a), w2(b), r1(b), r2(c), w1(b), w2(c)
- Schedule S3: r1(a), r2(b), w1(a), r1(b), w2(b), r2(c), w1(b), w2(c)

- Überprüfen Sie, ob der jeweilige Schedule konfliktserialisierbar ist oder nicht. Begründen Sie Ihre Entscheidung anhand Serialisierbarkeitsgraphen.

  Hinweis: Beachten Sie, dass die Konfliktserialisierbarkeit lediglich eine hinreichende, aber keine notwendige Bedingung für die Serialisierbarkeit ist. Somit können Sie aus der Tatsache, dass ein Schedule S nicht konfliktserialisierbar ist, nicht folgern, dass S nicht serialisierbar ist.

Aufgabe 1 - Lösung

S1:

T1	T2
r(A)	
w(A)	
r(B)	
w(B)	
	r(B)
	w(B)
	r(C)
	w(C)

T1 ⟶ T2

S2:

T1	T2
r(A)	
	r(B)
w(A)	
	w(B)
r(B)	
	r(C)
w(B)	
	w(C)

T1 ⟵ T2

S3:

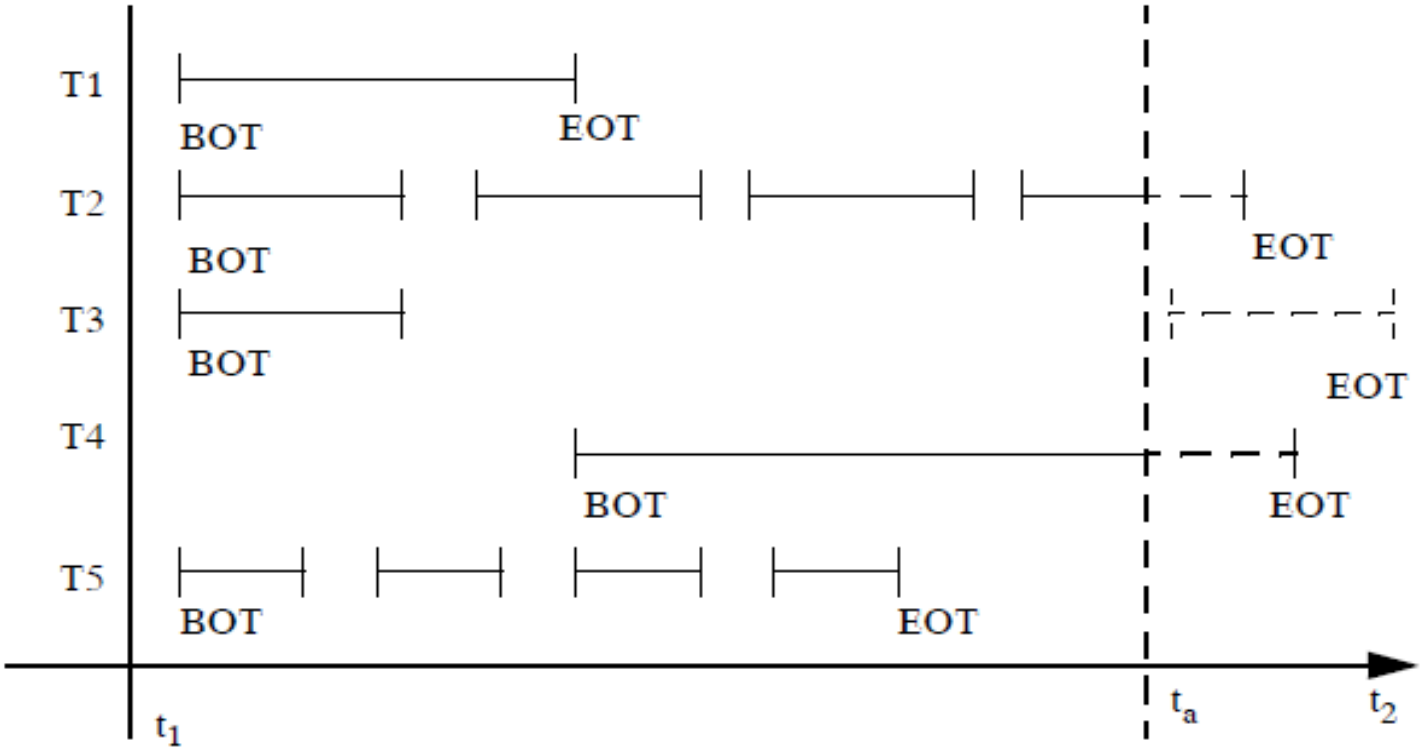
T1	T2
r(A)	
	r(B)
w(A)	
r(B)	
	w(B)
	r(C)
w(B)	
	w(C)

Schedule 2 ist ebenfalls und trotz der verschachtelung konfliktserialisierbar, auch hier ist der einzige konkurrierende Zugriff auf Element B. T2 ist vor T1:

Schedule 3 ist nicht konfliktserialisierbar, Hier überschreibt Transaktion 1 die Änderungen von Transaktion 2 (Lost Update). Der Serialisierbarkeitsgraph zeigt das mit einem Kreis.

Aufgabe 2

- In einem betrachteten Zeitraum [t1,t2] werden 5 Transaktionen (Tn) abgearbeitet. Zum Zeitpunkt ta erfolgt ein Systemabsturz.
- a) Erklären Sie, wie jede Transaktion behandelt werden muss.
- b) Gibt es einen Unterschied in der Behandlung von Systemabstürzen und Fehlern in Transaktionen?



Aufgabe 2

- a) Erklären Sie, wie jede Transaktion behandelt werden muss.

- **Lösung:** T1 und T5 sind Gewinner, da sie vor dem Systemabsturz terminierten. Alle drei anderen Transaktionen müssen vollständig zurückgesetzt werden, also auch bereits beendete Teilprozesse, da sonst gegen das Prinzip der Atomizität verstoßen würden.

- b) Gibt es einen Unterschied in der Behandlung von Systemabstürzen und Fehlern in Transaktionen?

- **Lösung:** Nach einem Systemabsturz müssen sämtliche zum Zeitpunkt des Absturzes noch nicht beendeten Transaktionen zurückgerollt werden. Selbiges gilt für die Änderungen einer später abgebrochenen Transaktion: Alle Änderungen müssen zurückgesetzt werden. Erlaubt die Datenbank das Lesen von Zwischenergebnissen(Dirty Read) muss überprüft werden, ob nicht bereits weitere Transaktionen mit den – jetzt nicht mehr korrekten – Zwischenergebnissen der abgebrochenen Transaktionen arbeiten (Non-Repeatable Read). Falls dem so ist, müssen auch diese zurückgerollt werden.