Ludwig Maximilians Universität München
Institut für Informatik
Lehr- und Forschungseinheit für Datenbanksysteme

Skript zur Vorlesung

Datenbanksysteme II

Sommersemester 2007

Kapitel 2: Synchronisation

Vorlesung: Christian Böhm

http://www.dbs.informatik.uni-muenchen.de/Lehre/DBSII

Synchronisation (Concurrency Control)

- **Serielle Ausführung** von Transaktionen ist unerwünscht, da die Leistungsfähigkeit des Systems beeinträchtigt ist (niedriger Durchsatz, hohe Wartezeiten)
- **Mehrbenutzerbetrieb** führt i.a. zu einer besseren Auslastung des Systems (z.B. Wartezeiten bei E/A-Vorgängen können zur Bearbeitung anderer Transaktionen genutzt werden)
- **Aufgabe der Synchronisation**
  Gewährleistung des logischen Einbenutzerbetriebs, d.h. innerhalb einer TA ist ein Benutzer von den Aktivitäten anderer Benutzer nicht betroffen

Anomalien im Mehrbenutzerbetrieb

- Verloren gegangene Änderungen (*Lost Updates*)
- Zugriff auf „schmutzige“ Daten (*Dirty Read / Dirty Write*)
- Nicht-reproduzierbares Lesen (*Non-Repeatable Read*)
- Phantomproblem

- **Beispiel:** Flugdatenbank

Passagiere	FlugNr	Name	Platz	Gepäck
	LH745	Müller	3A	8
	LH745	Meier	6D	12
	LH745	Huber	5C	14
	BA932	Schmidt	9F	9
	BA932	Huber	5C	14

Lost Updates

- Änderungen einer Transaktion können durch Änderungen anderer Transaktionen überschrieben werden und dadurch verloren gehen
- Bsp.: Zwei Transaktionen T1 und T2 führen je eine Änderung auf demselben Objekt aus
  - T1:
```
UPDATE Passagiere SET Gepäck = Gepäck+3
WHERE FlugNr = LH745 AND Name = „Meier“;
```
  - T2:
```
UPDATE Passagiere SET Gepäck = Gepäck+5
WHERE FlugNr = LH745 AND Name = „Meier“;
```
- Mgl. Ablauf:

T1	T2
`read(Passagiere.Gepäck, x1);`	`read(Passagiere.Gepäck, x2);`
	`x2 := x2 + 5;`
`x1 := x1+3;`	`write(Passagiere.Gepäck, x2);`
`write(Passagiere.Gepäck, x1);`	

- In der DB ist nur die Änderung von T1 wirksam, die Änderung von T2 ist verloren gegangen → Verstoß gegen *Durability*

Dirty Read / Dirty Write

- Zugriff auf „schmutzige“ Daten, d.h. auf Objekte, die von einer noch nicht abgeschlossenen Transaktion geändert wurden
- Bsp.:
  - T1 erhöht das Gepäck um 3 kg, wird aber später abgebrochen
  - T2 erhöht das Gepäck um 5 kg und wird erfolgreich abgeschlossen
- Mgl. Ablauf:

T1	T2

`UPDATE Passagiere`
`SET Gepäck = Gepäck+3;`
 	
`ROLLBACK;`
	
	`UPDATE Passagiere`
	`SET Gepäck = Gepäck+5;`
	`COMMIT;`

- Durch den Abbruch von T1 werden die geänderten Werte ungültig. T2 hat jedoch die geänderten Werte gelesen (*Dirty Read*) und weitere Änderungen darauf aufgesetzt (*Dirty Write*)
- Verstoß gegen ACID: Dieser Ablauf verursacht einen inkonsistenten DB-Zustand (*Consistency*) bzw. T2 muss zurückgesetzt werden (*Durability*)

Non-Repeatable Read

- Eine Transaktion sieht während ihrer Ausführung unterschiedliche Werte desselben Objekts
- Bsp.:
  - T1 liest das Gepäckgewicht der Passagiere auf Flug BA932 zwei mal
  - T2 bucht den Platz 3F auf dem Flug BA932 für Passagier Meier mit 5kg Gepäck
- Mgl. Ablauf:

T1	T2

`SELECT Gepäck FROM Passagiere`
`WHERE FlugNr = „BA932“;`
 	
`SELECT Gepäck FROM Passagiere`
`WHERE FlugNr = „BA932“;`
	
	`INSERT INTO Passagiere`
	`VALUES (BA932, Meier, 3F, 5);`
	`COMMIT;`

- Die beiden SELECT-Anweisungen von Transaktion T1 liefern unterschiedliche Ergebnisse, obwohl T1 den DB-Zustand nicht geändert hat → Verstoß gegen *Isolation*

Phantomproblem

- Ausprägung des nicht-reproduzierbaren Lesens, bei der neu generierte Daten, sowie meist bei der 2. TA Aggregat-Funktionen beteiligt sind
- Bsp.:
  - T1 druckt die Passagierliste sowie die Anzahl der Passagiere für den Flug LH745
  - T2 bucht den Platz 7D auf dem Flug LH745 für Phantomas
- Mgl. Ablauf:

T1	T2

`SELECT * FROM Passagiere`
`WHERE FlugNr = „LH745“;`
 	
`SELECT COUNT(*) FROM Passagiere`
`WHERE FlugNr = „ LH745“;`
	
	`INSERT INTO Passagiere`
	`VALUES (LH745, Phantomas, 7D, 2);`
	`COMMIT;`

- Für Transaktion T1 erscheint Phantomas noch nicht auf der Passagierliste, obwohl er in der danach ausgegebenen Anzahl der Passagiere berücksichtigt ist

Inhalt

1. Anomalien im Mehrbenutzerbetrieb
2. Serialisierbarkeit von Transaktionen
3. Sperrverfahren (Locking)
4. Behandlung von Verklemmungen
5. Synchronisation ohne Sperren

Schedules (1)

- Die nebenläufige Bearbeitung von Transaktionen geschieht für den Benutzer transparent, d.h. als ob die Transaktionen (in einer beliebigen Reihenfolge) hintereinander ausgeführt werden

- **Allgemeiner Schedule**:
  Ein **Schedule** (dt. auch „**Historie**“) für eine Menge $\{T_1, \ldots, T_n\}$ von Transaktionen ist eine Folge von Aktionen, die durch Mischen der Aktionen der Transaktionen T_i entsteht, wobei die Reihenfolge innerhalb der jeweiligen Transaktion beibehalten wird.
  - Frage: Warum darf die Reihenfolge der Aktionen innerhalb einer TA nicht verändert werden?

Schedules (2)

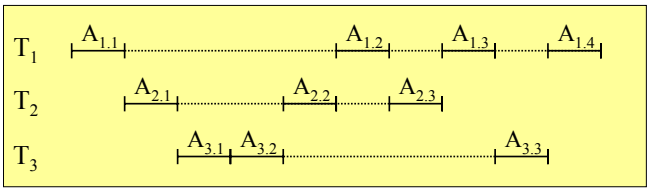
- **Serieller Schedule**:
  Ein **serieller Schedule** ist ein Schedule S von $\{T_1, \ldots, T_n\}$, in dem die Aktionen der einzelnen Transaktionen nicht untereinander verzahnt sondern in Blöcken hintereinander ausgeführt werden.

- **Serialisierbarer Schedule**:
  Ein Schedule S von $\{T_1, \ldots, T_n\}$ ist **serialisierbar**, wenn er dieselbe Wirkung hat wie ein beliebiger serieller Schedule von $\{T_1, \ldots, T_n\}$.

  Nur serialisierbare Schedules dürfen zugelassen werden!

Beispiel serieller Schedule

- Beliebiger Schedule:



- Serieller Schedule: