Transaktionsverwaltung

1. Schnellkurs: Serialisierbarkeit, Isolationslevel, Synchronisationsverfahren, Savepoints, Logging, Implementierungsaspekte
   → Härder, Rahm Buch
2. Erweiterte Transaktionskonzepte
   → Härder, Rahm Buch
3. Verteilte Transaktionsverwaltung (2PC, Deadlockerkennung)
4. Lastkontrolle
5. Replikationskontrolle, Cache Consistency
6. Recovery in Client/Server Umgebungen und von Anwendungsprogrammen
   → Franklin et al., SIGMOD 1992
   → Lomet, Weikum, SIGMOD 1998

- Das Thema verdient eigentlich eine eigene Vorlesung.

Transaktionskonzept

Eine Transaktion ist eine Folge von Operationen mit folgenden *ACID* Eigenschaften:

- **Atomicity:** Es werden alle Operationen oder gar keine Operation ausgeführt.
- **Consistency:** Wenn alle Operationen ausgeführt werden, dann überführt die Transaktion die Datenbasis von einem konsistenten in einen (anderen) konsistenten Zustand.
- **Isolation:** Im Mehrbenutzerbetrieb sind die Operationen der Transaktionen *serialisierbar*.
- **Durability:** Die Änderungen einer Transaktionen können nur durch eine andere Transaktionen überschrieben werden.

Komponenten der Transaktionsverwaltung:

- **Synchronisation:** Sorgt für *Isolation.*
- **Logging, Recovery, Commit-Behandlung:** Sorgt für *Atomicity* und *Durability.*
- **Integritätskontrolle:** Sorgt für *Consistency.*

Beispiele

Integritätsbedingung: Summe aller Kontostände ist eine Konstante.

Transaktion 1:

BOT;
andreas.konto += 100.000;
markus.konto –= 100.000;
COMMIT;

Transaktion 2:

BOT;
andreas.konto += 50.000;
ABORT;

Transaktion 3:

BOT;
andreas.konto += 70.000;
markus.konto –= 50.000;
COMMIT;

Nachweis der Serialisierbarkeit

- Eine Historie ist eine Folge von *(TID, Operation, Datum)* Elementen
- Beispiel:

$$(T_1, W, x); (T_2, R, y); (T_3, W, x); (T_1, R, y)$$

- Abhängigkeitsgraph zwischen Transaktionen. Kante, wenn zwei Transaktionen auf dasselbe Objekt mit nicht reihenfolgeunbhängigen Operationen zugreifen.
- Die Historie ist serialisierbar, wenn der Abhängigkeitsgraph keine Zyklen enthält.

Isolationslevel nach ANSI SQL 92

Level	**Dirty Read**	**Fuzzy Read**	**Phantom**
Read Uncommitted	möglich	möglich	möglich
Read Committed	unmöglich	möglich	möglich
Repeatable Read	unmöglich	unmöglich	möglich
Serializable	unmöglich	unmöglich	unmöglich

Dirty Read

$$x = 2; (T_1, W, x = 1); (T_2, R, x = 1); (T_1, abort, x = 2)$$

Fuzzy Read

$$(T_1, R, x = 1); (T_2, W, x = 2); (T_1, R, x = 2)$$

Phantom

$$(T_1, R, avg(x) = 2); (T_2, I, x = 1); (T_1, R, avg(x) = 1.5)$$

Zweiphasen Sperrprotokoll (2PL)

Prinzip

- vor jedem Objektzugriff muß Sperre mit ausreichendem Modus angefordert werden
- blockieren, falls Sperre nicht erteilt werden kann
- Zweiphasigkeit: Wachstums- und Schrumpfphase. D.h. Sperrfreigabe kann erst beginnen, wenn alle Sperren gehalten werden.
- Spätestens beim Commit oder Abort werden alle Sperren freigegeben.

Striktes 2PL

Beobachtungen

- Kann man Ende der Wachstumsphase genau bestimmen?
- Fehler während der Schrumpfphase können zu *Dirty Read* führen.

Striktes 2PL

- Freigabe aller Sperren erst mit dem Commit (oder Abort).

N.B.

- Striktes 2PL erzielt Isolationslevel 2.
- Serialisierbarkeit (Ausschluß von Phantomen) wird nur bei *Prädikatsperren* oder anderen besonderen Maßnahmen erreicht.

RX-Sperrverfahren

- Fordere R Sperre zum Lesen an.
- Fordere X Sperre zum Schreiben an.
- Ggf. Upgrade einer R zu einer X Sperre, wenn Objekt nach einem Lesen geschrieben werden soll.
- Kompatibilitätsmatrix:

angeforderter Modus \ aktueller Modus	NL	R	X
R	+	+	-
X	+	-	-

RUX-Sperrverfahren

- Sperrkonversionen führen oft zu Deadlocks

$$(T_1, R, x); (T_2, R, x); (T_1, W, x); (T_2, W, x)$$

- Erweitertes Sperrverfahren:
  - U-Sperre für Lesen mit Änderungsabsicht
  - Konversion $U \rightarrow X$ bei Änderung
  - Konversion $U \rightarrow R$ ansonsten

		aktueller Modus			
		NL	R	U	X
angeforderter Modus	R	+	+	-	-
	U	+	+	-	-
	X	+	-	-	-

Frage: Wieso ist die Matrix unsymmetrisch?

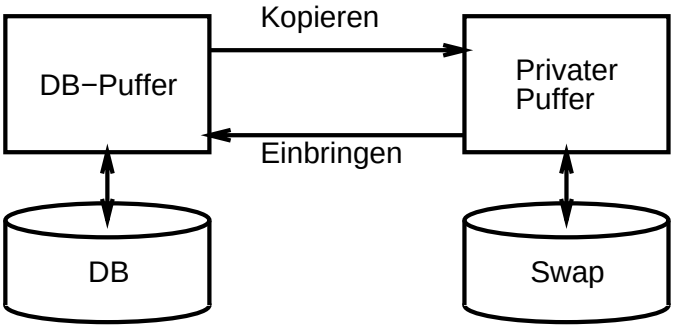
Optimistische Verfahren

Grundidee: 3-phasige Verarbeitung

- **Lesephase**
  - Durchführung der Operationen einer Transaktion.
  - Änderungen werden in einem privaten Puffer durchgeführt.
- **Validierungsphase**
  - Überprüfung, ob ein Lese-/Schreib- oder Schreib-/Schreib-Konflikt mit einer anderen Transaktion aufgetreten ist.
  - Konfliktauflösung durch Zurücksetzen der Transaktion.
  - Es wird in der Regel nur eine Transaktion gleichzeitig validiert.
- **Schreibphase** (bei positiver Validierung)
  - Propagieren der Änderungen

Optimistische Verfahren

Pufferverwaltung



Bewertung

- **Vorteile**
  - einfaches Rollback wegen NOSTEAL
  - keine Deadlocks
- **Nachteile**
  - Ressourcenverschwendung und unglückliche Benutzer durch mehr Rollbacks
  - Gefahr des Verhungerns von Transaktionen

Validierung bei optimistischen Verfahren

Voraussetzungen

- jede Transaktion merkt sich ihren *Read Set (RS)* und ihren *Write Set (WS)*
- eine Transaktion darf nur abgeschlossen werden, wenn sie alle Änderungen von zuvor abgeschlossenen Transaktionen gesehen hat
- N.B.: die Validierungsreihenfolge bestimmt die Serialisierungsreihenfolge

Backward Oriented Validierung BOCC

- Schließe T ab, nur wenn für alle T_j, die seit dem BOT von T abgeschlossen haben, gilt:

$$\mathrm{RS}(T) \cap \mathrm{WS}(T_j) = \emptyset$$

Deadlock-Behandlung

- Deadlockvermeidung durch "Preclaiming" oder "Ordnung auf Datenobjekte" (in der Regel ist beides nicht realisierbar).
- Lastkontrolle zur Verringerung der Anzahl der Deadlocks.
- Deadlock Detection: Modelliere Wartebeziehung zwischen Transaktionen durch Graph. Deadlock bei Zyklus im Wartegraph. Verschiedene Heuristiken zum Auflösen von Deadlocks (z.B. breche jüngste Transaktion ab.
- N.B. Deadlock Detection bei optimistischen Synchronisationsverfahren nicht notwendig.

Zusammenfassung: Synchronisation

- Vermeidung von Anomalien:
  - zu ändernde Objekte werden dem Zugriff aller anderen Transaktionen entzogen
  - zu lesende Objekte werden vor Änderungen geschützt
- Standardverfahren: striktes hierarchisches Zweiphasen-Sperrprotokoll
  - mehrere Sperrgranulate
  - Deadlock-Erkennung
  - besondere Tricks zur Reduzierung von Deadlocks (z.B. RUX)
  - besondere Tricks bei Hot Spots
  - Vermeidung von Phantomen sehr aufwendig.
- Effiziente Implementierung der Sperrtabelle ist fieselig und kritisch!
- Erhöhung des Durchsatzes (der Parallelität) durch niedrigeren Isolationslevel in vielen Anwendungen in Ordnung.

- Änderungen auf Objektkopien und Nutzung mehrerer Objektversionen in Kombination mit "optimistischen" Ansätzen wird immer populärer (z.B. Oracle); aber Vorsicht:
  - Bleibt Serialisierbarkeit wirklich gewahrt?
  - Wie groß ist der Hauptspeicherverbrauch? Wie hoch ist der Aufwand, die Kopien zu erzeugen.