Beispiel-Transaktion

1. Lese den Kontostand von A in die Variable a: **read**(A,a);
2. Reduziere den Kontostand um 50,– DM: $a := a - 50$;
3. Schreibe den neuen Kontostand in die Datenbasis: **write**(A,a);
4. Lese den Kontostand von B in die Variable b: **read**(B,b);
5. Erhöhe den Kontostand um 50,– DM: $b := b + 50$;
6. Schreibe den neuen Kontostand in die Datenbasis: **write**(B,b);

Anforderungen an die Transaktionsverwaltung

- gleichzeitig (nebenläufig) ablaufende Transaktionen
- Synchronisation
- Datenbanken gegen Soft- und Hardwarefehler schützen
- Abgeschlossene Transaktionen müssen erhalten bleiben
- Nicht abgeschlossene Transaktionen müssen vollständig revidiert (zurückgesetzt) werden.

Operationen auf Transaktions-Ebene

- **begin of transaction** (**BOT**): Mit diesem Befehl wird der Beginn einer eine Transaktion darstellenden Befehlsfolge gekennzeichnet.
- **commit**:
  - Alle Änderungen der Datenbasis werden durch diesen Befehl *festgeschrieben*,
  - sie werden **dauerhaft** in die Datenbank eingebaut.
- **abort**:
  - Selbstabbruch der Transaktion
  - Datenbanksystem muß sicherstellen, daß die Datenbasis wieder in den Zustand zurückgesetzt wird, der vor Beginn der Transaktionsausführung existierte.
- **define savepoint**:
  - Sicherungspunkt definiert, auf den sich die (noch aktive) Transaktion zurücksetzen läßt.
- **backup transaction**:
  - die noch aktive Transaktion wird auf den jüngsten – also den zuletzt angelegten – Sicherungspunkt zurückgesetzt.
  - evtl. ist auch ein Rücksetzen auf weiter zurückliegende Sicherungspunkte möglich.

Abschluß einer Transaktion

1. erfolgreicher Abschluß durch ein **commit**
2. erfolgloser Abschluß durch ein **abort** oder durch Fehler

BOT

op_1

op_2

$\vdots$

op_n

commit

BOT

op_1

op_2

$\vdots$

op_j

abort

BOT

op_1

op_2

$\vdots$

op_k

$\sim\sim\sim\sim\sim\sim$ Fehler

ACID-Paradigma

Atomicity (Atomarität)

- Transaktion ist kleinste, nicht mehr weiter zerlegbare Einheit
- Entweder werden alle Änderungen der Transaktion festgeschrieben oder gar keine
- Man kann sich dies auch als „alles-oder-nichts"-Prinzip merken

Consistency

- Transaktion hinterläßt einen konsistenten Datenbasiszustand
- Anderenfalls wird sie komplett (siehe *Atomarität*) zurückgesetzt
- Zwischenzustände während der TA-Bearbeitung dürfen inkonsistent sein
- Endzustand muß die im Schema definierten Konsistenzbedingungen (z.B. referentielle Integrität) erfüllen

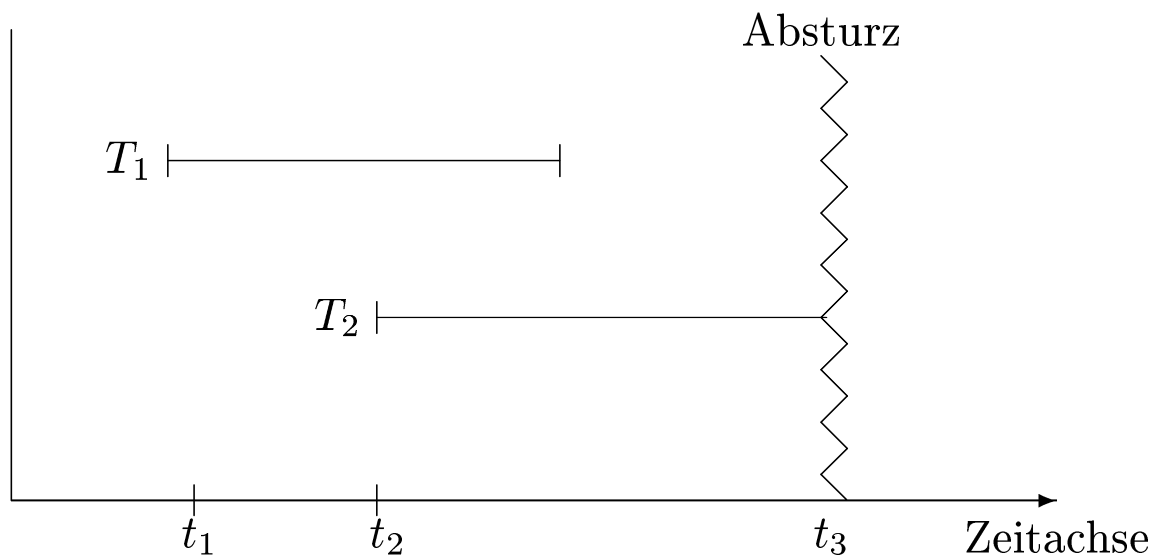
Isolation

- nebenläufig (parallel, gleichzeitig) ausgeführte Transaktionen dürfen sich nicht gegenseitig beeinflussen
- alle anderen parallel ausgeführten Transaktionen bzw. deren Effekte dürfen nicht sichtbar sein

Durability (Dauerhaftigkeit)

- Wirkung einer erfolgreich abgeschlossenen Transaktion bleibt dauerhaft in der Datenbank erhalten
- Transaktionsverwaltung muß sicherstellen, daß dies auch nach einem Systemfehler (Hardware oder Systemsoftware) gewährleistet ist
- Wirkungen einer einmal erfolgreich abgeschlossenen Transaktion kann nur durch eine sogenannte kompensierende Transaktion aufgehoben werden

Transaktionsbeginn und -ende



1. Die Wirkungen der zum Zeitpunkt t_3 abgeschlossenen Transaktion T_1 müssen in der Datenbasis vonhanden sein.

2. Die Wirkungen der zum Zeitpunkt des Systemabsturzes noch nicht abgeschlossenen Transaktion T_2 müssen vollständig aus der Datenbasis entfernt sein. Diese Transaktion kann man nur durch ein erneutes Starten durchführen.

Transaktionsverwaltung in SQL

- **commit work**:
  - Änderungen werden – falls keine Konsistenzverletzungen oder andere Probleme aufgedeckt werden – festgeschrieben.
  - Das Schlüsselwort **work** ist optional,
  - d.h. das Transaktionsende kann auch einfach mit **commit** „befohlen“ werden.
- **rollback work**:
  - Alle Änderungen sollen zurückgesetzt werden.
  - Anders als der **commit**-Befehl muß das DBMS die „erfolgreiche“ Ausführung eines **rollback**-Befehls immer garantieren können.
- **Beispiel-Transaktion**

insert into Vorlesungen
  values (5275, 'Kernphysik', 3, 2141);

insert into Professoren
  values (2141, 'Meitner', 'C4', 205);

commit work

- Man beachte: ein **commit**-Versuch nach dem ersten **insert** könnte nicht erfolgreich durchgeführt werden, da zu diesem Zeitpunkt die referentielle Integrität verletzt ist.

Zustandsübergänge einer Transaktion

- *potentiell*:
- *aktiv*:
- *wartend*:
- *abgeschlossen*:
- *persistent*:
- *gescheitert*:
- *wiederholbar*:
- *aufgegeben*:

Zustandsübergangs-Diagramm für Transaktionen

potentiell

inkarnieren

aktiv

verdrängen

einbringen

wartend

beenden

neustarten

abbrechen

wiederholbar

zurücksetzen

gescheitert

zurücksetzen

abgeschlossen

abbrechen

aufgegeben

festschreiben

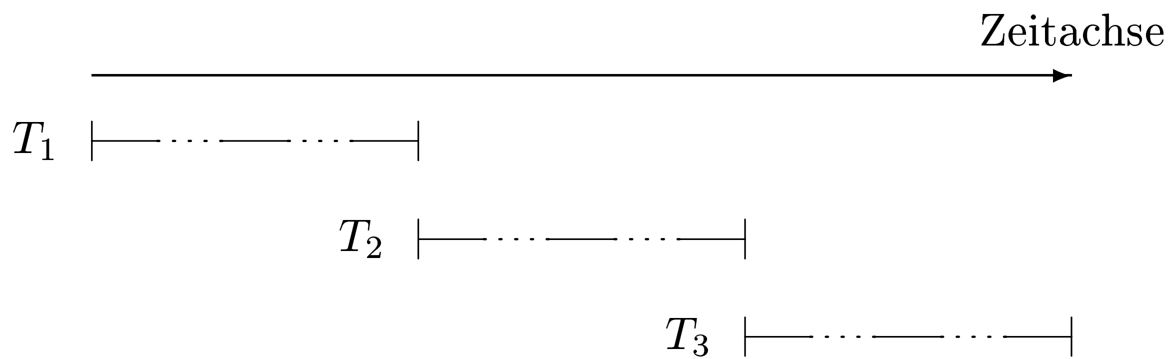
persistent

Fehlerklassifikation

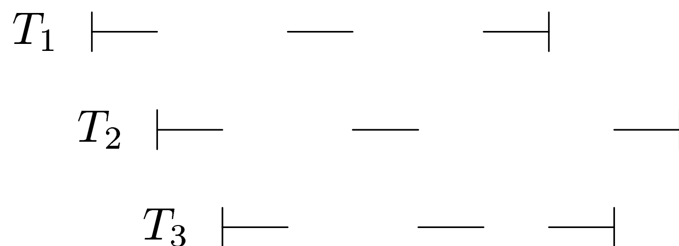
1. Lokaler Fehler in einer noch nicht festgeschriebenen (committed) Transaktion
   - Wirkung muß zurückgesetzt werden
   - $R1$-Recovery
2. Fehler mit Hauptspeicherverlust
   - abgeschlossene TAs müssen erhalten bleiben (R2-Recovery)
   - noch nicht abgeschlossene TAs müssen zurückgesetzt werden (R3-Recovery)
3. Fehler mit Hintergrundspeicherverlust
   - R4-Recovery

Ausführung der drei Transaktionen T_1, T_2 und T_3:

(a) im Einbenutzerbetrieb und



(b) im (verzahnten) Mehrbenutzerbetrieb (gestrichelte Linien repräsentieren Wartezeiten)



Fehler bei unkontrolliertem Mehrbenutzerbetrieb

Verlorengegangene Änderungen (*lost update*)

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)	

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.	$\cdots$	
9.	**abort**	

Fehler . . . (Forts.)

Phantomproblem

T_1	T_2
	select sum(KontoStand)
	from Konten
insert into Konten	
values $(C, 1000, \ldots)$	
	select sum(KontoStand)
	from Konten

Serialisierbarkeit

- Historie ist „äqivalent“ zu einer seriellen Historie
- dennoch parallele (verzahnte) Ausführung möglich

Serialisierbare Historie von T_1 und T_2

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**

Serielle Ausführung von T_1 vor T_2, also $T_1 \mid T_2$

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**

Nicht serialisierbare Historie

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**	

Zwei verzahnte Überweisungs-Transaktionen

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**	

Eine Überweisung (T_1) und eine Zinsgutschrift (T_3)

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**	

Transaktionsverwaltung in SQL92

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

- **read uncommitted**: Dies ist die schwächste Konsistenzstufe. Sie darf auch nur für **read only**-Transaktionen spezifiziert werden. Eine derartige Transaktion hat Zugriff auf noch nicht festgeschriebene Daten. Zum Beispiel ist folgender Schedule möglich:

T_1	T_2
	read(A)
	. . .
	write(A)
read(A)	
. . .	
	rollback

- **read committed**: Diese Transaktionen lesen nur festgeschriebene Werte. Allerdings können sie unterschiedliche Zustände der Datenbasis-Objekte zu sehen bekommen:

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

- **repeatable read**: Das oben aufgeführte Problem des *non repeatable read* wird durch diese Konsistenzstufe ausgeschlossen. Allerdings kann es hierbei noch zum Phantomproblem kommen. Dies kann z.B. dann passieren, wenn eine parallele Änderungstransaktion dazu führt, daß Tupel ein Selektionsprädikat erfüllen, das sie zuvor nicht erfüllten.

- **serializable**: Diese Konsistenzstufe fordert die Serialisierbarkeit. Dies ist der Default.