

Kapitel 8: Transaktionen und ihre Realisierung

Lernziele und Überblick:

- Transaktionen als Kooperationsmodell, Eigenschaften von Transaktionen
- Isolation von Transaktionen:
 - Terminologie und Formalisierung
 - Klassifikation der Probleme durch Verletzung der Isolation
 - Isolation durch Sperrprotokolle
- Diskussion: Vor- und Nachteile niedriger Isolationsgrade

Datenmodelle und Kooperationsmodelle

Datenmodelle:

- Datenbanksprachen unterscheiden sich hauptsächlich in ihrer Mächtigkeit bei der Beschreibung der Struktur und des Zustandes der Datenbanken.
- Ziel: (vgl. Kapitel 3-6): Problemangepasste Datenstrukturen und deklarative Anfragen für Nutzer der Datenbank

Kooperationsmodelle:

- Weitgehend orthogonal zur Wahl des Datenmodells ist das Kooperationsmodell.
- Ziel: Problemangepasste Mechanismen zur Synchronisation und Fehlererholung von nebenläufigen Aktivitäten auf dem gemeinsamen Datenbankzustand
- Transaktionen* sind das allgemein in Theorie und Praxis akzeptierte Kooperationsmodell.
- Andere Ansätze: Private & öffentliche Arbeitsumgebungen (*workspaces* bei CASE), *Message Passing* (nebenläufige Programmierung), Replizierte Datenbanken mit manueller Konfliktauflösung (*Groupware*, sehr gut geeignet für autonome mobile Aktoren).

Transaktionen für kooperierende Aktivitäten

- ❑ Kooperation mehrerer Geschäftsprozesse über *Seiteneffekte* auf gemeinsamen Daten
- ❑ Idee: Der gemeinsame Datenbankzustand ist ein Abbild des aktuellen Zustands der gemeinsamen Realität der nebenläufigen Geschäftsprozesse.
 - Beispiele: Buchhaltung, Lagerwirtschaft, Logistik, ...
- ❑ Das Datenbanksystem sorgt *implizit* für Synchronisation und Fehlererholung der *Zustandsänderungen*, die durch die Geschäftsprozesse ausgelöst werden.
- ❑ Der Programmierer ergreift keine expliziten Maßnahmen zur Synchronisation und Fehlererholung.

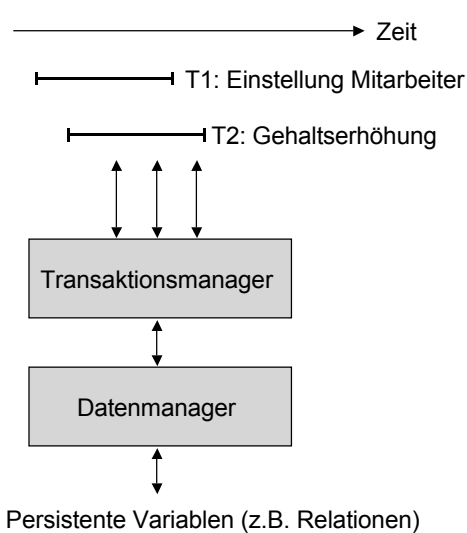
Grundsätzlich sind Transaktionen auch als Bausteine für komplexere, anwendungsbezogene Kooperationsmodelle geeignet (s. Beispiel "betriebswirtschaftliche Transaktionen" in SAP R/3).

Definition: Transaktion

Eine Transaktion ist eine Folge von "zusammengehörigen" Operationen auf dem Systemzustand, die die folgenden Kriterien erfüllt (**ACID**-Eigenschaften):

- ❑ **Atomarität**: Die Änderungen des Systemzustands sind atomar, d.h. entweder finden alle oder keine statt. Diese Änderungen umfassen Datenbankzustandsänderungen, Nachrichten und Aktionen auf externen Geräten.
- ❑ **Konsistenz**: Eine Transaktion stellt eine konsistente Transformation des Datenbankzustands dar, wenn die Gesamtheit ihrer Aktionen keine der Integritätsbedingungen, die für den Datenbankzustand definiert sind, verletzt.
- ❑ **Isolation**: Obwohl Transaktionen nebenläufig ausgeführt werden, erscheint es für jede Transaktion T , als ob jede einzelne der nebenläufigen Transaktionen entweder als ganzes vor oder als ganzes nach T ausgeführt wird.
(\Rightarrow *Serialisierbarkeit*)
- ❑ **Dauerhaftigkeit**: Wenn eine Transaktion erfolgreich abgeschlossen ist, überleben ihre Änderungen des Datenbankzustands auch alle späteren Systemausfälle.

Beispiel: Projektdatenbank



Einstellung Mitarbeiter:

```
BEGIN_WORK
SQL-Anfragen
SQL-Änderungsoperationen
END_WORK
```

Vorteile von Transaktionen:

- Programmierer können Nebenläufigkeit und Fehlersituationen weitgehend "ignorieren". Sie sehen nur die Schlüsselworte `BEGIN_WORK`, `END_WORK` und `ROLLBACK_WORK`.
- Mit `ROLLBACK_WORK` können bei Bedarf in der Datenbank ausgeführte Änderungen durch die laufende Transaktion rückgängig gemacht werden.

ACID: Atomarität

Vorteile:

- Klare Fehlersemantik in komplexen Systemen.
- Schutz des gesamten Datenbestands, der meist wesentlich wertvoller als die einzelnen Zustandsänderungen einer Transaktion ist.
- Möglichkeit zum Rücksetzen von Anwendungstransaktionen im Falle von Verklemmungen, Überlastsituationen, ...

Probleme:

- Rücksetzbarkeit von "realen" Operationen (Geld auszahlen, Rakete abschießen, ...)
- Programmgesteuerte Reaktion auf `ROLLBACK`
 - Restaurieren des Bildschirminhalts, des Zustands von Dateien, ...
 - Benachrichtigung des Benutzers, ...

ACID: Konsistenz

- ❑ In der Praxis ein rein syntaktisches Kriterium: Jede sequentielle Anweisungsfolge zwischen `BEGIN_WORK` und `COMMIT_WORK` erzeugt im Einbenutzerbetrieb aus einem konsistenten Datenbankzustand einen konsistenten Datenbankzustand.
- ❑ Alternativ: Test statischer semantischer Konsistenzbedingungen durch den Datenmanager (Schlüsselintegrität, ...)
- ❑ Außerdem: Transaktionen wahren dynamische Konsistenzbedingungen (z.B. "Gehälter fallen nie").
- ❑ Das Rücksetzen von Transaktionen führt immer zu einem konsistenten Zustand.

Probleme:

- ❑ Wer sichert die Konsistenz zwischen Datenbankzustand und Realität ("Korrektheit von Transaktionen")?

ACID: Isolation

Durch die ungeschützte parallele Ausführung der Transaktionsanweisungen können Inkonsistenzen entstehen, auch wenn die einzelnen Transaktionen konsistent sind.

Idee:

- ❑ Transaktionen enthalten keine expliziten Synchronisationsbefehle (Zugriff auf Semaphore, Mutexe, Sperren, Nachrichtenschlangen, ...)
- ❑ Es existieren keine relevanten Reihenfolgeabhängigkeiten zwischen nebenläufigen Transaktionen (s. Beispiel: Einstellung / Gehaltserhöhung), d.h. jede serielle Ausführungsreihenfolge ist "erlaubt".

Überblick: (s. folgende Folien)

- ❑ Modellbildung (Transaktionsabhängigkeiten)
- ❑ Klassifikation der Probleme aufgrund fehlender Isolation (Anomalien)
- ❑ Formalisierung des Begriffs der Isolation (Serialisierbarkeit)
- ❑ Techniken zur Isolation von Transaktionen (Sperren)

ACID: Dauerhaftigkeit und Fehlererholung

Rücksetzen der Transaktion (ROLLBACK, UNDO), falls

- ❑ Systemfehler oder Integritätsverletzung während der Transaktionsausführung oder während COMMIT_WORK
- ❑ ROLLBACK_WORK Befehl

Wiederherstellen der Transaktionseffekte (REDO), falls

- ❑ Systemfehler nach erfolgreichem COMMIT_WORK basierend auf Logdateien, Archivkopien, dem Zustand von Spiegelsystemen, ...

Betrifft eine Transaktion mehr als einen Datenmanager, so muß am Transaktionsende ein *Zwei-Phasen-Commit-Protokoll* stattfinden:

- ❑ **Phase 1:** Feststellen der Commitfähigkeit:
Der Transaktionsmanager schickt allen Datenmanagern eine PREPARE TO COMMIT Nachricht. Nur wenn alle Datenmanager mit "JA" antworten, wird die Commitfähigkeit festgestellt.
- ❑ **Phase 2:** Propagieren der getroffenen Entscheidung:
Der Transaktionsmanager schickt allen Datenmanagern eine COMMIT bzw. eine ABORT Nachricht.

Transaktionen

Transaktionen sind Folgen von Einzelaktionen, die zwischen begin und commit oder rollback stattfinden.

T1	begin	
	slock	A
	xlock	B
	read	A
	write	B
	commit	

T2	begin	
	slock	A
	read	A
	xlock	B
	write	B
	rollback	

Symbolische Repräsentation einer Transaktion:

$\langle \langle t, a_i, o_k \rangle \mid i = 1, \dots, n; k = 1, \dots, m \rangle$

Der i -te Schritt der Transaktion t führt die Einzelaktion a_i auf dem Objekt o_k durch.

Isolation von Transaktionen

Ziel:

- ❑ *Datenbankhistorien*: Beschreibung der nebenläufigen ("verzahnten") Ausführung von Transaktionen.

Hilfsbegriffe:

- ❑ *Datenbankzustand*
- ❑ *Operation* auf *Objekt* (Komponente des Datenbankzustands)
- ❑ *Abhängigkeiten* zwischen Operationen
- ❑ *Transaktion* (mit impliziter und expliziter Fehlererholung)

Parallel wird das Konzept der *Sperren* auf Objekten eingeführt.

Anschließend wird gezeigt, daß durch geeignete Sperrprotokolle (= Sperralgorithmen) nur "erwünschte" Datenbankhistorien zugelassen werden.

Einzelaktionen

Generische Aktionen:

- ❑ `begin` Legt den Anfang einer Transaktion fest.
- ❑ `commit` Beendet eine erfolgreich ausgeführte Transaktion.
- ❑ `rollback` Auf Objekten innerhalb einer Transaktion durchgeführte Manipulationen werden rückgängig gemacht, anschließend wird die Transaktion beendet.

Aktionen auf Objekten:

- ❑ `read A` Liefert den Wert des benannten Objekts *A*.
- ❑ `write B` Weist dem benannten Objekt *B* einen Wert zu und erzeugt neue Objektversion.
- ❑ `xlock B` Setzt X-Sperre (*exklusiv*) für Manipulation des Objekts *B*.
- ❑ `slock A` Setzt S-Sperre (*shared*) für Lesen des Objekts *A*.
- ❑ `unlock B` Gibt Sperre für Objekt *B* frei.

Sperrkompatibilitätsmatrix

Ziel von Sperrprotokollen:

- Erzeugung von serialisierbaren Historien
- Dazu müssen Lese- und Schreibaktionen in einer bestimmten Systematik verwendet werden (Zweiphasen-Sperrprotokoll).

Kompatibilität		Sperrmodus	
		shared	exklusiv
Sperranforderung	shared	kompatibel	Konflikt
	exklusiv	Konflikt	Konflikt

- Ist eine Sperranforderung gewährt worden, wird die Ausführung fortgesetzt.
- Ist eine Sperranforderung nicht gewährt worden, wird die Ausführung vorerst angehalten.

Notation

- Folgen S, T

$S = \langle a, b, c \rangle \quad T = \langle d, e, f, \rangle$

- Konkatenation $S \parallel T$

$S \parallel T = \langle a, b, c, d, e, f \rangle$

- i -tes Element einer Folge S

$S[3] = c$

- Teilfolge S' von S

$S' = \langle S[i] \mid \text{Prädikat}(S[i]) \rangle$

- Geordnetes Paar als zweielementige Folge

$\langle \text{Name}, \text{Wert} \rangle \quad \langle A, 3 \rangle, \langle B, 4 \rangle$

- Zustand Z eines Systems

$Z = \{ \langle \text{Name}, \text{Wert} \rangle \} \quad Z = \{ \langle A, 3 \rangle, \langle B, 4 \rangle \}$

Transaktionsvereinfachung (1)

Transaktionen können dadurch vereinfacht werden, daß die Aktionen `begin`, `commit` und `rollback` durch die Aktionen `read`, `write`, `lock` und `unlock` substituiert sind:

Transaktionsvereinfachung:

❑ Verzichte auf `begin`.

❑ Ersetze `commit` durch:

<unlock A | falls slock A oder xlock A in T vorkommt >

❑ Ersetze `rollback` durch:

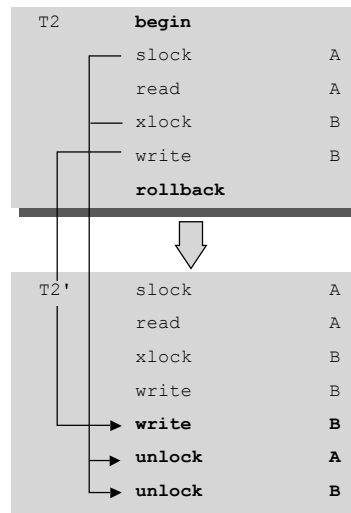
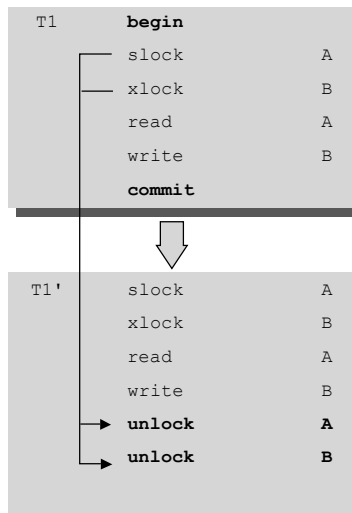
<write A | falls write A in T vorkommt > ||

<unlock A | falls slock A oder xlock A in T vorkommt >

❑ Beispiel: T1' und T2' auf der nächsten Folie

Transaktionsvereinfachung (2)

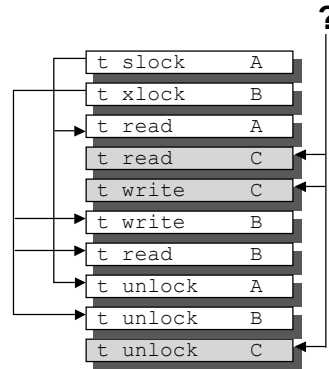
Beispiel:



Zweiphasige und wohlgeformte Transaktionen (1)

Wohlgeformte Transaktion:

- ❑ Jede lock Aktion korrespondiert in systematischer Weise mit den Lese- und Schreibaktionen der Transaktion.
 - Vor Leseaktionen erfolgt ein slock.
 - Vor Schreibaktionen erfolgt ein xlock.
- ❑ Eine Transaktion fordert eine Sperre, die sie schon besitzt, nicht erneut an.
- ❑ Sperren anderer Transaktionen müssen beachtet werden (Sperrkompatibilitätsmatrix).
- ❑ Jede read, write und unlock Aktion korrespondiert mit einer lock Aktion.
- ❑ Auf jede lock Aktion erfolgt eine korrespondierende unlock Aktion.
- ❑ Bei EOT muß eine Transaktion spätestens alle ihre Sperren zurückgeben.



Zweiphasige und wohlgeformte Transaktionen (2)

Zweiphasen-Sperrprotokoll:

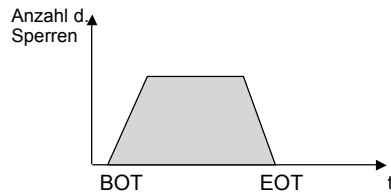
- ❑ Alle lock Aktionen gehen allen unlock Aktionen einer Transaktion voraus.
- ❑ In der *Wachstumsphase* $T[1], \dots, T[j]$ der Transaktion T werden Sperren angefordert (lock).
- ❑ In der *Schrumpfungsphase* $T[j+1], \dots, T[n]$ werden die Sperren zurückgegeben (unlock). Während der Schrumpfungsphase werden keine Sperren mehr angefordert.
- ❑ Wohlgeformte, zweiphasige Transaktionen genügen dem Zweiphasen-Sperrprotokoll.

Striktes Zweiphasen-Sperrprotokoll:

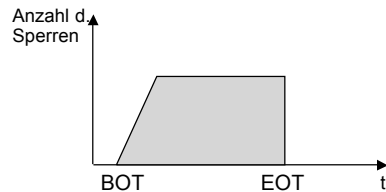
- ❑ Spezielle Zusatzanforderungen an den Verlauf der Phasen (s. nächste Folie) lösen Probleme, die über die Anforderungen der Isolation von Transaktionen hinausgehen (Dauerhaftigkeit, Verklemmung, ...).

Zweiphasige und wohlgeformte Transaktionen (3)

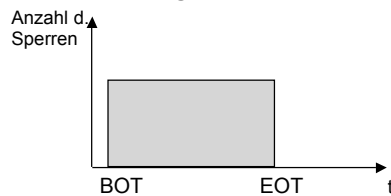
Zweiphasen-Sperrprotokoll (Dauerhaftigkeit, Verklemmung?)



Striktes Zweiphasen-Sperrprotokoll



Striktes Zweiphasen-Sperrprotokoll mit preclaiming



Historien

- ❑ **Ziel:** Modellierung der parallelen Ausführung von Transaktionen
- ❑ Folge von Aktionen bestehend aus einer Menge von Transaktionen unter Beibehaltung der Teilfolgen innerhalb der einzelnen Transaktionen
- ❑ **Notation:** $H = \langle \langle t, a_i, o_k \rangle \mid i = 1, \dots, n; k = 1, \dots, m \rangle$
Ein Schritt einer Historie entspricht einer Aktion a_i der Transaktion t auf dem Objekt o_k .
- ❑ **Serielle Historie:** Die Transaktionen werden vollständig nacheinander ausgeführt. Es entstehen keine Inkonsistenzen aufgrund von Nebenläufigkeit.
- ❑ **Serialisierbare Historie:** Eine Historie, die äquivalent zu einer seriellen Historie ist. (Äquivalenz noch zu definieren!)
- ❑ **Sperr-Bedingungen** schränken die Menge der erlaubten Historien ein. Z.B. kann eine Exklusivsperrung auf ein Objekt für eine Aktion nur dann genehmigt werden, wenn für dieses Objekt nicht schon eine Sperrung im Rahmen einer anderen Transaktion besteht (s. Kompatibilität zwischen *Anforderungsmodus* und *Sperrmodus* auf Folie Sperrkompatibilitätsmatrix).
- ❑ **Legale Historie:** Sperren werden berücksichtigt (s. Sperrkompatibilitätsmatrix).

Ausführungshistorien, Beispiele (1)

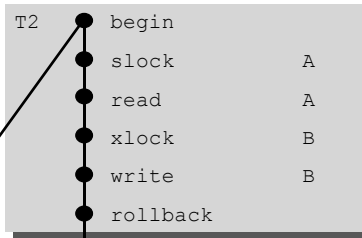
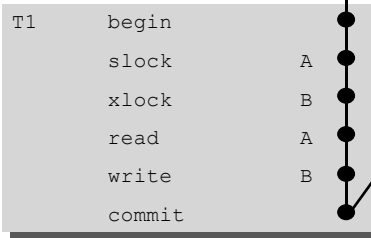
Legal und seriell:

T1 slock A
 T1 xlock B
 T1 read A
 T1 write B
 T1 unlock A
 T1 unlock B

Historie

T2 slock A
 T2 read A
 T2 xlock B
 T2 write B
 T2 write B
 T2 unlock A
 T2 unlock B

Transaktionen



Ausführungshistorien, Beispiele (2)

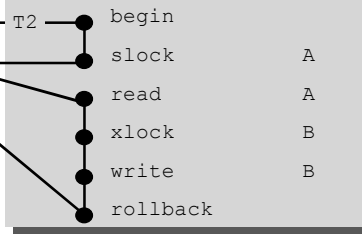
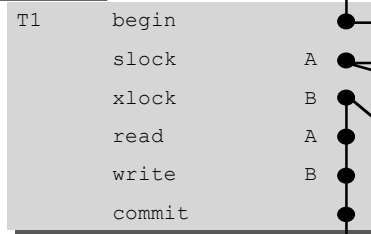
Legal und nicht seriell:

T2 slock A
 T1 slock A
 T2 read A
 T2 xlock B
 T2 write B
 T2 write B

Historie

T2 unlock A
 T2 unlock B
 T1 xlock B
 T1 read A
 T1 write B
 T1 unlock A
 T1 unlock B

Transaktionen



Ausführungshistorien, Beispiele (3)

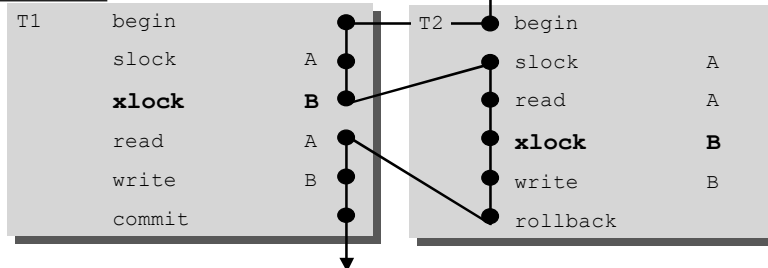
Nicht legal und nicht seriell:

T1	slock	A
T1	xlock	B
T2	slock	A
T2	read	A
T2	xlock	B
T2	write	B

Historie

T2	write	B
T2	unlock	A
T2	unlock	B
T1	read	A
T1	write	B
T1	unlock	A
T1	unlock	B

Transaktionen



Einführung in Datenbanksysteme

Transaktionen 8.23

Abhängigkeiten zw. Transaktionen in Historien

- ❑ **Ziel:** Definition des Datenflusses zwischen Transaktionen und Vergleich zweier Historien miteinander
- ❑ Eine Transaktion *T2* hängt von einer Transaktion *T1* innerhalb einer Historie *H* ab, falls
 - *T1* Objekte beschrieben hat, die anschließend von *T2* gelesen oder beschrieben werden oder
 - *T1* Objekte gelesen hat, die anschließend von *T2* beschrieben werden.
- ❑ **Äquivalente Historie:** Zwei Historien besitzen die selben Abhängigkeiten, d.h. ihre Transaktionen greifen lesend und schreibend in beiden Historien in der gleichen Reihenfolge auf die gleichen Objekte zu.
- ❑ **Abhängigkeitsgraphen** beschreiben die Abhängigkeiten zwischen Transaktionen in Historien (s. nächste Folie).

Einführung in Datenbanksysteme

Transaktionen 8.24

Abhängigkeitsgraphen: Abhängigkeit

Eventuell kritische Elementarsituationen:

A (i): Objekt A mit Versionsnummer i

Ausführungssequenzen:

READ → WRITE

T1 read A (1)
T2 write A (2)

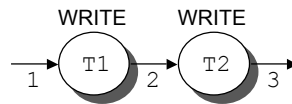
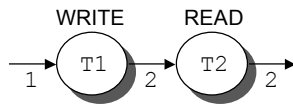
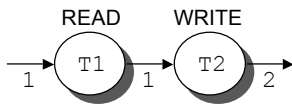
WRITE → READ

T1 write A (2)
T2 read A (2)

WRITE → WRITE

T1 write A (2)
T2 write A (3)

Abhängigkeitsgraphen:



Liegt einer dieser drei Fälle vor, so „hängt T2 von T1 ab“ oder formal $\langle T1, (A(i)), T2 \rangle \in DEP (H)$, bzw. $T1 \lll T2$.
Leseabhängigkeiten (READ → READ) werden nicht berücksichtigt.

Abhängigkeiten bei verzahnter Transaktionsausführung

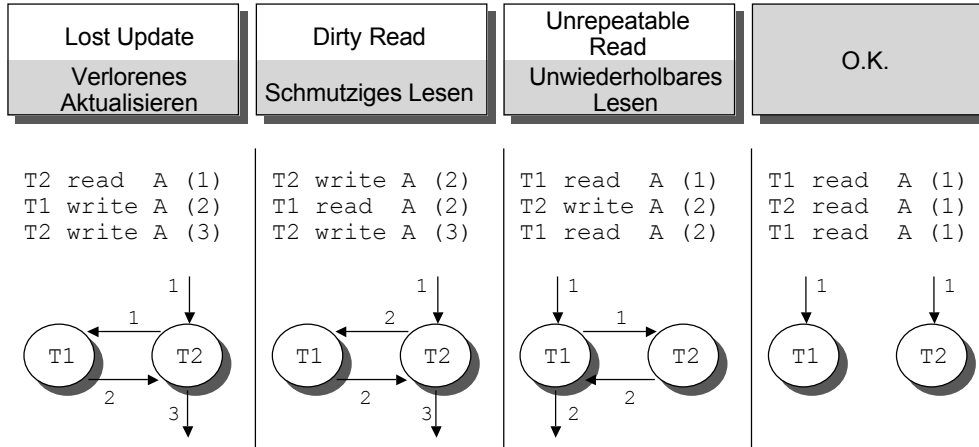
□ Beispiel:

- Zwei Transaktionen greifen in einer Historie abwechselnd auf dasselbe Objekt zu.
- Betrachtet werden drei Zugriffe auf dieses Objekt, die in der Historie aufeinander folgen.
- Der erste und letzte Zugriff erfolgt durch T1, der zweite Zugriff durch T2.

1. Zugriff	2. Zugriff	3. Zugriff			
T1	T2	T1	Abhängigkeiten	Zyklus	Anomalie
Read	Read	Read	DEP = \emptyset	nein	
Read	Read	Write	DEP = { $\langle T2, A(1), T1 \rangle$ }	nein	
Read	Write	Read	DEP = { $\langle T1, A(1), T2 \rangle, \langle T2, A(2), T1 \rangle$ }	ja	unrepeatable read
Read	Write	Write	DEP = { $\langle T1, A(1), T2 \rangle, \langle T2, A(2), T1 \rangle$ }	ja	lost update (2. Zugriff)
Write	Read	Read	DEP = { $\langle T1, A(1), T2 \rangle$ }	nein	
Write	Read	Write	DEP = { $\langle T1, A(1), T2 \rangle, \langle T2, A(1), T1 \rangle$ }	ja	dirty read
Write	Write	Read	DEP = { $\langle T1, A(1), T2 \rangle, \langle T2, A(2), T1 \rangle$ }	ja	lost update (1. Zugriff)
Write	Write	Write	DEP = { $\langle T1, A(1), T2 \rangle, \langle T2, A(2), T1 \rangle$ }	ja	lost update (2. Zugriff)

Abhängigkeitsgraphen: Zyklen

Anormale T1/T2-Abhängigkeiten (Zyklen durch Kombination kritischer Elementarsituationen)



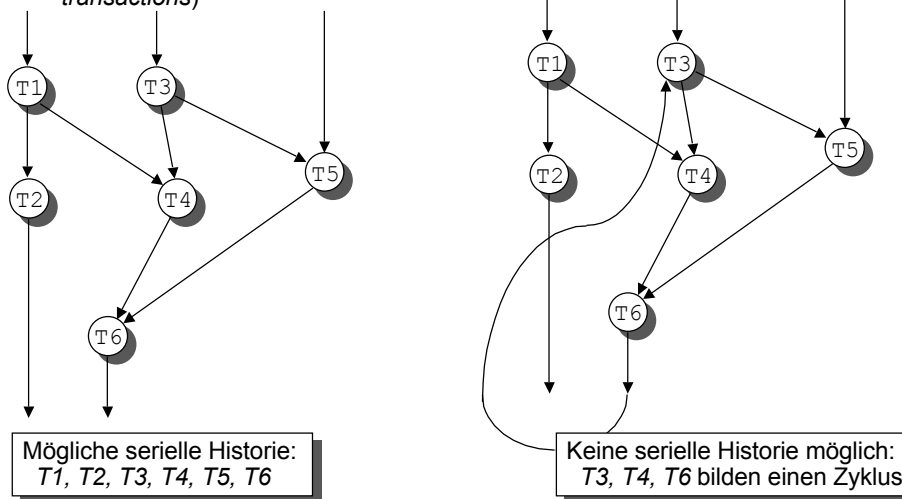
Daten gelten als *schmutzige Daten*, falls auf sie bereits schreibend zugegriffen wurde, die schreibende Transaktion aber noch nicht beendet ist.

Einführung in Datenbanksysteme

Transaktionen 8.27

Abhängigkeitszyklen (1)

Kritisch: Transaktionen, die zu Abhängigkeitszyklen beitragen (engl. *wormhole transactions*)



Einführung in Datenbanksysteme

Transaktionen 8.28

Abhängigkeitszyklen (2)

Für die linke Seite (s. vorige Folie) gilt:

- ❑ $VOR(T4) = \{T1, T3\}$ und $NACH(T4) = \{T6\}$. $T2$ und $T5$ sind deshalb weder vor noch nach $T4$ und können deshalb nebenläufig zu $T4$ ausgeführt werden. $T2$, $T4$ und $T5$ können in beliebiger Reihenfolge und beliebig überlappend ausgeführt werden.
- ❑ Die Relation \lll hat keine Zyklen und definiert eine partielle Ordnung der Transaktionen.

Für die rechte Seite (s. vorige Folie) gilt:

- ❑ $VOR(T4) = \{T1, T3, T4, T5, T6\}$
- ❑ Die Relation \lll hat Zyklen \square keine Serialisierbarkeit

Abhängigkeitszyklen (3)

Folgende Aussagen sind äquivalent (ohne Beweis):

- ❑ Wenn für zwei Transaktionen $T1$ und $T2$ gilt, daß Teile von $T1$ vor $T2$ und Teile von $T2$ vor $T1$ ausgeführt werden, dann liegt eine Nebenläufigkeitsanomalie vor, anderenfalls gibt es keine Nebenläufigkeitsanomalie.
- ❑ Hat der Abhängigkeitsgraph Zyklen, gibt es keine äquivalente serialisierbare Historie; ist er zyklensfrei, so existiert mindestens eine serialisierbare Historie. (Wormhole-Theorem, **Graphsicht** für Beweis)
- ❑ Eine serialisierbare Historie entsteht dann, wenn alle nebenläufigen Transaktionen wohlgeformt sind und dem Zweiphasen-Sperrprotokoll genügen (Locking-Theorem, **Protokollsicht** für Implementierung).

Locking Theorem:

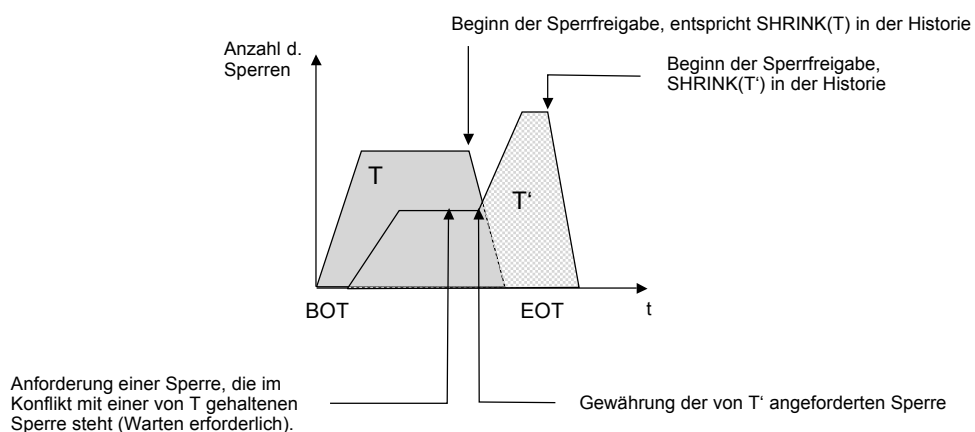
- ❑ Wenn alle Transaktionen wohlgeformt und zweiphasig sind, dann ist jede legale Historie serialisierbar.

Details: J. Gray, A. Reuter : Transaction Processing - Concepts and Techniques, The Morgan Kaufmann Series in Data Management Systems, Morgan Kaufman, 1993.

Locking Theorem: Shrink-Lemma

- Sei $SHRINK(T) = \min(i \mid H[i] = \langle T, UNLOCK, o \rangle \text{ für ein Objekt } o)$
 $SHRINK(T)$ bezeichnet also den Index i der ersten UNLOCK-Aktion einer Transaktion T innerhalb einer Historie H .
- T und T' seien wohlgeformt und zweiphasig
- Lemma: Wenn $T \ll T'$, dann gilt $SHRINK(T) < SHRINK(T')$
 Beweis des Lemmas:
 - (1) $T \ll T'$ bedeutet, daß beide Transaktionen im Laufe der Historie H mindestens einmal eine Aktion auf demselben Objekt ausführen, wobei mindestens einer dieser Zugriffe schreibend ist.
 (z.B. $H[i] = \langle T, WRITE, o \rangle$, $H[j] = \langle T', READ, o \rangle$, mit $i < j$)
 - (2) Da beide Transaktionen wohlgeformt sind, wird jeder Zugriff durch entsprechende Sperren gesichert, wodurch laut (1) zumindest bei einem Objekt ein Sperrkonflikt auftritt.
 - (3) T und T' sind außerdem zweiphasig. Alle Sperraktionen der jeweiligen Transaktion finden statt, bevor die Transaktion die erste ihrer Sperren wieder freigibt.
 - (4) Da die Historie H legal sein soll, muß ein Zeitpunkt bzw. Index k_1 existieren, zu dem T die Sperre für das Objekt freigibt, der vor dem Zeitpunkt k_2 liegt, zu dem T' eine Sperre für dasselbe Objekt bekommen kann, also im Beispiel:
 $H[i] = \langle T, WRITE, o \rangle$, $H[k_1] = \langle T, UNLOCK, o \rangle$, $H[k_2] = \langle T', SLOCK, o \rangle$, $H[j] = \langle T', READ, o \rangle$
 mit $i < k_1 < k_2 < j$.
 - (5) Es gilt $k_2 < SHRINK(T')$, und aus (3) folgt $SHRINK(T) \leq k_1$. Zusammen mit der Feststellung $k_1 < k_2$ aus (4) ergibt sich $SHRINK(T) \leq k_1 < k_2 < SHRINK(T')$ und damit $SHRINK(T) < SHRINK(T')$.

Locking Theorem: Veranschaulichung – Shrink-Lemma



Locking Theorem: Beweisskizze

□ Gegeben:

- Wormhole-Theorem (ohne Beweis): Eine Historie über beliebigen ("nicht protokollierten") Transaktionen ist genau dann serialisierbar, wenn sie keine „Wormhole“-Transaktionen enthält (keine Zyklen im Abhängigkeitsgraphen □ Graphsicht).
- Lemma der vorigen Folien: Wenn zwei wohlgeformte, zweiphasige Transaktionen die Relation $T \lll T'$ erfüllen, dann gilt $SHRINK(T) < SHRINK(T')$

□ Widerspruchsbeweis. Zu widerlegende Behauptung: Wenn alle Transaktionen *wohlgeformt und zweiphasig* sind, dann existiert eine legale Historie H, die nicht serialisierbar ist (Protokollsicht).

- Wenn H nicht serialisierbar wäre, müßte der zugehörige Abhängigkeitsgraph Zyklen enthalten, z.B. $T_1 \lll T_2 \lll \dots \lll T_n \lll T_1$
- Aus dem Shrink-Lemma würde dann folgen $SHRINK(T_1) < SHRINK(T_2) < \dots < SHRINK(T_n) < SHRINK(T_1)$, also die Aussage $SHRINK(T_1) < SHRINK(T_1)$, die einen Widerspruch darstellt.

Grade von Sperrprotokollen

Sperrprotokolle haben die folgenden Eigenschaften:

- **Grad 0:** Wohlgeformt in bezug auf Schreiboperationen (*anarchy, chaos*)
- **Grad 1:** Zweiphasig in bezug auf WRITE-Sperren und wohlgeformt in bezug auf Schreiboperationen (*browse*)
- **Grad 2:** Zweiphasig in bezug auf WRITE-Sperren und wohlgeformt (*cursor stability*)
- **Grad 3:** Zweiphasig und wohlgeformt (*isolated, serializable, repeatable reads*)

Die Isolationsgrade berücksichtigen die Abhängigkeiten:

1° □ WRITE □ WRITE

2° □ WRITE □ WRITE, WRITE □ READ

3° □ WRITE □ WRITE, WRITE □ READ, READ □ WRITE

Frage: Auswirkungen der Isolationsgrade auf die möglichen Abhängigkeitsgraphen ?

		Wohlgeformt 2°, 3°	0°, 1°

Grade der Isolation (1)

Auswirkungen des Sperrverhaltens des jeweiligen Isolationsgrads:

- Grad 0:** Eine 0° Transaktion überschreibt keine schmutzigen Daten einer anderen Transaktion, wenn diese mindestens Grad 1 hat.
 - Grad 1:** Änderungen von 1°-Transaktionen gehen für die Dauer der Transaktion nicht verloren.
 - Grad 2:** Änderungen von 2°-Transaktionen gehen für die Dauer der Transaktion nicht verloren und das Lesen schmutziger Daten anderer Transaktionen wird vermieden.
 - Grad 3:** Änderungen von 3°-Transaktionen gehen für die Dauer der Transaktion nicht verloren und alle Leseoperationen in 3°-Transaktionen sind wiederholbar.
-
- Isolation wird von den meisten Systemen aus Performanzgründen nicht in vollem Umfang geboten. Es werden nur WRITE \square WRITE und WRITE \square READ Abhängigkeiten berücksichtigt. READ \square WRITE Abhängigkeiten werden meist ignoriert.
 - Der ANSI SQL-Standard schreibt volle Isolation vor (Grad 3).

Grade der Isolation (2)

Isolationstheorem:

Erfüllt eine Transaktion das 0°, 1°, 2° oder 3° Sperrprotokoll, so erreicht sie Isolationsgrad 1°, 2° oder 3° in jeder legalen Historie, wenn alle anderen Transaktionen mindestens vom Grad 1° sind.

- In der Praxis bedeutet dies: Der Grad der eigenen Transaktion kann unabhängig von anderen Transaktionen gewählt werden.
- Ausnahme:** Daten, welche von 1° Transaktionen aus der Datenbank gelesen werden, sind möglicherweise schmutzig. Werden diese Daten zum Aktualisieren der Datenbank verwendet, können andere Transaktionen inkonsistente Daten bekommen. Das Theorem geht deshalb davon aus, daß 1° Transaktionen "wissen, was sie tun".
- Konklusion:** Auch wenn eine Transaktion schmutzig Daten liest, nimmt man an, daß sie die Datenbank in einen konsistenten Zustand transformiert. Viele Systeme lassen deshalb 1° Isolationsverhalten nur für Lese-Transaktionen zu.

Grade der Isolation in SQL

- ❑ Transaktionen mit niedrigen Isolationsgraden benötigen weniger Sperren, halten diese für eine kürzere Zeit und sorgen somit für bessere Performanz.
- ❑ 2° Isolation erlaubt es Transaktionen, die gesamte Datenbank abzusuchen und dabei wenig Sperren zu halten und somit andere Transaktionen nicht zu stören.

```
select count(*)
from mitarbeiter
where gehalt > 10000;
```



3°: Alle Datensätze aus `mitarbeiter` bzw. die gesamte Tabelle werden bis Transaktionsende gesperrt.



2°: Jeder Datensatz wird nach dem Lesen vom System automatisch freigegeben.

- ❑ Volle Isolation kann nur bei Grad 3 erreicht werden.

- ❑ SQL stellt die folgenden Optionen bereit:
**set transaction
 isolation level**

read uncommitted	Grad 1
read committed	Grad 2
repeatable read	Grad 2,999 (Phantome)
serializable	Grad 3

Cursor stability

- ❑ In SQL wird üblicherweise eine Isolation implementiert, die mehr Konsistenz als Grad 2 bietet (*cursor stability*).

Cursor stability vermeidet, dass Änderungen durch andere Transaktionen verloren gehen:

```
exec sql declare cursor c for
select gehalt
from mitarbeiter
where no = :nr;

exec sql open c;
exec sql fetch c into :gehalt;
gehalt = gehalt * 1,5;
exec sql update mitarbeiter
set gehalt = :gehalt;
where no = :nr;

exec sql close c;
... /* weitere SQL-Befehle */
exec sql commit;
```

Lesesperre, solange Cursor offen

Überschreiben fremder Änderungen bei 2° Isolation:

```
exec sql select gehalt
into :gehalt
from mitarbeiter
where no = :nr;

/* Datensatz ist nicht gesperrt und kann
von nebenläufigen Transaktionen geändert
werden */
gehalt = gehalt * 1,5;
/* update überschreibt zwischenzeitliche
Änderungen anderer (beendeter)
Transaktionen */
exec sql update mitarbeiter
set gehalt = :gehalt
where no = :nr;

exec sql commit;
```

Searched Update is OK 1° oder 2°:

```
exec sql update mitarbeiter
set gehalt = gehalt * 1,5
where no = :nr;
```

Zusammenfassung: Niedrige Isolationsgrade

Vorteile:

- Daten sind nur kurze Zeit gesperrt.
- Weniger Daten werden gesperrt.
- Potentiell höhere Nebenläufigkeit

Nachteile:

- Möglicherweise inkonsistente Eingabe- und Ausgabedaten
- Eine ROLLBACK Operation muß für ihre Aktionen Sperren setzen.
- 0° Transaktionen unterstützen kein ROLLBACK.
- Für reine Lesetransaktionen gilt 0° = 1°.

Die Eigenschaften niedriger Isolationsgrade sind in den folgenden Tabellen nochmals gegenübergestellt.

Einführung in Datenbanksysteme

Transaktionen 8.39

Überblick über die Isolationseigenschaften (1)

	Grad 0	Grad 1
Name	<i>Chaos</i>	<i>Browse</i>
Schutz	Überschreibt keine Änderungen von laufenden Transaktionen höheren Grades	wie 0° und keine verlorengangenen Änderungen möglich
Commit Zeitpunkt	Schreiboperation sofort sichtbar	Schreiboperation am Ende der Transaktion sichtbar
Schmutzige Daten	Schmutzige Daten anderer Transakt. höheren Gr. werden nicht übersch.	Eigene schmutzige Daten werden nicht von anderen T. übersch.
Sperrprotokoll	Kurze exklusive Sperren beim Schreiben, keine beim Lesen	Lange exklusive Sperren beim Schreiben, keine beim Lesen
Transaktionsstruktur	Wohlgeformt, bzgl. Schreibzugr.	Wohlgeformt und zweiphasig bzgl. Schreibzugr.
Nebenläufigkeit	Größtmöglich: kurze Schreibsperren	Groß: nur Schreibsperren
Aufwand	Kleinstmöglich: kurze Schreibsperren	Klein: nur Schreibsperren
Rücksetzen	Nicht möglich	Nicht beendete Transaktionen
Beachtete Abhängigkeiten	Keine	WRITE <input type="checkbox"/> WRITE

Einführung in Datenbanksysteme

Transaktionen 8.40

Überblick über die Isolationseigenschaften (2)

	Grad 2	Grad 3
Name	<i>Cursor Stability</i>	<i>(Serialisierbar) Repeatable Read</i>
Schutz	Keine verlorengel. Änderungen Keine schmutzigen Leseoperationen	Keine verlorengel. Änderungen, keine schmu. Leseoper., wiederh. Leseoper.
Commit Zeitpunkt	Schreiboperation am Ende der Transaktion sichtbar	Schreiboperation am Ende der Transaktion sichtbar
Schmutzige Daten	wie 0°, 1° und es werden keine schmutzigen Daten anderer T. gel.	wie 0°, 1°, 2° und andere beschreiben keine in der 3°-Transaktion gel. Daten
Sperrprotokoll	1° und kurze Sperren beim Lesen	1° und lange Lesesperren
Transaktionsstruktur	Wohlgeformt für Lese- und Schreibz. und zweiphasig für Schreibzugr.	Wohlgeformt und zweiphasig für Lese- und Schreibzugriffe
Nebenläufigkeit	Mittel: wenige Lesesperren	Klein: alle betroffenen Daten werden gesperrt
Aufwand	Mittel: Schreib- und Lesesperren, Lesesperren werden nicht gehalten	Mittel: hält beide Arten von Sperren
Rücksetzen	Nicht beendete Transaktionen	Nicht beendete Transaktionen
Beachtete Abhängigkeiten	WRITE \square WRITE, WRITE \square READ	WRITE \square WRITE, WRITE \square READ READ \square WRITE

Einführung in Datenbanksysteme

Transaktionen 8.41

Beispiel: Aktienkauf

Gegeben: Ein Bankkonto mit einem Kontostand von 1000 €.

Es soll ein Aktienkauf verbucht werden.

- Die Kauftransaktion beinhaltet die Bezahlung der Aktien selbst,
- die Zahlung einer Bearbeitungsgebühr.
- Aktienkaufpreis und Bearbeitungsgebühr erscheinen als zwei getrennte Buchungen.
- Die Gesamttransaktion gilt nur dann als erfolgreich, wenn beide Buchungen ausgeführt wurden.

Beispiel (100 € Kaufpreis, 10 € Gebühr): persistenter Kontostand in DB:

1. READ Kontostand	1000 €
2. Kontostand := Kontostand – Kaufpreis	1000 €
3. WRITE Kontostand	900 €
4. READ Kontostand	900 €
5. Kontostand := Kontostand – Bearbeitungsgebühr	900 €
6. WRITE Kontostand	890 €

Einführung in Datenbanksysteme

Transaktionen 8.42

Beispiel: Aktienkauf

Annahme für die nächsten Beispiele: Zwei Transaktionen, die dasselbe Konto betreffen, sollen nebenläufig ausgeführt werden:

T1 (100 € Kaufpreis, 10 € Gebühr),
T2 (200 € Kaufpreis, 20 € Gebühr)

Hinweis: Konsistente Endzustände sind im folgenden grün, inkonsistente Endzustände rot hinterlegt.

Serielle Historien

T1	persistenter Kontostand	T2
READ K	1000 €	
K := K - 100	1000 €	
WRITE K	900 €	
READ K	900 €	
K := K - 10	900 €	
WRITE K	890 €	
	890 €	READ K
	890 €	K := K - 200
	690 €	WRITE K
	690 €	READ K
	690 €	K := K - 20
	670 €	WRITE K

T1	persistenter Kontostand	T2
	1000 €	READ K
	1000 €	K := K - 200
	800 €	WRITE K
	800 €	READ K
	800 €	K := K - 20
	780 €	WRITE K
READ K	780 €	
K := K - 100	780 €	
WRITE K	680 €	
READ K	680 €	
K := K - 10	680 €	
WRITE K	670 €	

Einführung in Datenbanksysteme

Transaktionen 8.43

Lost Update

Lost Update bei Isolationsgrad 0

T1	persistenter Kontostand	T2
READ K	1000 €	
K := K - 100	1000 €	
	1000 €	READ K
	1000 €	K := K - 200
WRITE K	900 €	
	800 €	WRITE K
	800 €	READ K
	800 €	K := K - 20
	780 €	WRITE K
READ K	780 €	
K := K - 10	780 €	
WRITE K	770 €	

T2 kann erst mit dem Schreiben beginnen, nachdem T1 die Schreibsperre freigegeben hat.

Isolationsgrad 1: Für Schreiboperationen werden Sperren angefordert und bis zum Transaktionsende gehalten.

T1	persistenter Kontostand	T2
READ K	1000 €	
K := K - 100	1000 €	
	1000 €	READ K
	1000 €	K := K - 200
XLOCK K	1000 €	
WRITE K	900 €	
		Sperranforderung (T2 wartet)
READ K	900 €	
K := K - 10	900 €	
WRITE K	890 €	
UNLOCK K	890 €	
	890 €	XLOCK K
	800 €	WRITE K
	800 €	READ K
	800 €	K := K - 20
	780 €	WRITE K
	780 €	UNLOCK K

Obwohl im Beispiel die Änderungen von T1 durch T2 überschrieben werden, gilt der DB-Zustand als konsistent im Sinne von 1°-Isolation, da beide T in einem konsistenten Zustand starten und am jeweiligen Transaktionsende einen im Vergleich zum ihrem Startzustand konsistenten Zustand hinterlassen.

Einführung in Datenbanksysteme

Transaktionen 8.44

Dirty Read

Dirty Read (hier bei Isolationsgrad 1)

T1	persistenter Kontostand	T2
READ K	1000 €	
K := K - 100	1000 €	
XLOCK K	1000 €	
WRITE K	900 €	
	900 €	READ K ←
	900 €	K := K - 200
WRITE K (ROLLBACK)	1000 €	
UNLOCK K	1000 €	
	1000 €	XLOCK K
	700 €	WRITE K
	700 €	UNLOCK K

Da T1 mit ROLLBACK abgebrochen wurde, hat T2 einen inkonsistenten Zustand gelesen und dadurch einen inkonsistenten Endzustand erzeugt.

T2 darf von T1 veränderte Daten erst nach Freigabe der Schreibsperre selber sperren und lesen.

Isolationsgrad 2: Zusätzlich zu 1° werden für Leseoperationen Sperren angefordert.

T1	persistenter Kontostand	T2
SLOCK K	1000 €	
READ K	1000 €	
UNLOCK K	1000 €	
K := K - 100	1000 €	
XLOCK K	1000 €	
WRITE K	900 €	
		Sperranforderung (T2 wartet)
WRITE K (ROLLBACK)	1000 €	
UNLOCK K	1000 €	
	1000 €	SLOCK K
	1000 €	READ K
	1000 €	K := K - 200
	1000 €	XLOCK K
	800 €	WRITE K
	800 €	UNLOCK K

Einführung in Datenbanksysteme
Transaktionen 8.45

Nicht wiederholbares Lesen

Beispiel von Folie 44 als 2°-Transaktionen

T1	persistenter Kontostand	T2
SLOCK K	1000 €	
READ K	1000 €	
UNLOCK K	1000 €	
K := K - 100	1000 €	
	1000 €	SLOCK K
	1000 €	READ K
	1000 €	UNLOCK K
	1000 €	K := K - 200
XLOCK K	1000 €	
WRITE K	900 €	
		X-Sperranforderung (T2 wartet)
READ K	900 €	
K := K - 10	900 €	
WRITE K	890 €	
UNLOCK K	890 €	
	890 €	XLOCK K ←
	800 €	WRITE K
	800 €	READK
	800 €	K := K - 20
	780 €	WRITE K
	780 €	UNLOCK K

T1 wurde erfolgreich beendet, T2 hat einen konsistenten, aber veralteten Zustand gelesen. Ein erneutes READ von T2 würde einen anderen Kontostand liefern.

Einführung in Datenbanksysteme
Transaktionen 8.46

Isolationsgrad 3

Beispiel von Folie 44 als 3°-Transaktionen

T1	persistenter Kontostand	T2
SLOCK K	1000 €	
READ K	1000 €	
K := K - 100	1000 €	
	1000 €	SLOCK K
	1000 €	READ K
	1000 €	K := K - 200
X-Sperranforderung (T1 wartet)		X-Sperranforderung (T1 wartet)

← Zweite Lesesperre wird gewährt.

Häufiges Problem bei 3° - Verklemmung/Deadlock:

Beide Transaktionen halten bis zum Transaktionsende eine Lesesperre auf dasselbe Objekt und benötigen als nächste Operation eine Schreibsperre. Die Schreibsperre wird nur gewährt, wenn eine der beiden Transaktionen ihre Lesesperre wieder freigibt.

Die Lesevorgänge innerhalb der Transaktionen sind wiederholbar, da auf zum Lesen gesperrte Objekte keine Schreibsperren durch fremde Transaktionen erlaubt werden.

Mögliche 3°-Historie

T1	persistenter Kontostand	T2
SLOCK K	1000 €	
READ K	1000 €	
K := K - 100	1000 €	
XLOCK K	1000 €	
		Sperranforderung (T2 wartet)
WRITE K	900 €	
READ K	900 €	
K := K - 10	900 €	
WRITE K	890 €	
UNLOCK K	890 €	
	890 €	SLOCK K
	890 €	K := K - 200
	890 €	XLOCK K
	690 €	WRITE K
	690 €	READ K
	690 €	K := K - 20
	670 €	WRITE K
	670 €	UNLOCK K

← Sperre wird gewährt.

Lesesperre für T2 wird nicht gewährt, solange T1 eine Schreibsperre hält. T2 muss bis zum Ende von T1 warten.