7. Transaktionen

Einordnung in den Vorlesungsverlauf

- ER-Modell
- Relationenmodell
- relationale Anfragesprachen
- SQL
- Entwurfstheorie
- **Transaktionen**

7. Transaktionen

Überblick

- Transaktionsbegriff
- Probleme im Mehrbenutzerbetrieb
- Serialisierbarkeit
- Sperrprotokolle zur Synchronisation
- Transaktionen in SQL-DBMS

Beispiele

Beispielszenarien

- Platzreservierung für Flüge quasi gleichzeitig aus vielen Reisebüros
  $\rightsquigarrow$ Platz könnte mehrfach verkauft werden, wenn mehrere Reisebüros den Platz als verfügbar identifizieren
- überschneidende Kontooperationen einer Bank
- statistische Datenbankoperationen (...länger andauernde Berechnungen)
  $\rightsquigarrow$ Ergebnisse sind verfälscht, wenn während der Berechnung Daten geändert werden

7.1. Transaktionsbegriff

Eine *Transaktion* ist eine Folge von Operationen (Aktionen), die die Datenbank von einem konsistenten Zustand in einen konsistenten, eventuell veränderten, Zustand überführt, wobei die ACID-Eigenschaften eingehalten werden müssen.

Aspekte:

- semantische Integrität: korrekter (konsistenter) DB-Zustand nach Ende der Transaktion
- Ablaufintegrität: Fehler durch „gleichzeitigen“ Zugriff mehrerer Benutzer auf dieselben Daten vermeiden

ACID-Eigenschaften

A **Atomicity** (Atomarität):
Transaktion wird entweder ganz oder gar nicht ausgeführt.

C **Consistency** (Konsistenz oder auch Integritätserhaltung):
Datenbank ist vor Beginn und nach Beendigung einer Transaktion jeweils in einem konsistenten Zustand.

I **Isolation** (Isolation):
Nutzer*in, der/die mit einer Datenbank arbeitet, sollte den Eindruck haben, dass er/sie mit dieser Datenbank alleine arbeitet.

D **Durability** (Dauerhaftigkeit / Persistenz):
nach erfolgreichem Abschluss einer Transaktion muss das Ergebnis dieser Transaktion „dauerhaft“ in der Datenbank gespeichert werden.

Transaktion i

Kommandos zur Transaktionssteuerung

- Beginn einer Transaktion:
  Begin-of-Transaction–Kommando *BOT* (in SQL implizit!)
- *commit*:
  die Transaktion soll erfolgreich beendet werden
- *abort*:
  die Transaktion soll abgebrochen werden

Transaktion ii

Integritätsverletzung

- Beispiel:
  - Übertragung eines Betrages *B* von einem Konto *K1* auf ein anderes Konto *K2*
  - Bedingung: Summe der Kontostände bleibt konstant
- vereinfachte Notation:
  $Transfer = < K1{:=}K1\text{-}B;\ K2{:=}K2{+}B >;$
- Realisierung in SQL:
  als Sequenz zweier elementarer Änderungen

  $\leadsto$ Bedingung ist zwischen den einzelnen Änderungen nicht unbedingt erfüllt!

Transaktion iii

Vereinfachtes Modell für Transaktion

Repräsentation von Datenbankänderungen einer Transaktion

- $read(A,x)$: weise den Wert des DB-Objektes A der Variablen x zu
- $write(x, A)$: speichere den Wert der Variablen x im DB-Objekt A

Beispiel:

T_1: $read(A, x)$; $x := x - 200$; $write(x, A)$;
T_2: $read(B, y)$; $y := y + 100$; $write(y, B)$;

7.2. Probleme im Mehrbenutzerbetrieb i

Übersicht:

- inkonsistentes Lesen: Nonrepeatable Read
- Abhängigkeiten von nicht freigegebenen Daten: Dirty Read
- das Phantom-Problem
- verloren gegangene Änderungen: Lost Update

7.2. Probleme im Mehrbenutzerbetrieb ii

Nonrepeatable Read (inkonsistentes Lesen)

Beispiel:

- Zusicherung $X = A + B + C$ am Ende der Transaktion T_1
- X und Y seien lokale Variablen
- T_i ist die Transaktion i
- Integritätsbedingung $A + B + C = 0$

7.2. Probleme im Mehrbenutzerbetrieb iii

Beispiel:

T_1	T_2
$X := A;$	
	$Y := A/2;$
	$A := Y;$
	$C := C + Y;$
	$commit;$
$X := X + B;$	
$X := X + C;$	
$commit;$	

7.2. Probleme im Mehrbenutzerbetrieb iv

Dirty Read

T_1	T_2
$read(X)$;	
$X := X + 100$;	
$write(X)$;	
	$read(X)$;
	$Y := Y + X$;
	$write(Y)$;
	$commit$;
$abort$;	

[Anmerkung: Abkürzung $read(X)$, wenn lokale Variable und DB-Objekt gleichen Namen haben]

Das Phantom-Problem

T_1	T_2
select count $(*)$ *into X* *from Mitarbeiter*;	
	insert *into Mitarbeiter* *values* (*Meier*, 50000, ...); *commit*;
update Mitarbeiter *set Gehalt* = *Gehalt* + 10000/*X*; *commit*;	

7.2. Probleme im Mehrbenutzerbetrieb vi

Lost Update

T_1	T_2	X
$read(X);$		10
	$read(X);$	10
$X := X + 1;$		10
	$X := X + 1;$	10
$write(X);$		11
	$write(X);$	11

7.3. Einführung in die Serialisierbarkeit

$$
\begin{aligned}
T_1 \quad &: \quad \mathit{read}\, A;\ \ A := A - 10;\ \ \mathit{write}\, A;\ \ \mathit{read}\, B; \\
&\qquad\ \ B := B + 10;\ \ \mathit{write}\, B; \\
T_2 \quad &: \quad \mathit{read}\, B;\ \ B := B - 20;\ \ \mathit{write}\, B;\ \ \mathit{read}\, C; \\
&\qquad\ \ C := C + 20;\ \ \mathit{write}\, C;
\end{aligned}
$$

Ausführungsvarianten für zwei Transaktionen:

- seriell, etwa T_1 vor T_2
- „gemischt“, d.h. abwechselnd Schritte von T_1 und T_2

Beispiele für verschränkte Ausführungen i

Ausführung 1		Ausführung 2		Ausführung 3	
T_1	T_2	T_1	T_2	T_1	T_2
$read\ A$		$read\ A$		$read\ A$	
$A - 10$			$read\ B$	$A - 10$	
$write\ A$		$A - 10$			$read\ B$
$read\ B$			$B - 20$	$write\ A$	
$B + 10$		$write\ A$			$B - 20$
$write\ B$			$write\ B$	$read\ B$	
	$read\ B$	$read\ B$			$write\ B$
	$B - 20$		$read\ C$	$B + 10$	
	$write\ B$	$B + 10$			$read\ C$
	$read\ C$		$C + 20$	$write\ B$	
	$C + 20$	$write\ B$			$C + 20$
	$write\ C$		$write\ C$		$write\ C$

Beispiele für verschränkte Ausführungen ii

Effekt der unterschiedlichen Ausführungen

	A	B	C	$A+B+C$
initialer Wert	10	10	10	30
nach Ausführung 1	0	0	30	30
nach Ausführung 2	0	0	30	30
nach Ausführung 3	0	20	30	50

Serialisierbarkeit i

Eine verschränkte Ausführung mehrerer Transaktionen heißt serialisierbar, wenn ihr Effekt identisch zum Effekt einer (beliebig gewählten) seriellen Ausführung dieser Transaktionen ist.

Serialisierbarkeit ii

Das Read/Write-Modell

- Transaktion T ist eine endliche Folge von Operationen (Schritten) p_i der Form $r(x_i)$ oder $w(x_i)$:

$$T = p_1 p_2 p_3 \cdots p_n \quad \text{mit} \quad p_i \in \{r(x_i), w(x_i)\}$$

- vollständige Transaktion T hat als letzten Schritt entweder einen Abbruch a oder ein Commit c:

$$T = p_1 \cdots p_n a$$

oder

$$T = p_1 \cdots p_n c.$$

Serialisierbarkeit iii

Schedule

Ein Schedule ist ein Präfix eines vollständigen Schedules.

$$\overbrace{\underbrace{r_1(x)r_2(x)w_1(x)}_{\text{ein Schedule}}\, r_2(y)a_1w_2(y)c_2}^{\text{ein vollständiger Schedule}}$$

Ein vollständiger Schedule ist eine Folge von DB-Operationen, so dass alle Operationen zu vollständigen Transaktionen gehören und alle Operationen dieser Transaktionen im Schedule in derselben relativen Reihenfolge auftreten wie in der Transaktion.

Serialisierbarkeit iv

Serieller Schedule

Ein serieller Schedule s für T ist ein vollständiger Schedule der folgenden Form:

$$s := T_{\rho(1)} \cdots T_{\rho(n)} \qquad \text{für eine Permutation } \rho \text{ von } \{1, \ldots, n\}$$

serielle Schedules für zwei Transaktionen $T_1 := r_1(x)w_1(x)c_1$ und $T_2 := r_2(x)w_2(x)c_2$:

$$s_1 := \underbrace{r_1(x)w_1(x)c_1}_{T_1}\underbrace{r_2(x)w_2(x)c_2}_{T_2}$$

$$s_2 := \underbrace{r_2(x)w_2(x)c_2}_{T_2}\underbrace{r_1(x)w_1(x)c_1}_{T_1}$$

Serialisierbarkeit v

Korrektheitskriterium

Ein Schedule s ist korrekt, wenn der Effekt des Schedules s (Ergebnis der Ausführung des Schedules) äquivalent dem Effekt eines (beliebigen) seriellen Schedules s' bzgl. derselben Menge von Transaktionen ist (in Zeichen $s \approx s'$).

Ist ein Schedule s äquivalent zu einem seriellen Schedule s', dann ist s serialisierbar (zu s').

Serialisierbarkeit vi

Konflikte

T_1	T_2
$read\ A$	
	$read\ A$

unabhängig von Reihenfolge

T_1	T_2
$read\ A$	
	$write\ A$

abhängig von Reihenfolge

T_1	T_2
	$read\ A$
$write\ A$	

abhängig von Reihenfolge

T_1	T_2
	$write\ A$
$write\ A$	

abhängig von Reihenfolge

Serialisierbarkeit vii

Konfliktserialisierbarkeit

- Zwei Schedules s und s' heißen konfliktäquivalent, wenn die Reihenfolge zweier in Konflikt stehender Operationen in beiden Schedules gleich ist.
- andernfalls: unterschiedliche Effekte, z.B.

$$w_1(x)w_2(x) \text{ vs. } w_2(x)w_1(x)$$

Ein Schedule s ist genau dann konfliktserialisierbar, wenn s konfliktäquivalent zu einem seriellen Schedule ist.

Graphbasierter Test i

Konfliktgraph $G(s) = (V, E)$ von Schedule s:

1. Knotenmenge V enthält alle in s vorkommende Transaktionen
2. Kantenmenge E enthält alle gerichteten Kanten zwischen zwei in Konflikt stehenden Transaktionen

Graphbasierter Test ii

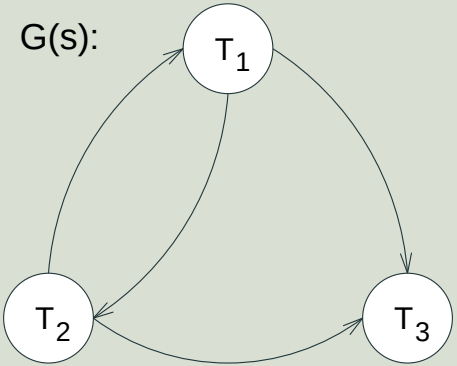
Zeitlicher Verlauf dreier Transaktionen

T_1	T_2	T_3
r(y)		
		r(u)
	r(y)	
w(y)		
w(x)		
	w(x)	
	w(z)	
		w(x)

$$s = r_1(y)r_3(u)r_2(y)w_1(y)w_1(x)w_2(x)w_2(z)w_3(x)$$

Graphbasierter Test iii

Konfliktgraph



Graphbasierter Test iv

Eigenschaften eines Konfliktgraphen $G(s)$

1. Ist s ein serieller Schedule, dann ist der vorliegende Konfliktgraph ein azyklischer Graph.
2. Für jeden azyklischen Graphen $G(s)$ lässt sich ein serieller Schedule s' konstruieren, so dass s konfliktserialisierbar zu s' ist (Test bspw. durch topologisches Sortieren).
3. Enthält ein Graph Zyklen, dann ist der zugehörige Schedule nicht konfliktserialisierbar.

7.4. Sperrprotokolle i

- Sichern der Serialisierbarkeit durch exklusiven Zugriff auf Objekte (Synchronisation der Zugriffe)
- Implementierung über Sperren und Sperrprotokolle
- Sperrprotokoll garantiert (Konflikt-) Serialisierbarkeit ohne zusätzliche Tests!

7.4. Sperrprotokolle ii

Sperrmodelle

Schreib- und Lesesperren in folgender Notation:

- $rl(x)$: Lesesperre (engl. *read lock*) auf einem Objekt x
- $wl(x)$: Schreibsperre (engl. *write lock*) auf Objekt x

Entsperren $ru(x)$ und $wu(x)$, oft zusammengefasst $u(x)$ für engl. *unlock*

7.4. Sperrprotokolle iii

Kompatibilitätsmatrix

- für elementare Sperren

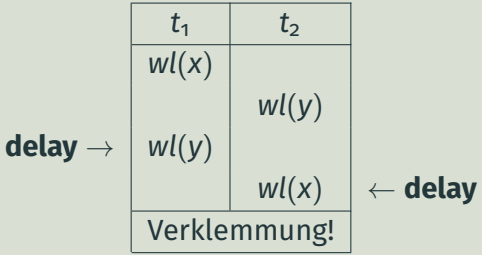
	$rl_i(x)$	$wl_i(x)$
$rl_j(x)$	$\surd$	—
$wl_j(x)$	—	—

7.4. Sperrprotokolle iv

Sperrdisziplin

- Schreibzugriff $w(x)$ nur nach Setzen einer Schreibsperre $wl(x)$ möglich
- Lesezugriffe $r(x)$ nur nach $rl(x)$ oder $wl(x)$ erlaubt
- nur Objekte sperren, die nicht bereits von einer anderen Transaktion gesperrt sind
- nach $rl(x)$ nur noch $wl(x)$ erlaubt, danach auf x keine Sperre mehr; Sperren derselben Art werden maximal einmal gesetzt
- nach $u(x)$ durch t_i darf t_i kein erneutes $rl(x)$ oder $wl(x)$ ausführen
- vor einem `commit` müssen alle Sperren aufgehoben werden

Verklemmungen (deadlocks)

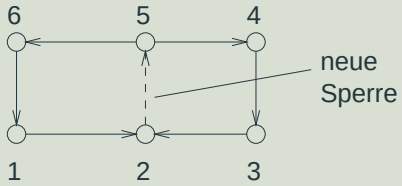


Alternativen:

- Verklemmungen werden erkannt und beseitigt
- Verklemmungen werden von vornherein vermieden

Verklemmungserkennung und -auflösung

Wartegraph



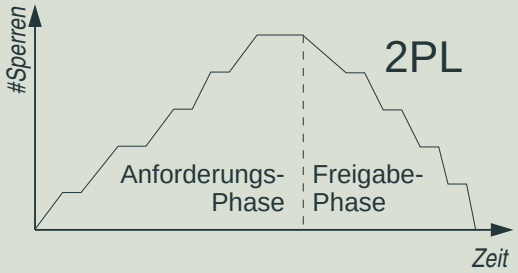
Auflösen durch Abbruch einer Transaktion, Kriterien:

- Anzahl der aufgebrochenen Zyklen
- Länge einer Transaktion
- Rücksetzaufwand einer Transaktion
- Wichtigkeit einer Transaktion
- ...

Sperrprotokolle: Notwendigkeit

T_1	T_2
$wl(x)$	
$w(x)$	
$u(x)$	
	$wl(x)$
	$w(x)$
	$u(x)$
	$wl(y)$
	$w(y)$
	$u(y)$
$wl(y)$	
$w(y)$	
$u(y)$	

Zwei-Phasen-Sperr-Protokoll

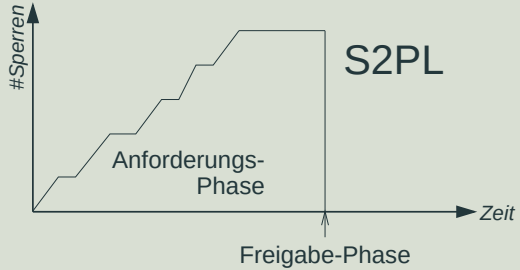


[2PL = 2 Phase Locking]

Konflikt bei Nichteinhaltung des 2PL

T_1	T_2
$u(x)$	
	$wl(x)$
	$wl(y)$
	$\vdots$
	$u(x)$
	$u(y)$
$wl(y)$	
$\vdots$	

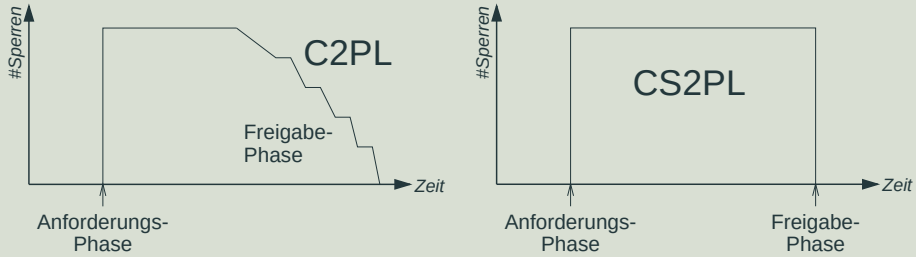
Striktes Zwei-Phasen-Sperr-Protokoll



vermeidet kaskadierende Abbrüche!

[S2PL = Strict 2 Phase Locking]

Konservatives 2PL-Protokoll



vermeidet Deadlocks!

[C2PL = Conservative 2 Phase Locking]

[CS2PL = Conservative (and) Strict 2 Phase Locking]

Aufweichung von ACID in SQL-92: Isolationsebenen

```
set transaction
   [ { read only | read write }, ]
   [isolation level
      {  read uncommitted | read committed |
         repeatable read | serializable }, ]
   [ diagnostics size …]
```

7.5. Transaktionen in SQL-DBMS ii

Standardeinstellung:

```
set transaction read write,
   isolation level serializable
```

Bedeutung der Isolationsebenen i

- *read uncommitted*
  - schwächste Stufe: Zugriff auf nicht geschriebene Daten, nur für *read only* Transaktionen
  - statistische und ähnliche Transaktionen (ungefährer Überblick, nicht korrekte Werte)
  - keine Sperren $\rightarrow$ effizient ausführbar, keine anderen Transaktionen werden behindert
- *read committed* (Standard)
  - nur Lesen endgültig geschriebener Werte, aber *nonrepeatable read* möglich

Bedeutung der Isolationsebenen ii

- `repeatable read`
  - kein *nonrepeatable read*, aber Phantomproblem kann auftreten
- `serializable`
  - garantierte Serialisierbarkeit
  - Transaktion sieht nur Änderungen, die zu Beginn der Transaktion committed waren (plus eigene Änderungen)

Bedeutung der Isolationsebenen iii

Isolationsebene	Dirty Read	Nonrepeatable Read	Phantom Read
Read Uncommitted	+	+	+
Read Committed	–	+	+
Repeatable Read	–	–	+
Serializable	–	–	–

Bedeutung der Isolationsebenen iv

Isolationsebenen: *read committed*

T_1	T_2
select A from R	
$\rightarrow$ *alter Wert*	
	update R set A = neu
select A from R	
$\rightarrow$ *alter Wert*	
	commit
select A from R	
$\rightarrow$ *neuer Wert*	

Bedeutung der Isolationsebenen v

Isolationsebenen: *serializable*

T_1	T_2
set transaction *isolation level* *serializable* *update R set A = neu* *where C = 42* $\rightarrow$ *Fehler*	 *set transaction …* *update R set A = neu* *where C = 42* *commit*