

Informations- und Wissensmanagement

Kapitel 6: Nebenläufigkeit und Transaktionen



Gliederung

Einleitung/
Probleme
Definitionen
Locking

- Einleitung/Probleme,
- Definitionen (Transaktion, History, Konflikt, Äquivalenz, Serialisierbarkeit, ...),
- Locking.



Transaktionseigenschaften (1)

Einleitung/
Probleme
Definitionen
Locking

• ACID-Eigenschaften:

- **Atomicity** (Constraints)
- **Consistency**
- **Isolation**
- **Durability** (Logging, Recovery)



Transaktionseigenschaften (2)

Einleitung/
Probleme
Definitionen
Locking

• Atomarität

- Beispiel Bank:

Name	Konto	Guthaben
Erik B.	1892	1000
Mirco S.	2434	2000

- Überweisung besteht technisch aus zwei Elementaroperationen.

- Abbuchung(Erik B., 500),
- Einzahlung(Mirco S., 500).

• Isolation

- was passiert, wenn parallele Transaktionen
- die gleichen Datensätze schreiben?
- Datensätze von nicht abgeschlossenen Transaktionen lesen?



```

SQL> SELECT * FROM Bankkonten;
KONTNR  INHABER          STAND
-----  -
1234  Klemens Boehm      500
5678  Gunter Saake       500

SQL> UPDATE Bankkonten SET Stand=Stand+200 WHERE Kontonr=1234;
1 row updated.

SQL> SELECT * FROM Bankkonten;
KONTNR  INHABER          STAND
-----  -
1234  Klemens Boehm      300
5678  Gunter Saake       500

SQL> UPDATE Bankkonten SET Stand=Stand+200 WHERE Kontonr=5678;
1 row updated.

SQL> SELECT * FROM Bankkonten;
KONTNR  INHABER          STAND
-----  -
1234  Klemens Boehm      300
5678  Gunter Saake       700
    
```

```

SQL> SET AUTOCOMMIT OFF;
SQL> SELECT * FROM Bankkonten;
KONTNR  INHABER          STAND
-----  -
1234  Klemens Boehm      500
5678  Gunter Saake       500

1 SQL> UPDATE Bankkonten SET Stand=Stand+200 WHERE Kontonr=1234;
1 row updated.

2 SQL> SELECT * FROM Bankkonten;
KONTNR  INHABER          STAND
-----  -
1234  Klemens Boehm      300
5678  Gunter Saake       500

3 SQL> UPDATE Bankkonten SET Stand=Stand+200 WHERE Kontonr=5678;
1 row updated.

SQL> SELECT * FROM Bankkonten;
KONTNR  INHABER          STAND
-----  -
1234  Klemens Boehm      300
5678  Gunter Saake       700

SQL> COMMIT;
Commit complete.

4 SQL> SELECT * FROM Bankkonten;
KONTNR  INHABER          STAND
-----  -
1234  Klemens Boehm      300
5678  Gunter Saake       700
    
```

Benutzer 1

Benutzer 2

```

1 SQL> SET AUTOCOMMIT OFF;
  SQL> SELECT * FROM Bankkonten;
-----
KONTNR  INHABER          STAND
-----
1234 Klewens Boehn      500
5678 Gunter Saake       500
SQL>
SQL>
SQL>
2 SQL> SELECT * FROM Bankkonten;
-----
KONTNR  INHABER          STAND
-----
1234 Klewens Boehn      500
5678 Gunter Saake       500
SQL>
SQL>
SQL>
3 SQL> SELECT * FROM Bankkonten;
-----
KONTNR  INHABER          STAND
-----
1234 Klewens Boehn      500
5678 Gunter Saake       500
SQL>
SQL>
SQL>
4 SQL> SELECT * FROM Bankkonten;
-----
KONTNR  INHABER          STAND
-----
1234 Klewens Boehn      300
5678 Gunter Saake       700
SQL>

```

... und Transaktionen - 7

```

SQL> SET AUTOCOMMIT OFF;
SQL> SELECT * FROM Bankkonten;
-----
KONTNR  INHABER          STAND
-----
1234 Klewens Boehn      500
5678 Gunter Saake       500
SQL> UPDATE Bankkonten SET Stand=Stand+200 WHERE Kontonr=1234;
1 row updated.
SQL> SELECT * FROM Bankkonten;
-----
KONTNR  INHABER          STAND
-----
1234 Klewens Boehn      700
5678 Gunter Saake       500
SQL> UPDATE Bankkonten SET Stand=Stand+200 WHERE Kontonr=5678;
1 row updated.
SQL> SELECT * FROM Bankkonten;
-----
KONTNR  INHABER          STAND
-----
1234 Klewens Boehn      300
5678 Gunter Saake       700
SQL> ROLLBACK;
Rollback complete.
SQL> SELECT * FROM Bankkonten;
-----
KONTNR  INHABER          STAND
-----
1234 Klewens Boehn      500
5678 Gunter Saake       500
SQL>

```

... Transaktionen - 8

Rollback
als Gegensatz
zum Commit.

Synchronisation in Datenbanken

Einleitung/
Probleme
Definitionen
Locking

- Zentrales Leistungsmerkmal von Datenbanken: Viele Benutzer können die gleichen Daten gleichzeitig sowohl lesend als auch schreibend zugreifen.
- Konsistenz muß sichergestellt sein – Aufgabe der **Synchronisationskomponente**.
- Benutzer sollen vom Multi-User Betrieb so wenig wie möglich merken. *Nebenläufigkeit* soll transparent sein. ‚Illusion‘, daß man der einzige Nutzer ist.
- Engl. *concurrency, concurrency control*.

Synchronisationsprobleme

Einleitung/
Probleme
Definitionen
Locking

- Unkontrollierte **nicht-serielle Ausführung** führt zu Inkonsistenz:
 - **Lost Updates**,
 - **Non-repeatable reads**, inkonsistente Lesezugriffe, unterschiedliche Ergebnisse bei identischer Anfrage
 - **Dirty Reads**, Lesen von Updates, die noch nicht committet sind.
 - **Phantome**, Lesen von neu angelegten Datentupeln aus einer Transaktion, die noch nicht committet hat

Lost Update

Einleitung/
Probleme
Definitionen
Locking

- Programm T_1 transferiert EUR 300,- von A auf B, Programm T_2 schreibt Konto A 3 % Zinsen gut.
- Zinsen aus Schritt 5 von Programm T_2 gehen verloren, weil T_1 den Wert Schritt 6 überschreibt.

Schritt	T1	T2
1	Read(A, a1)	
2	a1 := a1-300	
3		Read(A, a2)
4		a2 := a2 *1.03
5		Write(A, a2)
6	Write(A, a1)	
7	Read(B, b1)	
8	b1 := b1 + 300	
9	Write(B, b1)	

Dirty Read

Einleitung/
Probleme
Definitionen
Locking

- *Commit, Abort*.
- Programm T_2 berechnet Zinsen auf einem Wert aus einem inkonsistenten Zustand

Schritt	T1	T2
1	Read(A, a1)	
2	a1 := a1-300	
3	Write(A, a1)	
4		Read(A, a2)
5		a2 := a2 *1.03
6		Write(A, a2)
7		commit
8	Read(B, b1)	
9	...	
10	abort	

Non-Repeatable Reads

Einleitung/
Probleme
Definitionen
Locking

- Programm liest Datenobjekt mehr als einmal und sieht Änderung, die eine andere Transaktion durchgeführt hat.

Schritt	T1	T2
1	Read(A, a1)	
2	a1 := a1-300	
3	Write(A, a1)	
4		Read(A, a2)
5		a2 := a2 * 1.03
6		Write(A, a2)
7	Read(A, a3)	
8	...	

z

Phantom Reads

Einleitung/
Probleme
Definitionen
Locking

- während einer Transaktion wiederholte Anfragen ergeben unterschiedliche Ergebnismengen
- entspricht Non-repeatable Reads auf Mengen statt Werten

T1	T2
select count(*) from Bank	
	insert into Bank (Name, Konto, Wert) values ('Erik B', 29384, 100.00)
select count(*) from Bank	
...	...

Lösbar ohne Synchronisation, aber...

Einleitung/
Probleme
Definitionen
Locking

- **Serielle Ausführung** von Transaktionen
 - neue Transaktion kann erst starten, wenn vorhergehende alle Daten geschrieben hat
 - Datenbank ist am Ende jeder Transaktion konsistent.
 - kein Aufwand durch Sperren und Scheduling
 - **Aber:** Extreme Wartezeiten und unbefriedigende Ausnutzung der Ressourcen.
 - während Festplatte arbeitet, steht der Rest des Rechners ungenutzt still

Maßnahmen zur Synchronisation

Einleitung/
Probleme
Definitionen
Locking

- Zusammenfassen von mehreren Zugriffen zu **Transaktionen**
 - commit/rollback bestätigt/verwirft **alle** Änderungen der Transaktion
- Sperren von einzelnen Lese / Schreibzugriffen
- Umsortieren der Reihenfolge von parallelen Transaktionen
- Abbrechen von Transaktionen, die mit anderen in Konflikt stehen
 - DBMS kann Transaktion selbst zurücksetzen

Problem: Tradeoff zwischen Leistung der DB und Garantien zur Isolation!

Nebenläufigkeit in existierenden DBMS

Einleitung/
Probleme
Definitionen
Locking

- Beispiel: Isolation Levels in Oracle
Read committed ist die Voreinstellung

Anm.: Lost Updates immer ausgeschlossen

Isolation Level	Dirty Read	Nonrepeatable Read	Phantom Read
Read uncommitted	Possible	Possible	Possible
Read committed	Not possible	Possible	Possible
Repeatable read	Not possible	Not possible	Possible
Serializable	Not possible	Not possible	Not possible



Transaktionen und Histories

Transaktionen

Einleitung/
Probleme
Definitionen
Locking

- **Transaktion** := Ausführung eines Programms, das auf die Datenbank (lesend oder schreibend) zugreift.
- Programmausführung \neq Programm-Code.
- Genauer: Repräsentation der Ausführung, die folgende Charakteristika umfasst:
 - Gelesene und geschriebene Datenobjekte,
 - Reihenfolge ihrer Ausführung,
 - kommt am Ende ein Commit (bzw. Abort) oder nicht?

Zielstellung dieses Kapitels

Einleitung/
Probleme
Definitionen
Locking

- Wie muss ein Ablaufplan (**Schedule**) für mehrere Transaktionen aussehen, der
 - das gleiche Resultat zeigt wie eine serielle Ausführung, aber
 - die Ressourcen des Rechners durch parallele Ausführung besser ausnutzt?

Beispiel für Transaktionen

Einleitung/
Probleme
Definitionen
Locking

- Beispiel:
Procedure P begin
 Start;
 temp := Read(x);
 temp := temp + 1;
 Write(x, temp);
 Commit
end
- Repräsentation: $r_i[x] \rightarrow w_i[x] \rightarrow c_i$
- Transaktion ist partielle Ordnung $(\Sigma, <)$
 (Σ wird im Folgenden meistens weggelassen.)

Operationen

Konflikte

Einleitung/
Probleme
Definitionen
Locking

- **Zwei Operationen p, q konfliktieren** := p, q greifen auf das gleiche Datenobjekt zu, und p oder q ist eine Schreiboperation.
- Weitere Operationen – Definition von 'Konflikt' muß erweitert werden.
- Beispiel. Kompatibilitätsmatrix:

	Read	Write	Increment	Decrement
Read	y	n	n	n
Write	n	n	n	n
Increment	n	n	y	y
Decrement	n	n	y	y

Transaktion – formale Definition

Einleitung/
Probleme
Definitionen
Locking

Transaktion ist partielle Ordnung mit Ordnungsrelation $<$, so daß gilt

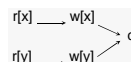
1. $T_i \subseteq \{r_i[x], w_i[x] \mid x \text{ ist ein Datenobjekt}\} \cup \{a_i, c_i\}$,
2. $a_i \in T_i \Leftrightarrow c_i \notin T_i$;
3. wenn es sich bei t um c_i oder a_i handelt, dann gilt für jede andere Operation $p \in T_i$: $p <_i t$; und
4. wenn $r_i[x], w_i[x] \in T_i$, dann $r_i[x] <_i w_i[x]$ oder $w_i[x] <_i r_i[x]$.

Ordnungsbeziehung ist essentiell um Konflikte zu bestimmen!

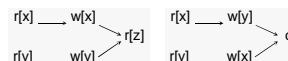
Transaktion – weitere Beispiele

Einleitung/
Probleme
Definitionen
Locking

- Gegeben: Halbordnung von Operationen



ist Transaktion.



ist keine TA.

ist keine TA.

Beziehungen zwischen Transaktionen

Einleitung/
Probleme
Definitionen
Locking

- Im DBMS viele parallele Transaktionen möglich
- Reads-from-Beziehung:**
Transaktion T_i liest von Transaktion T_j wenn
 - T_i liest x , nachdem T_j x geschrieben hat;
 - T_i abortet nicht, bevor T_j liest; und
 - Jede Transaktion, die x schreibt, bevor T_i liest, und nachdem T_j x überschreibt, abortet, bevor T_i liest.

Beispiele für Beziehungen

Einleitung/
Probleme
Definitionen
Locking

- $w_1[x] w_2[x] r_3[x] r_4[x] c_1 c_2 c_3 c_4$
reads-from Beziehungen:
 T_3 von T_2 , T_4 von T_2 .
- $w_1[x] w_2[x] r_3[x] r_4[x] c_1 a_2 c_3 c_4$
reads-from Beziehungen:
 T_3 von T_2 , T_4 von T_2 .
- $w_1[x] w_2[x] a_2 r_3[x] r_4[x] c_1 c_3 c_4$
reads-from Beziehungen:
 T_3 von T_1 , T_4 von T_1 .

Histories

Einleitung/
Probleme
Definitionen
Locking

- Ausführung der Operationen mehrerer Transaktionen, die miteinander 'verzahnt' sind, d.h. nebenläufig ablaufen.
- Formal: $H = \{T_1, T_2, \dots, T_n\}$ ist eine Menge von Transaktionen.
- Anm.: in der Literatur werden *Histories* auch häufig als *Schedules* bezeichnet

Vollständige Histories

Einleitung/
Probleme
Definitionen
Locking

- Vollständige History H über T :** := partielle Ordnung mit Ordnungsbeziehung $<_H$, so dass
 - $H = \bigcup_{i=1}^n T_i$
 - $<_H \supseteq \bigcup_{i=1}^n <_i$
 - $p, q \in H$ konfiglieren $\rightarrow p <_H q$ oder $q <_H p$

Probleme mit vollst. Histories

Einleitung/
Probleme
Definitionen
Locking

- vollständige Histories erst nach Abschluss aller Transaktionen definiert
 - für *Scheduling* nutzlos
- praktische Herangehensweise im folgenden:
 - History:** Präfix einer vollständigen History.

„Arbeitsversion“, um eine vollständige History zu bekommen

Histories – Beispiele (1)

Einleitung/
Probleme
Definitionen
Locking

- Gegeben zwei Transaktionen:

$$T_1 \quad r_1[x] \text{---} w_1[x] \quad c_1$$

$$T_2 \quad r_2[x] \text{---} w_2[x] \text{---} c_2$$
- Vollständige History:

$$r_2[x] \text{---} w_2[x] \text{---} c_2$$

$$r_1[x] \text{---} w_1[x] \quad c_1$$

Histories – Beispiele (2)

- Gegeben zwei Transaktionen:

$$T_1 \begin{array}{l} r_1[x] \text{---} w_1[x] \\ r_1[y] \text{---} w_1[y] \end{array} \xrightarrow{c_1} \quad T_2 \begin{array}{l} r_2[x] \text{---} w_2[x] \text{---} c_2 \end{array}$$
- Keine Histories:

$$\begin{array}{l} r_2[x] \text{---} w_2[x] \text{---} c_2 \\ r_1[x] \text{---} w_1[x] \\ r_1[y] \text{---} w_1[y] \end{array} \xrightarrow{c_1} \quad \begin{array}{l} r_2[x] \text{---} w_2[x] \text{---} c_2 \\ r_1[x] \text{---} w_1[x] \\ r_1[y] \text{---} w_1[y] \end{array} \xrightarrow{c_1}$$

Einleitung/
Probleme
Definitionen
Locking

IWM: Nebenläufigkeit und Transaktionen – 31

Histories – Beispiele (3)

- Gegeben zwei Transaktionen:

$$T_1 \begin{array}{l} r_1[x] \text{---} w_1[x] \\ r_1[y] \text{---} w_1[y] \end{array} \xrightarrow{c_1} \quad T_2 \begin{array}{l} r_2[x] \text{---} w_2[x] \text{---} c_2 \end{array}$$
- History, aber nicht *vollständig*:

$$\begin{array}{l} r_2[x] \\ r_1[x] \text{---} w_1[x] \\ r_1[y] \text{---} w_1[y] \end{array} \xrightarrow{c_1}$$

Einleitung/
Probleme
Definitionen
Locking

IWM: Nebenläufigkeit und Transaktionen – 32

Äquivalenz von Histories

- mehrere Definitionen möglich:
 - (Konflikt-)Äquivalenz,
 - (Sicht-)Äquivalenz.
- im Folgenden nur Konflikt-Äquivalenz betrachtet
- Definition **Konflikt-Äquivalenz**:
Histories H, H' sind *Konfliktäquivalent*, wenn
 - gleiche Transaktionen, gleiche Operationen;
 - gleiche Ordnung konfligierender Operationen.
z.B. gehören p_i und q_j zu T_i bzw T_j ,
 $a_i, a_j \in H$. Wenn $p_i <_H q_j$, dann $p_i <_{H'} q_j$.

Einleitung/
Probleme
Definitionen
Locking

IWM: Nebenläufigkeit und Transaktionen – 33

Äquivalenz von Histories – Beispiele (1)

- Gegeben zwei Transaktionen:

$$T_1 \begin{array}{l} r_1[x] \text{---} w_1[x] \\ r_1[y] \text{---} w_1[y] \end{array} \xrightarrow{c_1} \quad T_2 \begin{array}{l} r_2[x] \text{---} w_2[x] \text{---} c_2 \end{array}$$
- Eine vollständige History, OK:

$$\begin{array}{l} r_2[x] \text{---} w_2[x] \text{---} c_2 \\ r_1[x] \text{---} w_1[x] \\ r_1[y] \text{---} w_1[y] \end{array} \xrightarrow{c_1}$$
- Beispiel für eine Konflikt-äquivalente History, die nicht identisch ist?

Einleitung/
Probleme
Definitionen
Locking

IWM: Nebenläufigkeit und Transaktionen – 34

Äquivalenz von Histories – Beispiele (2)

- History 1:

Schritt	T1	T2	T3
1	Read(A)		
2		Write(A)	
3	Write(A)		
4			Write(A)
- History 2:

Schritt	T1	T2	T3
1	Read(A)		
2	Write(A)		
3		Write(A)	
4			Write(A)
- Sind diese Histories Konflikt-äquivalent?

Alle Transaktionen committen

Einleitung/
Probleme
Definitionen
Locking

IWM: Nebenläufigkeit und Transaktionen – 35

Überlegungen zur Korrektheit

- History muß nicht korrekt sein, kann z. B. Lost Updates etc. enthalten.
- Ziel im folgenden: suche eine Definition 'Korrektheit' für Histories.

Einleitung/
Probleme
Definitionen
Locking

IWM: Nebenläufigkeit und Transaktionen – 36

Committed Projections

Einleitung/
Probleme
Definitionen
Locking

- Committed projection einer History H – Abkürzung: $C(H)$:= resultiert aus H durch Löschen aller Operationen, die nicht committed sind.
- Illustration:

IWM: Nebenläufigkeit und Transaktionen – 37

Präfixe von Histories

Einleitung/
Probleme
Definitionen
Locking

$H = o_1 \dots o_n$ (linear, der Einfachheit halber) $H' = o_1 \dots o_m$ $H'' = o_1 \dots o_m$	α = "History enthält weniger als 10 Operationen"	β = "Alle Operationen sind Lese-Operationen"	γ = "History enthält mehr als 10 Operationen"
Präfixe: Wenn H α erfüllt, erfüllen es auch H' , H'' und jeder andere Präfix. α ist <i>prefix commit-closed</i> .	Wenn H β erfüllt, erfüllen es auch H' , H'' und jeder andere Präfix. β ist <i>prefix commit-closed</i> .	H' muß γ nicht erfüllen, selbst wenn H es erfüllt. γ ist nicht <i>prefix commit-closed</i> .	

Weitere Beispiele für Eigenschaften, die *prefix commit-closed* sind?

IWM: Nebenläufigkeit und Transaktionen – 38

Präfixe von Histories (2)

Einleitung/
Probleme
Definitionen
Locking

- Eigenschaft von Histories ist *prefix commit-closed*, wenn gilt:
 H erfüllt die Eigenschaft.
 $\Rightarrow C(H')$ erfüllt die Eigenschaft, H' ist Präfix von H .

$C(H)$:= *committed projection*, d. h. nur Operationen von Transaktionen, die committed sind.

- Vorangegangene Folie hat so getan, als ob man alle Präfixe betrachtet. Es reicht, committed projections der Präfixe zu betrachten.

IWM: Nebenläufigkeit und Transaktionen – 39

Korrektheit von Histories

Einleitung/
Probleme
Definitionen
Locking

- Überlegung hinter vorangegangener Definition:
 - Korrektheitskriterium für Histories muss *prefix commit-closed* Eigenschaft haben
 - Scheduler generiert History; generiert aber auch jedes Präfix.

- Absturz des DBMS – History nach Wiederaufnahme des Betriebs hat diese Eigenschaft \rightarrow *History ist korrekt!*

IWM: Nebenläufigkeit und Transaktionen – 40

Eigenschaften von Histories

Einleitung/
Probleme
Definitionen
Locking

- Conflict-Serializability (**CSR**)
 - Konflikt-Serialisierbarkeit
- Recoverability (**RC**)
 - Rücksetzbarkeit
- Avoids Cascading Aborts (**ACA**)
 - keine kaskadierenden Abbrüche
- Strictness (**ST**)
 - keine Zwischenergebnisse von uncommitted Transaktionen lesen

IWM: Nebenläufigkeit und Transaktionen – 41

Konflikt-Serialisierbarkeit (1)

Einleitung/
Probleme
Definitionen
Locking

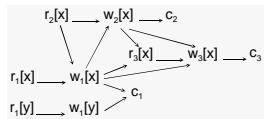
- Committed projection einer History H : $C(H)$:= resultiert aus H , indem alle Operationen gelöscht werden, die nicht committed sind.
- H ist serialisierbar, wenn $C(H)$ zu serieller History H_s äquivalent ist.
- Ist diese History serialisierbar? Wenn ja, wie sieht äquivalente serielle History aus?

IWM: Nebenläufigkeit und Transaktionen – 42

Konflikt-Serialisierbarkeit (2)

Einleitung/
Probleme
Definitionen
Locking

- Konflikt-Serialisierbarkeit ist offensichtlich prefix commit-closed.
- Illustration:



z

Recoverability (1)

Einleitung/
Probleme
Definitionen
Locking

- 'Commit einer Transaktion' – kein Abort mehr mgl.
- Commit nur, wenn alle Änderungen an Datenobjekten, die T gelesen hat, committet sind.
- Gegenbeispiel: Dirty Read.

Serialisierbar?

Schritt	T1	T2
1	Read(A, a1)	
2	a1 := a1 - 300	
3	Write(A, a1)	
4		Read(A, a2)
5		a2 := a2 + 1.03
6		Write(A, a2)
7		commit
8	Read(B, b1)	
9	...	
10	abort	

Recoverability (2)

Einleitung/
Probleme
Definitionen
Locking

- Ausführung ist recoverable := Commit von T nach Commit aller Transaktionen, von denen T gelesen hat.
- folgendes darf **nie** passieren:
 $r_1[x] w_1[x] r_2[x] w_2[x] c_2 a_1$

Cascading Aborts (1)

Einleitung/
Probleme
Definitionen
Locking

- Transaktion T abortet, wenn sie nicht korrekt beenden kann.
- T muß zurückgesetzt werden:
 - Writes von T,
 - dto. alle anderen Transaktionen, die solche Änderungen gelesen haben.
 Undo kann zu cascading abort führen.

Cascading Aborts (2)

Einleitung/
Probleme
Definitionen
Locking

- Beispiel für cascading abort:
 - x und y haben zunächst Wert 1.
 - Zwei Transaktionen T₁ und T₂
 $r_1[x] w_1[x] r_2[x] a_1 \dots$
 - Abort von T₁, undo w₁[x]
⇒ T₂ muß ebenfalls aborten.
- Cascading aborts sind möglich, auch wenn die History recoverable ist.

Cascadelessness

Einleitung/
Probleme
Definitionen
Locking

- Cascading aborts sind unerwünscht:
 - Sie ziehen 'Buchhaltung' nach sich,
 - Zahl der Transaktionen, die aborten müssen, ist nicht beschränkt.
- DBMS ist cascadeless := Jede Transaktion liest Datenobjekte nur von committeten Transaktionen.



Strictness

Einleitung/
Probleme
Definitionen
Locking

- **Strictness** := keinen Wert einer nicht abgeschlossenen Transaktion lesen oder überschreiben
- Problem: Undo basiert auf *Before-Images*.
- Veranschaulichung:
 1. $w_1(x, 2)$; $w_2(x, 3)$; a_1
 2. $w_1(x, 2)$; $w_2(x, 3)$; a_1 ; a_2
- Strikte History: $w_i(x, v)$ wird verzögert, bis alle Transaktionen, die x geschrieben haben, entweder committet oder abortet haben
 $w_1(x, 2)$; a_1 ; $w_2(x, 3)$



Beispiele

Einleitung/
Probleme
Definitionen
Locking

- $T_1 = w_1[x] w_1[y] w_1[z] c_1$
- $T_2 = r_2[u] w_2[x] r_2[y] w_2[y] c_2$
- $H_7 = w_1[x] w_1[y] r_2[u] w_2[x] r_2[y] w_2[y] c_2 w_1[z] c_1$
- $H_8 = w_1[x] w_1[y] r_2[u] w_2[x] r_2[y] w_2[y] w_1[z] c_1 c_2$
- $H_9 = w_1[x] w_1[y] r_2[u] w_2[x] w_1[z] c_1 r_2[y] w_2[y] c_2$
- $H_{10} = w_1[x] w_1[y] r_2[u] w_1[z] c_1 w_2[x] r_2[y] w_2[y] c_2$
- Welche Histories sind recoverable, welche kommen ohne cascading abort aus, welche sind strict?



Serialisierbarkeitsgraph (1)

Einleitung/
Probleme
Definitionen
Locking

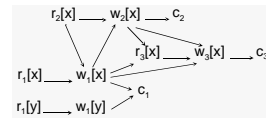
- Test, ob ein Schedule (= Transaktionen, zusammen mit ihren Operationen) serialisierbar ist:
 - Erzeuge Serialisierbarkeitsgraphen bzw. **Abhängigkeitsgraphen**.
 - Knoten = Transaktionen,
 - (gerichtete) Kante = Abhängigkeit zwischen zwei Transaktionen: Transaktionen greifen auf das gleiche Datenobjekt zu, und Operationen konfliktieren.



Serialisierbarkeitsgraph (2)

Einleitung/
Probleme
Definitionen
Locking

- Wie sieht der Serialisierbarkeitsgraph aus?



Serialisierbarkeitsgraph (3)

Einleitung/
Probleme
Definitionen
Locking

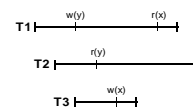
- Theorem: Schedule ist serialisierbar, wenn entsprechender Abhängigkeitsgraph **zyklfrei** ist.
- Denn: Partielle Ordnung, zu totaler Ordnung erweiterbar – äquivalenter serieller Schedule.



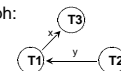
Serialisierbarkeitsgraph – Beispiel

Einleitung/
Probleme
Definitionen
Locking

- $r(x)/w(x)$ – Lese-/Schreibzugriff auf Datenobjekt x .
- Schedule:



- Abhängigkeitsgraph:



- azyklisch → Schedule ist serialisierbar.
- Serialisierungsreihenfolge: $T3 < T1 < T2$





Serialisierbarkeitsgraph – Diskussion

Einleitung/
Probleme
Definitionen
Locking

- Ansatz ist nicht praktikabel.
- Serialisierbarkeit von Schedules nur im nachhinein überprüfbar.
 - Administrativer Overhead ist zu hoch: Abhängigkeiten zu bereits terminierten Transaktionen müssen ebenfalls berücksichtigt werden. z.B. wird Beziehung zwischen T1 und T3 erst nach Commit von T3 klar.



Sperrungen und Sperrprotokolle



Verzögern und Zurücksetzen

Einleitung/
Probleme
Definitionen
Locking

- Verzögern und Zurücksetzen mittels Locking sind übliche Techniken in der Transaktionsverwaltung.
- Locking:
 - Datenobjekte werden zum Schreiben und/oder Lesen gesperrt
 - Verzögerte Transaktionen müssen warten bis Lock aufgehoben
 - zahlreiche Strategien zum Locking möglich, im folgenden: 2-Phase-Locking

Im Allgemeinen immer noch schneller als rein serielle Ausführung der Transaktionen



Locking

Einleitung/
Probleme
Definitionen
Locking

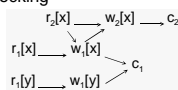
- Transaktion i setzt Lock für Operation o auf Datenobjekt x – Notation: $o_i[x]$
- Einfachster Fall: Nur Read Locks und Write Locks („RX Locking Scheme“).
 - read lock (rl): Lesesperre
 - write lock (wl): Schreibsperre
 - read/write unlock (ru/wu): Sperre aufheben



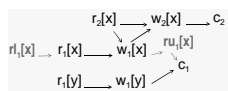
Locking - Beispiel

Einleitung/
Probleme
Definitionen
Locking

- ohne Locking



- T_1 setzt Read-Lock auf x



Sperrdisziplin

Einleitung/
Probleme
Definitionen
Locking

- Schreibzugriff $w[x]$ nur nach $w[x]$
- Lesezugriff $r[x]$ nur nach $rl[x]$ oder $w[x]$
- Transaktion darf nur Objekte sperren, die keine Locks von anderen Transaktionen tragen, sonst: verzögert bis ru/wu
- nach $rl[x]$ ist noch ein $w[x]$ erlaubt, sonst keine weiteren Sperren
- nach dem Entsperrten darf eine Transaktion das selbe Objekt nicht erneut sperren
- vor Commit alle Sperren aufheben

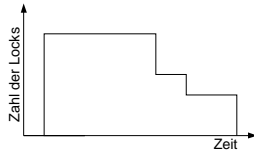




Konservatives 2PL

Einleitung/
Probleme
Definitionen
Locking

- stellt Deadlock-Freiheit sicher:
Alle Locks werden am Anfang der Transaktion
gesetzt



- Kombination möglich: **Konservatives Strenges
Zwei-Phasen-Sperrprotokoll**



Zusammenfassung

Einleitung/
Probleme
Definitionen
Locking

- Nebenläufiger Zugriff
– fundamental wichtiges Anliegen.
- Serielle Ausführung wäre *immer* korrekt,
ist wegen mangelhafter Performance
aber nicht akzeptabel.
- Korrektheitskriterium
– Äquivalenz zu serieller Ausführung.
- Two-Phase Locking stellt Serialisierbarkeit sicher.

