TU München, Fakultät für Informatik
Lehrstuhl III: Datenbanksysteme
Prof. Alfons Kemper, Ph.D.

Übung zur Vorlesung *Einsatz und Realisierung von Datenbanken* im SoSe20

Maximilian {Bandle, Schüle}, Josef Schmeißer (i3erdb@in.tum.de)
http://db.in.tum.de/teaching/ss20/impldb/

Blatt Nr. 02

Hinweise Beliebige Historien können auf `https://transactions.db.in.tum.de/` auf ihre Eigenschaften hin getestet werden. Dort können Sie Ihre eigene Einschätzung überprüfen und sehen, ob Sie richtig liegen.

Hausaufgabe (wird nicht in der Übung besprochen)

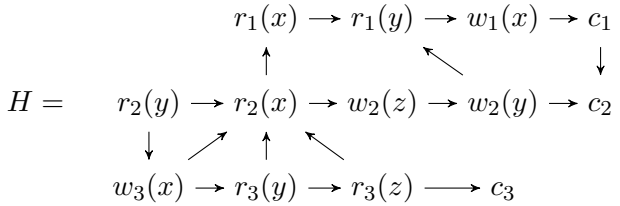
Wir definieren $r_i(A)$ als das Lesen, $w_1(A)$ als das Schreiben des Datenobjektes A durch Transaktion T_i, sowie a_i als **abort** und c_i als **commit** der Transaktion T_i. Die verzahnte Ausführung mehrerer Transaktionen bezeichnen wir als *Historie*. Geben Sie mögliche Konfliktoperationen bezüglich eines Datenobjektes A an!

Geben Sie außerdem an, wann eine Transaktion T_i von einer Transaktion T_j liest.

Wann ist eine Historie H serialisierbar (SR), rücksetzbar (RC), kaskadierendes Rücksetzen vermeidend (ACA) oder strikt (ST)?

Hausaufgabe 1

Die Historie H für die Transaktionen T_1, T_2 und T_3 sei durch das folgende Diagramm gegeben:



a) Geben Sie alle Konfliktoperationen von H an.

b) Geben Sie eine total geordnete Historie H' an (also eine „lineare“ Abfolge von Operationen), die konfliktäquivalent zu H ist.

c) Geben Sie an, welche Transaktionen voneinander lesen.

d) Geben Sie den Serialisierbarkeitsgraphen von H an.

e) Geben Sie eine serielle Historie H'' an, die konfliktäquivalent zu H ist.

Hausaufgabe 2

a) Geben Sie alle Eigenschaften an, die von der Historie erfüllt werden.

$$w_1(x), r_2(y), w_3(y), w_2(x), w_3(z), c_3, w_1(z), c_2, c_1$$

richtig	falsch	Aussage
		Serialisierbar (SR)
		Rücksetzbar (RC)
		Vermeidet kaskadierendes Zurücksetzen (ACA)
		Strikt (ST)

b) Geben Sie alle Eigenschaften an, die von der Historie erfüllt werden.

$$r_1(x), r_1(y), w_2(x), w_3(y), r_3(x), a_1, r_2(x), r_2(y), c_2, c_3$$

richtig	falsch	Aussage
		Serialisierbar (SR)
		Rücksetzbar (RC)
		Vermeidet kaskadierendes Zurücksetzen (ACA)
		Strikt (ST)

c) Gegeben die unvollständige Historie:

$$H = w_1(x), w_1(y), r_2(x), r_2(y)$$

1) Fügen Sie `commit`s in H so ein, dass die Historie RC aber nicht ACA erfüllt.
2) Fügen Sie `commit`s in das ursprüngliche H so ein, dass die Historie ACA erfüllt.

Hausaufgabe 3

a) Erläutern Sie kurz die zwei Phasen des 2PL-Protokolls.

b) Inwiefern unterscheidet sich das *strenge* 2PL?

c) Welche Eigenschaften (SR,RC,ACA,ST) haben Historien, welche vom 2PL und vom strengen 2PL zugelassen werden?

d) Wäre es beim strengen 2PL-Protokoll ausreichend, alle Schreibsperren bis zum EOT (Transaktionsende) zu halten, aber Lesesperren schon früher wieder freizugeben?

Hausaufgabe 4

SQL-92 spezifiziert mehrere Konsistenzstufen (*isolation level*) durch welche der Benutzer (bzw. die Anwendung) festlegen kann, wie "stark" eine Transaktion von anderen parallel laufenden Transaktionen isoliert werden soll.

a) Nennen und erläutern Sie kurz die Isolation Level. Geben Sie an, welche Nebenläufigkeitsprobleme mit dem jeweiligen Level vermieden werden.

b) Warum kann zwischen den Konsistenzstufen gewählt werden?

Hausaufgabe 5

Ein inhärentes Problem der sperrbasierten Synchronisationsmethoden ist das Auftreten von Verklemmungen (Deadlocks). Zur Erkennung von Verklemmungen wurde der Wartegraph eingeführt. Dabei wird eine Kante $T_i \rightarrow T$ eingefügt, wenn T_i auf die Freigabe einer Sperre durch T wartet.

Skizzieren Sie einen Ablauf von Transaktionen, bei dem ein Deadlock auftritt, der einen Zyklus mit einer Länge von mindestens 3 Kanten im Wartegraphen erzeugt.

Gruppenaufgabe 6

Gegeben die Relation „Aerzte“, die den Bereitschaftsstatus von Ärzten modelliert

Name	Vorname	...	Bereit
House	Gregory	...	ja
Green	Mark	...	nein
Brinkmann	Klaus	...	ja

sowie die folgende Transaktion in Pseudocode:

```
dienstende(arzt_name)
  select count(*) into anzahl_bereit from aerzte where bereit='ja'
  if anzahl_bereit > 1 then
    update aerzte set bereit='nein' where name=arzt_name
```

Die Transaktion soll sicherstellen, dass immer mindestens ein Arzt bereit ist.

Betrachten Sie einen Ablauf, bei dem zwei zur Zeit bereite Ärzte zum gleichen Zeitpunkt entscheiden, ihren Status auf „nein “, d.h. nicht bereit zu ändern:

T_1: execute dienstende('House')
T_2: execute dienstende('Brinkmann')

Gehen Sie beispielsweise davon aus, dass das DBMS versucht, die Transaktion jeweils abwechselnd zeilenweise abzuarbeiten.

Diskutieren Sie:

a) Was kann bei Snapshot Isolation passieren?

b) Warum ist dies bei optimistischer Synchronisation nicht möglich?

c) Wie verhält sich die Zeitstempel-basierte Synchronisation?

d) Wie verhält sich das strenge 2PL?