Lösungsvorschlag

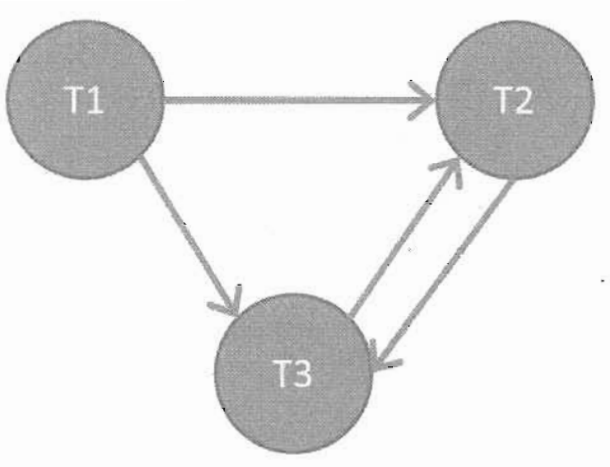
Klausur 1672

SS 2016

Aufgabe 1: Serialisierbarkeit — 12 Punkte

(pro Teilaufgabe 4 Punkte)

a)



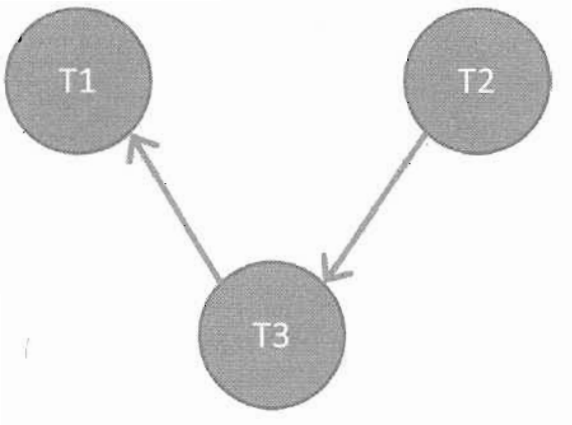
Da der Graph einen Zyklus enthält, ist die Schedule nicht serialisierbar.

Der Zyklus entsteht durch

(1) ... r3(B); ...; r2(B); ... ; w2(B): impliziert T3 vor T2

(2) ... r2(C); ... ; w2(C); r3(C); w3(C): impliziert T2 vor T3

b)



Da der Graph keinen Zyklus enthält, ist die Schedule serialisierbar. Eine äquivalente serielle Schedule ist T2, T3, T1.

c)

Im Fall a) liegt die Ursache der Nicht-Serialisierbarkeit darin, dass T2 vor T3 auf das Objekt C zugreift, auf das Objekt B aber T3 vor T2. Da T3 auf B nur lesend zugreift,

könnte man r3(B) hinter w2(B) schieben. Die entstehende Schedule ist dann serialisierbar, da keine Zyklen in dem Graphen vorliegen. Ursprüngliche Schedule:

S_1: r1(C); **r3(B);** r1(A); r2(B); w1(A); w2(B); r3(A); r2(C); w3(A); w2(C); r3(C); w3(C);

Neue Schedule:

S_1: r1(C); r1(A); r2(B); w1(A); w2(B); **r3(B);** r3(A); r2(C); w3(A); w2(C); r3(C); w3(C);

Aufgabe 2: Hierarchische Sperren (17 Punkte)

Im Folgenden arbeiten wir mit den folgenden Kürzeln:

T1, T2, T3: Transaktionen
K1-K15: Knoten (siehe Graphik in der Aufgabenstellung)

a) Wir benötigen die folgenden Sperrtypen: (5 Punkte)
   - W Schreibsperre (write lock)
   - R Lesesperre (read lock)
   - IW Intentionssperre zum Schreiben
   - IR Intentionssperre zum Lesen

   Die Kompatibilitätsmatrix sieht folgendermaßen aus:

	W	R	IW	IR
W	Nein	Nein	Nein	Nein
R	Nein	Ja	Nein	Ja
IW	Nein	Nein	Ja	Ja
IR	Nein	Ja	Ja	Ja

b) T1 möchte das Tupel 8 mit einer Schreibsperre (W) versehen.

   (T1, K1, IW)
   (T1, K2, IW)
   (T1, K4, IW)
   (T1, K8, W)

c) T2 möchte das Tupel 10 zum Lesen sperren.

   (T2, K1, IR)
   (T2, K2, IR)
   (T2, K5, IR)
   (T2, K10, R)

d) T1 möchte die Relation 5 zum Schreiben sperren.

(T1, K1, IW)
(T1, K2, IW)
(T1, K5, W) Diese Sperre wird nicht gewährt, da sie in Konflikt zu der in c) gewährten Lese-Sperre steht.

e) T3 möchte das Tupel 9 zum Schreiben sperren.

(T3, K1, IW)
(T3, K2, IW)
(T3, K4, IW)
(T3, K9, W)

f) T1 möchte die ganze Relation 6 zum Schreiben sperren.

(T1, K1, IW)
(T1, K3, IW)
(T1, K6, W)

g) T2 möchte das Tupel 12 zum Lesen sperren.

(T2, K1, IR)
(T2, K3, IR)
(T2, K6, IR) Diese Sperre wird nicht gewährt, da sie in Konflikt zu der in f) gegebenen W Sperre auf K6 steht.

(b) – g) jeweils 2 Punkte)

Aufgabe 3: Recovery (22 Punkte)

a) Die Daten der abgeschlossenen Transaktion sind nicht notwendigerweise bereits sicher in der Datenbank abgespeichert. Die Transaktion schreibt die Daten lediglich in den Systempuffer, der ein Teil des flüchtigen Hauptspeichers ist. Verantwortlich für die Übertragung der Daten aus dem Systempuffer in die Datenbank auf der externen Platte ist der Systempuffer-Manager. Er nutzt dabei Wissen über Daten und Transaktionen, insbesondere, um unnötige Aus- und Wiedereinlagerung von Seiten zu vermeiden. Damit können Daten von abgeschlossenen Transaktionen bei einem Systemcrash (=Verlust des Hauptspeichers) verloren gehen. (5 Punkte)

b) (9 Punkte)
   (1) Diese Aussage stimmt nicht. Ein UNDO-Log-Eintrag muss spätestens vor dem Speichern der Änderung in den Datenbank-Puffer auf die Platte geschrieben werden. Nur dann ist sichergestellt, dass bei einem Crash die evtl. bereits ausgelagerte Änderung wieder rückgängig gemacht werden kann.

(2) Diese Aussage stimmt nicht. Entscheidend für die Notwendigkeit eines REDO-Logs ist nicht der Zeitpunkt, wann die Transaktion die Daten schreibt, sondern wann die Daten vom Systempuffer-Manager auf die Platte geschrieben werden. Dies kann beliebig lange dauern, daher muss in diesem Fall ein REDO-Log geführt werden.
(3) Diese Aussage stimmt nicht. Beim Checkpoint werden die Daten aller abgeschlossenen Transaktionen auf die Festplatte übertragen (damit braucht beim Recory keine Transaktion wiederholt werden, die vor dem Checkpoint beendet wurde), aber die geänderten Daten der laufenden Transaktionen werden nicht übertragen.

c) T2 muss generell nicht berücksichtigt werden, da die Transaktion zum Zeitpunkt des Checkpoints bereits abgeschlossen war und damit auf der externen Platte gespeichert ist. Das gilt für alle Verfahren.

Im Falle der UNDO/REDO Recovery-Strategie muss das Recovery folgendermaßen aussehen: (2 Punkte)

- T4 und T6 sind laufende Transaktionen, sie müssen zurückgesetzt werden (Rollback), d.h. die Before-Images müssen in die Datenbank eingebracht werden.
- T1, T3 und T5 sind abgeschlossene Transaktionen, sie müssen wiederholt werden, d.h. die After Images werden in die Datenbank eingebracht.

Im Falle der UNDO/NOREDO Recovery-Strategie muss das Recovery folgendermaßen aussehen: (2 Punkte)

- T4 und T6 sind laufende Transaktionen, sie müssen zurückgesetzt werden (Rollback), d.h. die Before-Images müssen in die Datenbank eingebracht werden.
- T1, T3 und T5 sind abgeschlossene Transaktionen. Bei der NOREDO-Strategie werden zum Commit-Zeitpunkt die Daten der Transaktion in die Datenbank (also auf die Festplatte) übertragen. Damit ist für diese Transaktionen beim Recovery nichts zu tun.

Im Falle der NOUNDO/REDO Recovery-Strategie muss das Recovery folgendermaßen aussehen: (2 Punkte)

- T4 und T6 müssen nicht zurückgesetzt werden (Rollback), da alle Änderungen erst zum Commit-Zeitpunkt in den Systempuffer geschrieben werden. Da T5 und T6 diesen Punkt noch nicht erreicht haben, können auch keine Daten dieser Transaktionen im Systempuffer stehen und damit auch noch nicht in die externe Datenbank übertragen worden sein.
- T1, T3 und T5 sind abgeschlossene Transaktionen, sie müssen wiederholt werden, d.h. die After Images werden in die Datenbank eingebracht.

Im Falle der NOUNDO/NOREDO Recovery-Strategie muss das Recovery folgendermaßen aussehen: (2 Punkte)

- T4 und T6 müssen nicht zurückgesetzt werden (Rollback), da alle Änderungen erst zum Commit-Zeitpunkt in den Systempuffer geschrieben

werden. Da T4 und T6 diesen Punkt noch nicht erreicht haben, können auch keine Daten dieser Transaktionen im Systempuffer stehen und damit auch noch nicht in die externe Datenbank übertragen worden sein.

- T1, T3 und T5 sind abgeschlossene Transaktionen. Bei der NOREDO-Strategie werden zum Commit-Zeitpunkt die Daten der Transaktion in die Datenbank (also auf die Festplatte) übertragen. Damit ist für diese Transaktionen beim Recovery nichts zu tun.

Aufgabe 4: Sperren (21 Punkte)

a) (2 Punkte pro Teilaufgabe)

1. T_2 darf die Sperre auf B nicht setzen, da B noch von T_3 gesperrt ist. Damit kann T_2 auch nicht die folgenden Operationen ausführen ($L_2(C)$, $U_2(A)$). Diese müssen zumindest verzögert werden.

2. T_1 darf die Sperre nicht auf E setzen, da E noch von T_3 gesperrt ist. Damit kann T_1 auch nicht die Sperre auf C freigeben.

3. T_2 darf die Sperre auf C nicht setzen, da C bereits von T_3 gesperrt ist. Ebenso darf T_3 keine Sperre auf A setzen, da A bereits von T_2 gesperrt ist.
   T_2 darf keine Sperren auf B setzen, da B bereits von T_3 gesperrt ist.

b) (5 Punkte pro Teilaufgabe)

1. $L_2(A)$; $L_3(B)$; $L_3(D)$;
   ~~$L_2(B)$~~; Diese Sperranfrage führt dazu, dass T_2 geblockt wird und zunächst nicht weitermachen kann. Die Sperranfrage wird also nicht ausgeführt
   $U_3(D)$;
   ~~$L_2(C)$~~; Da T_2 geblockt ist, wird diese Sperranforderung nicht ausgeführt
   $L_1(D)$;
   ~~$U_2(A)$~~; Da T_2 geblockt ist, wird diese Freigabe nicht ausgeführt
   $U_1(D)$;
   $U_3(B)$; Jetzt ist die Sperre auf D freigegeben, damit kann jetzt T_2 zum Zuge kommen. Somit werden die verzögerten Operationen jetzt ausgeführt
   $L_2(B)$; $L_2(C)$; $U_2(A)$;
   $U_2(C)$; $U_2(B)$.

   Damit kann die Folge korrekt abgearbeitet werden.

2. $L_2(A)$; $L_2(B)$; $L_3(C)$; $U_2(B)$; $L_1(D)$; $U_2(A)$; $L_3(E)$; $L_1(A)$; $U_3(C)$; $L_1(C)$
   ~~$L_1(E)$~~; Diese Sperranfrage führt dazu, dass T_1 geblockt wird und zunächst nicht weitermachen kann. Die Sperranfrage wird also nicht ausgeführt
   ~~$U_1(C)$~~; Da T_1 geblockt ist, wird diese Freigabe nicht ausgeführt.
   $U_3(E)$; Jetzt ist die Sperre auf C freigegeben, damit kann jetzt T_1 zum Zuge kommen. Somit werden alle nicht ausgeführten Operationen jetzt ausgeführt.
   $L_1(E)$; $U_1(C)$;
   $U_1(D)$; $U_1(A)$; $U_1(E)$.

Damit kann die Folge korrekt abgearbeitet werden

3. $L_3(B)$; $L_2(A)$; $L_3(C)$; $L_1(D)$;
   ~~$L_2(B)$~~; Diese Sperranfrage führt dazu, dass T_2 geblockt wird und zunächst nicht weitermachen kann. Die Sperranfrage wird also nicht ausgeführt
   $L_1(E)$; $U_1(D)$;
   ~~$L_2(C)$~~; Da T_2 geblockt ist, wird diese Sperranforderung nicht ausgeführt.
   ~~$L_3(A)$~~; Diese Sperranfrage führt dazu, dass T_3 geblockt wird und zunächst nicht weitermachen kann. Die Sperranfrage wird also nicht ausgeführt
   ~~$U_3(B)$~~; ~~$U_2(A)$~~; T_2 und T_3 sind geblockt, die Operationen werden also zunächst nicht ausgeführt
   $U_1(E)$;
   ~~$U_3(C)$; $U_2(B)$; $U_2(C)$; $U_3(A)$~~.
   T_2 und T_3 sind immer noch geblockt, die Operationen werden also nicht ausgeführt

   Damit können T_2 und T_3 nicht ausgeführt werden, da sie sich gegenseitig blockieren. Wir haben es hier also mit einem Deadlock zu tun.

Aufgabe 5: Transaktionsfehler (28 Punkte)

a) Ein paralleles System von Transaktionen ist dann korrekt synchronisiert, wenn es serialisierbar ist, d.h. wenn es mindestens eine (gedachte) serielle Ausführung derselben Transaktionen gibt, die
   (1) denselben Datenbankzustand und
   (2) dieselben Ausgabedaten der Transaktionen liefern.

   Wichtig bei dieser Definition ist, dass es irgendeine serielle Ausführung mit gleicher Wirkung geben muss; die Reihenfolge der Transaktionen ist unwichtig. (4 Punkte)

b) (12 Punkte)
   (1) Lost Update tritt auf, wenn zwei Transaktionen beide den alten Wert eines Objektes lesen und daraufhin einen neuen Wert schreiben. Die Transaktion, die als letzte schreibt, gewinnt, der Wert der anderen Transaktion geht verloren.

   Beispiel:
   - Objekt a habe den Wert 100.
   - T1 liest Objekt a (Wert 100)
   - T2 liest Objekt a (Wert 100)
   - T2 addiert 100 zu a und schreibt den Wert 200 zurück
   - T1 subtrahiert 20 von a und schreibt den Wert 80 zurück
   - Damit ist die Änderung von T2 verloren gegangen.

   (2) Inkonsistente Sicht tritt auf, wenn zwei Transaktionen auf (mindestens) zwei Objekten gleichzeitig arbeiten und die eine Transaktion den Zustand des einen Objektes vor der Änderung durch die andere Transaktion sieht, den

Zustand des anderen Objektes aber nach der Änderung der anderen Transaktion. Wenn auf dieser Basis Änderungen durchgeführt werden, kann es zu einer inkonsistenten Datenbank führen.

Beispiel:

- Objekt a und b haben beide den Wert 100.
- T1 liest Objekt a (Wert 100) und addiert 100, schreibt also 200 zurück
- T2 liest Objekt b (Wert 100) und addiert 10%, schreibt also 110 zurück
- T1 liest Objekt b (Wert 110) und addiert 100, schreibt also 210 zurück
- T2 liest Objekt a (Wert 200) und addiert 10%, schreibt also 220 zurück
- Objekt a hat jetzt den Wert 220, Objekt b den Wert 210.
- Wir haben es damit mit einer inkonsistenten Datenbank zu tun – in einer konsistenten Datenbank hätten beide Objekte nach Anwendung der gleichen Operationen weiterhin den gleichen Wert. Ursache ist, dass sowohl T1 als auch T2 zunächst den alten Wert eines Objektes gesehen haben und als Basis für ihre Änderungsoperation genommen haben und anschließend beim anderen Objekt jeweils den Zustand nach der ersten Änderung durch die jeweils andere Transaktion.

Man beachte den Unterschied zum Lost Update: Da ging es um ein und dasselbe Objekt, hier geht es um unterschiedliche Objekte.

(3) Phantome können beim wiederholten Lesen von gleichen Datenmengen auftauchen: Beim ersten Lesen ist das Objekt noch nicht vorhanden, durch zwischenzeitliches Einfügen durch eine andere Transaktion ist es dann beim zweiten Lesen vorhanden.

Beispiel:

- T1 liest alle Angestelltensätze
- T2 fügt parallel einen neuen Angestellten in die Datenbank ein
- T2 liest wiederum alle Angestelltensätze (oder eine Teilmenge davon) und findet dabei den neuen Angestelltensatz, der vorher nicht sichtbar war.

c) (12 Punkte)

(1) Lost Update

Das normale Sperrverfahren, das einer Transaktion den Zugriff auf ein Objekt erst nach dem Erwerb einer Sperre erlaubt, verhindert bereits das Auftreten von Lost Updates.

- Objekt a habe den Wert 100.
- T1 erhält eine Sperre für a
- T1 liest Objekt a (Wert 100)
- T2 fordert eine Sperre für a, wird aber zunächst auf die Warteliste gesetzt. Damit werden die Aktionen von T2 verzögert, und T1 kann zunächst weitermachen:
- T1 subtrahiert 20 von a und schreibt den Wert 80 zurück
- T1 gibt die Sperre auf a frei
- T2 erhält nun die Sperre auf a

- T2 liest Objekt a (Wert 80)
- T2 addiert 100 zu a und schreibt den Wert 180 zurück
- T2 gibt die Sperre zurück
- Damit ist die Änderung von T2 diesmal korrekt in das Ergebnis eingegangen.

(2) Inkonsistente Sicht, inkonsistente Datenbank
Hier geht es um die korrekte Synchronisation von Zugriffen auf mehrere Objekte. Dazu ist das Zwei-Phase-Sperrprotokoll (2PL) eingeführt worden, das verhindert, dass Transaktionen wechselseitig auf unterschiedliche Objekte zugreifen. Im konkreten Beispiel führt dies zu einem Deadlock:

Beispiel:
- Objekt a und b haben beide den Wert 100.
- T1 erhält die Sperre auf a, liest Objekt a (Wert 100) und addiert 100, schreibt also 200 zurück. Aufgrund des 2PL kann T1 die Sperre noch nicht zurückgeben (will ja noch b bearbeiten)
- T2 erhält die Sperre auf b, liest Objekt b (Wert 100) und addiert 10%, schreibt also 110 zurück. Aufgrund des 2PL kann T2 die Sperre noch nicht zurückgeben (will ja noch a bearbeiten)
- T1 versucht, die Sperre für das Objekt b zu bekommen, muss aber auf die Freigabe von T2 warten
- T2 versucht, die Sperre für das Objekt a zu bekommen, muss aber auf die Freigabe von T1 warten
- Damit warten beide Transaktionen aufeinander, sind also in einem Deadlock. Nach Auflösung des Deadlocks kann eine Transaktion weitermachen, die andere muss neu starten. Damit werden die Transaktionen hintereinander ausgeführt, und der Fehler tritt nicht auf.

(3) Phantome
Zur Vermeidung von Phantomen muss verhindert werden, dass andere Transaktionen in eine Objektmenge neue Objekte einfügen können. Dies kann durch eine Sperre auf größeren Dateneinheiten geschehen, etwa einer Sperre auf der ganzen Relation Angestellte. Dann wäre ein Einfügen nicht möglich gewesen, und beide Lesevorgänge hätten die gleichen Sätze zurückgeliefert.