ETH Eidgenössische Technische Hochschule Zürich
Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Swiss Federal Institute of Technology Zurich

Dr. G. Alonso *C. Hagen, B. Kemme*

1. SCHNELLÜBUNG zu AR I

29. 5. 1997

Dauer: 45 Minuten

DON'T PANIC!!
Diese Prüfung enthält viele Fragen. Wir erwarten jedoch nicht, dass alle Aufgabe gelöst werden

---

Name, Vorname (Druckschrift) .....

ETH-Nummer / Abteilung .....

Name (Unterschrift) .....

Aufgabe	Max. Punktzahl	Erreichte Punktzahl
1	15	
2	20	
3	15	
4	15	
5	20	
6	15	
Summe:	100	

Note:

Aufgabe 1: Analyse eine Schedules

Gegeben seien die Transaktionen T_1, T_2, und T_3 und der (unvollständige) Schedule S:

$$\begin{aligned} T_1 &= < w_1^1(y)\ w_2^1(x)\ r_3^1(z) > \\ T_2 &= < r_1^2(z)\ w_2^2(x) > \\ T_3 &= < r_1^3(x)\ w_2^3(y) > \\ S &= < r_1^2(z)\ w_1^1(y)\ w_2^1(x)\ w_2^2(x)\ r_1^3(x)\ r_3^1(z)\ w_2^3(y) > \end{aligned}$$

a) Beantworten Sie folgende Fragen.

1. Ist S konfliktserialisierbar? Begründen Sie Ihre Entscheidung. (3P)
   Lösung:
   Die Abhängigkeitsrelation lautet:

   $$\rightarrow_S = \{(w_1^1(y), w_2^3(y)), (w_2^1(x), w_2^2(x)), (w_2^1(x), r_1^3(x)), (w_2^2(x), r_1^3(x))\}$$

   Der Schedule ist konfliktserialisierbar, da der dazugehörige Abhängigkeitsgraph keinen Zyklus enthält. Ein äquivalenter serieller Schedule wäre $T_1\ T_2\ T_3$.
2. Kann S durch das 2-Phasen-Sperrprotokoll (2PL) entstanden sein? Begründen Sie Ihre Entscheidung. Geben Sie gegebenenfalls den erweiterten Schedule an, der die Sperr- und Freigabeoperationen enthält. (3P)
   Ja, der Schedule kann durch 2PL entstanden sein. In dem erweiterten Schedule

   $$\begin{aligned} S = < & \underline{rl_1^2(z)}\ r_1^2(z)\ \underline{wl_1^1(y)}\ \underline{wl_2^1(x)}\ \underline{rl_3^1(z)} w_1^1(y)\ w_2^1(x)\ \underline{wu_2^1(x)} \\ & \underline{wl_2^2(x)}\ w_2^2(x)\ \underline{wu_2^2(x)}\ \underline{rl_1^3(x)}\ r_1^3(x) r_3^1(z)\ \underline{wu_1^1(y)}\ \underline{ru_3^1(z)} \\ & \underline{wl_2^3(y)}\ w_2^3(y)\ \underline{unlock(rest)} > \end{aligned}$$

   werden alle Sperren angefordert, bevor eine Sperre freigegeben wird. Indem Transaktion T_1 die Lesesperre auf z vorzeitig anfordert, kann 2PL eingehalten werden.
3. Kann S durch striktes 2PL entstanden sein? Begründen Sie Ihre Entscheidung. Geben Sie gegebenenfalls den erweiterten Schedule an, der die Sperr- und Freigabeoperationen enthält. (3P)
   Nein, der Schedule kann nicht durch striktes 2PL entstanden sein, denn T_1 muss die Sperre für x vor Ende der Transaktion aufheben.

b) Fügen Sie die fehlenden Commit-Operationen so in S ein, dass er

1. rücksetzbar, aber nicht kaskadenfrei ist. (3P)

   $$S =< Sc_1c_2c_3 >$$

2. kaskadenfrei ist. (3P)

   $$S =< r_1^2(z)\ w_1^1(y)\ w_2^1(x)\ w_2^2(x)\ c_2\ r_1^3(x)\ r_3^1(z)\ c_1\ w_2^3(y)c_3 >$$

Aufgabe 2: Beispielschedules

Geben Sie für jede der folgenden Anforderungen einen Beispielschedule an oder kennzeichnen Sie die Anforderung als nicht erfüllbar. Sie müssen dabei die einzelnen Operationen einer Transaktion nicht durchnummerieren. Es genügt pro Unterpunkt also eine Notation wie z.B. $S = r_1(x)w_2(x)r_2(y)c_2c_1$, wobei der Index die Transaktion identifiziert. Sind Sperrverfahren involviert, geben Sie auch die Sperr- und Freigabeoperationen an.

1. Der Schedule ist strikt, aber nicht serialisierbar. (2P)

   $$S =< r_1(x)\ w_2(x)\ c_2\ r_1(x)\ c_1 >$$

2. Der Schedule ist serialisierbar, aber nicht strikt. (2P)

   $$S =< w_1(x)\ w_2(x)\ c_1\ c_2 >$$

3. Der Schedule ist strikt, aber nicht rücksetzbar. (2P)
   nicht möglich, da $ST \subset RC$

4. Der Schedule ist 2PL, aber nicht serialisierbar. (2P)
   nicht möglich, da alle 2PL Schedules serialisierbar sind

5. Der Schedule ist serialisierbar, aber nicht 2PL. (2P)

   $$S =< wl_1(x)\ w_1(x)\ wu_1(x)\ wl_1(y)\ w_1(y)\ wu_1(y)\ c_1 >$$

6. Der Schedule ist 2PL, aber nicht strikt. (2P)

   $$S =< wl_1(x)\, wl_1(y) w_1(x)\, wu_1(x)\, wl_2(x)\, w_2(x)\, wu_2(x)\, c_2 w_1(y)\, wu_1(y)\, c_1 >$$

7. Der Schedule ist 2PL, aber hat *Lost Updates*. (2P)
   nicht möglich, da all 2PL Schedules serialisierbar sind

8. Der Schedule kann durch ein Cursor-Stability Sperrverfahren entstanden sein, ist aber nicht serialisierbar. (2P)
   Bei Cursor-Stability Sperrverfahren werden Lesesperren freigegeben, sobald der Cursor auf das nächste Datenobjekt wandert. Will man ein gelesenes Objekt schreiben, muss sofort eine Schreibsperre gesetzt werden. Schreibsperren werden bis EOT gehalten. Deshalb muss ein nicht serialisierbarer Schedule auf Leseoperationen arbeiten ($sc = set-cursor, dc = drop-cursor$)

   $$S =< sc_1(x)\, r_1(x)\, dc_1(x)\, wl_2(x)\, w_2(x)\, wu_2(x)\, c_2\, sc_1(x)\, r_1(x)\, dc_1(x)\, c_1 >$$

9. Geben Sie ein Beispiel für ein Phantom. Benützen Sie dazu einfache *Select*- und *Insert*-Statements. (4P)
   T1: select halter from auto where farbe = 'rot'
   T2: insert into auto values ("Käfer", "rot", "Alonso")
   T2: commit
   T1: select halter from auto where farbe = 'rot'

Aufgabe 3: Ein einfacher Beweis

Gegeben sei der folgende Schedule:

$$S =< w_1(x)\ r_2(x)\ c_2\ c_1 >$$

Dieser Schedule kann durch 2PL entstehen. S ist nicht ordnungserhaltend: Transaktion 2 wird nach Transaktion 1 serialisiert, obwohl das Commit von T_1 nach dem von T_2 stattfindet.

Zeigen Sie: Durch striktes 2PL entstandene Schedules sind immer ordnungserhaltend. Dabei heisst strikt, dass sowohl Lese- als auch Schreibsperren bis zum Ende der Transaktion gehalten werden. (15P)

Lösung:

Zu zeigen: Jeder durch S2PL entstandene Schedule ist ordnungserhaltend. Sei S ein Schedule, der durch S2PL enstanden ist. Für je zwei konfligierende Operationen $o_1(x), o_2(x)$ zweier Transaktionen mit Serialisierungsfolge T_1, T_2 gilt also:

$$o_1(x) < c_1 < o_2(x) \tag{1}$$

Annahme: S sei nicht ordnungserhaltend. Das heisst: Es gibt zwei Transaktionen T_1, T_2 in S mit Commit-Reihenfolge T_2, T_1 ...

$$c_2 < c_1 \tag{2}$$

... und Serialisierungsreihenfolge T_1, T_2 – es muss also ein Datenobjekt x und zwei konfligierende Operationen $o_1(x), o_2(x)$ geben mit

$$o_1(x) < o_2(x). \tag{3}$$

Aus 2 und 3 folgt

$$o_1(x) < o_2(x) < c_2 < c_1, \tag{4}$$

da für jede Transaktion T_i und jede ihrer Operationen o_i gilt: $o_i(y) < c_i$. Nun haben wir einen Widerpruch zwischen 4 und 1. (Einerseits $o_2(x) < c_1$, andererseits $c_1 < o_2(x)$.) Damit kann unser nicht ordnungserhaltender Schedule S nicht durch S2PL entstanden sein. Dies beweist den Satz.

Aufgabe 4: Sperren in der Praxis

In Version 7.3 des Oracle-DBMS wurde ein neues Sperrprotokoll *(Snapshot-Isolation)* eingeführt. Jede Transaktion arbeitet auf einem *snapshot* der Datenbank, der den Zustand aller Daten zum Zeitpunkt des Transaktionsbeginns reflektiert. Leseoperationen lesen also Werte so, wie sie zu Beginn der Transaktion vorlagen. Um *lost updates* zu vermeiden, wird für Schreiboperationen die folgende Regelung eingeführt: Wenn eine Transaktion T_1 ein Objekt schreiben will, dass von einer anderen Transaktion T_2 geschrieben wurde, so wird T_1 abgebrochen, falls das Commit von T_2 *nach* dem Beginn von T_1 stattfand.

Beispiel: $S =< r_1(x)\ w_2(y)\ c_2\ w_1(y) >$
Transaktion 1 wird hier wegen der Schreiboperation w_1 abgebrochen.

Zeigen Sie, dass dieses Protokoll nichtserialisierbare Schedules erzeugen kann, indem Sie einen Beispielschedule konstruieren. (15P)

(Hinweise: (a) Benützen Sie KEINE reinen Lesetransaktionen. (b) Es genügen zwei Transaktionen mit je zwei Operationen, wobei jeweils eine Operation eine Schreiboperation ist.)

Konflikte müssen zwischen Lese- und Schreiboperationen hergestellt werden, da das Protokoll Konflikte zwischen Schreibern erkennt. Damit keine Serialisierung vorgenommen werden kann, müssen zwei Konflikte hergestellt werden, um sowohl Abhängigkeit T1 vor T2, als auch Abhängigkeit T2 vor T1 zu erhalten.

$$
\begin{aligned}
T_1 &= < r_1(x)\ w_1(y) > \\
T_2 &= < r_2(y)\ w_2(x) >
\end{aligned}
$$

$$
S = < r_1(x)\ r_2(y)\ w_1(y)\ w_2(x)\ c_1\ c_2 >
$$

Aufgabe 5: Sperrprotokolle und optimistische Verfahren

1. Was ist der Unterschied zwischen *Tree Locking* und *DAG Locking*? (6P)

   Sowohl Tree Locking als auch DAG Locking sind Multigranularity-*Sperrverfahren. Beim Tree-Locking hat jedoch jedes Objekt einer feineren Granularitätsstufe genau ein Elternobjekt in der nächsthöheren Granularitätsstufe, während es beim DAG-Locking mehrere Eltern geben kann. Der Graph der Verfeinerungsbeziehung bildet im einen Fall also einen Baum, im zweiten Fall einen gerichteten, azyklischen Graph.*

   Unterschiede ergeben sich dadurch, dass beim Tree-Locking genau ein Pfad von einem Objekt zur Wurzel existiert, beim DAG-Locking aber mehrere solche Pfade vorhanden sein können. Damit Transaktionen korrekt isoliert werden, müssen unterschiedliche Sperralgorithmen verwendet werden:

   Um beim Tree-Locking auf ein Objekt x (lesend bzw. schreibend) zugreifen zu können, muss das Objekt selbst oder einer seiner Vorfahren (mit einer R- bzw. W-Sperre) gesperrt werden. Beim DAG-Locking muss auf jedem Pfad vom Objekt zur Wurzel *eine (R- oder W-) Sperre existieren.*

   Um eine W-Sperre auf einem Objekt x zu erhalten, müssen IW- oder RIW-Sperren auf sämtlichen *Vorfahren von x gesetzt werden. Dies gilt sowohl für Tree- als auch für DAG-Locking. Beim DAG-Locking bedeutet dies jedoch, dass saemtliche Pfade zur Wurzel gesperrt werden. Analoges gilt für R-Sperren.*

2. Was ist *Lock Escalation*? (6P)

   Lock Escalation wird von einigen Datenbanksystemen angewandt, wenn eine Transaktion sehr viele Sperren feiner Granularität hält. In diesem Fall wird eine Sperre höherer Granularität erworben und die untergeordneten Sperren können freigegeben werden. Hat eine Transaktion z.B. die Hälfte aller Tupel einer Relation gesperrt, so wird eine Sperre auf die gesamte Relation gesetzt. Lock Escalation veringert den Verwaltungsaufwand und Speicherbedarf, ist aber Deadlock-anfällig.

3. Was muss getan werden, um einen *Timestamp Order* Algorithmus strikt zu machen? (8P)

   Ein TO-Scheduler verwaltet für jedes Objekt x einen Zähler w-in-transit(x), *der die Schreiboperationen auf x zählt, die an den Datenmanager geschickt, aber noch nicht beendet wurden. Dieser Zähler darf nicht grösser als 1 werden und verhindert damit, daß mehrere writes gleichzeitig ausgeführt werden. Striktheit erreicht man, indem man das zurücksetzen des Zählers bis zum commit der Transaktion, die zuletzt geschrieben hat, verzögert.*

Aufgabe 6: Recovery

1. In dem DBMS "Exam 1" kann eine Seite zur Festplatte geschrieben werden, unabhängig davon, ob alle Transaktionen, die die Seite modifiziert haben, beendet wurden oder noch laufen. Welche Aktivitäten sind bei der Recovery (mindestens) notwendig? Begünden Sie Ihre Aussage. (7.5 P)

   Das DBMS benützt die **STEAL**-*Strategie, d.h. es können Modifikationen von nicht committeten Transaktionen in die Datenbank auf die Festplatte eingebracht werden. Deshalb müssen bei einem Rechnerabsturz die Modifikationen von Transaktionen, die zum Zeitpunkt des Absturzes aktiv waren, rückgängig gemacht werden (***UNDO***). Damit dies möglich ist, muss während des normalen Betriebes das* **Before-Image** *einer Seite im Log gespeichert werden, bevor die modifizierte Seite auf die Festplatte geschrieben wird.*

   Darüber, ob **REDO**, *d.h. das Wiederherstellen von Modifikationen von Transaktionen, die zum Zeitpunkt des Absturzes bereits committet waren, nötig ist, kann keine Aussage gemacht werden, da keine Information vorliegt, was bei EOT einer Transaktion geschieht.*

2. In dem DBSM "Exam 2" werden modifizierte Seiten nicht auf die Festplatte zurückgeschrieben, wenn eine Transaktion committet. Welche Aktivitäten sind bei der Recovery (mindestens) notwendig? Begründen Sie Ihre Aussage (7.5 P)

   Das DBMS benützt die **No Flush**-*Strategie, um Seiten in die Datenbank auf Fesplatte zu schreiben, d.h. es können Transaktionen committen, ohne dass ihre Änderungen in die Datenbank auf Festplatte eingebracht worden sind. Bei einem Rechnerabsturz müssen diese Modifikationen nachträglich durchgeführt werden (***REDO***). Damit dies möglich ist, muss während des normalen Betriebes das* **After-Image** *einer Seite im Log gespeichert werden, bevor die modifizierende Transaktionen committet.*

   Darüber, ob **UNDO**, *d.h. das Rückgängigmachen von Transaktionen, die zum Zeitpunkt des Absturzes gerade aktiv waren, nötig ist, kann keine Aussage gemacht werden, da keine Information vorliegt, ob uncommittete Daten auf die Platte geschrieben werden.*