



7. Übungszettel (Abgabe 09.06.2008)

Einführung in Datenbanksysteme
Datenbanken für die Bioinformatik

Heinz Schweppe, Jürgen Broß, Manuel Scholz

Übungsaufgaben

1. Aufgabe (Konfliktserialisierbarkeit)

Gegeben sind die drei Schedules für die beiden Transaktionen T1 und T2. Überprüfen Sie, ob diese serialisierbar sind. Zeichnen Sie dazu jeweils den entsprechenden Konfliktgraphen auf.

- a) $r1(a), w1(a), r1(b), w1(b), r2(b), w2(b), r2(c), w2(c)$
- b) $r1(a), r2(b), w1(a), w2(b), r1(b), r2(c), w1(b), w2(c)$
- c) $r1(a), r2(b), w1(a), r1(b), w2(b), r2(c), w1(b), w2(c)$

a) JA

Konfliktpaare: $r1(b) \rightarrow w2(b)$, $w1(b) \rightarrow r2(b)$, $w1(b) \rightarrow w2(b)$

Konfliktgraph: $T1 \rightarrow T2$

b) JA

Konfliktpaare: $r2(b) \rightarrow w1(b)$, $w2(b) \rightarrow r1(b)$, $w2(b) \rightarrow w1(b)$

Konfliktgraph: $T2 \rightarrow T1$

c) Nein

Konfliktpaare: $r2(b) \rightarrow w1(b)$, $r1(b) \rightarrow w2(b)$, $w2(b) \rightarrow w1(b)$

Konfliktgraph: $T1 \leftrightarrow T2$ (Zyklus)

2. Aufgabe (Historien)

Gegeben sind die beiden Transaktionen T1 und T2, sowie der Konfliktgraph C(S).

$T1 = r1[x], r1[y], w1[y], r1[z], w1[x], c1$

$T2 = r2[y], r2[z], w2[z], r2[x], w2[x], r2[z], c2$

$C(S) = \{(r2[y], w1[y]), (r1[x], w2[x]), (w1[x], w2[x]), (r2[x], w1[x])\}$

- a) Geben Sie für die Transaktionen TA1 und TA2 eine Historie an, die den Konfliktgraphen C(S) besitzt.
- b) Wie viel Historien gibt es, die diese Bedingung erfüllen?

a) Ein möglicher Schedule ist z.B.

$r1(x), r2(y), r1(y), r2(z), w1(y), w2(z), r1(z), r2(x), w1(x), w2(x), c1, r2(z), c2$

b) Die Anzahl der Historien, welche dem Konfliktgraphen $C(S)$ entsprechen lässt sich mit Hilfe der Topologischen Sortierung finden. Die Anordnung der nicht in Konflikt stehenden Operationen ist für die Äquivalenz von Historien irrelevant. Sie können also getauscht werden, solange die Reihenfolge der Operationen innerhalb einer Transaktion invariant bleibt.
Wendet man diesen Algorithmus auf das Problem an, so kommt man auf 65 mögliche Varianten.

3. Aufgabe (Isolationslevel)

Gegeben sind folgende Relationen:

Produkt (Hersteller, Modell, Typ)

PC (Modell, Geschwindigkeit, RAM, Festplatte, Preis)

Mit Hilfe von embedded SQL (z.B. in JAVA) wurden nun kleine Programme geschrieben, die folgende Funktionen umgesetzt haben:

1. Gegeben sei die Geschwindigkeit und der RAM eines PC's (Funktionsargumente). Ausgegeben werden sollen das Modell, die Geschwindigkeit und der RAM der PC's mit diesen Werten.
2. Für eine gegebene Modell-Nr. sollen die Tupel des Modells aus den Relationen Produkt und PC gelöscht werden.
3. Für eine gegebene Modell-Nr. soll der Preis um 100 Euro verringert werden.
4. Gegeben sind der Hersteller, die Modell-Nr., die Geschwindigkeit, der RAM, die Festplatte und der Preis. Es soll überprüft werden, ob ein Produkt mit dieser Modell-Nr. existiert. Ist dies der Fall, so soll eine Fehlermeldung ausgegeben werden. Existiert das Modell nicht, so sollen die Informationen in die Produkt und die PC Tabelle eingetragen werden.

Angenommen es wird eines dieser 4 Programme als Transaktion T ausgeführt, während dieses oder andere (der 4) Programme gleichzeitig ausgeführt werden.

Welches Verhalten kann bei der Ausführung der Transaktion T festgestellt werden, wenn alle Transaktion mit Isolationslevel „READ UNCOMMITTED“ arbeiten, das nicht möglich wäre wenn alle Transaktionen mit Isolationslevel „SERIALIZABLE“ arbeiten würden.

Betrachten sie alle Fälle für jedes der vier Programme einzeln.

T1||T2 evtl. Ausgabe eines Modells, das bereits gelöscht wurde.

T1||T3 evtl. Ausgabe eines falschen Preises

T1||T4 evtl. wird Modell noch nicht angezeigt.

T2||T1 OK

T2||T3 OK

T2||T4 Modell evtl. noch nicht eingefügt

T3||T1 OK

T3||T2 Modell evtl. nicht mehr vorhanden

T3||T4 Modell evtl. noch nicht vorhanden

T4||T1 OK

T4||T2 Modell evtl. noch nicht gelöscht

T4||T3 OK