Gradiance Online Accelerated Learning

Submission number: 155783
Submission certificate: FE562646
Submission time: 2014-03-03 01:26:59 PST (GMT - 8:00)

Number of questions: 7
Positive points per question: 3.0
Negative points per question: 1.0
Your score: 21

1. Which of the following schedules is serializable?

a) $r_1(X),w_1(Y),r_2(X),r_2(Y),w_1(X),c_1,w_2(Y),w_2(Z),c_2$

b) $r_1(Y),r_2(Y),w_1(Z),w_1(Y),w_1(X),c_1,r_2(Z),w_2(X),c_2$

c) $r_2(X),r_1(X),w_2(Y),w_2(X),w_2(Z),c_2,r_1(Y),w_1(Z),c_1$

d) $r_2(Z),w_2(X),r_1(Z),r_2(Y),r_1(X),w_2(Y),c_2,w_1(Z),c_1$

Answer submitted: **d)**

You have answered the question correctly.

Question Explanation:

A schedule is serializable if it is equivalent to some serial schedule. The usual test for serializability is to check for *conflict serializability*. We build a precedence graph in which transaction T precedes transaction S if T accesses some value before S, and at least one of those accesses is a write. The schedule is (conflict) serializable if and only if this graph has no cycles. Thus, we can check for serializability by looking at every data element accessed in the schedule, and seeing if its accesses imply any precedences among transactions. For example, one of the correct answers is:

$r_2(X),r_1(X),w_2(Y),r_2(Z),r_1(Y),w_2(Z),c_2,w_1(X),c_1$

If we look at accesses to X, we see that T_1 writes X after T_2 reads X. Thus, T_2 must precede T_1. Y is written by T_2 before T_1 reads Y, so again, T_2 must precede T_1. Finally, Z is accessed only by T_2, so it imposes no constraints. The serial order T_2, T_1 is thus equivalent to the given schedule, and the schedule is serializable.

2. The relation R(x) consists of a set of integers --- that is, one-component tuples with an integer component. Alice's transaction is a query:

```
SELECT SUM(x) FROM R;
COMMIT;
```

Betty's transaction is a sequence of inserts:

```
INSERT INTO R VALUES(10);
INSERT INTO R VALUES(20);
INSERT INTO R VALUES(30);
COMMIT;
```

Carol's transaction is a sequence of deletes:

```
DELETE FROM R WHERE x=30;
DELETE FROM R WHERE x=20;
COMMIT;
```

Before any of these transactions execute, the sum of the integers in R is 1000, and none of these integers are 10, 20, or 30. Alice's, Betty's, and Carol's transactions run at about the same time. Which sums could be returned by Alice's transaction if all three transactions run under isolation level `READ UNCOMMITTED`, but not if they run under isolation level `SERIALIZABLE`? Identify one of those sums from the list below.

a) 1020

b) 1000

c) 1010

d) 1040

Answer submitted: **d)**

You have answered the question correctly.

Question Explanation:

First, if the transactions all run SERIALIZABLE, the effect must be something that could happen if the three transactions each ran completely, in some order. The possibilities are: ABC, CAB, or ACB: 1000; BAC or CBA: 1060; BCA: 1010.

If each runs under READ UNCOMMITTED, there are many more things that can happen. In fact, the only constraints are caused by the order that Betty and Carol execute their steps. That is, Betty inserts 30 after 20, which is inserted after 10, and Carol deletes 30 before 20. Alice can see R at any time and read whatever integers are in it at the time. We need to enumerate all the possible sequences of events up to the time Alice reads:

1. Carol does nothing or has executed her deletes before Betty inserted the corresponding integers. Then Alice could see 1000, 1010, 1030, or 1060, depending on how far Betty has progressed.
2. Carol has deleted 30 after Betty inserted 30, but Carol has not deleted 20 yet. Then Alice sees 1030.
3. Carol deleted 30 before Betty inserted 30, but deleted 20 after Betty inserted 20. Then Alice sees 1010 or 1040, depending on whether or not Betty has yet inserted 30.
4. Carol deleted 20 and 30 after Betty inserted them. Then Alice sees 1010.

The conclusion is that Alice can get a sum of 1000, 1010, 1030, 1040, or 1060. However, we observed that under SERIALIZABLE, sums of 1000, 1010, or 1060 can be obtained. Thus only 1030 and 1040 are in the difference.

---

3. The relation R(x) consists of a set of integers --- that is, one-component tuples with an integer component. Alice's transaction is a query:

```
SELECT SUM(x) FROM R;
COMMIT;
```

Betty's transaction is a sequence of inserts:

```
INSERT INTO R VALUES(10);
INSERT INTO R VALUES(20);
INSERT INTO R VALUES(30);
```

```
COMMIT;
```

Carol's transaction is a sequence of deletes:

```
DELETE FROM R WHERE x=30;
DELETE FROM R WHERE x=20;
COMMIT;
```

Before any of these transactions execute, the sum of the integers in R is 1000, and none of these integers are 10, 20, or 30. If Alice's, Betty's, and Carol's transactions run at about the same time, and each runs under isolation level `READ COMMITTED`, which sums could be produced by Alice's transaction? Identify one of those sums from the list below.

a) 990

b) 1000

c) 1050

d) 1080

Answer submitted: **b)**

You have answered the question correctly.

Question Explanation:

In order for Alice to see any of Betty's inserts, Betty must commit, which means that all three inserts are in R. Thus, Alice could compute the sum before Betty or Carol does anything, in which case she gets answer 1000, or after Betty commits but before Carol deletes anything, in which case she gets answer 1060.

However, we also need to consider the timing of Carol's deletes. She could complete her deletes and commit before Betty starts, in which case Carol has no effect. She could do the deletes after Betty commits, but before Alice reads, in which case, Alice sees only 10 from among Betty's inserts and gets the answer 1010.

Another possibility is that Carol starts before Betty commits, deleting 30 (which has no effect on R, since 30 is not in R, at least as far as Carol can see, since Betty has not committed the insert). Then, Betty does her inserts and commits, and Carol deletes 20, leaving 10 and 30 in R. Thus, when Alice reads, she gets the answer 1040.

The only other possibility is that Carol starts after Betty commits, but does not get to commit before Alice reads. In that case, Alice cannot see Carol's deletes, and gets the answer 1060.

---

4. Consider the following transactions:

T_1: $r_1(X)$, $w_1(X)$, $r_1(Y)$, $w_1(Y)$
T_2: $r_2(X)$, $w_2(X)$, $r_2(Y)$, $w_2(Y)$

Some schedules of these transactions will be conflict-equivalent to serial schedules, and of those, some are equivalent to the serial order (T_1, T_2) and others to the order (T_2, T_1). Your problem is to count how many of each kind of conflict-serializable schedule there are. Then, identify which of the following is a TRUE statement about schedules of T_1 and T_2.

a) There are exactly eight conflict-serializable schedules

b) There are exactly six schedules that are conflict equivalent to (T_2, T_1)

c) There are exactly 24 schedules that are conflict equivalent to (T_2, T_1)

d) There are exactly 70 conflict-serializable schedules

Answer submitted: **b)**

You have answered the question correctly.

Question Explanation:

First, consider the schedules that are conflict-equivalent to (T_1, T_2). In these schedules, $w_1(X)$ must precede $r_2(X)$. Thus, the first two actions must be $r_1(X)$, $w_1(X)$. Similarly, $w_1(Y)$ precedes $r_2(Y)$, so the last two actions are $r_2(Y)$, $w_2(Y)$. Only the four actions in the middle --- $r_1(Y)$, $w_1(Y)$, $r_2(X)$, $w_2(X)$ --- may be interleaved in any way such that $r_1(Y)$ precedes $w_1(Y)$ and $r_2(X)$ precedes $w_2(X)$. There are six such ways, which you can see by noting that of the four positions for these actions, two must be allocated to T_1, and these uniquely determine where each of the four actions belongs. The number of ways to choose 2 positions out of 4 is {4 choose 2} = 6.

Likewise, there are six schedules that are conflict-equivalent to the serial order (T_2, T_1). There are thus 12 conflict-serializable schedules.

---

5. Six transactions, T1 through T6, are requesting locks in a single-lock-mode system. Here is the sequence of lock requests so far:

1. T1: lock A
2. T2: lock B
3. T3: lock C
4. T3: lock A
5. T4: lock D
6. T4: lock B
7. T5: lock E
8. T5: lock C
9. T6: lock F
10. T6: lock B

Draw the waits-for graph at this time. Then, identify from the list below the request that will cause the requesting transaction to roll back.

a) T1: lock A

b) T1: lock D

c) T2: lock D

d) T1: lock F

Answer submitted: **c)**

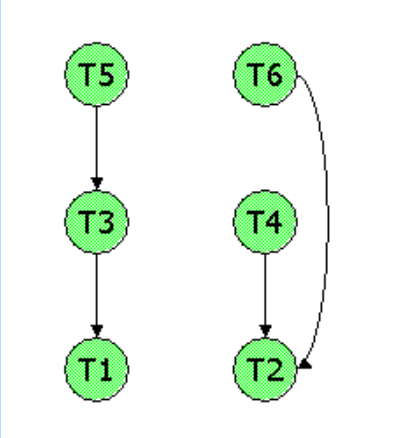
You have answered the question correctly.

Question Explanation:

The first three requests are granted, and cause T1 to hold (a lock on) A, T2 to hold B, and T3 to hold C. The fourth request (T3: lock A) causes T3 to wait for T1, the holder of A. Now, the waits-for graph has only an arc from T3 to T1. Then, the request by T4 for a lock on D is granted, but the next request causes an arc from T4 to T2, the holder of B.

The request by T5 for a lock on E is granted, but then T5 waits for T3, and there is an arc from T5 to T3. The request by T6 for a lock of F is granted, but T6 then is made to wait for T2, and there is an arc from T6 to T2. The waits-for

graph is:



Now, only T1 and T2 are not waiting. If T1 requests a lock on C or E, there will be a cycle, so T1 will have to roll back. However, if T1 requests a lock on B, D, or F, it will have to wait, but there is no cycle created in the waits-for graph. Similarly, if T2 requests D or F it must roll back, while if it requests A, C, or E, it merely waits.

---

6. Consider the following schedule of operations:

$r_1(X), r_2(Y), w_1(X), w_2(Y), r_2(Z), r_1(Y), w_1(Y), w_2(Z)$

Certain other interleavings of the steps of these two transactions are conflict-equivalent to the above schedule. In fact, there is a simple rule that tells whether or not another schedule of the same transactions will be conflict-equivalent. Determine this rule, and then identify from the list below, the one schedule that is conflict-equivalent to the above schedule.

a) $r_1(X), w_1(X), r_1(Y), r_2(Y), w_1(Y), w_2(Y), r_2(Z), w_2(Z)$

b) $r_2(Z), w_2(Z), r_2(Y), w_2(Y), r_1(Y), w_1(Y), r_1(X), w_1(X)$

c) $r_1(X), w_1(X), r_1(Y), w_1(Y), r_2(Y), w_2(Y), r_2(Z), w_2(Z)$

d) $r_2(Y), w_2(Y), r_1(X), w_1(X), r_1(Y), r_2(Z), w_2(Z), w_1(Y)$

Answer submitted: **d)**

You have answered the question correctly.

Question Explanation:

The only conflict in the given schedule is that caused by $w_2(Y)$ preceding $r_1(Y)$. No conflict involving X or Z is possible, since these are written by only one transaction. That is, as long as $w_2(Y)$ precedes $r_1(Y)$ in any other schedule, and that schedule consists of an interleaving of the same two transactions, it will be conflict-equivalent to the given schedule. Other schedules will not be conflict-equivalent.

---

7. Consider the following transactions:

- S: [X := X + 10; Y := Y - 10]
- T: [X := X * 2; Y := Y * 2]
- U: [Y := Y + 10; X := X - 10]

Assuming initial values of X = 15 and Y = 25, serializable schedules of

these three transactions can leave the database in various states. Determine all such possible states. Then, identify from the list below, which of the following is a possible state of the database resulting from a serializable execution of S, T and U.

a) X = 40; Y = 40

b) X = 30; Y = 70

c) X = 30; Y = 60

d) X = 20; Y = 50

Answer submitted: **a)**

You have answered the question correctly.

Question Explanation:

We may as well consider only serial schedules, since by definition any serializable schedule has the same effect as some serial schedule. There are six possible orders of the three transactions, but notice that S and U are inverses of each other, so whenever they appear in either order, consecutively, it is as if they were not there, and the net effect is that of T. In all these cases, we wind up doubling X and Y; i.e., the final database state is X=30; Y=50.

The only schedules in which S and U are not adjacent are STU and UTS. In STU, the state after S is (X=25; Y=15), after T is is (X=50; Y=30), and after U it is (X=40; Y=40). For UTS, the states are (X=5; Y=35), (X=10, Y=70), and the final (X=20; Y=60).

Thus, the only possible database states at the end are (X=30; Y=50), (X=Y=40), and (=20; Y=60).
