Weekly exercises IN3020 and IN4020
Week 10

March 28, 2022

1 Review

1. What is (conflict-)serialisability and how can serialisability be used as a way to ensure concurrency control?

   Solution: A schedule S is serialisable if it is conflict-equivalent to some serial schedule S'. If S is serialisable it is considered correct. It may not, however, be serial by itself. Thus a serialisable schedule gives the benefit of allowing concurrent transactions, but at the same time being correct.

2. What does it mean for two schedules to be conflict-equivalent?

   Solution: The two schedules are of the same transactions and the relative order of every two conflicting operations is the same in both schedules.

3. Discuss the four levels of isolation in SQL. What is snapshot isolation?

   Solution:

   - Read uncommitted:
   - Read committed:
   - Repeatable read:
   - Serialisable:

   Snapshot isolation is that a transaction sees the data items that it reads based on the committed values of the items in the database snapshot when the transaction starts. This ensures that the phantom record problem does not occur.

4. What is the two-phase locking protocol?

Solution: The two-phase locking protocol guarantees that all locking operations precede the first unlock operation in the transaction.

5. If you are using the two-phase locking protocol, can you be sure that a schedule is serialisable?

   Solution: The two-phase locking protocol can be proven to always give a serialisable schedule (if this schedule is complete).

6. In what cases can we have a deadlock? What are the approaches to this problem?

   Solution: We have a deadlock when several transactions are waiting for unlocking some items in a circular manner. Overall, there are deadlock prevention, deadlock detection, and timeout approaches to deal with deadlocks.

2 Serialisability

1. Let T_1 be the transaction $r_1(X), w_1(X), r_1(Y), w_1(Y)$ and T_2 the transaction $r_2(X), w_2(X)$. List all possible schedules of T_1 and T_2 and decide which are serialisable.

   Solution:

   - $r_1(X), w_1(X), r_1(Y), w_1(Y), r_2(X), w_2(X)$ Serializable, it is serial
   - $r_1(X), w_1(X), r_1(Y), r_2(X), w_1(Y), w_2(X)$ Serializable
   - $r_1(X), w_1(X), r_2(X), r_1(Y), w_1(Y), w_2(X)$ Serializable
   - $r_1(X), r_2(X), w_1(X), r_1(Y), w_1(Y), w_2(X)$ Not serialisable
   - $r_2(X), r_1(X), w_1(X), r_1(Y), w_1(Y), w_2(X)$ Not serialisable
   - $r_2(X), r_1(X), w_1(X), r_1(Y), w_2(X), w_1(Y)$ Not serialisable
   - $r_2(X), r_1(X), w_1(X), w_2(X), r_1(Y), w_1(Y)$ Not serialisable
   - $r_2(X), r_1(X), w_2(X), w_1(X), r_1(Y), w_1(Y)$ Not serialisable
   - $r_2(X), w_2(X), r_1(X), w_1(X), r_1(Y), w_1(Y)$ serialisable, it is serial
   - $+$ some others

2. Below are two transactions, described in terms of their effect on the database elements A and B, which we may assume are integers.

T_1	T_2
$r_1(A)$	$r_2(B)$
$A := A + 1$	$B := B * 2$
$w_1(A)$	$w_2(B)$
$r_1(B)$	$r_2(A)$
$B := B * 3$	$A := A + 3$
$w_1(B)$	$w_2(A)$

(a) Give examples of a serialisable schedule and a non-serialisable schedule of the these transactions

(b) It turns out that both serial schedules have the same effect on the database, that is, (T_1, T_2) and (T_2, T_1) are equivalent. Demonstrate this fact by showing the effect of the two transactions on an arbitrary initial database state.

Solution:
$T_1 : T_2$ gives $(A + 1) + 3 = A + 4$ and $B = (B * 3) * 2 = 6B$
$T_2 : T_1$ gives $(A + 3) + 1 = A + 4$ and $B = (B * 2) * 3 = 6B$

3. Consider the following two schedules:

S1 $w_3(A); r_1(A); w_1(B); r_2(B); w_2(C); r_3(C)$

S2 $r_1(A); r_2(A); r_3(B); w_1(A); r_2(C); r_2(B); w_2(B); w_1(C)$

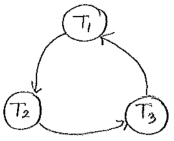
Answer the following questions:

(a) What is the precedence graph for the schedule?

(b) Is the schedule serialisable? If so, what are the equivalent serial schedules?

(c) Are there any serial schedules that must be equivalent (regardless of what the transactions do to the data), but are not conflict-equivalent?

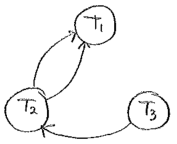
Solution:

(a) S1 gives a cyclic graph 

. This means that the schedule is not serialisable. Is it equivalent to one of these plans $T_1; T_2; T_3$ | $T_1; T_3; T2$ | $T_2; T_3; T1$?

If T_1 calculates a new value in B depending on what it reads from A, then it is of importance if T_3 has written a new value in A

before or after T_1 reads it. The same applies for T_2 and T_3. Thus. This plan is not equivalent with any serial plan.

(b) S2 This graph is without cycles 

. This means the schedule is conflict serialisable. It is equivalent with $T_3; T_2; T_1$. Since T_3 only reads values, the serial schedule $T_2; T_1; T_3$ will have the same effect on the database. It is, however, not conflict equivalent with the plan above. The application that initiated T_3 will get a different return value than the original plan gave.

3 Locks

1. Again consider the two schedules:

   (a) $w_3(A); r_1(A); w_1(B); r_2(B); w_2(C); r_3(C)$

   (b) $r_1(A); r_2(A); r_3(B); w_1(A); r_2(C); r_2(B); w_2(B); w_1(C)$

   Assume that at transaction takes a lock on each database element immediately before it reads or writes the element, and that each transaction releases its locks immediately after the last time it accesses an element. Tell what the locking scheduler would do with each of these schedules; i.e., what request would get delayed, and when would they be allowed to resume? Then give an example of how the scheduler can look like.

   Solution:

   (a)
   - $T_1 : l_1(A), r_1(A), l_1(B), w_1(B), u_1(A), u_1(B)$
   - $T_2 : l_2(B), r_2(B), l_2(C), w_2(C), u_2(B), u_2(C)$
   - $T_3 : l_3(A), w_3(A), l_3(C), r_3(C), u_3(A), u_3(C)$

T_1	T_2	T_3
		$l_3(A)$
		$w_3(A)$
$l_1(A)$ – wait		
	$l_2(B)$	
	$r_2(B)$	
	$l_2(C)$	
	$w_2(C)$	
	$u_2(B)$	
	$u_2(C)$	
		$l_3(C)$
		$r_3(C)$
		$u_3(A)$
$l_1(A)$ – aquired		
		$u_3(C)$
$r_1(A)$		
$l_1(B)$		
$w_1(B)$		
$u_1(A)$		
$u_1(B)$		

(b)
- $T_1 : l_1(A), r_1(A), w_1(A), l_1(c), w_1(c), u_1(A), u_1(C)$
- $T_2 : l_2(A), r_2(A), l_2(C), r_2(C), l_2(B), w_2(B), u_2(A), u_2(C), u_2(B)$
- $T_3 : l_3(B), r_3(B), u_3(B)$

T_1	T_2	T_3
$l_1(A)$		
$r_1(A)$		
	$l_2(A)$ – wait	
		$l_3(B)$
		$4_3(B)$
		$u_3(B)$
$w_1(A)$		
$l_1(C)$		
$w_1(C)$		
$u_1(A)$		
$u_1(C)$		
	$l_2(A)$ – aquired	
	$r_2(A)$	
	$l_2(C)$	
	$r_2(C)$	
	$l_2(B)$	
	$w_2(B)$	
	$u_2(A)$	
	$u_2(C)$	
	$u_2(B)$	

2. Prove that two-phase locking always gives serialisable schedules (Hint: if a serialisability graph for a schedule has a cycle, at least one of the transactions in the schedule does not obey the two-phase locking protocol)

   Solution: Let T be a set of transactions $T_1, T_2, \ldots, T_n$ that all obey the basic two-phase locking rule. Let S be a non-serialisable schedule for T. Then, the serialisation graph for S must have a cycle. Thus, there must be some sequence within the schedule on the form: $S \ldots; [o_1(X); \ldots; o_2(X)] \ldots; [o_2(Y); \ldots; o_3(Y)] \ldots; [o_n(Z); \ldots; o_1(Z)] \ldots$. Where each pair of operations between square brackets $[o, o]$ are conflicting (either [w,w], or [w,r] or [r,w]) in order to create an arc in the serialization graph. This implies that in transaction T_1 a sequence on the following form occurs: $T_1; \ldots; o_1(X); \ldots; o_1(Z) \ldots$ Furthermore, T_1 has to unlock item X (so T_2 can lock it before applying $o_2(X)$ to follow the rules of locking) and has to lock item Z (before applying $o_1(Z)$, but this must occur after T_n has unlocked it). Hence, a sequence in T_1 of the following form occurs: $T_1 : \ldots o_1(X); \ldots; \mathit{unlock}(X); \ldots; \mathit{lock}(Z); \ldots; o_1(Z); \ldots$ This implies that T_1 does not obey the two-phase locking protocol since $\mathit{lock}(Z)$ follows $\mathit{unlock}(Z)$, contradicting our assumption that all transactions in S follow the two-phase locking protocol.