Homework-5

Chapter 12 Query Processing (Pg-576)

Name: ABID ALI

Roll : 2019380141

and that transactions can be serialized according to their lock points.

12.8 Consider the following two transactions:

T_{34}: read(A);
read(B);
if $A = 0$ then $B := B + 1$;
write(B).

T_{35}: read(B);
read(A);
if $B = 0$ then $A := A + 1$;
write(A).

Add lock and unlock instructions to transactions T_{31} and T_{32}, so that they observe the two-phase locking protocol. Can the execution of these transactions result in a deadlock?

12.8

Solution:

Lock and unlock instructions:

T_{34}:

lock-S(A)

read(A)

lock-X(B)

read(B) if A = 0

then B := B + 1

write(B)

unlock(A)

unlock(B)

T_{35}:

lock-S(B)

read(B)

lock-X(A)

read(A)

if B = 0

then A := A + 1

write(A)

unlock(B)

unlock(A)

Execution of these transactions can result in deadlock. For example, consider the following partial schedule:

T_{31}	T_{32}
lock-S(A)	
	lock-S(B)
	read(B)
read(A)	
lock-X(B)	
	lock-X(A)

The transactions are now deadlocked.

both situations, and compare the relative amount of concurrency allowed.

12.12 For each of the following protocols, describe aspects of practical applications that would lead you to suggest using the protocol, and aspects that would suggest not using the protocol:

- Two-phase locking.
- Two-phase locking with multiple-granularity locking.
- Timestamp ordering.
- Validation.

12.12

Solution:

• Two-phase locking: Use for simple applications where a single granularity is acceptable. If there are large read-only transactions, multiversion protocols would do better. Also, if deadlocks must be avoided at all costs, the tree protocol would be preferable.

• Two-phase locking with multiple-granularity locking: Use for an application mix where some applications access individual records and others access whole relations or substantial parts thereof. The drawbacks of 2PL mentioned above also apply to this one

• Timestamp ordering: Use if the application demands a concurrent execution that is equivalent to a particular serial ordering (say, the order of arrival), rather than any serial ordering. But conflicts are handled by roll-back of transactions rather than waiting, and schedules are not recoverable. To make them recoverable, additional overheads and increased response time have to be tolerated. Not suitable if there are long read-only transactions, since they will starve. Deadlocks are absent.

• Validation: If the probability that two concurrently executing transactions conflict is low, this protocol can be used advantageously to get better concurrency and good response times with low overheads. Not suitable under high contention, when a lot of wasted work will be done.

12.15 **Explain the purpose** of the checkpoint mechanism. How often should checkpoints be performed? How does the frequency of checkpoints affect:

- System performance when no failure occurs?
- The time it takes to recover from a system crash?
- The time it takes to recover from a media (disk) failure?

12.15

Solution:

A checkpoint log record indicates that a log record and its modified data has been written to stable storage and that the transaction need not to be redone in case of a system crash. Obviously, the more often checkpoints are performed, the less likely it is that redundant updates will have to be performed during the recovery process.

• System performance when no failure occurs— If no failures occur, the system must incur the cost of performing checkpoints that are essentially unnecessary. In this situation, performing checkpoints less often will lead to better system performance.

• The time it takes to recover from a system crash—The existence of a checkpoint record means that an operation will not have to be redone during system recovery. In this situation, the more often checkpoints were performed, the faster the recovery time is from a system crash.

• The time it takes to recover from a disk crash—The existence of a checkpoint record means that an operation will not have to be redone during system recovery. In this situation, the more often checkpoints were performed, the faster the recovery time is from a disk crash.

12.16 List the ACID properties. Explain the usefulness of each.

12.16

Solution:

• Consistency: Execution of a transaction in isolation (that is, with no other transaction executing concurrently) preserves the consistency of the database. This is typically the responsibility of the application programmer who codes the transactions.

• Atomicity: Either all operations of the transaction are reflected properly in the database, or none are. Clearly lack of atomicity will lead to inconsistency in the database.

• Isolation: When multiple transactions execute concurrently, it should be the case that, for every pair of transactions T_i and T_j , it appears to T_i that either T_j finished execution before T_i started, or T_j started execution after T_i finished. Thus, each transaction is unaware of other transactions executing concurrently with it. The user view of a transaction system requires the isolation property, and the property that concurrent schedules take the system from one consistent state to another. These requirements are satisfied by ensuring that only serializable schedules of individually consistency preserving transactions are allowed.

• Durability: After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures.

12.19 Consider the following two transactions:

T_{13}: read(A);
read(B);
if $A = 0$ **then** $B := B + 1$;
write(B).

T_{14}: read(B);
read(A);
if $B = 0$ **then** $A := A + 1$;
write(A).

Let the consistency requirement be $A = 0 \lor B = 0$, with $A = B = 0$ the initial values.

a. **Show that every serial execution involving these two transactions preserves the consistency of the database.**

b. **Show a concurrent execution of T_{13} and T_{14} that produces a nonserializable schedule.**

c. **Is there a concurrent execution of T_{13} and T_{14} that produces a serializable schedule?**

serializable schedule with two transactions such

12.19

Solution:

a.

There are two possible executions: T13 T14 and T14 T13.

Case 1:

	A	B
Initially	0	0
After T_{13}	0	1
After T_{14}	0	1

Consistency met: A = 0 ∨ B = 0 ≡ T ∨ F = T

Case 2:

	A	B
Initially	0	0
After T_{14}	1	0
After T_{13}	1	0

Consistency met: A = 0 ∨ B = 0 ≡ F ∨ T = T

b.

Any interleaving of T_{13} and T_{14} results in a non-serializable schedule

T_1	T_2
read(A)	
	read(B)
	read(A)
read(B)	
if A = 0 then B = B + 1	
	if B = 0 then A = A + 1
	write(A)
write(B)	

T_{13}	T_{14}
read(A)	
	read(B)
	read(A)
read(B)	
if A = 0 then B = B + 1	
	if B = 0 then A = A + 1
	write(A)
write(B)	

c.

There is no parallel execution resulting in a serializable schedule. From part a. we know that a serializable schedule results in $A = 0 \vee B = 0$. Suppose we start with T_{13} read(A). Then when the schedule ends, no matter when we run the steps of T_2, B = 1. Now suppose we start executing T_{14} prior to completion of T_{13}. Then T_2 read(B) will give B a value of 0. So, when T_2 completes, A= 1. Thus

$B = 1 \wedge A = 1 \rightarrow \neg (A = 0 \vee B = 0)$.

Similarly, for starting with T_{14} read(B).