Ivan Ric Rebato

Concurrent Access Explanation

In this scenario, Transaction 1 (T1) first locks *data(A)* with an exclusive lock, meaning no other transaction can access or modify it. T1 then reads and writes to *data(A)*. After this, T1 locks *data(B)* with a shared lock. A shared lock allows multiple transactions to hold the lock at the same time, but only for reading, not writing. T1 reads *data(B)* while it's shared.

Once T1 finishes its operations, it releases the locks on both *data(A)* and *data(B)*. This makes the data available for other transactions, like Transaction 2 (T2) and Transaction 3 (T3).

Next, T2 locks *data(A)* exclusively, meaning no other transaction can access or change it while T2 is working. T2 reads and writes to *data(A)*, making some changes. After completing these operations, T2 unlocks *data(A)*.

Finally, T3 locks *data(A)*, but only with a shared lock. Since T3 is only reading, this shared lock allows it to access *data(A)* without blocking other read-only operations. The process continues this way for other transactions.

So, the problem is that if T3 does not unlock data(A), it will remain locked by T3, which means that neither T1 nor T2 can modify it because the lock is still held. This situation can lead to delays and inefficiencies in the overall transaction process. and the possible problem is that T1 only sets a shared lock on *data(B)*, which allows other transactions to also set a shared lock. If another transaction is modifying *data(B)* at the same time (and hasn't committed yet), T1 could read uncommitted or inconsistent data, leading to an incorrect outcome.