Transactions - Concurrency

Exercise 16.3 Consider a database with objects *X* and *Y* and assume that there are two transactions *T*1 and *T*2. Transaction *T*1 reads objects *X* and *Y* and then writes object *X*. Transaction *T*2 reads objects *X* and *Y,* then reads X once more, and finally writes objects *X* and *Y* (i.e. T1: R(X), R(Y), W(X); T2: R(X), R(Y), R(X), W(X), W(Y))

1. Give an example schedule with actions of transactions *T*1 and *T*2 on objects *X* and *Y* that results in a write-read conflict.
2. Give an example schedule with actions of transactions *T*1 and *T*2 on objects *X* and *Y* that results in a read-write conflict.
3. Give an example schedule with actions of transactions *T*1 and *T*2 on objects *X* and *Y* that results in a write-write conflict.
4. For each of the three schedules, show that Strict 2PL disallows the schedule.

Exercise 16.6 Answer the following questions: SQL supports four isolation-levels and two access-modes, for a total of eight combinations of isolation-level and access-mode. Each combination implicitly defines a class of transactions; the following questions refer to these eight classes:

1. Consider the four SQL isolation levels. Describe which of the phenomena can occur at each of these isolation levels: *dirty read, unrepeatable read, phantom problem*.
2. For each of the four isolation levels, i) explain a lock-based implementation and ii) give an example of transaction that can run safely at that isolation level (i.e. doesn't expose any of the phenomena associated with that level of isolation), but doesn't run safely one level below (i.e. weaker level with more phenomena).
3. Why does the access mode of a transaction matter?

Exercise 17.4 Consider the following sequences of actions, listed in the order they are submitted to the DBMS:

- **Sequence S1:** T1:R(X), T2:W(X), T2:W(Y), T3:W(Y), T1:W(Y), T1:Commit, T2:Commit, T3:Commit
- **Sequence S2:** T1:R(X), T2:W(Y), T2:W(X), T3:W(Y), T1:W(Y), T1:Commit, T2:Commit, T3:Commit

For each sequence and for each of the following concurrency control mechanisms, describe how the concurrency control mechanism handles the sequence.
Assume that the timestamp of transaction *Ti* is *i*. For lock-based concurrency control mechanisms, add lock and unlock requests to the previous sequence of actions as per the locking protocol. The DBMS processes actions in the order shown. If a transaction is blocked, assume that all its actions are queued until it is resumed; the DBMS continues with the next action (according to the listed sequence) of an unblocked transaction.

1. Strict 2PL with timestamps used for deadlock prevention.
2. Strict 2PL with deadlock detection. (Show the waits-for graph in case of deadlock.)
3. Conservative (and Strict, i.e., with locks held until end-of-transaction) 2PL.