Brian Dang
Spire ID: 32628683

1) 1. (56 points) Analyzing schedules For each of the following schedules, state which of the following properties hold: c*onflict serializable, recoverable, cascadeless*, or *strict*.

A schedule is strict if a value written by a transaction T is not read or overwritten by other transactions until T either aborts or commits. Please display your answer as a table with one row per schedule, and one column per property. Indicate the satisfied properties with a check mark.

Reminder: a conflict serializable schedule is one that has no conflicting operations amongst its committed transactions. Aborted transactions are ignored when determining conflict serializability.

Schedules	Conflict Serializable	Recoverable	Cascadeless	Strict
A) T1:W(X), T2:R(X), T1:W(X), T2:Commit, T1:Commit				
B) T1:R(X), T2:W(X), T1:W(X), T2:Abort, T1:Commit		✅	✅	
C) T1:R(X), T2:W(X), T1:W(X), T2:Commit, T1:Commit		✅	✅	
D) T1:R(X), T2:R(X), T1:W(X), T1:Commit, T2:W(X), T2:Commit		✅	✅	✅
E) T1:W(X), T2:R(X), T1:W(X), T2:Commit, T1:Abort				
F) T1:W(X), T2:R(Y), T1:R(Y), T2:R(X), T1:Commit, T2:Commit	✅	✅		

G) T1:R(X), T2:R(X), T1:Commit, T2:W(X), T2:Commit	✅	✅	✅	✅

2) (10 Points) Locking 1

Consider the following schedule:

T1:R(A), T1:W(A), T2:R(A), T2:W(A), T2:R(B), T2:W(B), T2:Commit, T1:Abort

a) Is this schedule allowed by Two Phase Locking? If not explain why. If so, show the schedule with lock requests and releases obeying 2PL.

This schedule is not allowed by Two Phase Locking (2PL) because it violates the "growing phase" and "shrinking phase" rules of the protocol. In 2PL, a transaction must obtain all its locks in the growing phase and release all its locks in the shrinking phase.

New Schedule:

T1:Lock(S,A) - T1 acquires a shared lock on A.
T1:R(A) - T1 reads A.
T1:Lock(X,A) - T1 upgrades its shared lock to an exclusive lock on A.
T1:W(A) - T1 writes to A.
T1:Unlock(A) - T1 releases the exclusive lock on A.
T2:Lock(S,A) - T2 acquires a shared lock on A.
T2:R(A) - T2 reads A.
T2:Lock(X,B) - T2 acquires an exclusive lock on B.
T2:W(B) - T2 writes to B.
T2:Commit - T2 commits the transaction.
T1:Abort - T1 aborts the transaction.

b) Is this schedule allowed by Strict Two Phase Locking? If not explain why. If so, show the schedule with lock requests obeying Strict 2PL.

This schedule is not allowed by Strict Two Phase Locking (Strict 2PL) because it violates the strictness rule of the protocol. In Strict 2PL, a transaction must hold all its locks until it commits or aborts.

There is no way to modify the schedule to comply with Strict 2PL because T2 attempts to read a conflicting object before T1 releases its locks.

3. (30 points) Locking 2
Consider the following two transactions:

T1 = R(A) W(A) R(B) W(B)
T2 = R(B) R(A) W(A) W(B)

Assume that exclusive lock (X) and unlock (U) actions are inserted by the scheduler, resulting in the following annotated transactions:

T1 = X(A) R(A) W(A) X(B) R(B) W(B), AFTER COMMIT: U(A) U(B)
T2 = X(B) R(B) X(A) R(A) W(A) W(B), AFTER COMMIT: U(A) U(B)

Consider concurrent execution of these two transactions and answer the following questions:

a) Is conflict serializability guaranteed? Why or why not?

Conflict serializability is not guaranteed because there is a conflict when T1 writes to A and then T2 reads A or when T1 writes to B and then T2 reads B.

b) Is deadlock possible? If so, then assuming that T1 starts first, which transaction(s) would be rolled back (aborted) under the wait-die deadlock prevention scheme?

Deadlock is possible because T1 acquires an exclusive lock (X) on resource A and then attempts to acquire an exclusive lock on resource B. At the same time, T2 acquires an exclusive lock on resource B and then attempts to acquire an exclusive lock on resource A which then causes deadlock.

Assuming that T1 starts first, T2 is rolled back because T1 is allowed to wait.

c) Is cascading rollback possible? If not, explain why not. If so, show a scenario that results in cascading rollback.

Because deadlock is possible, cascading rollback is possible.

Now, say that lock and unlock actions are inserted in the following way instead:
T1 = X(A) R(A) W(A) X(B) U(A) R(B) W(B) U(B)
T2 = X(A) X(B) R(B) R(A) W(A) U(A) W(B) U(B)

d) Is conflict serializability guaranteed? Why or why not?

Conflict serializability is guaranteed in this case because the order of conflicting operations between T1 and T2 is the same in any serial schedule.

e) Is deadlock possible? If so, then assuming that T1 starts first, which transaction(s) would be rolled back (aborted) under the wound-wait deadlock prevention scheme?

Deadlock isn't possible because there are no conflicting operations between T1 and T2.

f) Is cascading rollback possible? If not, explain why not. If so, show a schedule that results in cascading rollback.

Cascading rollback isn't possible because in the given schedule, there are no shared data items being modified after they have been read by another transaction. Therefore, cascading rollback is not possible.

Consider the following schedule:

T1: X(A) (T1 acquires an exclusive lock on A)
T1: R(A) (T1 reads A)
T2: X(A) (T2 attempts to acquire an exclusive lock on A but is blocked by T1)
T1: W(A) (T1 modifies A)
T1: X(B) (T1 acquires an exclusive lock on B)
T1: U(A) (T1 releases the lock on A)
T2: R(B) (T2 reads B)
T2: R(A) (T2 attempts to read A, but it has been modified by T1)
T1: R(B) (T1 reads B)
T2: W(A) (T2 attempts to modify A, but it has been modified by T1)
T1: W(B) (T1 modifies B)
T1: U(B) (T1 releases the lock on B)
T2: U(A) (T2 releases the lock on A)
T2: U(B) (T2 releases the lock on B)
In this schedule, T2 attempts to read A (step 8) after T1 has modified it (step 4), resulting in a conflict. As a result, T2 would need to be rolled back (aborted) to maintain consistency, resulting in cascading rollback.

Therefore, the given schedule results in cascading rollback, where T2 is rolled back due to conflicts with the modifications made by T1.