Winter 2016 #4

(a) Consider a concurrency control manager that uses strict two phase locking that schedules three transactions:

- *T1 : R1(A), R1(B), W1(A), W1(B), Co1*
- *T2 : R2(B), W2(B), R2(C), W2(C), Co2*
- *T3 : R3(C), W3(C), R3(A), W3(A), Co3*

Each transaction begins with its first read operation, and commits with the *Co* statement. Answer the following questions for each of the schedules below:

- Is the schedule conflict-serializable? If yes, indicate a serialization order.
- Is this schedule possible under a strict 2PL protocol?
- If strict 2PL does not allow this schedule because it denies a read or a write request, is the system in a deadlock at the time when the request is denied?

i. Schedule 1:

R2(B), W2(B), R3(C), W3(C), R3(A), W3(A), Co3, R2(C), W2(C), Co2, R1(A), R1(B), W1(A), W1(B), Co1

α) Is this schedule conflict-serializable? If yes, indicate a serialization order.

β) Is it possible under strict 2PL?

γ) Does strict 2PL lead to a deadlock?

ii. Schedule 2:

R2(B), W2(B), R3(C), W3(C), R1(A), R1(B), W1(A), W1(B), Co1, R2(C), W2(C), Co2, R3(A), W3(A), Co3

α) Is this schedule conflict-serializable? If yes, indicate a serialization order.

β) Is it possible under strict 2PL?

γ) Does strict 2PL lead to a deadlock?

(b) Consider the following three transactions:

- *T1 : R1(A), W1(B), C01*
- *T2 : R2(B), W2(C), C02*
- *T3 : R3(C), W3(D), C03*

Given an example of a conflict-serializable schedule that has the following properties: transaction T1 commits before transaction T3 starts, and the equivalent serial order is T3, T2, T1.

(c) A read-only transaction is a transaction that only reads from the database, without writing/inserting deleting. Answer the questions below by circling the correct answer.

i. If all transactions are read-only, then every schedule is serializable.
TRUE or FALSE

ii. If no transaction reads the same element twice, then the serialization level READ COMMITTED is equivalent to REPEATABLE READS.
TRUE or FALSE

iii. If no transaction inserts or deletes records to/from the database, then the serialization level REPEATABLE READS is equivalent to SERIALIZABLE.
TRUE or FALSE

iv. The reason why some applications use serialization levels other than SERIALIZABLE is because they would not be correct under the SERIALIZABLE isolation level.
TRUE or FALSE

v. In Sqlite phantoms are not possible.
TRUE or FALSE

vi. The difference between Two Phase Locking and Strict Two Phase Locking is that the latter avoids deadlocks, while the former may allow deadlocks.
TRUE or FALSE

vii. Only one transaction can hold a shared lock at any time.
TRUE or FALSE

viii. Only one transaction can hold an exclusive lock at any time.
TRUE or FALSE

Autumn 2016 #4
Given the following three transactions:

```
T1: R(A), W(B), I(D), R(C)
T2: R(B), R(D), W(C)
T3: R(D), R(C), R(D), W(A)
```

Assume that `R(X)` reads all tuples in table `X`, `W(X)` updates all tuples in `X`, and `I(X)` inserts one new tuple in `X`. `Co` and `Ab` mean commit and abort, respectively. In the following, indicate the **strongest** isolation level that can generate each schedule, with serializable being the strongest isolation level. If serializable, then also give the serial schedule.

a) `R1(A); W1(B); I1(D); R3(D); R2(B); R3(C); R3(D); R2(D); R1(C); W2(C); W3(A); Co1; Co2; Co3;`

None Read uncommitted Read committed Repeatable read Serializable

Serial schedule:

b) `R2(B); R1(A); R3(D); R3(C); R2(D); W2(C); Co2; R3(D); W3(A); Ab3; W1(B); I1(D); R1(C); Co1;`

None Read uncommitted Read committed Repeatable read Serializable

Serial schedule:

c) `R1(A); R2(B); R3(D); R3(C); R2(D); I1(D); W2(C); Co2; R3(A); W1(B); W3(D); Co3; R1(C); Ab1;`

None Read uncommitted Read committed Repeatable read Serializable

Serial schedule:

Transactions copied here for your reference.

```
T1: R(A), W(B), I(D), R(C)
T2: R(B), R(D), W(C)
T3: R(D), R(C), R(D), W(A)
```

d) Does there exist a schedule of the above transactions that would result in a deadlock if executed under strict 2PL with **both shared and exclusive table locks**? If so write such a schedule with lock / unlock ops, and explain why the transactions are deadlocked. Otherwise write "No". Use `L1(A)` to refer to `T1` locking table `A`, and `U1(A)` for unlocking.

e) Does there exist a schedule of the above transactions that would result in a deadlock if executed under strict 2PL with **only exclusive table locks**? If so write such a schedule with lock and unlock operations and indicate why the transactions are deadlocked. Otherwise write "No". Use `L1(A)` to refer to `T1` locking table `A`, and `U1(A)` for unlocking.

f) Does there exist a schedule of the above transactions that would result in a manifestation of the phantom problem under non-strict 2PL with exclusive table locks? If so write out such a schedule with lock and unlock operations and indicate why there is a phantom problem. Otherwise write "No".

g) Suppose we change locking granularity to tuple rather than table level, where we only lock the tuples that are read / written / inserted from the affected table. Is there a schedule of the above transactions that would result in a manifestation of the phantom problem under non-strict 2PL with exclusive tuple locks? If so write out such a schedule with lock and unlock operations and indicate why there is a phantom problem. Otherwise write "No".