10.03.2025: Week 4 exercises: Transactions

1) For each of the following schedules, state if they are conflict serializable or not.

Schedule S1	
T1	**T2**
R(A)	
W(A)	
	R(A)
	W(A)
R(B)	
W(B)	
C	
	R(B)
	W(B)
	C

Schedule S2	
T1	**T2**
R(A)	
	R(A)
	W(A)
	C
W(A)	
C	

Schedule S3		
T1	**T2**	**T3**
R(A)		
	W(A)	
	C	
W(A)		
C		
		W(A)
		C

Schedule S4		
T1	**T2**	**T3**
R(A)		
	W(A)	
	C	
W(A)		
C		
		R(A)
		C

Answer:

- S1 is conflict serializable
- S2 is not conflict serializable

- S3 is not conflict serializable
- S4 is not conflict serializable

2) Given T1, T2, T3, state which of the following schedules S1, S2, S3 are serializable and which are conflict serializable. For all the schedules that are serializable, write their equivalent serial schedules.

T1: R(X) W(X) C
T2: W(X) W(Y) C
T3: R(X) R(Y) C

Schedule S1		
T1	**T2**	**T3**
R(X)		
		R(X)
	W(X)	
W(X)		
C		
		R(Y)
	W(Y)	
	C	
		C

Schedule S2		
T1	**T2**	**T3**
	W(X)	
R(X)		
W(X)		
		R(X)
	W(Y)	
		R(Y)
C		
	C	
		C

Schedule S3		
T1	**T2**	**T3**
R(X)		
	W(X)	
		R(X)
W(X)		
		R(Y)
	W(Y)	
	C	
		C
C		

Answer:

- S1 is not serializable. However, this is a case where serializability (not conflict serializability) can be enforced by certain protocols, e.g. the timestamp-based protocol with Thomas-Write Rule. In this case, when T1 tries to write X, it is detected

that X has already been updated by a newer transaction (T2) so this write is ignored. The equivalent order is T3 -> T1 -> T2.

- S2 is conflict serializable. The equivalent order is T2 -> T1 -> T3.
- S3 is not serializable (R(X) in T3 reads the value written by T2, but R(Y) in T3 does not read the value written by T2).

3) Is the following schedule allowed by:

a) 2PL?
b) Strict 2PL?

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

Answer:
The schedule is allowed by 2PL but it is not allowed by strict 2PL.

Potential schedule for 2PL:

T1	**T2**
Lock(A)	
R(A)	
W(A)	
Unlock(A) // Assuming that T1 does not need those locks any more	
	Lock(A)
	Lock(B)
	R(A)
	W(A)
	R(B)
	W(B)
	Unlock(A)
	Unlock(B)
	Commit
Abort	

4) Consider the following sequence of actions, listed in the order they are submitted to the DBMS:

T1: R(X), T2: W(X), T2: W(Y), T3: W(Y), T1: W(Y), T1: Commit, T2: Commit, T3: Commit

Describe how the sequence is handled by:

a) Strict 2PL with timestamps used for deadlock prevention and Wait-Die policy;
b) Strict 2PL with deadlock detection.

Answer:
a)

- T1 acquires shared lock on X.
- T2 asks for an exclusive lock on X. Since T2 has a lower priority, it will be aborted.
- T3 gets exclusive lock on Y.

- T1 also asks for an exclusive lock on Y, which is still held by T3. Since T1 has higher priority, T1 will be blocked waiting.
- T3 completes the write, commits and releases all the locks.
- T1 wakes up, acquires the lock, proceeds and finishes.
- T2 is restarted.

b)

- T1 gets a shared lock on X.
- T2 blocks waiting for an exclusive lock on X.
- T3 gets an exclusive lock on Y.
- T1 blocks waiting for an exclusive lock on Y.
- T3 finishes, commits and releases its locks.
- T1 wakes up, gets an exclusive lock on Y, finishes up and releases all the locks.
- T2 gets exclusive locks on X and Y and proceeds to finish.

No deadlock.

5) Are the following schedules allowed by Optimistic Concurrency Control?

Schedule S1	
T1	**T2**
W(B)	
R(A)	
	W(A)
	C
C	

Schedule S2	
T1	**T2**
	W(A)
W(A)	
C	
	C

Schedule S3	
T1	**T2**
R(A)	
	W(A)
R(A)	
C	
	C

Answer:
S2 and S3 is allowed.
In S1, T2 finishes first the read phase so it acquires a smaller timestamp. There is a conflict since WriteSet(T2) ∩ ReadSet(T1) = A.
In S2, test 2 passes, that is, T1 completes before T2 begins validation and T2 does not read dirty data from T1.

6) State if the following schedule is allowed by a) Timestamp-based CC, b) MVTO.

T1	T2
R(A)	
	W(A)
	C
R(A)	
C	

Answer:
a) NO, b) YES

7) Is the following schedule allowed by a) 2PL, b) Strict 2PL, c) Timestamp-based CC, d) OCC, e) MVTO?

T1	T2	T3
R(Y)		
	R(Y)	
R(X)		
		W(Y)
W(Z)		
	R(Y)	
W(X)		
	R(Y)	
	C	
C		
		C

Answer:
Tip: First "cleanup" reads and writes that do not affect consistency (e.g. only T1 touches X and Z)
a) NO, b) NO, c) NO, d) YES, e) YES.

8) Answer each of the following questions briefly. The questions are based on the following relational schema:

Emp(eid: integer, ename: string, age: integer, salary: real, did: integer)
Dep(did: integer, dname: string, floor: integer)

and on the following update command:

replace (salary = 1.1 * EMP.salary) where EMP.ename = "Santa"

1. Give an example of a query that would conflict with this command (in a concurrency control sense) if both were run at the same time. Explain what could go wrong, and how locking tuples would solve the problem.
2. Give an example of a query or a command that would conflict with the update command such that the conflict could not be resolved by just locking individual tuples of pages but requires index locking.
3. Explain what index locking is and how it resolves the preceding conflict.

Answer:

1. One example query that would conflict with the above command is:

Select eid From Emp Where salary = 10,000

Suppose there are two employees who both have salary of 10,000. If the update command is carried out first, neither of them will be selected; if the select command is done first, both of them will be selected. However, if the two commands were run at the same time, one tuple may be selected while the other may not. By locking tuples, it is ensured that either all the tuples are updated first or all the tuples go through the selection query first.

2. Insert (EID, “Santa”, AGE, SALARY, DID) Into Emp
3. If there is an index on a particular field of a relation, a transaction can obtain a lock on one or more index page(s). This will effectively lock all the existing records with the field having a certain range of values, and it will also prevent insertion of new records with the field value in that particular range. This will prevent the so-called phantom problem.