1. (20 points) Consider a database that has two integer objects, A and B. DBMS is given two programs:

P1	P2
A = A + 1 B = B + 1	A = B B = B + 1

Before the program is execution, A = 0 and B = 0. For each of the following schedules, determine if it is a serializable schedule. You must explain why.

S1

T1	T2
A = A + 1 B = B + 1	A = B B = B + 1

There are two serial schedules:

1) SS1: Execute P1 and then P2. Then result is A=1 and B=2.
2) SS2: Execute p2 and then P2. The result is A=1 and B=2.

The result of S1 is A=1 and B=2, which is equivalent to one of the two serial schedules, which are same. Therefore, S1 is a serializable schedule.

S2

T1	T2
A = A + 1 B = B + 1	A = B B = B + 1

The result of S2 is A=0 and B=2. It is not equivalent to any of the two serial schedules. Therefore, S2 is NOT a serializable schedule.

S3

T1	T2
A = A + 1 B = B + 1	A = B B = B + 1

The result of S3 is A=1 and B=2. It is equivalent to one of the two serial schedules. Therefore, S3 is a serializable schedule.

2. (20 points) Consider the following two programs.

P1	P2
R(A)	W(A)
R(C)	R(B)
W(C)	W(B)
c/o	c/o

For each of the following execution, determine if it is a schedule, a serial schedule, and/or a strict schedule. You must explain why.

S1

T1	T2
R(A)	
	W(A)
R(C)	
	R(B)
W(C)	
	W(B)
c/o	
	c/o

S2

T1	T2
R(A)	
R(C)	
W(C)	
	W(A)
c/o	
	R(B)
	W(B)
	c/o

S3

T1	T2
R(A)	
R(C)	
W(C)	
c/o	
	W(A)
	R(B)
	W(B)
	c/o

S4

T1	T2
	W(A)
R(A)	
W(C)	
c/o	
	R(B)
	W(B)
	c/o

S1: It is a schedule (all reads/writes are in the same order as in their original programs), not a serial schedule (the execution of T1 and T2 interleaves), not a strict schedule (T1 has a R(A), before T1 commits/aborts, W(A) in T2 is executed).

S2: It is a schedule (all reads/writes are in the same order as in their original programs), not a serial schedule (the execution of T1 and T2 interleaves), not a strict schedule (T1 has a R(A), before T1 commits/aborts, W(A) in T2 is executed).

S3: It is a schedule (all reads/writes are in the same order as in their original programs), a serial schedule (T1 and T2 are executed one by one, without interleaving), and a strict schedule (a serial schedule must be a strict schedule).

S4: It is not a schedule (R(C) is missing in T1), not a serial schedule, not a strict schedule.

3. (60 points) Consider the following two schedules implemented by strict 2PL.

S1

T1	T2
S(A)	
R(A)	
X(A)	
W(A)	
commit	
	S(A)
	R(A)
	commit

S2

T1	T2
S(A)	
R(A)	
	S(A)
	R(A)
X(A)	
	commit
W(A)	
commit	

Recall strict 2PL is implemented with a lock table to keep 1) which data object is locked; 2) which transaction owns which lock; 3) which transaction is waiting for a lock on this object.

Lock table

Data	*Lock*	*Owner*	*Waiting*

For each of the two schedules, explain how the status of the lock table changes as the actions are executed. The table is assumed to be empty initially.

Answers are on the next page.

For S1 1) T1.S(A) asks for a share lock on A. Since there is no lock on A, the application is approved.

Data	*Lock*	*Owner*	*Waiting*
A	S	T1	

2) T1.R(A) reads A, the lock table remains the same.

Data	*Lock*	*Owner*	*Waiting*
A	S	T1	

3) T1.X(A) asks for an exclusive lock on A. Since A has a shared lock but this lock is owned by T1, the application is approved. The lock is upgraded to exclusive lock.

Data	*Lock*	*Owner*	*Waiting*
A	X	T1	

4) T1.W(A) writes A. The lock table remains the same.

Data	*Lock*	*Owner*	*Waiting*
A	X	T1	

5) T1.commit. T1 finished, so all locks it owns are released.

Data	*Lock*	*Owner*	*Waiting*

6) T2.S(A) asks for a shared lock on A. Since no lock on A exists, the application is approved.

Data	*Lock*	*Owner*	*Waiting*
A	S	T2	

7) T2.R(A) reads A. No change on the table.

Data	*Lock*	*Owner*	*Waiting*
A	S	T2	

8) T2.commits. T2 finishes. So, all locks that it owns are released.

Data	*Lock*	*Owner*	*Waiting*

For S2

1) T1.S(A) asks for a share lock on A. Since there is no lock on A, the application is approved.

Data	*Lock*	*Owner*	*Waiting*
A	S	T1	

2) T1.R(A) reads A, the lock table remains the same.

Data	*Lock*	*Owner*	*Waiting*
A	S	T1	

3) T2.S(A) asks for a shared lock on A. Since A has a shared lock, the application is approved.

Data	*Lock*	*Owner*	*Waiting*
A	S	T1, T2	

4) T2.R(A) reads A. The lock table remains the same.

Data	*Lock*	*Owner*	*Waiting*
A	S	T1, T2	

5) T1.X(A) asks for an exclusive lock on A. Since A has a shared lock from T2, T1 is suspended.

Data	*Lock*	*Owner*	*Waiting*
A	S	T1,T2	T1(X)

6) T2.commits. T2 finishes. All locks it owns are released. T1, which is waiting, resumes. It is asking for an exclusive lock, which is now approved.

Data	*Lock*	*Owner*	*Waiting*
A	X	T1	

7) T1.W(A) writes A. No change in the table.

Data	*Lock*	*Owner*	*Waiting*
A	X	T1	

8) T1.commits. T1 finishes. So, all locks it owns are now released.

Data	*Lock*	*Owner*	*Waiting*