Exercise 1: Consider the following transaction involving two bank accounts x and y.

read(x); x := x – 50; write(x); read(y); y := y + 50; write(y)

The given constraint is that the sum of the accounts x and y should remain constant is that of:

(A) Atomicity
(B) Consistency
(C) Isolation
(D) Durability

Answer. (B) Consistency in database systems refers to the requirement that any given database transaction must only change affected data in allowed ways, that is sum of x and y must not change.

Exercise 2: Consider the following two transactions:

*T*1: read(*A*);
read(*B*);
if *A* = 0 **then** *B* := *B* + 1;
write(*B*).
*T*2: read(*B*);
read(*A*);
if *B* = 0 **then** *A* := *A* + 1;
write(*A*).

Let the consistency requirement be $A = 0 \lor B = 0$, with $A = B = 0$the initial values.

Show that every serial execution involving these two transactions preserves the consistency of the database.

Answer

There are two possible executions: $T_1\ T_2$ and $T_2\ T_1$.

Case 1:

	A	B
initially	0	0
after T_1	0	1
after T_2	0	1

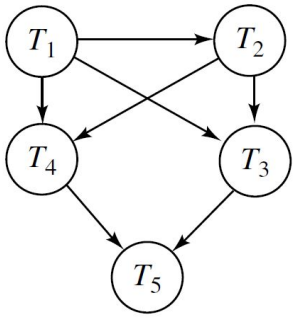
Consistency met: $A = 0 \vee B = 0 \equiv T \vee F = T$

Case 2:

	A	B
initially	0	0
after T_2	1	0
after T_1	1	0

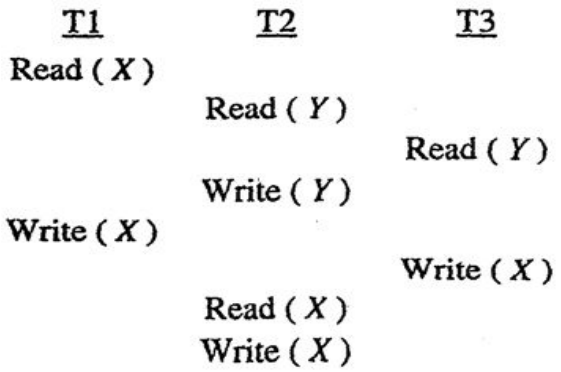
Consistency met: $A = 0 \vee B = 0 \equiv F \vee T = T$

Exercise 3: Consider the precedence graph below. Is the corresponding schedule conflict serializable?



Answer. Yes, as the graph is acyclic. A possible schedule can be obtained from a topological sort- one schedule is T_1, T_2, T_3, T_4, T_5

Exercise 4: Consider the following schedule for transactions T1, T2 and T3:



Write the correct serialization of the above schedule.

Answer. T1 -> T3 -> T2
Explanation: T1 can complete before T2 and T3 as there is no conflict between Write(X) of T1 and the operations in T2 and T3 which occur before Write(X) of T1 in the above diagram.
T3 should can complete before T2 as the Read(Y) of T3 doesn't conflict with Read(Y) of T2. Similarly, Write(X) of T3 doesn't conflict with Read(Y) and Write(Y) operations of T2.
Another way to solve this question is to create a dependency graph and topologically sort the dependency graph. After topologically sorting, we can see the sequence T1, T3, T2.