Question 1:

The following schedules are given: Check whether each of these schedules are **conflict serializable** and **view serializable**.

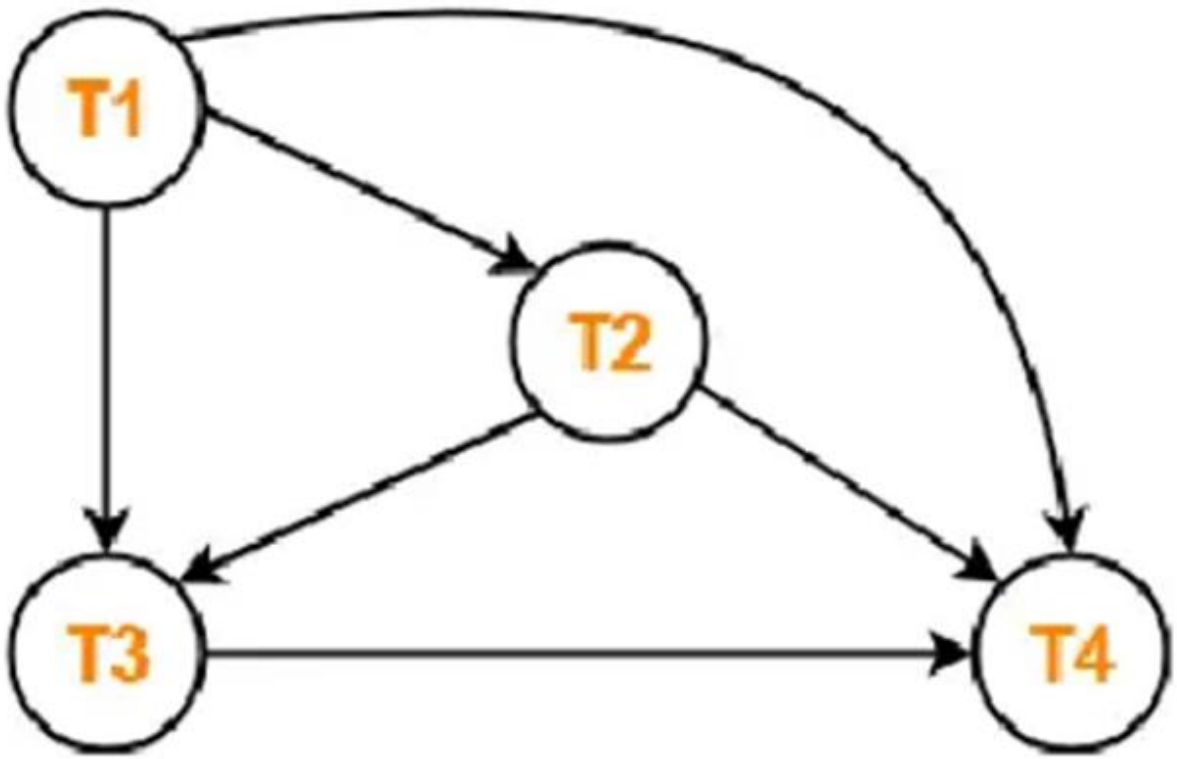
1. S1: $R_1(A)$, $R_2(A)$, $R_3(A)$, $R_4(A)$, $W_1(A)$, $W_2(A)$, $W_3(A)$, $W_4(A)$

2. S2: $R_1(A)$, $R_2(A)$, $W_3(A)$, $W_1(A)$

3. S3: $R_1(A)$, $R_2(A)$, $W_1(A)$, $W_2(A)$, $R_1(B)$, $R_2(B)$, $W_1(B)$, $W_2(B)$

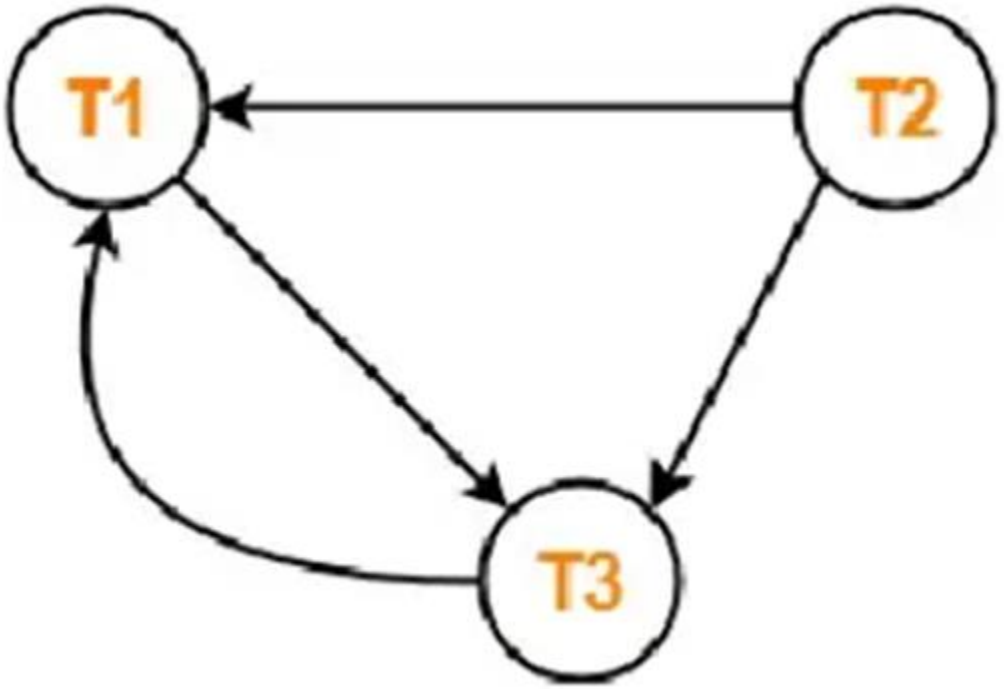
Solution:

S1:



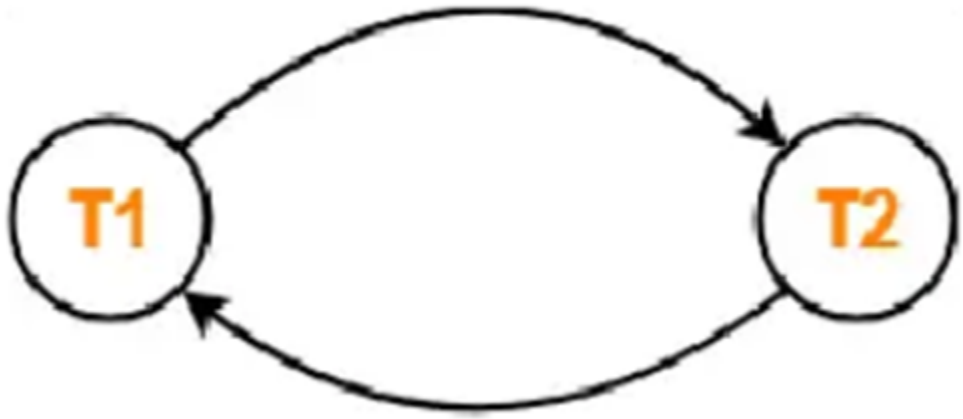
Clearly, there exists no cycle in the precedence graph. Therefore, the given schedule S is conflict serializable. Thus, we conclude that the given schedule is also view serializable.

S2:



There exists a cycle in the precedence graph. To check whether S is view serializable or not, check for blind writes. Therefore, the given schedule S is not conflict serializable and not view serializable .

S3:



There exists a cycle in the precedence graph. Therefore, the given schedule S is not conflict serializable. There exists no blind write in the given schedule S so not view serializable.

Question 2:

Consider a database table Accounts with columns Account_ID and Balance. The following transactions, T1 and T2, are meant to adjust the balance of an account based on its current value.

The interleaving of operations between T1 and T2 happens as follows:

1. T1 reads the current Balance (initially 500).
2. T2 reads the current Balance (initially 500).
3. T1 adds 100 to the balance, calculating it as 600.
4. T2 subtracts 50 from the balance, calculating it as 450.
5. T1 writes 600 back to Balance.
6. T2 writes 450 back to Balance.

1. Identify the Type of Problem(s): What concurrency issues are present in this scenario? Explain how they arise and what consequences they might have.
2. What is the final Balance of Account_ID = 101 after both transactions complete?
3. How does this result differ from the correct, serial outcome?

Solution:

1. This scenario involves a lost update problem because T2's write operation overwrites the results of T1's update without being aware of T1's actions.

2. Final Balance: 450 (incorrect) rather than the expected 550 if T1 and T2 executed sequentially.

3. Difference: The serial outcome would retain both modifications (adding 100 and subtracting 50 sequentially), but due to the concurrent writes without isolation, one update is effectively "lost."

Question 3:

Question

Consider two schedules, S1 and S2, involving two transactions, T1 and T2, on the same data items X and Y. The operations are as follows:

Schedule S1

T1: Read(X) T2: Read(X) T1: Write(X) T2: Write(X) T1: Read(Y) T2: Write(Y)

Schedule S2

T1: Read(X) T1: Write(X) T2: Read(X) T2: Write(X) T1: Read(Y) T2: Write(Y)

Questions:

1. List the conflicting operations in S1 and S2.

2. Determine if S1 and S2 are conflict equivalent. Justify your answer by comparing the order of conflicting operations in both schedules.

3. If S1 and S2 are conflict equivalent, state whether they are also conflict serializable. If they are not equivalent, explain how a serializable schedule can be derived from either of them.

Solution:

In both S1 and S2, the conflicting operations are:

T1’s Write(X) and T2’s Read(X) – there’s a conflict because T2 reads an updated value of X written by T1.

T1’s Write(X) and T2’s Write(X) – there’s a conflict because they write to the same data item, X, with different results.

Schedule S1 Conflicts

T1’s Read(X) happens before T2’s Read(X) — no conflict, as reads do not conflict.

T1’s Write(X) comes before T2’s Write(X).

T1’s Read(Y) comes before T2’s Write(Y).

Schedule S2 Conflicts

T1’s Read(X) happens before T2’s Read(X).

T1’s Write(X) comes before T2’s Write(X).

T1’s Read(Y) comes before T2’s Write(Y).

Since the conflicting operations in S1 and S2 appear in the same order, S1 and S2 are conflict equivalent. This means they will produce the same final result on the database as long as they follow the same order of conflicting operations.

In both schedules, T1 writes to X before T2 writes to X, indicating a dependency from T1 to T2 on X.

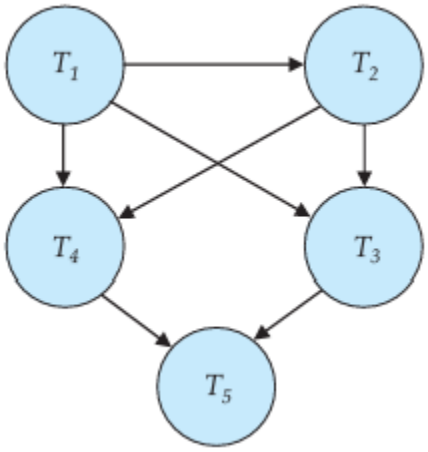
T1 also reads Y before T2 writes to Y, further enforcing T1 must precede T2.

Since all conflicts follow a single direction (from T1 to T2) in both S1 and S2, they can be serialized with T1 executing entirely before T2.

Therefore, S1 and S2 are both conflict serializable with the serial order T1 → T2.

Question: 4

Consider the given precedence graph. Is the corresponding schedule conflict serializable? Explain your answer



Answer: There is a serializable schedule corresponding to the precedence graph below, since the graph is acyclic. A possible schedule is obtained by doing a topological sort, that is, T1, T2, T3, T4, T5

Question: 5

The definition of a schedule assumes that operations can be totally ordered by time. Consider a database management system that runs on a system with multiple processors, where it is not always possible to establish an exact ordering between operations that are executed on different processors. However, operations on a data item can be totally ordered. Does the above situation cause any problem for the definition of conflict serializability? Explain your answer.

Ans:

The given situation will not cause any problem for the definition of conflict serializability since the ordering of operations on each data item is necessary for conflict serializability, whereas the ordering of operations on different data items is not important.

T_1	T_2
read(A)	
	read(B)
write(B)	