Tutorial 9

CSN-351/AID-523 Database Management Systems

1. Answer is option A.

   Create precedence graph and apply Topological sort on it to obtain T1->T3->T2

$$\begin{tikzcd}
T_1 \arrow[rr] \arrow[dr] & & T_2 \\
& T_3 \arrow[ur] &
\end{tikzcd}$$

2. (D) make a precedence graph for all the options, for option (D) only the graph will be acyclic, hence (D) is conflict serializable.

3. S is both conflict serializable and recoverable.

   Recoverable? Look if there are any dirty reads? Since there are no dirty read, it simply implies schedule is recoverable(if there were dirty read, then we would have taken into consideration the order in which transactions commit)

   Conflict serializable? Draw the precedence graph(make edges if there is a conflict instruction among Ti and Tj. But for the given schedule, no cycle exists in precedence graph, thus it's conflict serializable.

4.

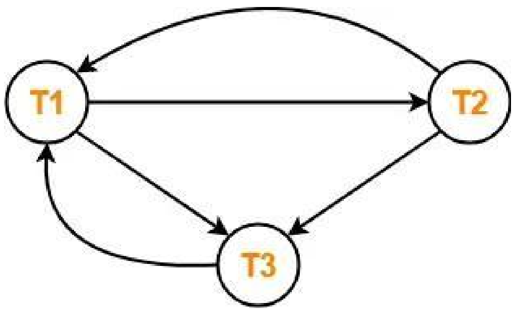
(A) Not correct, as there’s no dirty read; without a dirty read, the schedule remains recoverable.

(B) Incorrect, as no dirty read implies no cascading aborts.

(D) Incorrect, since T2 performs $W_2(X)$ before T1 commits or aborts.

(C) is the only correct option remaining.

5. Draw the precedence graph-

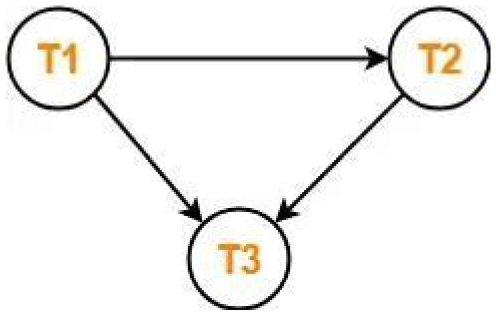


Clearly, there exists a cycle in the precedence graph.
Therefore, the given schedule S is not conflict serializable.

Now,

Since, the given schedule S is not conflict serializable, so, it may or may not be view serializable.
To check whether S is view serializable or not, let us use another method.
Let us check for blind writes.
There exists a blind write W2 (A) in the given schedule S.
Therefore, the given schedule S may or may not be view serializable.

Now,
To check whether S is view serializable or not, let us use another method.
Let us derive the dependencies and then draw a dependency graph.
T1 firstly reads A and T2 firstly updates A.
So, T1 must execute before T2.
Thus, we get the dependency **T1 → T2**.
Final updation on A is made by the transaction T3.
So, T3 must execute after all other transactions.
Thus, we get the dependency **(T1, T2) → T3**.
From write-read sequence, we get the dependency **T2 → T3**
Now, let us draw a dependency graph using these dependencies-



Clearly, there exists no cycle in the dependency graph.
Therefore, the given schedule S is view serializable.
The serialization order T1 → T2 → T3