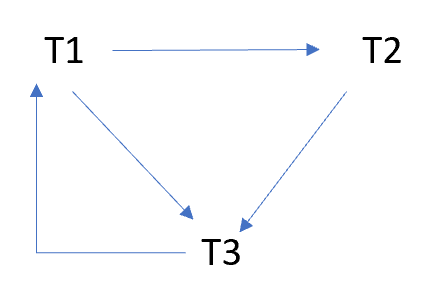
**Jaypee Institute of Information Technology**

**Database Systems & Web (15B11CI312)**

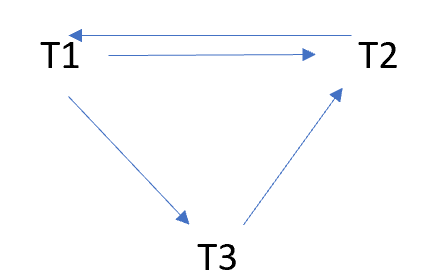
**Tutorial 14 -Solutions**

Q. 1 Which of the following schedules is (conflict) serializable? For each serializable schedule, determine the equivalent serial schedules.

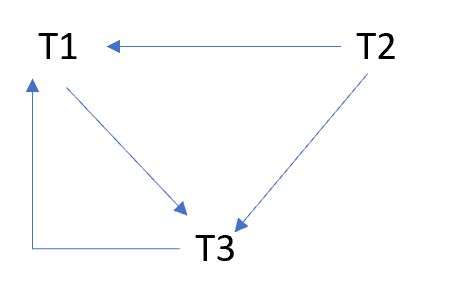
1. r1(X); r3(X); w1(X); r2(X); w3(X);



1. r1(X); r3(X); w3(X); w1(X); r2(X);

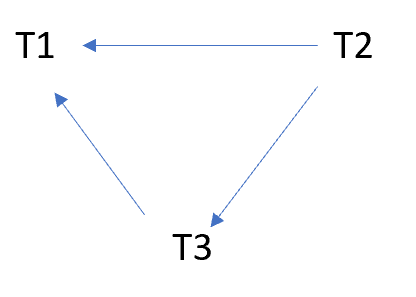


1. r3(X); r2(X); r1(x);w3(X); w1(X);



d. r3(X); r2(X); w3(X); r1(X); w1(X);

Ans: d r2(x);r3(x);w3(x);r1(x);w1(x)



Q.2 Consider schedules S3, S4, and S5 below. Determine whether each schedule is strict, cascadeless, recoverable, or nonrecoverable. (Determine the strictest recoverability condition that each schedule satisfies.)

S3: r1 (X); r2 (Z); r1 (Z); r3 (X); r3 (Y); w1 (X); c1; w3 (Y); c3; r2 (Y); w2 (Z); w2 (Y); c2;

r1 (X); r3 (X); w1 (X); c1; c3 c2

r2 (Z); r1 (Z); w2 (Z);

r3 (Y); w3 (Y); c3; r2 (Y); w2 (Y); c2;

***Sol: no dirty read, hence recoverable and casacadeless.***

***Read only the committed value, therefore, strict.***

S4: r1 (X); r2 (Z); r1 (Z); r3 (X); r3 (Y); w1 (X); w3 (Y); r2 (Y); w2 (Z); w2 (Y); c1; c2; c3;

Sol**: there is a dirty read between W3(Y) and R2(Y) hence not cascadeless, Not recoverable because T2 is committing before T3**

S5: r1 (X); r2 (Z); r3 (X); r1 (Z); r2 (Y); r3 (Y); w1 (X); c1; w2 (Z); w3 (Y); w2 (Y); c3; c2;

Recoverable, Cascadeless, not strict

**Solution 3**

1. Assume we use Wait-Die policy.

Sequence S1: T1:R(X), T2:W(X), T2:W(Y), T3:W(Y), T1:W(Y), T1:Commit, T2:Commit, T3:Commit Sequence S2: T1:R(X), T2:W(Y), T2:W(X), T3:W(Y), T1:W(Y), T1:Commit, T2:Commit, T3:Commit

**Sequence S1:**

T1 acquires shared-lock on X; for an exclusive-lock on X, since T2 has a lower priority, it will be aborted When T2 asks;

T3 now gets exclusive-lock on Y;

When 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 now finishes write, commits and releases all the lock;

T1 wakes up, acquires the lock, proceeds and finishes;

T2 now can be restarted successfully.

**Sequence S2:**

The sequence and consequence are the same with sequence S1, except T2 was able to advance a little more before it gets aborted.

1. In deadlock detection, transactions are allowed to wait, they are not aborted until a deadlock has been detected. (Compared to prevention schema, some transactions may have been aborted prematurely.)

**Sequence S1**:

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 locks;

T1 wakes up, get an exclusive-lock on Y, finishes up and releases lock on X and Y;

T2 now gets both an exclusive-lock on X and Y, and proceeds to finish. No deadlock.

**Sequence S2:** There is a deadlock. T1 waits for T2, while T2 waits for T