RICHFLEX

SRIDHAR SODHI-2021104
SOUMYA -2021103

A)
Transaction 1:

START TRANSACTION;

UPDATE n_product SET Quantity = Quantity - 10 WHERE P_ID = 1;
SELECT * from n_product;
INSERT INTO cart (C_ID, P_ID, Quantity) VALUES (1, 1, 10);

COMMIT;

Transaction 2:

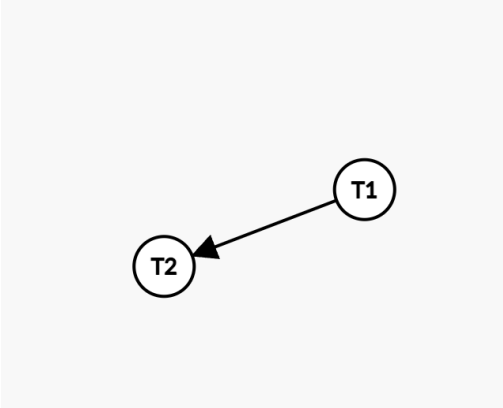
START TRANSACTION;

UPDATE n_product SET Quantity = Quantity - 5 WHERE P_ID = 1;
INSERT INTO cart (C_ID, P_ID, Quantity) VALUES (1, 2, 5);

COMMIT;

Schedule 1:

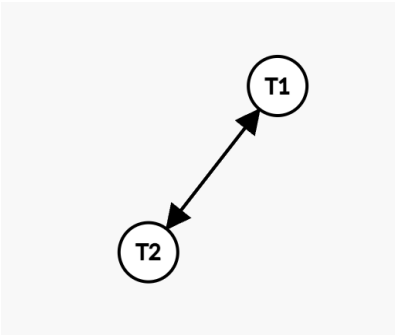
Transaction 1	Transaction 2
Write in n_prod(1)	
Read in n_product(1)	
	Write in n_prod(1)
	Write in cart(2)
Write in cart(1)	



To check if this schedule is Conflict Serializable or not we made a Precedence graph in which each vertex was a transaction. We draw the edges by finding the conflict pairs in other transactions. While doing this, we only found one conflict in lines 1 and 3 in which both were writing in the n_product, so we made an edge from T1 to T2 since there were no other, so the final graph didn't have any loops or cycles so we can say that the given schedule is Conflict Serializable.

Schedule 2:

Transaction 1	Transaction 2
Write in n_prod(1)	
	Write in n_prod(1)
	Write in cart(2)
Read in n_product(1)	
Write in cart(1)	



Now in this schedule we can proceed with the same method mentioned above to check if this schedule is Conflict Serializable or not we made a Precedence graph while doing that we found one conflict in line 1 and 2 from T1 to T2 and one in line 2 and 4 from T2

to T1 .and now we can see that these do make a loop so we can say that the given schedule is Non-Conflict Serializable.

B)

Transaction 1:

START TRANSACTION;

UPDATE n_product SET Quantity = Quantity - 2 WHERE P_ID = 1;
INSERT INTO cart (C_ID, P_ID, Quantity) VALUES (1, 1, 10);
SELECT * from customer;

COMMIT;

Transaction 2:

START TRANSACTION;

SELECT * from cart;
SELECT * from n_product;

COMMIT;

Schedule 1:

Transaction 1	Transaction 2
Write in n_prod(1)	
Write in cart(1)	
	Read in cart(1)
	Read in n_product(1)
Read in cart(1)	

To check whether this schedule is Conflict Serializable, we made a Precedence graph in which each vertex was a transaction. We draw the edges by finding the conflict pairs in other transactions. While doing this, we found one conflict in lines 1 and 4 from T1 TO

T2 and one in lines 2 and 3 from T1 TO T2, so the final graph didn't have any loops or cycles, so we can say that the given schedule is Conflict Serializable.

Schedule 2:

Transaction 1	Transaction 2
Write in n_prod(1)	
	Read in cart(1)
	Read in n_product(1)
Write in cart(1)	
Read in cart(1)	

Now in this schedule, we can proceed with the same method mentioned above to check if this schedule is Conflict Serializable or not. We made a Precedence graph; while doing that, we found one conflict in lines 1 and 3 from T1 to T2 and one in lines 2 and 4 from T2 to T1 .and now we can see that these do make a loop, so we can say that the given schedule is Non-Conflict Serializable.