Transactions

Schedule 1:

Transaction 1:

SQL:

```
BEGIN TRANSACTION;
UPDATE Cart Set Amount = Amount +1 WHERE Product_ID =1 AND Customer_ID = 1;
SELECT SUM(Amount) as total_amount FROM cart WHERE Customer_ID = 1;
SELECT * FROM Customer;
COMMIT;
```

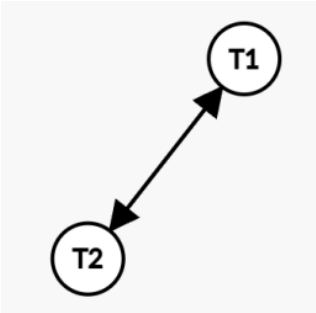
Transaction 2:

SQL:

```
BEGIN TRANSACTION;
SELECT * FROM Cart WHERE Customer_ID=1;
UPDATE Cart Set Amount = 5 WHERE Product_ID =1 AND Customer_ID = 1;
COMMIT;
```

Non-Conflict Serializable:

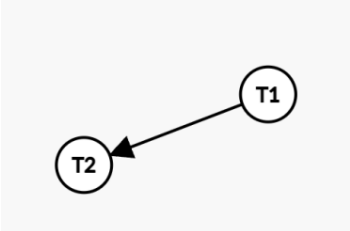
Transaction 1	Transaction 2
Begin Transaction	
Update Amount ←Amount+1	
	Begin Transaction
	Display entire cart of Customer_id:1
	Update Amount ← 5
	Commit
Display Sum total	
Display all the customers	
Commit	



This is non-conflict serializable because the precedence graph of the schedule has a cycle of length 2. Namely, there are Write-Read conflicts at time(2,4) and time(5,7). And a Write-Write conflict at time(2,5). We have conflicts from T1→T2 and T2→T1.

Conflict Serializable:

Transaction 1	**Transaction 2**
Begin Transaction	
Update Amount ←Amount+1	
Display Sum total	
	Begin Transaction
	Display entire cart of Customer_id:1
	Update Amount ← 5
	Commit
Show all the customers	
Commit	



This is conflict serializable because the precedence graph of the schedule has a valid sorting topological order, namely T1 followed by T2 as all the conflicts are from T1→T2.

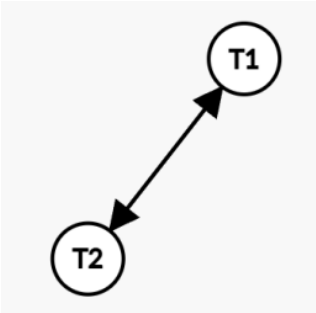
Schedule 2:

Transaction 1:
SQL:
BEGIN TRANSACTION;
SELECT Customer_Name from Customers WHERE Customer_ID = 1;
UPDATE Customers SET Customer_Email = 'sakshamp10@gmail.com' WHERE Customer_ID = 1;
SELECT * FROM Cart WHERE Customer_ID = 2;
SELECT * FROM Admin;
COMMIT;

Transaction 2:
SQL:
BEGIN TRANSACTION;
UPDATE Customers SET Customer_Name = 'Saksham' WHERE Customer_ID = 1;
INSERT INTO Cart(Product_ID,Amount,Customer_ID)
VALUES (1,2,4);
COMMIT;

Non-Conflict Serializable:

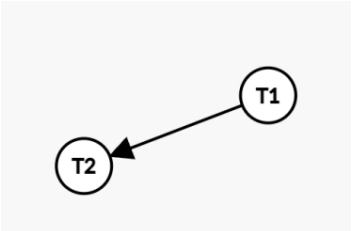
Transaction 1	Transaction 2
Begin Transaction	
Display Customer Name with id:1	
	Begin Transaction
	Update Customer Name with id:1
Update Customer email with id:1	
	Insert value into Cart
	Commit
Display the cart of customer id:2	
Display List of Admins	
Commit	



This is non-conflict serializable because the precedence graph of the schedule has a cycle of length 2. We have conflicts from T1→T2 and T2→T1.

Conflict Serializable:

Transaction 1	Transaction 2
Begin Transaction	
Display Customer Name with id:1	
Update Customer email with id:1	
Display the cart of customer id:2	
	Begin Transaction
	Update Customer Name with id:1
	Insert value into Cart
	Commit
Display List of Admins	
Commit	



This is conflict serializable because the precedence graph of the schedule has a valid sorting topological order, namely T1 followed by T2 as all the conflicts are from T1→T2.