Non - Conflicting Transactions:

1)Ride Cancellation:

```
start transaction;
set @customer=null;
set @driver=null;
set @amount=null;
set @trip=null;
select @customer:=Customer_ID, @driver:=Driver_ID from ride_request where
R_ID=70011;
select @trip:=T_ID , @amount:=Fare_Price from trip where R_ID=70011;
delete from trip where R_ID =70011;
delete from ride_request where R_ID=70011;
update customer set wallet =wallet-@amount/100 where Customer_ID= @customer;
update driver set wallet=wallet+@amount/100 where Driver_ID=@driver;
commit;
```

Description: If a ride is canceled then there is a penalty of 1 percent of the trip cost which is added to the driver wallet from the customer wallet. The trip with the corresponding R_ID is deleted.

2) Adding a new vehicle to a driver whose previous vehicle was damaged.

```
start transaction;
set @driv=NULL;
set @mystring =concat("HR",floor(rand()*100),"BF",floor(rand()*10000));
select @driv:= Driver_ID from vehicle where Number_Plate = "MH92IB9846";
DELETE FROM vehicle WHERE Number_Plate = "MH92IB9846";
INSERT INTO
vehicle(Driver_ID,Number_Plate,Seats_accomadation,Fuel,Color,Maintainance_state)
VALUES(@driv,@mystring,6,"EV","White","G");
commit;
```

Description: If a vehicle is damaged then a new vehicle is assigned to the driver.

3)Ride accepted:

```
start transaction;
set @customer=null;
set @driver=null;
set @amount=null;
set @ride=null;
select @customer:=Customer_ID, @driver:=Driver_ID from trip where R_ID=30011;
select @ride:=R_ID , @amount:=Fare_Price from trip where T_ID=30011;
update ride_request set status='A' where R_ID =@ride;

delete from trip where T_ID=30011;
delete from ride_request where R_ID=@ride;
delete from payment where T_ID=30011;
UPDATE customer
SET wallet = wallet+@amount where Customer_ID=@customer;

update driver set wallet=wallet+@amount where Driver_ID=@driver;

commit;
```

Description:
If a ride request is accepted by a driver, then the following trip is deleted along with the ride request.The customer and driver wallet are updated as per the fare price of the corresponding trip.

4) Updation of Customer status:

```
start transaction;
update customer set type="prime" where Customer_ID="50045";
update customer set wallet =wallet -4000 where Customer_ID="50045";
commit;
```

Description: If the status of the customer is updated then a fixed amount is deduced from the customer wallet.

Conflicting Transaction:

Schedule-1

T1: Add fare price to the driver's wallet

Read(Fare_Price)
Read(Driver's Wallet)
Driver's wallet is updated i.e Driver's wallet=Driver's wallet +Fare_Price
Write(Driver's Wallet)
Commit

T2: Add Cancellation fee to driver wallet

Read(Cancellation fee of the trip)
Read(Driver's Wallet)
Driver's Wallet is updated i.e Wallet + cancellation fee of the trip
Write(Driver's Wallet)
Commit

- Let's assume driver's wallet was 100 rupee and amount of Fare Price=1000 and cancellation fees is 100;
- Serial Schedule 1-> T1 –>T2 , wallet : 1200 at the end;
- Serial Schedule 2 -> T2 –>T1 , wallet : 1200 at the end;

Conflict Serializable:

T1	**T2**
Read(Fare_Price)	
	Read(Cancellation fee of the trip)

Read(Driver's Wallet)
Wallet=Wallet+Fare_Price
Write(Wallet)
Commit	

	Read(Driver's Wallet)
	Wallet=Wallet + cancellation fee of the trip
	Write(Wallet)
	Commit

Swapping non- conflicting steps to make it serializable.

T1	**T2**
Read(Fare_Price)	
Read(Driver’s Wallet)	
Wallet=Wallet+Fare_Price	
Write(Wallet)	
Commit	
	Read(Cancellation fee of the trip)
	Read(Driver’s Wallet)
	Wallet=Wallet + cancellation fee of the trip
	Write(Wallet)
	Commit

Non -Conflict serializable schedule:

T1	**T2**
Read(Fare_Price)	
	Read(Cancellation fee of the trip)
	Read(Driver’s Wallet)
Read(Driver’s Wallet)	
	Wallet=Wallet + cancellation fee of the trip
	Write(Wallet)
	Commit
Wallet=Wallet+Fare_Price	
Write(Wallet)	
Commit	

Here There is RW and WR anomalies which form a cycle causing the transaction to be non-conflicting serializable.

Solving the conflicts with locks

T1	**T2**
Lock1(Fare_Price)	
Read(Fare_Price)	
	Lock2(Cancellation fee of the trip)
	Read(Cancellation fee of the trip)
	Lock3(Driver’s Wallet)
	Read(Driver’s Wallet)
Lock3(Driver’s Wallet)	
Read(Driver’s Wallet)	
Wait..	Wallet=Wallet + cancellation fee of the trip
Wait…	Write(Wallet)
Wait..	**UnLock3**(Driver’s Wallet)
Wait..	**UnLock2**(Cancellation fee of the trip)
Wait..	Commit
Wallet=Wallet+Fare_Price	
Write(Wallet)	
UnLock3(Driver’s Wallet)	
UnLock1(Fare Price)	
Commit	

Schedule -2

T1: Updating status of customer:A fixed amount of 100 is deducted from the customer’s wallet when the status is updated from normal to prime.

Read(Status)
Write(Status)
Read(Cus_Wallet)
Cus_wallet=Cus_wallet-100;
Write(Cus_Wallet)
Commit

T2: Deducing the Fare Price of the trip based on customer status.

Read (Fare Price)
Read(Cus_wallet)
Cus_wallet=cus_wallet-Fare Price
Write (Cus_wallet)
Commit

- Let's assume there are Rs 1000 in the customer's wallet and the Fare Price of the trip is Rs 200.

T1→ T2 : serial : 1000-100-200=700
T2→ T1 : serial : 1000-200-100=700

Conflict Serializable:

T1	**T2**
Read(Status)	
Write(Status)	Read (Fare Price)
	Read(Cus_wallet)
Read(Cus_Wallet)	
Cus_wallet=Cus_wallet-100;	
Write(Cus_Wallet)	
	Cus_wallet=cus_wallet-Fare Price
	Write (Cus_wallet)
	Commit
Commit	

There is a W-W anomaly here which makes the conflicting however it can be converted into serializable form by swapping non-confliction reads and writes.

Swapping:

T1	**T2**
Read(Status)	
Write(Status)	
Read(Cus_Wallet)	
Cus_wallet=Cus_wallet-100;	

Write(Cus_Wallet)	
Commit	
	Read (Fare Price)
	Read(Cus_wallet)
	Cus_wallet=cus_wallet-Fare Price
	Write (Cus_wallet)
	Commit

Non-Conflicting serializable:

T1	**T2**
Read(Status)	
Write(Status)	
	Read (Fare Price)
	Read(Cus_wallet)
Read(Cus_Wallet)	
	Cus_wallet=cus_wallet-Fare Price
	Write (Cus_wallet)
Cus_wallet=Cus_wallet-100;	
Write(Cus_Wallet)	
Commit	
	Commit

Here There is RW and WR anomalies which form a cycle causing the transaction to be non-conflicting serializable.

Using locks to fix the non-conflicting serializable

T1	**T2**

Lock1(Staus)
Read(Status)
Write(Status)	

Unlock1(Status)	
	Lock2(Fare Price)
	Read (Fare Price)
	Lock3(Cus_wallet)
	Read(Cus_wallet)
Lock3(Cus_wallet)	
Wait..	
Wait…	Cus_wallet=cus_wallet-Fare Price
Wait..	Write (Cus_wallet)
Wait..	**Unlock3**(Cus_wallet)
Wait..	**Unlock3**(Fare Price)
Wait…	Commit
Read(Cus_Wallet)	
Cus_wallet=Cus_wallet-100;	
Write(Cus_Wallet)	
Unlock3(Cus_wallet)	
Commit	