INSTACAB -203

Case 1:-

Passenger makes a trip request

For non conflicting schedule :

Time	Transaction T1	Transaction T2
T1-1	Request by Passenger	
T1-2		Request by Driver 1
T1-3	Process request and assign Driver 1	
T1-4		
T1-5		Request by Driver 2
T1-6		Process request and assign Driver 2

Commit Transaction 1	
	Commit Transaction 2

In this schedule, both drivers can access the request at the same time without causing any conflict

Conflict Serializable Schedule :

Time	Transaction T1	Transaction T2
T1-1	Request by Passenger	
T1-2	Lock request	
T1-3		Request by Driver 1
T1-4	Process request and assign Driver 1	
T1-5	Unlock request	

T1-6		Lock request
T1-7		Request by Driver 2
T1-8		Process request and assign Driver 2

In this schedule, the second driver can only access the request after the first driver completes their access, ensuring that there is no conflict between the two transactions. However, this schedule is not recoverable, and if T2 were to fail after T1 unlocks the request but before T2 completes, then T1's changes would be lost.

Case:2)

Transaction 1:-

A passenger make a trip request and make a payment for it.

BEGIN TRANSACTION;

INSERT INTO TripRequest (DriverID, RiderID, RequestID, Date, Time, StartLocation, DropLocation, CarType) VALUES (1, 1, 1, '2023-04-25', '12:00:00', '123 Main St', '456 Oak St', 'Sedan');

INSERT INTO Payment (RiderID, TransactionID, PaymentDate, Amount) VALUES (1, 1, '2023-04-25', 25.00);

COMMIT;

Transaction 2:-

A driver accepts a trip request and updates the trip table with fare and rating

BEGIN TRANSACTION;

UPDATE TripRequest SET DriverID = 2 WHERE DriverID IS NULL AND RiderID = 1 AND RequestID = 1;

UPDATE Trip SET Fare = 25.00, TripRating = 4.5 WHERE DriverID = 2 AND RiderID = 1 AND TripID = 1;

COMMIT;

This schedule is conflict serializable because the two transactions do not conflict with each other. The first transaction only inserts data into the TripRequest and Payment tables, and the second transaction only updates data in the TripRequest and Trip tables. Therefore, the order of the transactions does not affect the outcome of the schedule.

Transaction 1	Transaction 2
Start Transaction 1	
Update Trip SET Fare=10 WHERE DriverID=1 AND RiderID=1 AND TripID=1	
	Start Transaction 2

	Update Payment SET Amount=10 WHERE RiderID=1 AND TransactionID=1
Commit Transaction 1	
	Commit Transaction 2

Here Transaction 1 updates the fare for a trip, and Transaction 2 updates the amount for a payment. Since these two transactions do not conflict with each other, the schedule is conflict serializable. This means that we can reorder the operations of the two transactions, and the final result will be the same.

For non conflict serializable:-

Transaction 1:-

A passenger requests a trip and updates their account balance.

BEGIN TRANSACTION;

INSERT INTO TripRequest (DriverID, RiderID, RequestID, Date, Time, StartLocation, DropLocation, CarType) VALUES (1, 1, 1, '2023-04-25', '12:00:00', '123 Main St', '456 Oak St', 'Sedan');

UPDATE Account SET Balance = Balance - 25.00 WHERE RiderID = 1;

COMMIT;

Transaction 2:-

A driver accepts the trip request and updates the trip table.

BEGIN TRANSACTION;

UPDATE TripRequest SET DriverID = 2 WHERE DriverID IS NULL AND RiderID = 1 AND RequestID = 1;

UPDATE Trip SET Fare = 25.00, TripRating = 4.5 WHERE DriverID = 2 AND RiderID = 1 AND TripID = 1;

COMMIT;

In this schedule, the two transactions conflict with each other because they both update the Account table. If Transaction 2 is executed before Transaction 1, it would result in an incorrect account balance, which violates the consistency of the database. Therefore, this schedule is not conflict serializable.

Transaction 1	Transaction 2
Start Transaction 1	
Update Payment SET Amount=10 WHERE RiderID=1 AND TransactionID=1	
	Start Transaction 2
	Update Account SET Balance=100 WHERE RiderID=1
Update Trip SET Fare=10 WHERE DriverID=1 AND RiderID=1 AND TripID=1	

Commit Transaction 1	
	Commit Transaction 2

It consists of the same two transactions as the first table. But, in this case, Transaction 1 updates the payment amount before Transaction 2 updates the account balance. This creates a conflict between the two transactions since they both affect the same row (i.e., the same RiderID). As a result, the schedule is not conflict serializable, and we cannot reorder the operations of the two transactions without changing the final result. This can lead to data inconsistencies and other issues, which is why it is important to ensure that schedules are conflict serializable