

Transactions

Non-Conflicting Transactions:-

1.Refund Transaction

TRANSACTION T1

START TRANSACTION;

```
UPDATE trip SET status = 'Cancelled' WHERE id = 789;
UPDATE payment SET status = 'Refunded' WHERE id = 456;
UPDATE wallet SET money = money + 500 WHERE id = 123;
IF ROW_COUNT() = 0 THEN
    ROLLBACK;
ELSE
    COMMIT;
END IF;
```

TRANSACTION T2

START TRANSACTION;

```
UPDATE trip SET status = 'Cancelled' WHERE id = 100;
UPDATE payment SET status = 'Refunded' WHERE id = 200;
UPDATE wallet SET money = money + 500 WHERE id = 300;
IF ROW_COUNT() = 0 THEN
    ROLLBACK;
ELSE
    COMMIT;
END IF;
```

SCHEDULE :-

T1	T2
Read(Trip)	
Write(Trip)	
	Read(Trip)
	Write(Trip)
Read(Payment)	
Write(Payment)	
	Read(Payment)
	Write(Payment)
Read(Wallet)	
Write(Wallet)	
COMMIT	
	Read(Wallet)
	Write(Wallet)
	COMMIT

EXPLANATION:- This transaction executes whenever a trip is cancelled, so it first updates the status of a trip to “Cancelled”, then updates the payment status to “Refunded”, and updates the customer's wallet with the amount he pays for the trip, then there is a consistency check condition to maintain consistency across database so it uses **ROW_COUNT()** function for this if it returns 0 then it means money is not refunded back to the customer hence it rolls back the entire transaction; otherwise, it commits the transaction.

2.Book a Ride

TRANSACTION T1

START TRANSACTION;

```
INSERT INTO booking (custId, pickupStreet,
destinationCity,bookingTime, price)
VALUES (1002, 'Govindpuri', 'Harkesh Nagar', NOW(), 500);
UPDATE wallet SET money = money - 500 WHERE id = 1002 AND
money >= 500;
IF ROW_COUNT() = 0 THEN
    ROLLBACK;
ELSE
    COMMIT;
END IF;
```

TRANSACTION T2

START TRANSACTION;

```
INSERT INTO booking (custId, pickupStreet,
destinationCity,bookingTime, price)
VALUES (1003, 'Kashmere Gate', 'Raj Bagh', NOW(), 600);
UPDATE wallet SET money = money - 600 WHERE id = 1003 AND
money >= 600;
IF ROW_COUNT() = 0 THEN
    ROLLBACK;
ELSE
    COMMIT;
END IF;
```

SCHEDULE :-

T1	T2
Write(Booking)	
	Write(Booking)
Read(Wallet)	
Write(Wallet)	
COMMIT	
	Read(Wallet)
	Write(Wallet)
	COMMIT

EXPLANATION:- This transaction executes whenever a new booking request is generated by the customer, so it first inserts a new booking record in the booking table, then it tries to deduct the booking amount from the customer's wallet, but if the current available amount in customer's wallet is less than the booking amount then whole transaction is rolled back; otherwise, it commits the transaction.

Conflicting Transactions:-

1.Location Update Conflict

TRANSACTION T1

START TRANSACTION;

UPDATE car

SET location = 'Govind Puri'

WHERE id = (SELECT RC FROM driver WHERE status = 'Active');

COMMIT;

TRANSACTION T2

START TRANSACTION;

UPDATE car

SET location = 'Kashmere Gate'

WHERE id = (SELECT RC FROM driver WHERE status = 'Active');

COMMIT;

SCHEDULE:-

T1	T2
Shared Lock(Driver)	
Read(Driver)	
	Shared Lock(Driver)
	Read(Driver)
Exclusive Lock(Car)	
Write(Car)	

COMMIT	
Release Shared Lock(Driver)	
Release Exclusive Lock(Car)	
	Exclusive Lock(Car)
	Write(Car)
	COMMIT
	Release Shared Lock(Driver)
	Release Exclusive Lock(Car)

EXPLANATION:- When these transactions are executing concurrently and assume there is only 1 available car, then if T1 comes first and updates the car location, then before it commits changes, the transaction manager switches to the T2 and starts executing it, and T2 also updates the location of the same car and commits after that T1 commits then changes made by T2 will become irrecoverable hence this leads to a conflict between two transactions.

2.Car Deletion and Booking Acceptance

TRANSACTION T1

START TRANSACTION;

UPDATE car

SET location = 'Govindpuri'

WHERE id = (SELECT RC FROM driver WHERE status = 'Active');

UPDATE booking

SET RC = (SELECT RC FROM driver WHERE status = 'Active' AND

RC = (SELECT RC FROM car WHERE location = 'Govindpuri')

LIMIT 1),status = 'Accepted' WHERE id = 456;

COMMIT;

TRANSACTION T2

START TRANSACTION;

UPDATE driver

SET status = 'Inactive'

WHERE id = (SELECT RC FROM booking WHERE id = 456);

DELETE FROM car WHERE id = (SELECT RC FROM driver WHERE id =
(SELECT RC FROM booking WHERE id = 456));

COMMIT;

SCHEDULE :-

T1	T2
Shared Lock(Driver)	
Read(Driver)	
Exclusive Lock(Car)	
Write(Car)	
	Shared Lock(Booking)
	Read(Booking)
	Exclusive Lock(Driver)
	Write(Driver)
Read(Car)	
Read(Driver)	
	Read(Booking)
	Read(Driver)
	Exclusive Lock(Car)
	Write(Car)
	COMMIT
	Release Shared Lock(Booking)
	Release Exclusive Lock(Driver)
	Release Exclusive Lock(Car)
Exclusive Lock(Booking)	
Write(Booking)	
COMMIT	

Release Shared Lock(Driver)	
Release Exclusive Lock(Car)	
Release Exclusive Lock(Booking)	

EXPLANATION:- When these transactions are executing concurrently, then T1 updates the location of a car and also updates the booking's RC to the assigned car's RC and status to "Active". Then before T1 commits changes, the transaction manager switches to T2 and starts executing it, where it updates the driver's status to "Inactive" and deletes the car assigned in T1, then gets committed. This would lead to a conflict because T2 deletes the car that has been assigned to T1, and then T1 will commit, and the booking is completed based on a car that no longer exists because T2 already deleted that car.