

(Q1)

- a) (i) Atomicity: The transaction did not complete fully; one part (wallet deduction) succeeded while the other (ride confirmation) did not.
- (ii) Consistency: The system ended up in an invalid state money deducted but no confirmed ride.
- (iii) Durability: The system failed to ensure that the correct final result (ride creation + wallet deduction) was stored persistently before the crash.
- b) The failure occurred because the steps involved in booking the ride were not executed as single atomic transaction. Since these were executed in separate database calls or microservice transactions, the server crashed between steps.
- c) (i) The customer requests a ride.
- (ii) The system opens a single atomic transaction.
 Step A: Receive (not deduct) wallet balance in a Pending state.
 Step B: Create a ride record with status "Pending Confirmation".
- (3) Only if both step A & step B succeed, the transaction is committed.
- (4) After commit:
 • The app shows the ride confirmation to the customer.
 • The reserved amount is converted to a final deduction when the ride starts.

- d) (1) Financial losses : Users charged without receiving a service cause forced refunds, disputes, & support overhead.
- (2) Loss of customer trust, user uninstall the app if money disappears without confirmed ride.
- (3) ~~few~~ drivers think they accepted ride that customers never saw. Leads to wasted time & frustration.

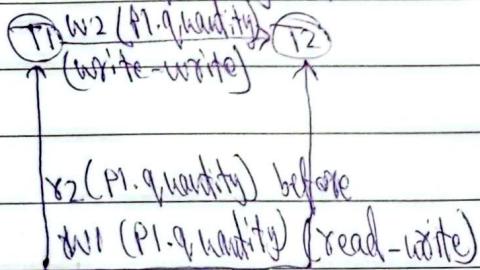
(Q2)

- a) Non-serial schedule :

$$r_1(P1.\text{quantity}), r_2(P1.\text{quantity}), w_1(P1.\text{quantity_sold_unit})$$

$$w_2(P1.\text{quantity} + \text{returned_units})$$

- b) $w_1(P1.\text{quantity})$ before



- c) The schedule is not conflict serializable because the Precedence graph contains a cycle, indicating no equivalent serial order without conflicts.

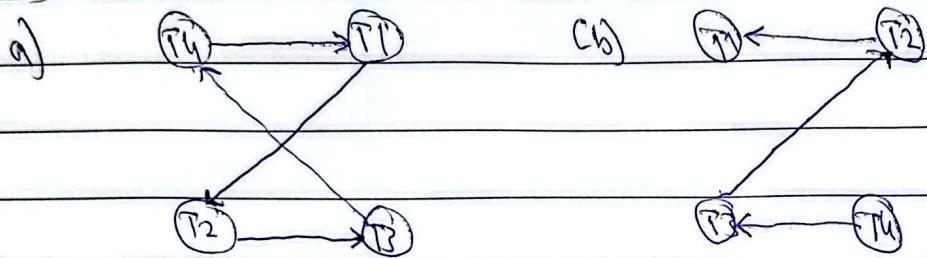
- d) Corrected Schedule :

$$r_1(P1.\text{quantity}), w_1(P1.\text{quantity_sold_unit}), r_2(P1.\text{quantity}),$$

$$w_2(P1.\text{quantity} + \text{returned_units})$$

c) Inventory inaccuracies could lead to overstocking / understocking, causing lost sales or excess holding costs. This might result in financial losses from return / refunds, damaged customer relationships.

(Q3)



b) Not conflict-serializable (cycle)

Conflict serializable (acyclic)

c) Not Serializable

The serial schedule

is $T_4 \rightarrow T_3 \rightarrow T_2 \rightarrow T_1$

(Q4)

a) The concurrency problems are lost update & dirty read (as more precisely a write-write conflict leading to lost update). They arise because both transactions read the same initial value (2000) without isolation, complete independently & overwrite each other.

b) The final amount is 2800 after both transactions finish.

c) $T_1 \rightarrow T_2$

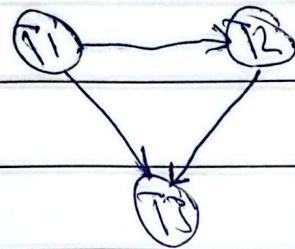
$$2000 - 300 \Rightarrow 1700 + 800 \Rightarrow 2800$$

 $T_2 \rightarrow T_1$

$$2000 + 800 \Rightarrow 2800 - 300 \Rightarrow 2500$$

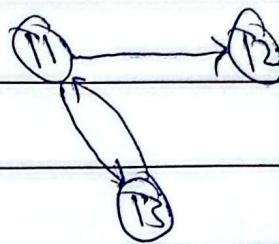
(a)

g)

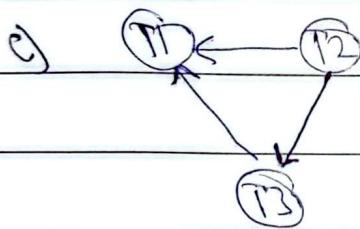


Conflict serializable (acyclic)

(b)

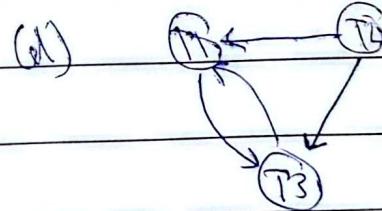
Not conflict serializable
(cyclic)

Serial schedule: T1 → T2 → T3

Not conflict serializable
(cycle)

Conflict serializable (acyclic)

Serial schedule: T2 → T3 → T1

Not conflict serializable
(cyclic)