Haris Ahmed

23K-6005

BSAI-5A

DBS Assignment 4

Date:

QUESTION 1:

(a) ACID property / properties violated

The main ACID property violated is Atomicity - the transaction did not execute as an "all or nothing" unit. Additionally, Consistency is also violated because the system ended up in an invalid state (driver accepted + money deducted but no ride confirmation)

(b) Technical explanation of why the violation occurred

The operations (driver acceptance write + wallet deduction write + ride confirmation) were not executed inside a single atomic database transaction.

Because of that

- Some statements were committed to the database individually (driver accepted, wallet deducted)
- Before the final part of the workflow (ride confirmation) could execute or commit, the server crashed.
- Since there was no atomic commit / rollback boundary, the system could not revert the earlier updates.

Thus the system reached a partially executed state, which breaks Atomicity, and therefore the database became inconsistent, breaking Consistency.

(c) Correct Scenario that preserves all ACID properties

To preserve ACID, the entire ride-booking workflow must run inside one atomic transaction:

1. BEGIN TRANSACTION
2. Record driver Acceptance.
3. Deduct amount from customer wallet.
4. Prepare and save ride confirmation
5. COMMIT only if all steps succeed.

If any error occurred at any steps (including a server crash):

- The database automatically performs a ~~rollback~~ ROLLBACK, undoing driver acceptance and wallet deduction.
- No partial updates remain → Atomicity + consistency preserved.
- Other concurrent operations cannot see intermediate states → Isolation preserved.
- Once committed, updates persist even if a crash happens → Durability preserved.

This ensures the system ends in either:

- State 1: Ride accepted & payment reserved & confirmation shown (everything committed), or
- State 2: Nothing changed (everything rolled back)

(d) Long-term business consequences of frequent violations

Frequent ACID violations can cause:

- Customers trust issues: Users may get charged without getting rides → loss of confidence
- Financial discrepancies: wallet refunds, reconcilictions, double-charging problems.
- High support /operational cost: more ticket's, manual corrections, and escalations.
- Reputation damage: Negative reviews → fewer customers and drivers.
- Regulatory /legal risks: Payment processors and financial regulators may penalize the company for inconsistent transaction handling.

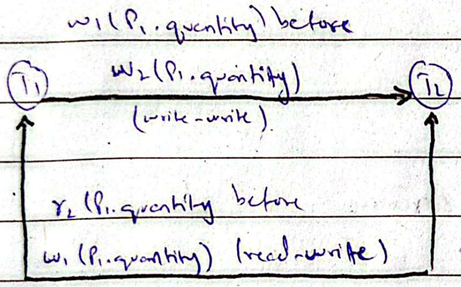
Ultimately, it can lead to loss of revenue, poor brand image, and potential legal consequences.

QUESTION 2:

(a) non-serial schedule:

r_1(P_1.quantity), r_2(P_1.quantity), w_1(P_1.quantity-sold_units), w_2(P_1.quantity + returned_units)

(b)



w_1(P_1.quantity) before
w_2(P_1.quantity)
(write-write)
T_1
T_2
r_2(P_1.quantity before
w_1(P_1.quantity) (read-write)

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

(d) Corrected schedule:

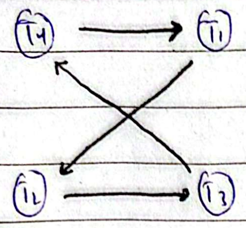
r_1(P_1.quantity), w_1(P_1.quantity-sold_units), r_2(P_1.quantity), w_2(P_1.quantity + returned_units)

(e) Inventory inaccuracies could lead to overstocking /understocking, causing lost sales or excess holding costs. This might ~~rest~~ result in financial losses, from return / refunds, damaged customer relationships, supply ~~dele~~ chain disruptions, and reputational harm if products are frequently out of stock

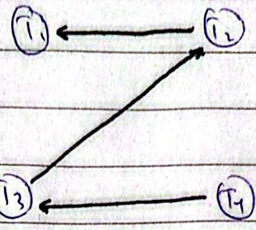
Date: ____________

QUESTION 3:

(a)



(b)



(b) Not conflict Serializable (cycle)

(c) Not serializable

conflict - serializable (acyclic)

The serial schedule is T4->T3->T2->T1

QUESTION 4:

(a) The concurrency problems are lost update and Dirty Read (as more precisely, a write-write conflict leading to lost update). They arise because both transaction read the same initial value (2000) without isolation, compute independently and overwrite each other

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

(c) T1 -> T2:

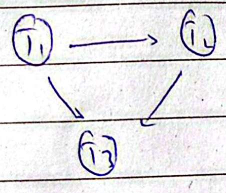
2000-300 = 1700 + 800 = 2500

T2 -> T1:

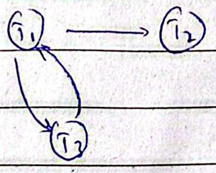
2000 + 800 = 2800 - 300 = 2500.

QUESTION 5:

(a)



(b)

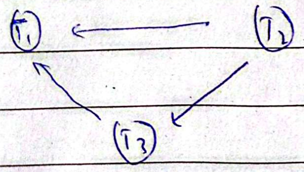


conflict Serializable (acyclic)

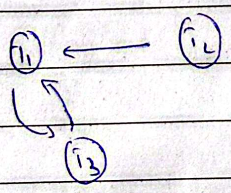
Serial schedule: T1->T2->T3

Not conflict Serializable (cycle)

(c)



(d)



Conflict Serializable (acyclic)

Serial schedule : T2 -> T3 -> T1

Not conflict Serializable. (cyclic)