23K0051 BS (AI)

a. Atomicity + Consistency

b. The system experienced a violation because it failed to implement an atomic transaction. It ended up booking in multiple steps. The driver acceptance and wallet deduction were written to the database, however, before the final ride confirmation was completed the server crashed leaving only the transaction partially executed. This violates Atomicity as the process did not adhere to an "all or none" rule, and it infringes on consistency because funds were withdrawn without a valid trip being established.

c. The system ought to initially secure the required information via appropriate locking mechanisms. Within the transaction block, log the driver's agreement, subtract from the account, and then finalize the reservation. These modifications remain hidden until the transaction is committed. Should the server fail prior to commitment, everything is reversed, preventing any incomplete alterations. Upon successful commitment, durability ensures the changes persist post failure.

d. Such problems can undermine user confidence, decrease reservations and lead to greater monetary setbacks from reimbursements and disputes. They also escalate customer service expenses, harm the brand's image, and could result in oversight or judicial repercussions.

Q2.

P1 = 100 , sold = 3 returned = 2

Non serial:

1. R1 (Q=100) 3. W1 (Q = 97)

2. R2 (Q=100) 4. W2 (Q = 102)

b.

R1 (Q) ⟶ W2 (Q) : T1 ⟶ T2

R2 (Q) ⟶ W1 (Q) : T2 ⟶ T1

W1 (Q) ⟶ W2 (Q) : T1 ⟶ T2

c. The graph has a loop (cycle). Plan is not conflict serizable.

d. T1 ⟶ T2 (serial Execution)

R1 (100) ⟶ W1 (97)

R2 (97) ⟶ W2 (99) Final Quantity = 99.

T2 ⟶ T1 (serial Execution)

R2 (100) ⟶ W2 (102)

R1 (102) ⟶ W1 (99) Final Quantity = 99

e. These irregularities may result in wrong inventory counts. causing excess sales, order cancellations, unhappy customers, higher reimbursements and lost income. In the long run, this erodes the business's standing and user loyalty.

Q3

<u>Schedule 1:</u>

1. r4(a) before w1(a) : T4 → T1

r1(b) before w2(b) : T1 → T2

r2(c) before w3(c) : T2 → T3

r3(d) before w4(d) : T3 → T4

T4 → T1 → T2 → T3 → T4

∴ Therefore cycle exists.

2. S1 → Not conflict seriazable

3. None Exists

<u>Schedule 2:</u>

1. w4(d) before r3(d) T4 → T3

w3(c) before r2(c) T3 → T2

w2(b) before r1(b) T2 → T1

r4(a) before w1(a) T4 → T1

2. Yes. S2 is conflict seriagable

3. T4 → T3 → T2 → T1

Q4

Schedule experiences a lost update problem. Both transactions (T1, T2) read the same initial value (2000) before either one writes back its update.

T1 : 2000 − 300 = 1700   T2 : 2000 − 800 = 2800

T1 is overwritten by T2 because there was no synchronization or locking to prevent both transactions from modifying the same data.

2. 2800 (Rs)

3. case 1:

T1 ⟶ T2

After T1 = 2000 − 300 = 1700

After T2 = 1700 + 800 = 2500

case 2:

T2 ⟶ T1

Start 2000

After T2 = 2000 + 800 = 2800

After T1 = 2800 − 300 = 2500

Q5

Conflicts and Edges:

w1(x) → r3(x) : T1 → T3

w1(x) → r2(x) : T1 → T2

r2(x) → w3(x) : T2 → T3

There is no cycle, conflict serializable

Equivalent serial order: T1 → T2 → T3

b. Conflicts and Edges

r1(x) → w3(x) : T1 → T3

r3(x) → w1(x) : T3 → T1

w3(x) → w1(x) : T3 → T1

cycle Exists, Not conflict serializable.

Precedence Edges

T1 → T3

T3 → 1

c. Conflicts and Edges

r2(x) → w3(x) : T2 → T3

r2(x) → w1(x) : T2 → T1

w3(x) → r1(x) : T3 → T1

Precedence

T2 → T3

T2 → T1

T3 → T1

No cycle, conflict serializable

Equivalent serial Order : T2 → T3 → T1

d. Conflict and Edges.

$r_3(x) \longrightarrow w_1(x)$ $T_3 \longrightarrow T_1$

$r_2(x) \longrightarrow w_3(x)$ $T_2 \longrightarrow T_3$

$r_2(x) \longrightarrow w_1(x)$ $T_2 \longrightarrow T_1$

$r_1(x) \longrightarrow w_3(x)$ $T_1 \longrightarrow T_3$

Cycle Exists Non conflict serializable.

Precedence Edges

$T_3 \longrightarrow T_2$

$T_1 \longrightarrow T_3$

$T_2 \longrightarrow T_3$

$T_2 \longrightarrow T_1$