Solutions

Marking: **M1** a correct method, **A1** the correct answer, **B1** an independent mark.

Each answer shows the steps, then the **result**.

2.1

atomicity, consistency

isolation, durability

e.g. a transfer takes money from one account and adds it to another: both happen or neither

2.2

a non-repeatable read

T1 read the same row twice in one transaction and got different values

2.3

READ COMMITTED

it reads only committed data, but another transaction may commit a change between two reads

2.4

two buyers can both read `n = 1` before either writes

`SELECT seats ... FOR UPDATE` locks the row until commit

or one statement: `UPDATE show SET seats = seats - 1 WHERE id = 1 AND seats > 0` and check one row changed

3.1

(a) $T_1 \to T_2$: $w_1(A)$ before $r_2(A)$; $T_2 \to T_1$: $w_2(B)$ before $r_1(B)$

(b) no: the graph has a cycle

(c) T1 holds its lock on A until it commits, so $r_2(A)$ waits, and T2 runs after T1

(d) yes: T1 holds A and waits for B while T2 holds B and waits for A; the database detects the cycle and aborts one

3.2

(a) T1 committed, so redo its updates of A and B from the log

(b) T2 did not commit, so undo any of its changes that reached disk, using the old values in the log

(c) otherwise a changed page could be on disk with no log record to undo it after a crash

3.3

(a) two requests both read `seats_left = 1`; both write 0 and both sell a seat: 1001 tickets

(b) `UPDATE concert SET seats_left = seats_left - 1 WHERE id = 1 AND seats_left > 0;` and sell only if one row was changed

(c) $1/0.002 = \mathbf{500}$ a second; $50\,000/500 = \mathbf{100}$ **s**

(d) put requests in a queue and answer "sold out" at once after 1000 succeed; the atomic update keeps it correct, and the queue stops 49 000 hopeless requests from waiting on the lock; or split the seats over several rows

exercises.lol exercises.com.cn