Agenda

- what are Transactions
- properties of Transactions
  - A, C, I, D.

practical
- How do Transactions work ✓
- commits and rollback. ✓

mostly next class.
- Transaction isolation levels. ✓
- deadlocks

begin @ 9:10

Transactions

- standalone
- operations (query) which have to done Together.

Initial Balance = 1000/-

Init Balance = 5000/-

accounts

id	name	balance	created_at	...
1	A	1000	...	
2	B	5000		

① cheaking balance of A.
(a) get balance of A. ⇒ x (DB call).
(b) transfer_amount <= x. (application).

② reduce 500 from A. (DB call). update.

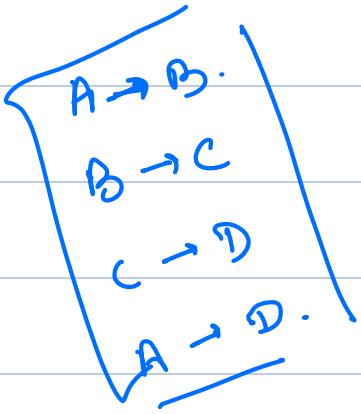
③ add / increase 500 to B. (DB. call).

A → B (operation)

3 queries
- get
- update
- update.

transfer_money (from, to, amount).

{ multiple SQL queries and some condition checks. }



Parallel calls.

A → B (500)

C → B. (5000).

init 1000 | 500 ?
500

C 10000 | 5000 ?
5000.

B 5000. | 10,000 || 5500 ?
10,500

10000 500
1000 5000
5000 5500
16K. 11K.

16K in Bank.
11K in Bank.

→ request (both) at same time.

→ get balance; math; update from amount ; update To amount.

A —500→ B.

C —5000→ B

① get Balance A = 1000.

② reduce Balance A To 500.

DB down

get balance = 10K.

reduce Blance. To 5000.

increase balance. To 5500

increase balance. To 15000

B = 5000.

B += x. ⇒

Temp = 5000

Temp = 5000 + x

B = (5000 + x)

① inconsistent (isolation)

② DB down. (atomicity)

all the queries were not executed.

x = 5.

a = 10 b = 15.

Transactions:

A set of dB operation cogically grouped together to perform a task.

Properties of Transactⁿ

A → atomicity

C → consistency

I → Isolation

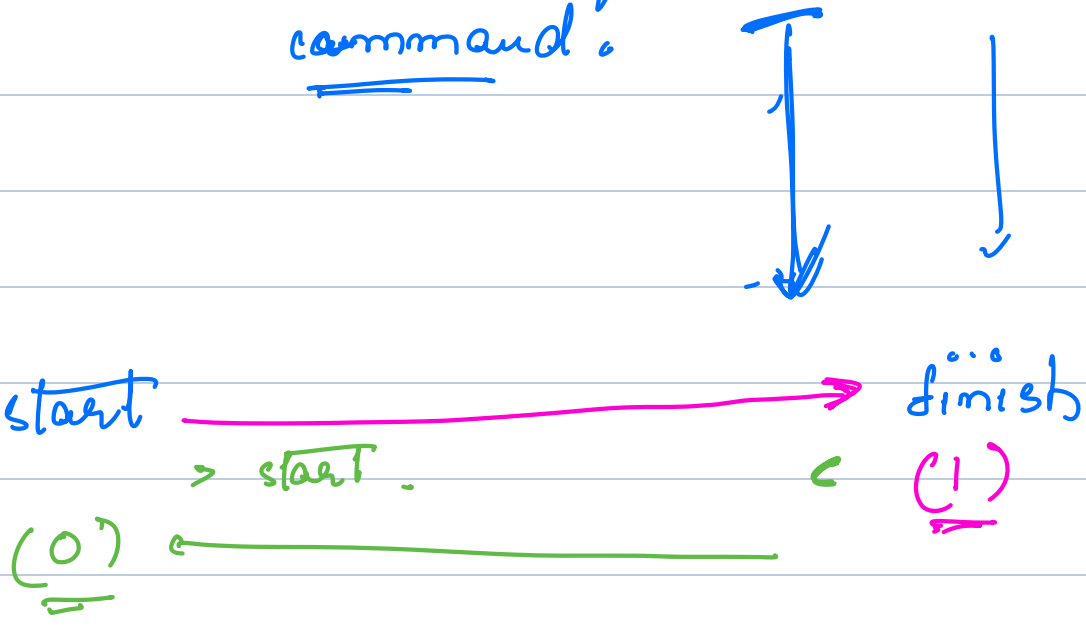
D → Durability

Atomicity

Atom

single smallest indivisible unit.

concurrent programming : no context switching → nothing can affect the execution of an atomic command.



its either (0, 1)

everything or nothing at all.

Transaction

- get ✓
- update A. ✓
- update B ✓

break
till 10:35

consistency
- correctness.
- reliability

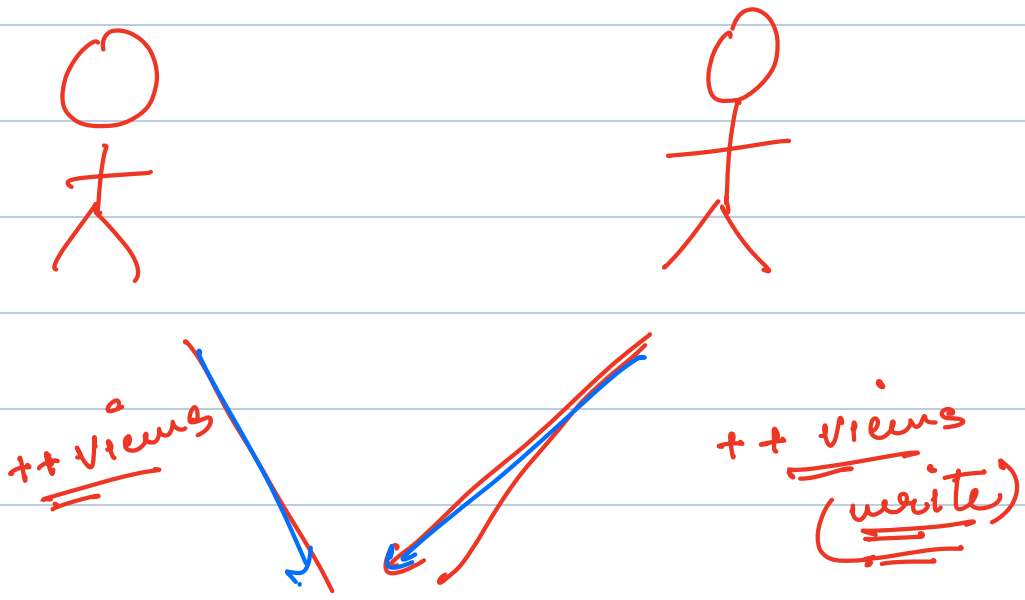
state (1) ⟶ state (2)

(1) ⟶ transitn (2) valid

consistency means that the integrity of DB is maintained after a transitn from a valid state to another valid state

after a transaction

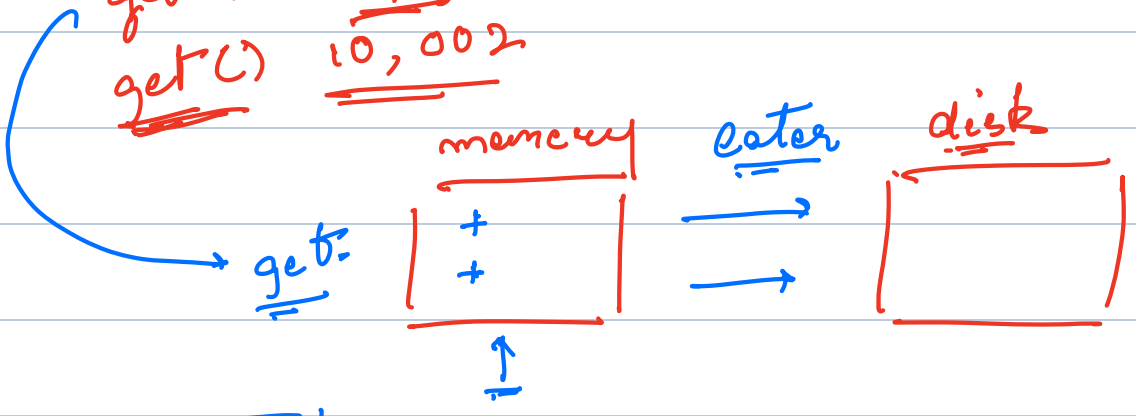
Money cost invalid state ✓



10,000 views

Tradeoff

get() 10,000
get() 10,002



CAP Theorem

consistency for availability

no sql

use-case

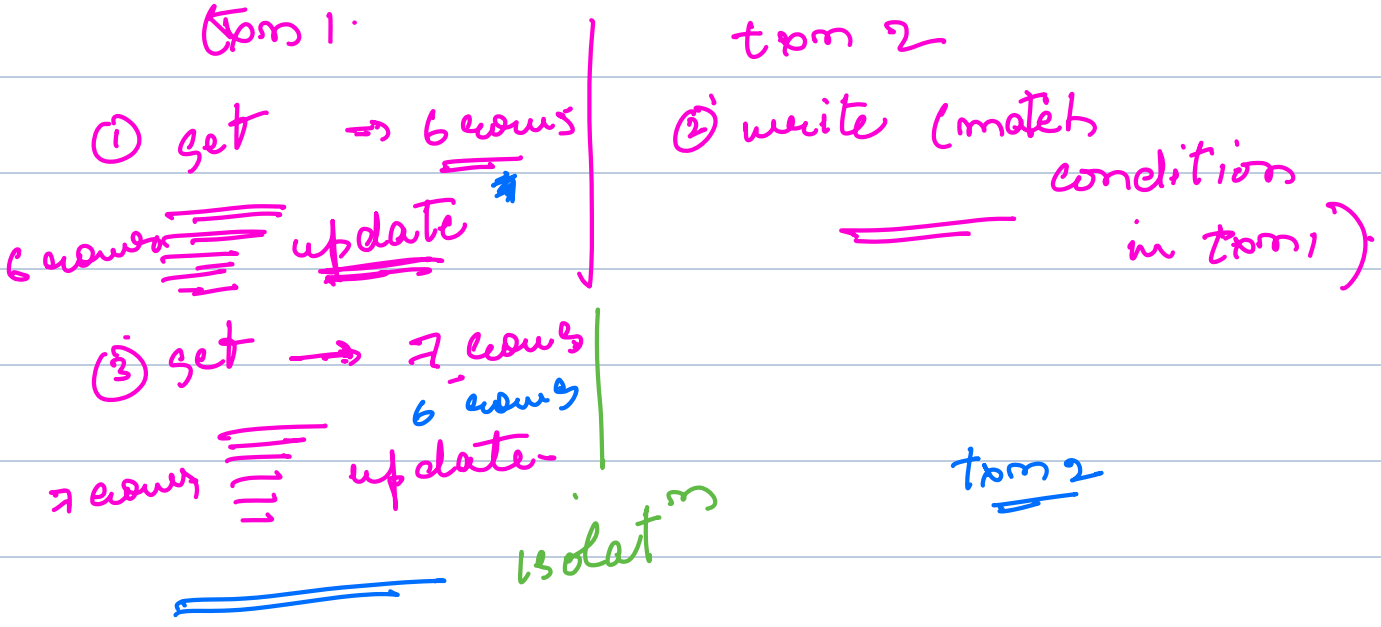
Isolation

Isolation means one transact^{n} should not affect another txn running at same time in DB in a wrong way.

no isolation led to no consistency

dirty read → read uncommitted data

Phantom read →.



Durability

once a transact^{n} is completed. we would want it to be persistant, {on disk}.

redis (K-V database).

(data in memory).

(tradeoff - durability
speed).

① Atomicity - Everything happen or nothing
② consistency - whatever happen, should be valid at the end.
③ Isolation - one txn should not interfere with another, to give unexpected outcomes.
④ Durability → once an outcome has been achieved, it should persist.

commits and rollbacks

id	name	psp	batch
1	Ayush	70	2

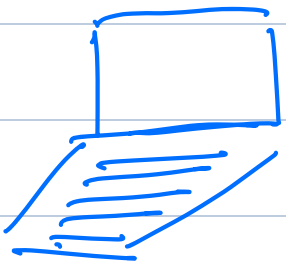
```
update students
set PSP = 80
where id = 1.
```

```
delete from students
where id = 1.
```

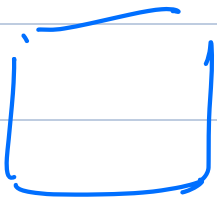
select * from ... ;

PSP ; id = 1

assumpt^{n} : whatever is updated in DB.



repo : Github



① → ②

state → state

update { q1, q2, q3

auto commit = false.

select

txn ①
update
select
commit

txn 2.
select *

what property is taken care of when we commit?

Durability

A → B 500.

tom !

① check balance.

② run async fraud check. (another Thread).

③ update A balance.

④ update B balance.

main.

A ✓ → commit

A (fraud).

finished updating A balance. ⑤ ..

rollback;