Transactions - I

Agenda

1. Transactions
2. Properties of a transaction
   - → A
   - → C
   - → I
   - → D

   } Imp for interviews
3. Read and write op"
4. Commit and Rollback
5. Transaction Isolation level
   - → lost updates
   - → dirty reads
   - → Phantom reads
   - → Non-repeatable reads

   } Imp for interviews
6. Deadlocks

Practical.

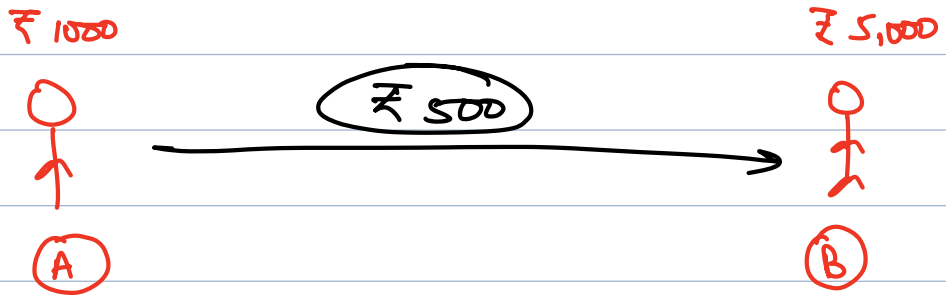
A bit more complex & theoritical class.

What are TRANSACTIONS

Till now we have used SQL for a single query. What if we need to perform a group of operations to achieve a task?

Bank Example

Transfer money



for this transaction / transfer to happen what changes do we need in the accounts table.

Accounts

id	name	balance	created-At	branch-id
1	A	₹ 1000	—	—
2	B	₹ 5000	—	—

1. Check balance of A → DB call → Read
2. Check > 500 ⇒ Application code
3. Reduce balance of A by 500 → DB call - Update
4. Increase balance of B by 500 → DB call - Update

```
transferMoney ( from, to , amount) {

    if ( bal >= amount) {
        ——
        ——
    } else {
    }

}
```

do you think only one person will be calling this function at a point of time?

① transferMoney may be getting executed by multiple people at the same time.

A → B (₹500)

C → B (₹500)

eg ecommerce – Amazon

① Check balance of A — Read A
② Check > 500 — Nor read/write
③ Reduce balance of A — write
④ Increase balance of B — read & write

B += 500 → This is NOT a single operation.

↓

B = B + 500 → first some has to read the previous value of B.

Read B → temp
temp → temp + 500
Write B ← temp

① A B 500
② C B 10000

transferMoney (a , b , amount) {

If database went down after this point?

Read A ⇒ X ① ⇒ 1000 ② ⇒ 15000
if (x >= amount) {
write A ← X - amount ① → 500 ② 1000
Read B ⇒ X ① → 5000 ② → 5000
x = x + amount
write B ⇐ X ① → 5500 ② → 15000

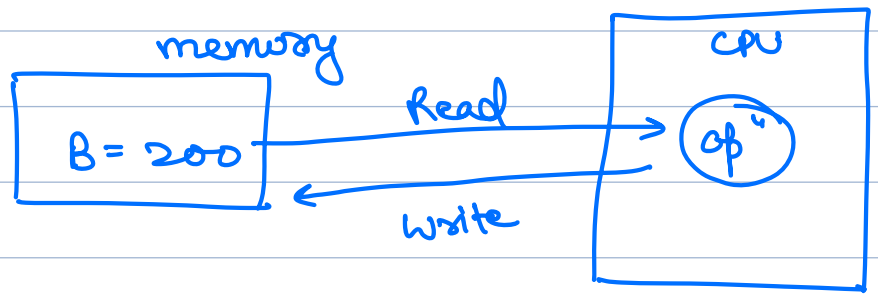
A → ~~10000~~ 500
B → ~~5000~~ ~~5500~~ 15000
C → ~~15000~~ 5000

21000 Initial sum
20500 final sum

}

write
Two operations can't happen at the exact same time. It will be one after the other.

How a system works



① Inconsistent / illogical state

② Entire operation may not complete

Transaction — A set of DB operations logically grouped together to perform a task.

Transfer money:

logical group of transaction

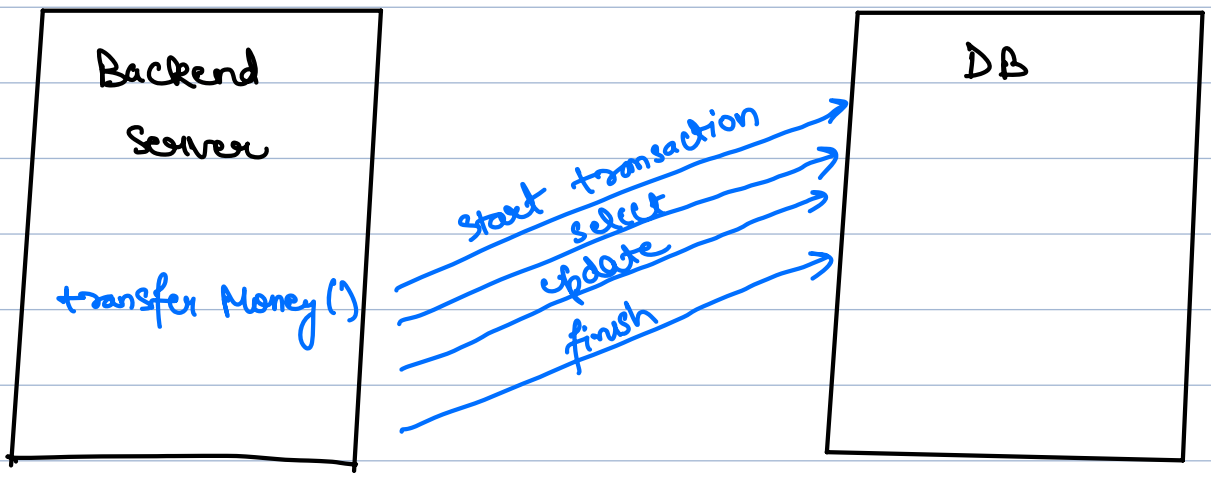
A → X

A ← X - _

B → X

B ← X + _

Start transaction



As we learnt when we group a lot of operations, a number of things can go wrong.

Expections / Properties of a Transaction.

#Properties that we MAY want from our transactions at times.

(ACID Properties).

A → Atomicity
C → Consistency
I → Isolation
D → Durability

ATOMICITY

↓

from the word Atom → Smallest 'indivisible' unit of mass.

⇒ Transaction should appear indivisible to an end user.

→ to an outsider it should feel that either nothing has happened or everything has happened.

[behave like ONE single unit / operation]

→ A transaction should never end in an intermediary state.

→ never the DB in between

CONSISTENCY

→ correctness

→ exactness

In the above example atomicity was handled → Transaction either happened & money transferred or nothing.

But work was not correct. Final outcome was wrong.

→ Accuracy

→ Logical correctness

Example where consistency might not be needed.

count → 1.6 cr

(A) → ← (B)

	A	B
①	16000000	16000000
②	16000001	16000001

① get current count

② update count

Hotstar

Live IPL match

stream_id	count
—	

Consistency hampered

BUT performance improved

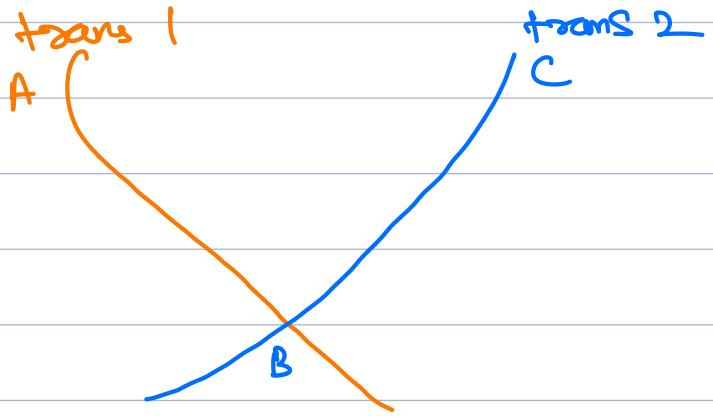
HAPPY INDIA.

The state of DB before trans" stored and also after tran" completed should be accurate.

ISOLATION

→ One transaction shouldn't affect other transaction running at the same time running on the same data.

In the above case, we would'nt have any issue if only one transaction was taking place. The 2 transactions were not isolated.



If you have 2 gf/bf if they dont interfere, you will not have any issue.

DURABILITY

Once a transaction is completed we would want its work to stay Persistent.

→ Store it on disk.

Seems obvious but may not be needed.

You might think when you do write operation to DB, it instantly writes to disk. But NO

In case where DB has to provide fast writes, it might do something like a batch write.

durability lost

→ Ques can be what if disk gets corrupted
→ Data centre catches fire
→ Alien invasion.

} Different levels of durability

[BREAK]

Commit and Rollback

Students

id	name	psp	batch_id
1	Ujjwal	80	2

```
update student
set psp = 100
where id = 1
```

```
select * from students.
```

we expect psp=100.

This is based on assumption, whatever we did in the first trans" worked.

This worked because our query's result got stored / saved to the dB.

WHY

Autocommit

→ every SQL query starts a transaction.
→ executes itself.
→ saves the changes // commit

What is commitment ?

→ Ask commitment from boss for promo
→ Committed relationship

1. Start transaction
2. Execute operation
3. Commit
   ↳ statement that is used to persist the results of an SQL query.

SQL query only makes changes but dont commit them to the database.
These are saved only after commit.

★ Commit takes care of durability.

Autocommit = true By default

[code] (make two sessions)

→ set autocommit = 0;

→ select * from film where film_id = 10;

→ update film
set title = 'Ujjwal'
where film_id = 10;

→ commit;

Workbench takes snapshots of database per session to achieve this.

—→ — r —

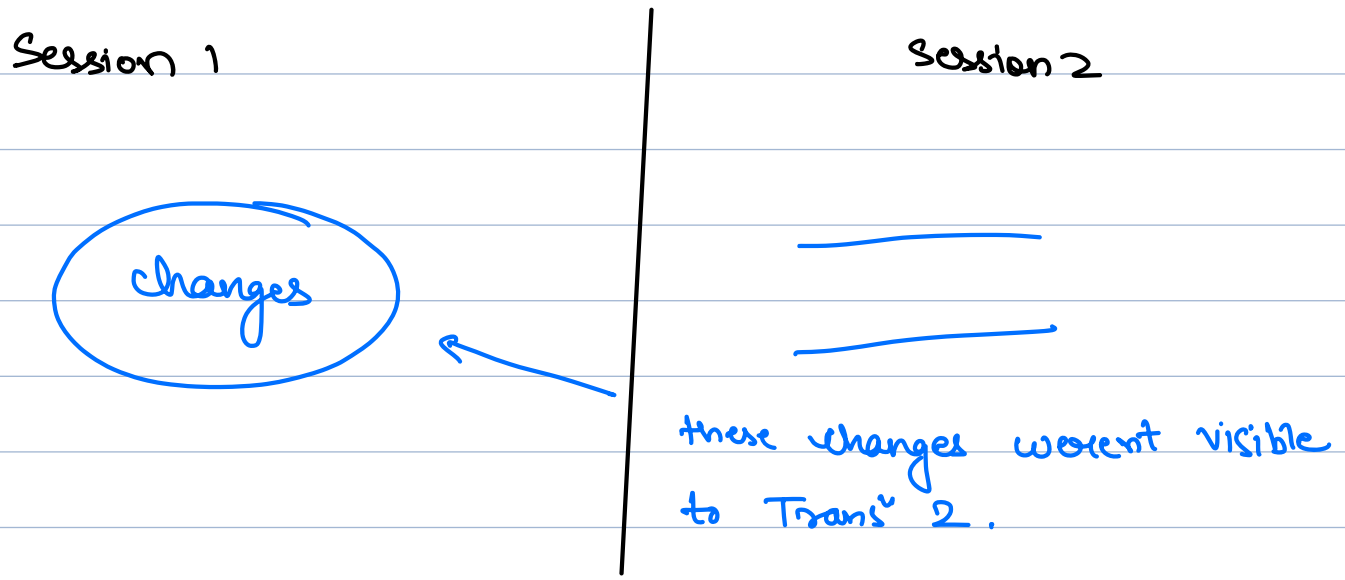
If a lot of people are transferring to B, A→B, C→B etc and we suspect B's account, we would want to block all transactions.
So we UNDO money reduced from A's account.

ROLLBACK

rollback;
↓
undo all changes done since the last commit.
(changes still in the memory)

commit → marriage → persisting the changes to db.
rollback → breakup → undo whatever changes were being done since the last commit.

→ You cannot rollback after a commit is done.



Both sessions are seeing diff things
Why? MySQL → each session is isolated

4 levels of isolation.

1. Read uncommitted → most relaxed
2. Read committed — PostgreSQL →
3. Repeatable read
4. Serializable → most strict

Default isolation level for MySQL

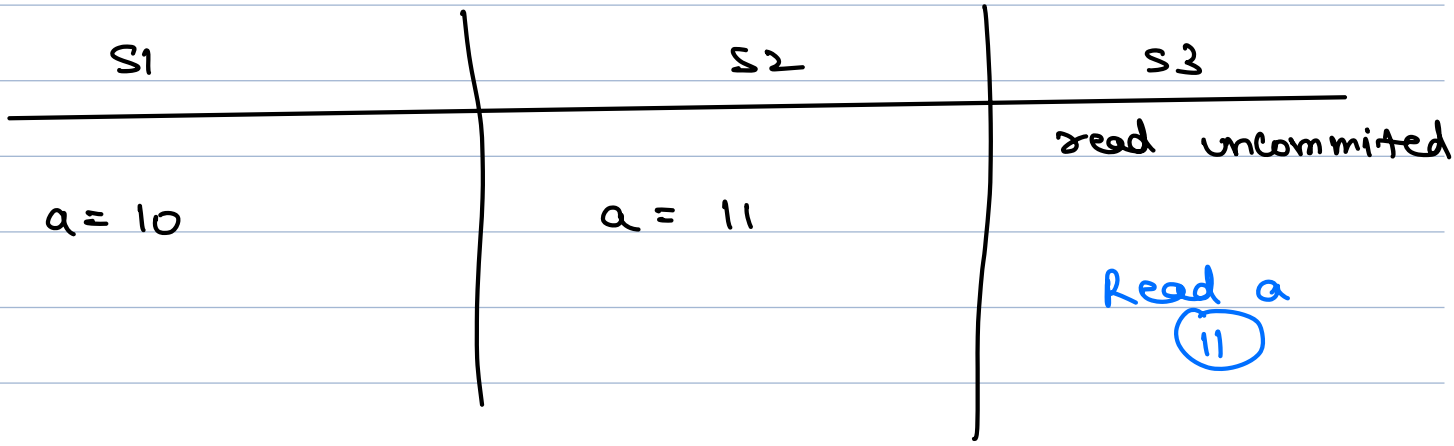
Increasing order of severity

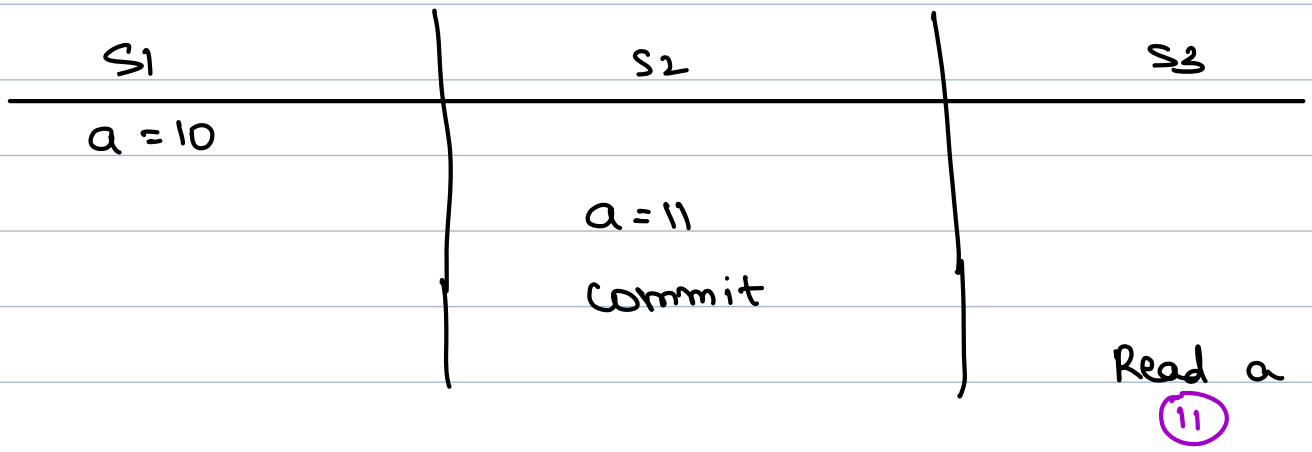
★ Isolation levels are talking about what data is read, not what is updated.

Read uncommitted

→ Allows a transaction to read even uncommitted data from another trans".

→ latest uncommitted data (commited or uncommitted)





☆ Isolation level only talks about how your session will act (read data).

↳ The isolation level of other trans" doesn't matter.

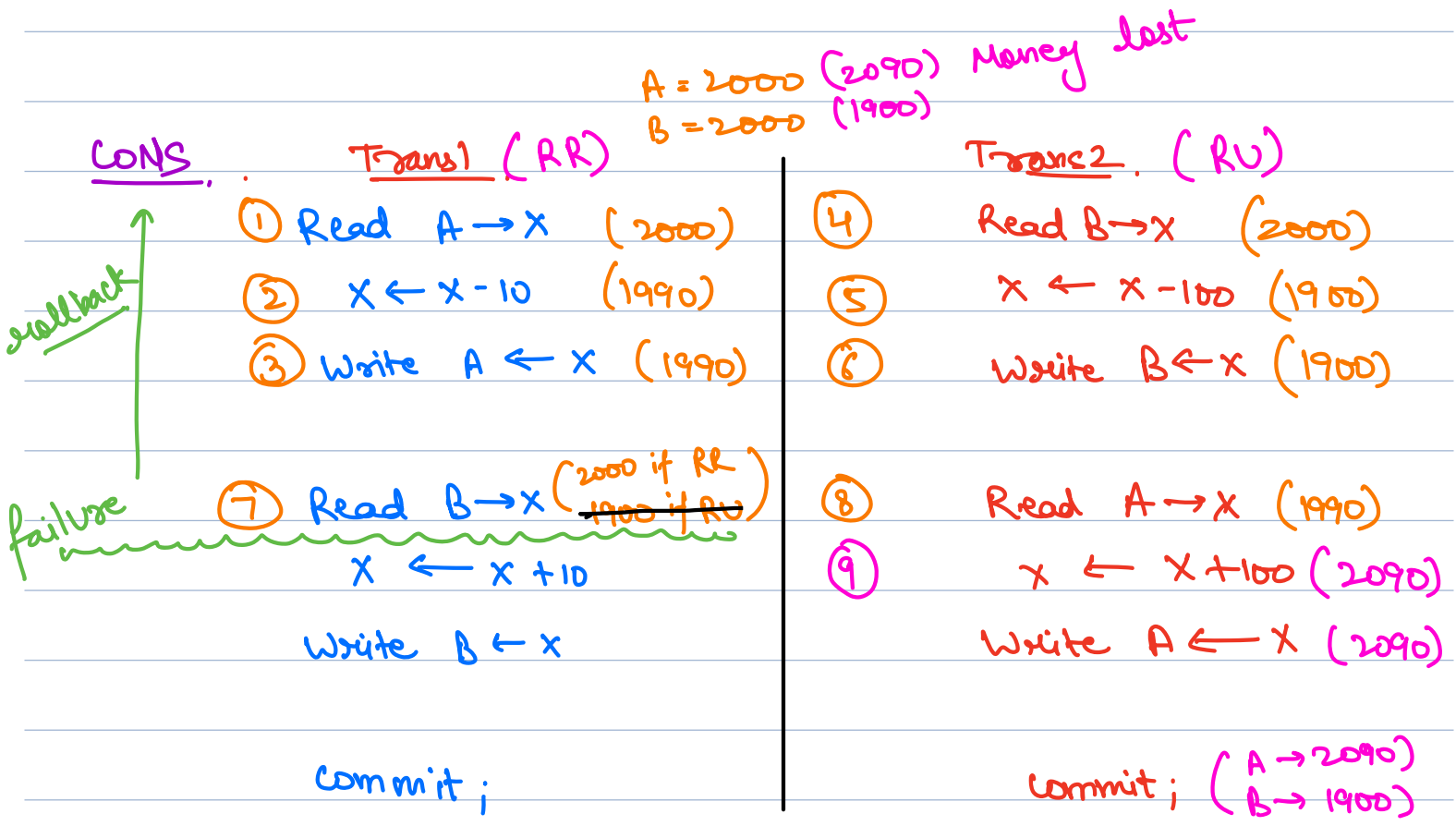
[code]

→ show variables like 'transaction_isolation';

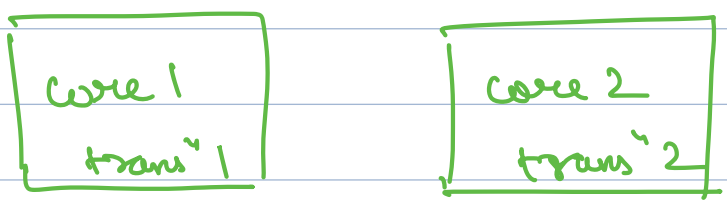
→ set session transaction isolation level read uncommitted;

PROs

① Fast → performance



Transactions happen on CPU, who runs lines one by one with context switches [Multi core machines]



Because 2 diff trans" were running in parallel

↓

ended up in inconsistent state

RU hampered consistency.

At line 8 ⇒ trans" 2 read data that was not committed and rolled back.

DIRTY READ

(Risky read)

When a trans" reads data that is not committed. That data might change | rollback | not get committed