Transactions - II

Agenda.

1) Read uncommitted
2) Read committed
3) Repeatable read
4) Serializable

} Isolation levels. (Locks)

Deadlocks

Dirty read - Reading some data that is not yet confirmed yet.

How to solve this?

A data is confirmed once it is committed.

Read committed → Your trans will only read the latest committed data.

[CODE]

L set session transaction isolation level read uncommitted.

R set sess transaction isolation level read committed.

L set autocommit = 0; R

select * from film where film_id = 10;

update film
set title = 'ujjwal 4'
where film_id = 10;

datagrip

Transaction isolation level of YOURS matters.
DONT care about others isolation level.

R select * from film where film_id = 10; ⇒ get latest committed value

When you start a transaction, it turns off the auto commit till the transaction is completed.

R update film
set title = 'ujjwal 6'
where film_id = 10;

⇒ It will go to waiting bcoz there is a lock taken on the other side.

L commit;

Read ⇒ Shared lock

Written ⇒ Exclusive lock

Some one has	Someone else can
Shared lock	⇒ read ⇒ write
exclusive lock ↳ given at a write query	→ read → write ✗

Read on left side in code & see right side can write as left one taken a shared lock.

If one transaction has written on a row (only row that was inserted / deleted / updated.), no other transaction will be allowed to write on that row.

focus on isolation levels, we talk about locks later.

Read committed

→ A trans' will read always the latest committed value
→ never read an uncommitted value.
→ No dirty reads.

Problems with this.

Users

id	name	psp	email_sent
1	A	60	false
2	B	~~80~~ 79	false
3	C	70	false
4	AB	45	false

Q Send email to all students with psp < 80.

```
list<users> = select * from users where psp < 80;
[1, 3, 4]
```

Send emails ⟹ At this point 2's psp goes to 79.

What can go wrong?

```
update users
    set email_sent = true
    where psp < 80;
[1, 2, 3, 4]
```

This is wrong as user 2 did not get an email.

Trans 1	Trans 2
① Read users with psp < 80	
② doing work	
	③ update psp of user 2
④ Read users with psp < 80	

within a trans, the value/result of a particular query can change.

⇒ within a transaction if I read the same row again, it must have the same value.

↓

when a trans" starts, it should take a snapshot of the db and only that value should be taken.

↓

Read committed will not have Repeatable reads.

↓

It has non-repeatable reads.

↓

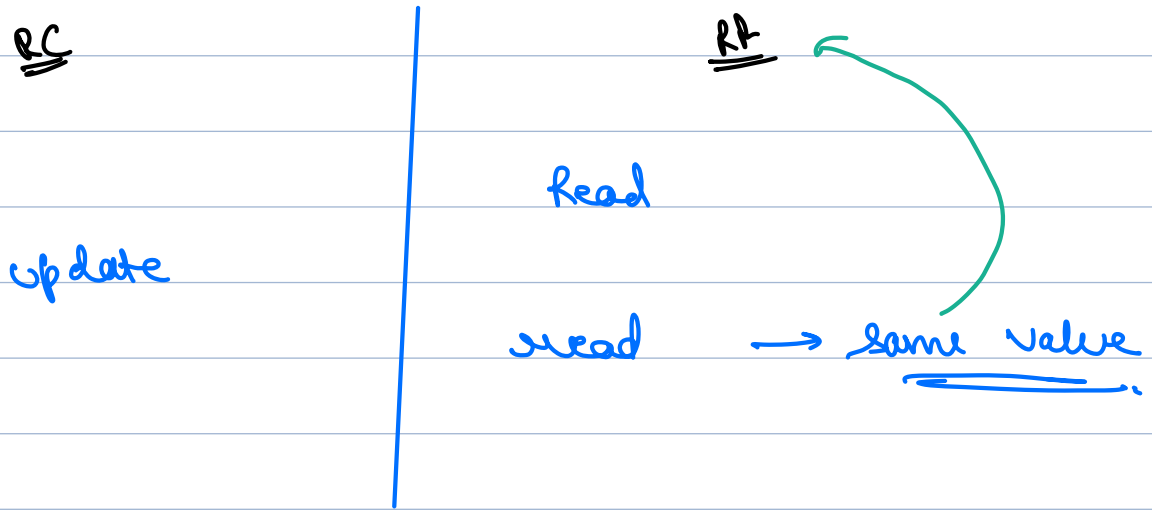
In read committed, it always reads the latest committed value, even if that committed value happened after the trans" started.

[CODE]

RC	RC
	Read
Update	
	Read → diff value from above.

Make sure autocommit is off.

Set session transaction isolation level repeatable read;



This will not work if autocommit is ON as after every statement it will commit & start a new transaction.

↓

Repeatable reads works only in the scope of a single transaction.

Auto commit = 0 :

↓

① write a query ⇒ starts a new trans"

② commit ⇒ end the trans".

Auto commit = 1

start trans" → executing → end trans"

MySQL / SQL DB → by default has autocommit = 1

⇒ every query executes as a part of new trans.

Start transaction ; } begin a trans manually ⇒ dont need autocommit = 0 ;

begin ;

{ ≡ commit / roll back }

[CODE]

RU	RR
begin ;	
Read - 12	
	Start trans ;
	Update - 12
	Read - 13
Update - 12 (cant perform this)	
Update - 13	
	Read - 13
commit ;	

RR

Read - 13 (still some value)

commit;

Read - 13 (new value)

Problem with Repeatable read.

solved by

first we learnt Dirty read → RC

then non-repeatable read → RR

Repeatable read stores values of which rows?

↓

of the rows it first read / set of rows around it.

(CODE)

RU

begin

Read - 12

Read - 180

Update - 180

commit

RR

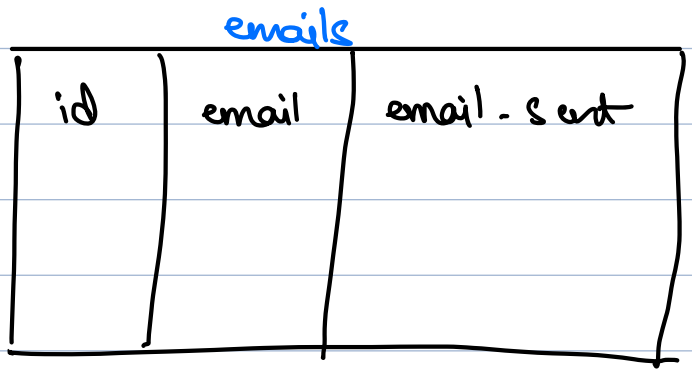
begin

Read - 12

Read - 180 (new value whereas should have been old value)

happened bcoz it takes a snapshot of subset (nearby rows)

Q Send an email to every user and after sending email set their email_sent = true.




```
select email,
    from emails;
```

In last psp example we assumed, no new student is added.

```
Send emails to them;
```

One new student inserted

```
Update emails
    set email_sent = true;
```

Repeatable read talks only about rows & their values that you initially read. It says nothing about rows that you never read.

[CODE]

RU	RR
Begin	Begin
Insert into lang	Get all (language)
commit;	Get all [New value missing]
	Update all names [new value also present]
	↓
	Came out of nowhere as a ghost.

[show above example again with more values]

This problem is known as Phantom read.

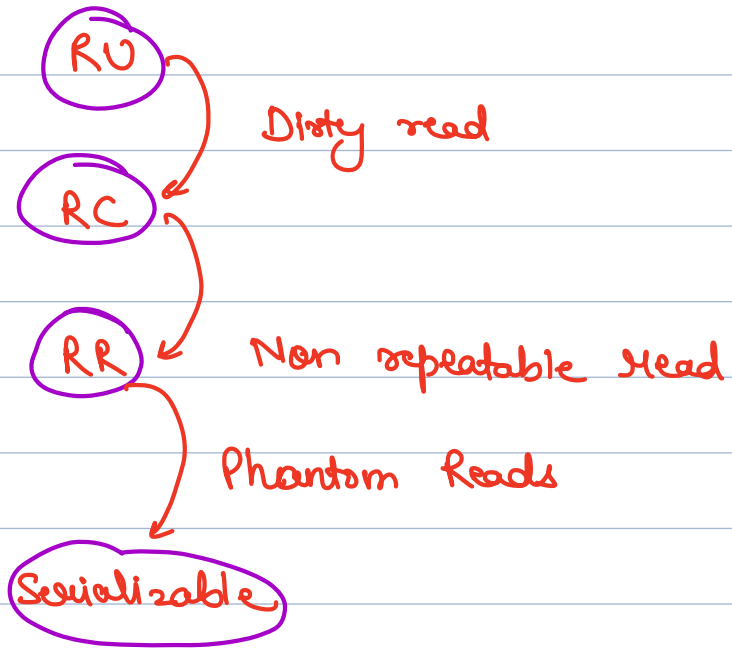
↓

A row comes magically that you had not intended earlier.

RR only guarantees about the rows that read initially.

→ Why this happens:
Every isolation level only talks about what it will read, not about what/how it will write/update.

★ Within same trans", it will always read your latest value.



RR only guaranteed that rows you had read will remain the same.

It never guaranteed about the complete table.

#
① you read rows of the table
② did some work on rows
③ updated rows of the table

↙ Someone inserted.

↑ That also got updated.

Another example

[CODE]

RU

begin

delete & commit

RR

begin

Read → reads 10 but there are only 9 in DB.

Update delete value [Nothing happens]

commit ;

Now data will be different.

This is because data is not there in actual dB.

Serializable.

↓

If I am doing anything on the table, no one else can work on it.

If somehow → no concurrency, at one time only one person can work on one table.

↓

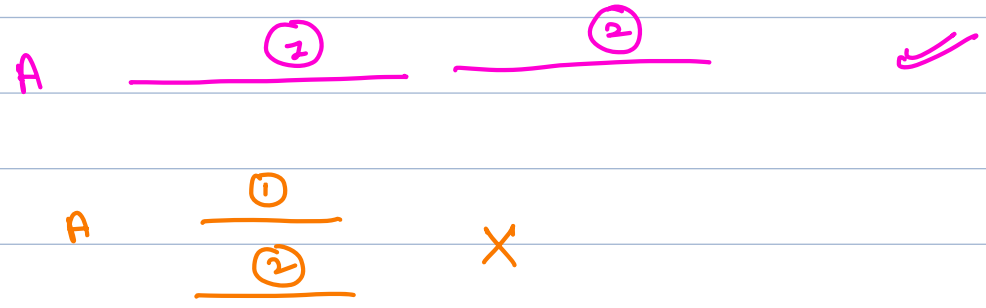
low performance

Remove concurrency

↳ Banks, BMS.

Serialized → sequential (one after other)

If a trans" has started for a row A, the other trans" can only start after the first trans is completed.



No concurrency | No parallelism

[CODE]

serializable	RR
begin	
	begin
	Read
Update	
	Read (on updated)

Same as before

Update (on same item) (not allowed)

Serializable	Serializable
Begin Read (12 & 13) select * from film where id in (12,13) for update; ↓ I am going to update these values, dont let anyone read them	Begin Read → 12 (not allowed) Read → 15 (allowed)

If one of them is not serializable, arent they doing harm to themselves by reading values they are not sure about.

↓

Thus both need to be serializable.

Serializable only allows us to read rows that are OK to be read.

In others even if you have a lock, others could atleast read the data, serializable is very strict and dont even allow that.

Serializable ⇒ Repeatabable read + Checks for lock even for read.

↓

If anyone else has a lock, it doesn't even read it.

Example

Book My show

RR or something

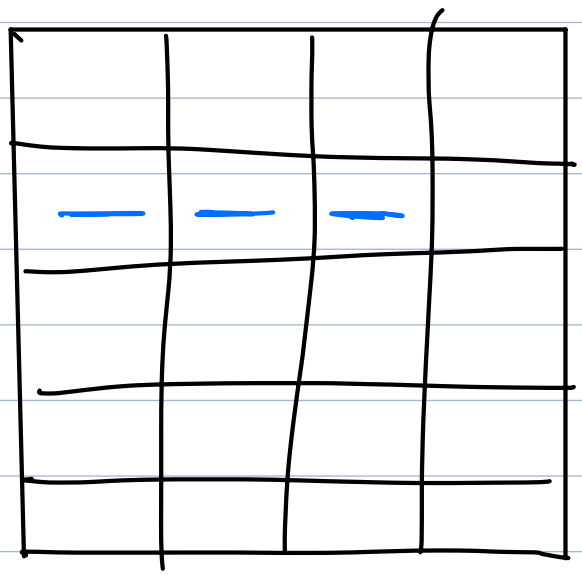
↑

① show tickets

② Book tickets

↓

Serializable



[show on website]

↓

One goes one blocked

★ In others you can read any row, in serializable you can only read rows with No Lock.
In others which version on data you read is the problem, read/ not read was not the problem.

In serializable

↓

You can't read → if other is writing or reading for update.

[CODE]

How banks might be working :—

Serializable	serializable
Read → film → 12, 13 for update.	Read → 12, 13 (Not allow)

Simply read doesn't take a lock.

If one person has taken lock for update only then others can't read, else they can.

serializable doesn't allow you to even read a row that has a lock.

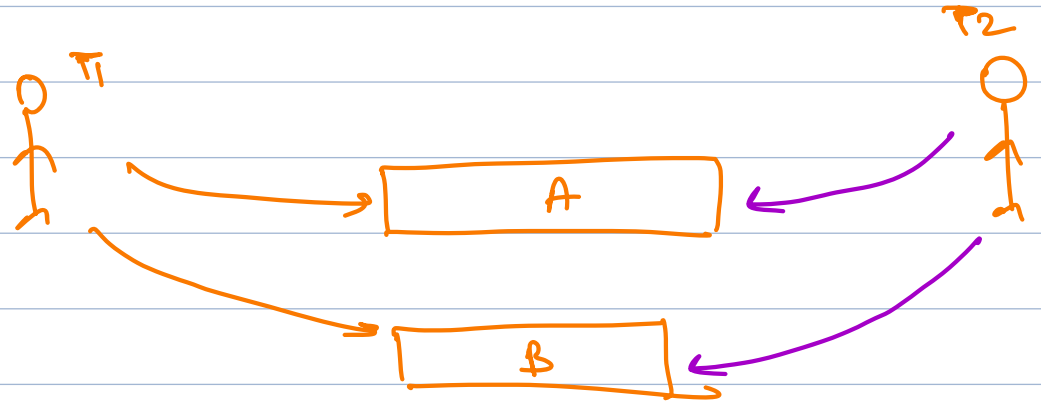
speed affected but works for bank cases.

For update → a way to basically say 'I will be writing' soon to this data.

Deadlocks

Eg → You have crush on her, she has crush on you. You are waiting for her to confess & she is waiting for you to confess.

We learnt in transaction when we write/read for update we take lock on row. (this belongs to me).



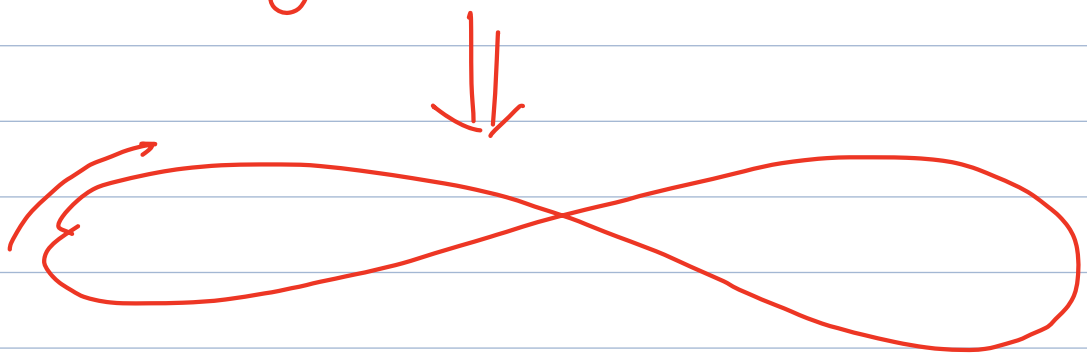
T1
✓ lock A
X lock B

T2
lock B ✓
lock A X

T1 will not get a lock till T2 completes

T2 will not get a lock till T1 completely.

T1 is waiting for T2 to complete but T2 is waiting for T1 to complete



Infinite cycle.

Deadlock → learn later in concurrency.

In OS we will talk how to handle deadlocks.

=> SQL detects deadlocks, it automatically kills and rolls back one of the trans.

[CODE]

begin	begin.
✓ Read 12 - for update	✓ Read 13- for update
x Read 13- for update (doesnt get lock) (this line wont lead to a deadlock, just waiting for resource)	Read 12 - for update (Deadlock) ("Trans" cancelled)