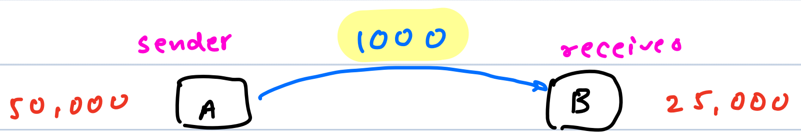
transactions

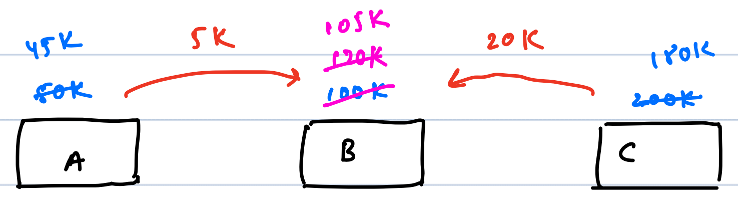
A transaction refers to a logical unit of work. In a single transaction there is one or more database operations. (Create, delete, update)



(accountA , accountB , amt)

```
int x = getacctBal( A);        50,000
    check( x , amt );
    update( A , x - amt);      49000
                               -> fail
int y = getacctBal( A);        25,000
    update( B, y + amt);
```

it causes data inconsistency because of partial execution.



transaction 1

```
1 int x = getacctBal(A); 50,000
  2 check( x , amt );
  3 update( A , x - amt); 45K
                            → fail
4 int y = getacctBal(B); 100K

  5 update( B, y + amt);

      fetch B Bal
        105K
```

transaction 2

```
1 int x = getacctBal(C); 200K
  2 check( x , amt );
  3 update( C , x - amt); 180K
                            → fail
4 int y = getacctBal(B); 100K

5      update( B, y + amt);
       fetch Bal B → 120K
```

* it follows ACID properties to solve these problems.

A : Atomicity

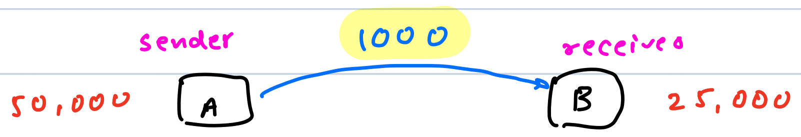
C : consistency

I : isolation

D : durability

1) **Atomicity :**

it ensures that a transaction is treated as single , indivisible unit of work.



(account A , account B , amt)

```
int x = getacctBal( A );      50,000
    check( x , amt );
    update( A , x - amt );     49000
                               → fail

int y = getacctBal( A );   25,000
    update( B, y + amt );
```

- either all the operations within the transaction are successfully completed or none of them.

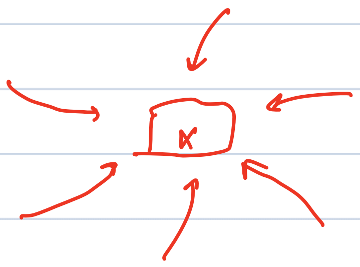
* if there is any failure the entire transaction is aborted and the database is left in its original state.

2) consistency :

- its ensure that data will be in consistent state.
- it is possible that data may be inconsisted during the transaction.
- it ensure that the database is remain in a valid state before and after the execution of transaction.

3) isolation

its . ensure that transaction operate independently.



its prvent to intefere to other transations.

4) durability

it is ensure that changes are permanently stored in a database when the transaction is complete.

- if any failure happen then its revert back to the last committed point.

- How to mark it is successfully
- How to revert back.

1) commit

it marks that the transaction is successfully completed or committed.

- it means all the changes made by the transaction are permanent saved to the database.

2) Roll back: when a transaction is rollback, any changes made by transaction are discarded. and database reverts to its previous state.

- If there is any failure happen so in that case it automatically roll back the transaction:

break till 9:17

transaction isolation level

- Read uncommitted
- .Read committed
- Repeatable read
- serializable.

- Read uncommitted (temporary data)

~~10 Lakh~~ ~~start~~ 10 lakh

t1	person	t2
start		start
get Bal (Person) 5L		get Bal (Person) 5L
send → 2L.		
update (person, 7L)		get Bal(Person) 7L
→ failure		4 lakh
		update(person, 11lak)
		commit;

11,00,000

~~5:00,000~~ → permanent data

[~~7,00,000~~] → temporary data

11,00,000

- this is the lowest isolation level. where transaction can see uncommitted changes made by other transactions before they committed.
- its allowed dirty read, becouse transaction can see changes that may be roll back later.

- there is low data consistency.

- **.Read committed**

9,00,000
~~7,00,000~~
(~~5,00,000~~) T
~~7L~~ 9L
(~~5,00,000~~) P

t1	Person	t2
~~10,00,000~~ 8,00,000		8,00,000
start		start
• getBal (Person) 5L		• getBal (Person) 5L
2L →		
• update (Person, 7Lac)		• getBal (Person) = 5L
		4 Lac
• commit		• update (person, 9L)
		commit

- It prevents dirty read.
- In this isolation level transaction can only see changes committed by other transaction.

- Repeatable read { default }

~~10,00,000~~ 8,00,000 t1	Person	t 2 8,00,000
start { 5L)		start { 5L }
- get Bal (Person) 5L		- get Bal (Person) 5L
- update (person, 7Lak)		- get Bal (Person) 5L
- commit		- get Bal (Person) 5L
		update (person, 9L)
		commit ;

perm [~~5,00,000~~] ~~7L~~ 9L

temp [~~5,00,000~~] 7L

- data will be same throughout transaction
- before and after committed in other transaction its still same.

- Serilizable S, L

person P1

t1	t2
start	start
get amt(p1) = 5L,	get amt (p1) = 7L
update(p1, 7L)	wait.....
↑ LOCK	
commit ;	

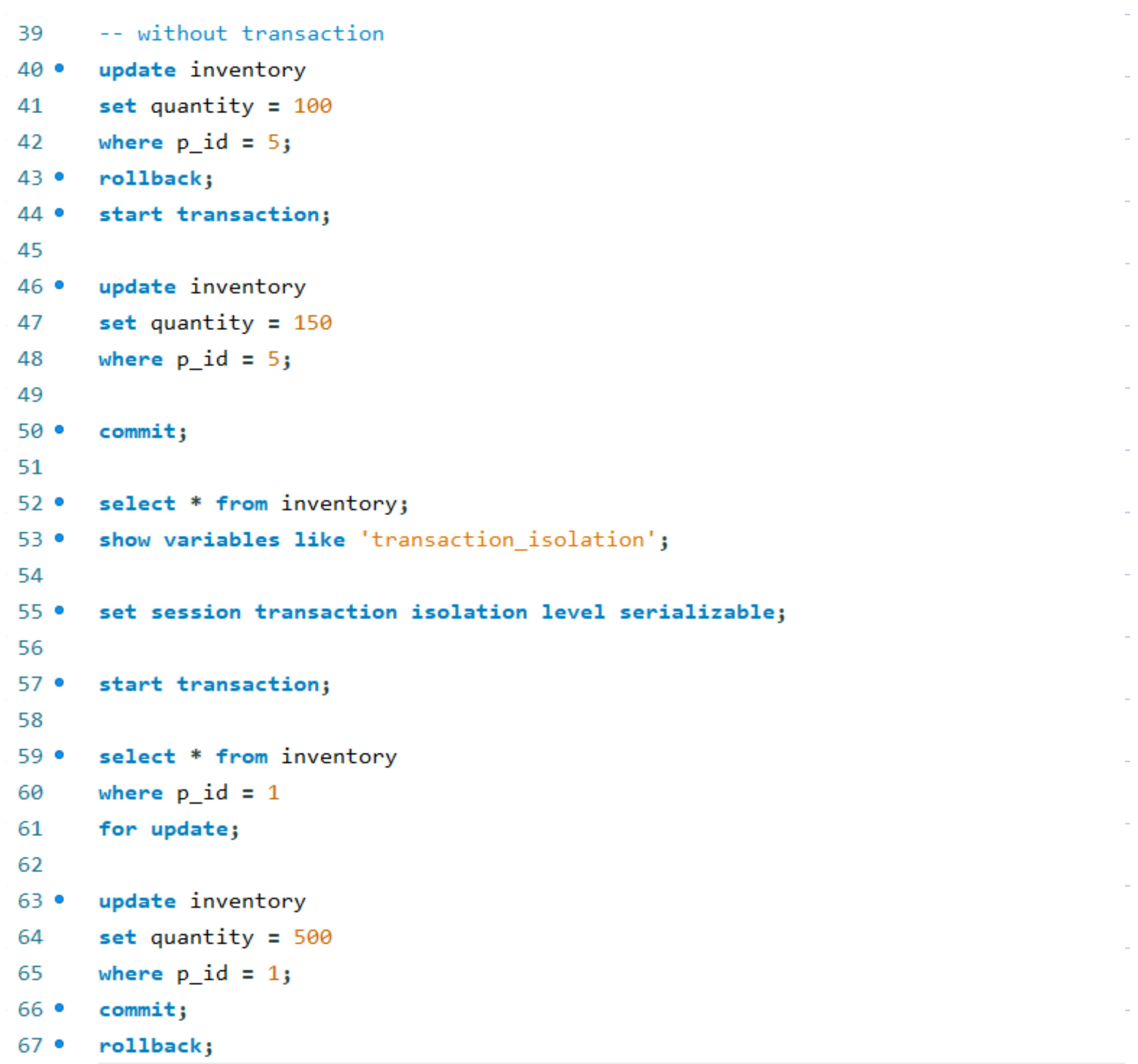
- serializable is the highest level of isolation. and consistency
- it locks the particular rows uses in the transaction and no transaction can do any operation on that row till it's committed.

serilizable

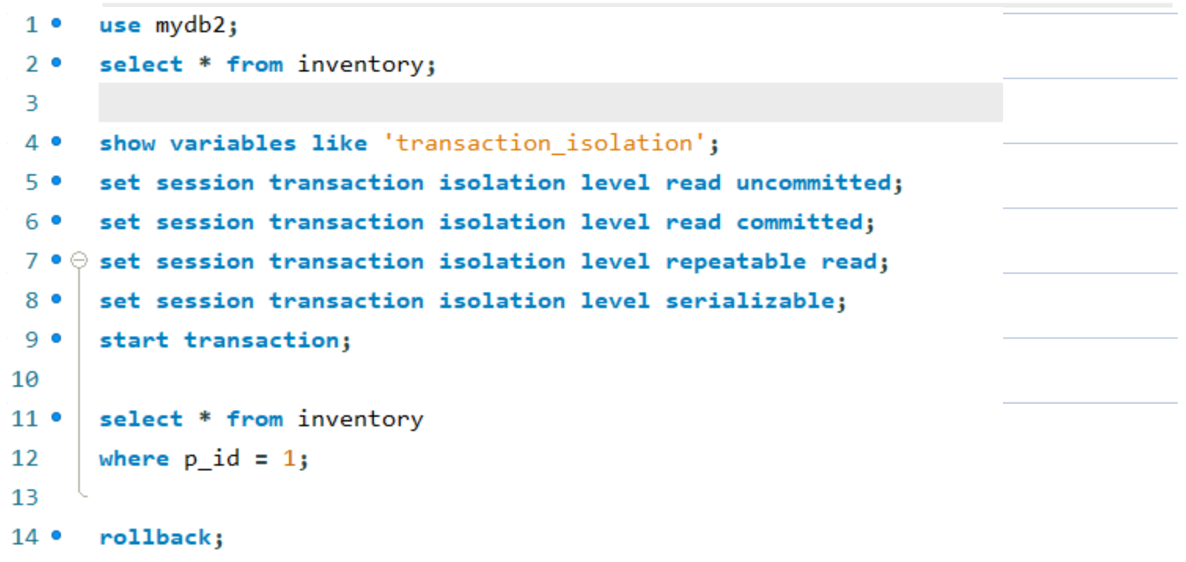
t1	t2
start	start
update P1 {lock}	update P2 {lock}
update P 2	update P1

deadlock

In my sql it cancel one of the transaction.



```
-- without transaction
update inventory
set quantity = 100
where p_id = 5;
rollback;
start transaction;

update inventory
set quantity = 150
where p_id = 5;

commit;

select * from inventory;
show variables like 'transaction_isolation';

set session transaction isolation level serializable;

start transaction;

select * from inventory
where p_id = 1
for update;

update inventory
set quantity = 500
where p_id = 1;
commit;
rollback;
```



```
use mydb2;
select * from inventory;

show variables like 'transaction_isolation';
set session transaction isolation level read uncommitted;
set session transaction isolation level read committed;
set session transaction isolation level repeatable read;
set session transaction isolation level serializable;
start transaction;

select * from inventory
where p_id = 1;

rollback;
```