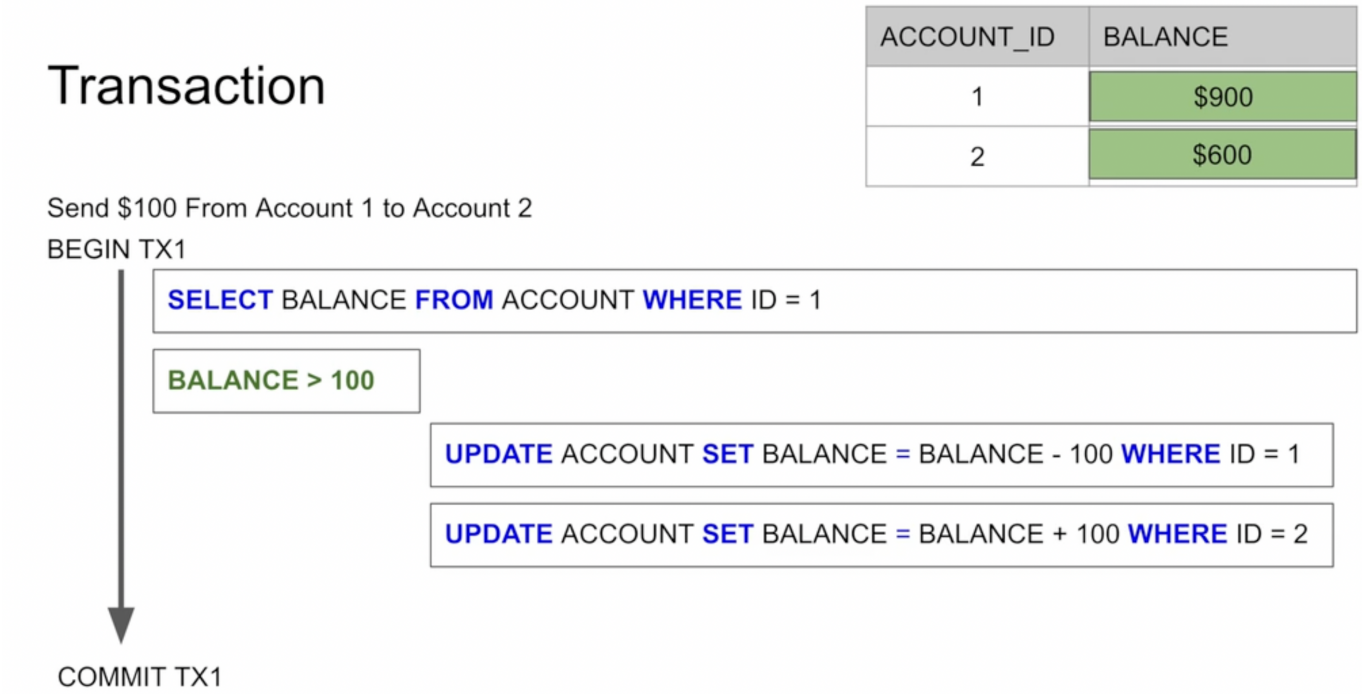
DBMS Class on Transactions

#dbms

Transaction

- collection of QUERIES
- property of these queries is that they are single unit of work
- transferring money from account A -> B
    - check if sender has enough money
    - debit money from sender's account
    - credit the amount to the receiver's account
- eg: DEPOSIT some cash into an account
-



1. Does account 1 even have 100 dollars? I don't know. Yes, sir, alright account 1 has 100 dollar `SELECT BALANCE FROM Account WHERE ID = 1`
2. It's a dirty change, debited the amount from account 1, yet not received on the receiver's side `UPDATE Account SET BALANCE = BALANCE - 100 WHERE ID = 1`

---

3. Add another 100 dollar to 2nd account `UPDATE Account SET BALANCE = BALANCE + 100 WHERE ID = 2`
4. Commit, turn it green => TRANSACTION is complete!!!

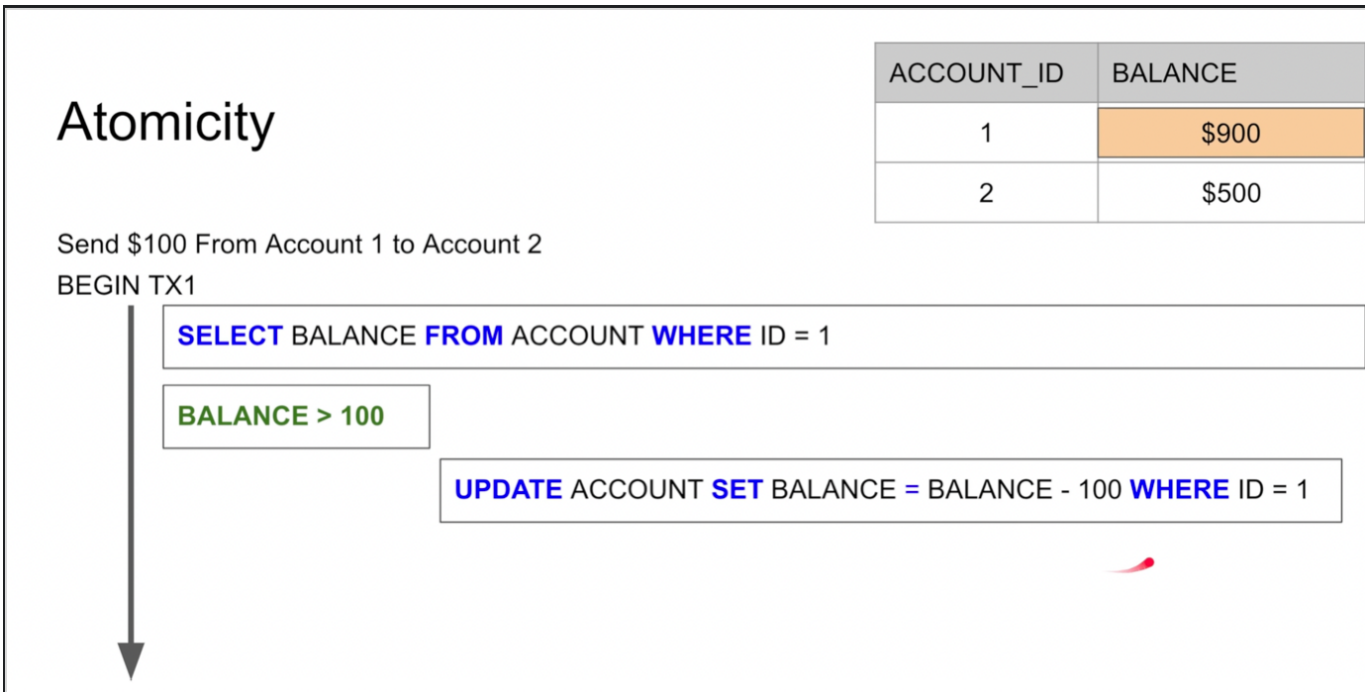
If something goes wrong Rollback, abort, abort, abort

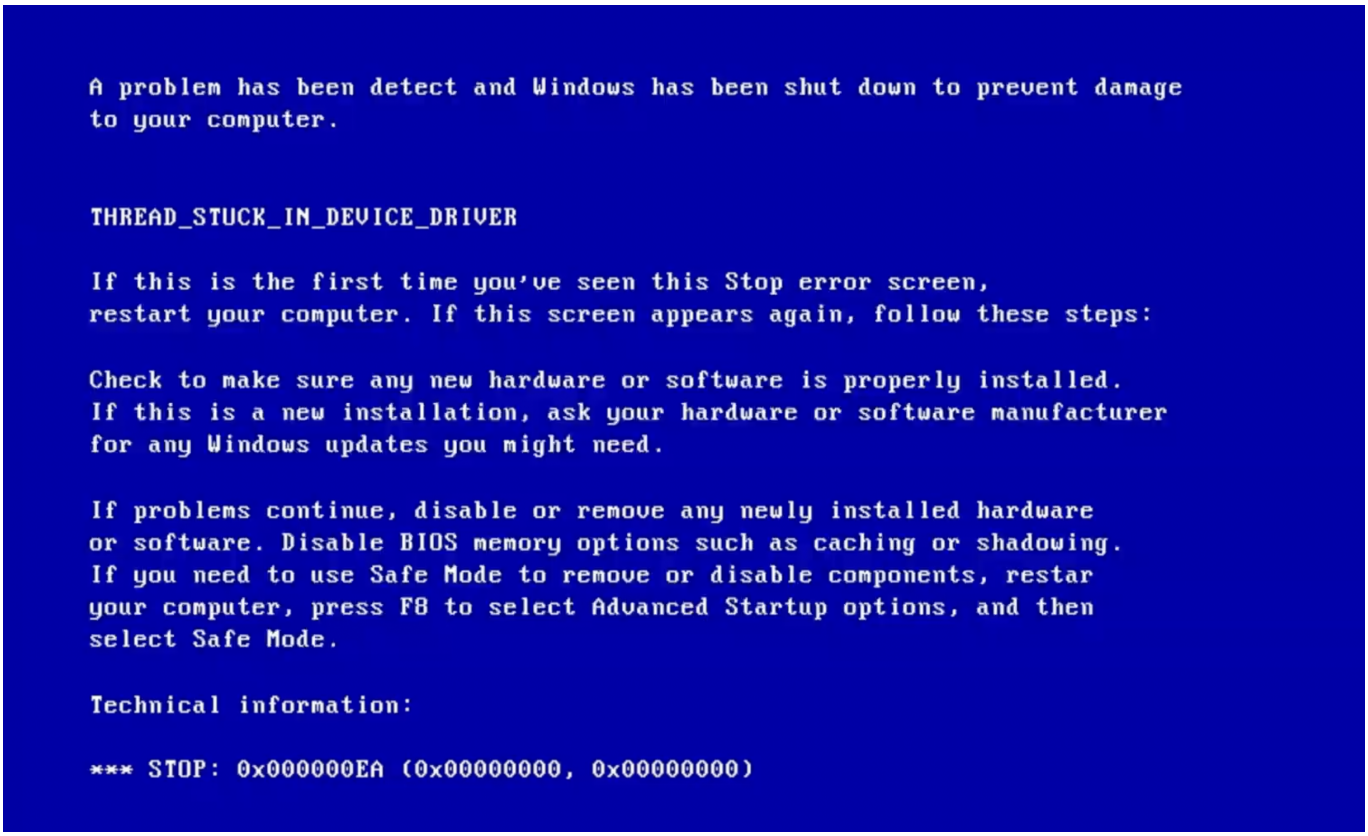
ACID: Properties of transactions

- Atomicity
- Consistency
- Isolation
- Durability

Atomicity

- a transaction should be atomic
- either all or none
- if one fails you better fail the whole thing





- the amount was not credited to the receiver
- 100 dollars just went into thin air
- DB internally provides atomicity
- if you start a transaction and one step fails, the complete transaction gets rolled back,
    - the other operations that were complete they will be reversed

---

Let's see it in action!!

Products

```
mysql> describe products;
+-----------+---------+------+-----+---------+----------------+
| Field     | Type    | Null | Key | Default | Extra          |
+-----------+---------+------+-----+---------+----------------+
| pid       | int(11) | NO   | PRI | NULL    | auto_increment |
| name      | text    | YES  |     | NULL    |                |
| price     | float   | YES  |     | NULL    |                |
| inventory | int(11) | YES  |     | NULL    |                |
+-----------+---------+------+-----+---------+----------------+
4 rows in set (0.00 sec)

mysql> desc sales;
+--------+---------+------+-----+---------+----------------+
| Field  | Type    | Null | Key | Default | Extra          |
+--------+---------+------+-----+---------+----------------+
| saleid | int(11) | NO   | PRI | NULL    | auto_increment |
| pid    | int(11) | YES  |     | NULL    |                |
| amount | float   | YES  |     | NULL    |                |
```

```
| qty    | int(11) | YES  |     | NULL    |                |
+--------+---------+------+-----+---------+----------------+
4 rows in set (0.00 sec)
```

Selling an iPhone

1. Check if we have enough inventory
2. Update inventory
3. Create a sales entry

Sell 5 units of phone Current inventory is 10 units

Isolation

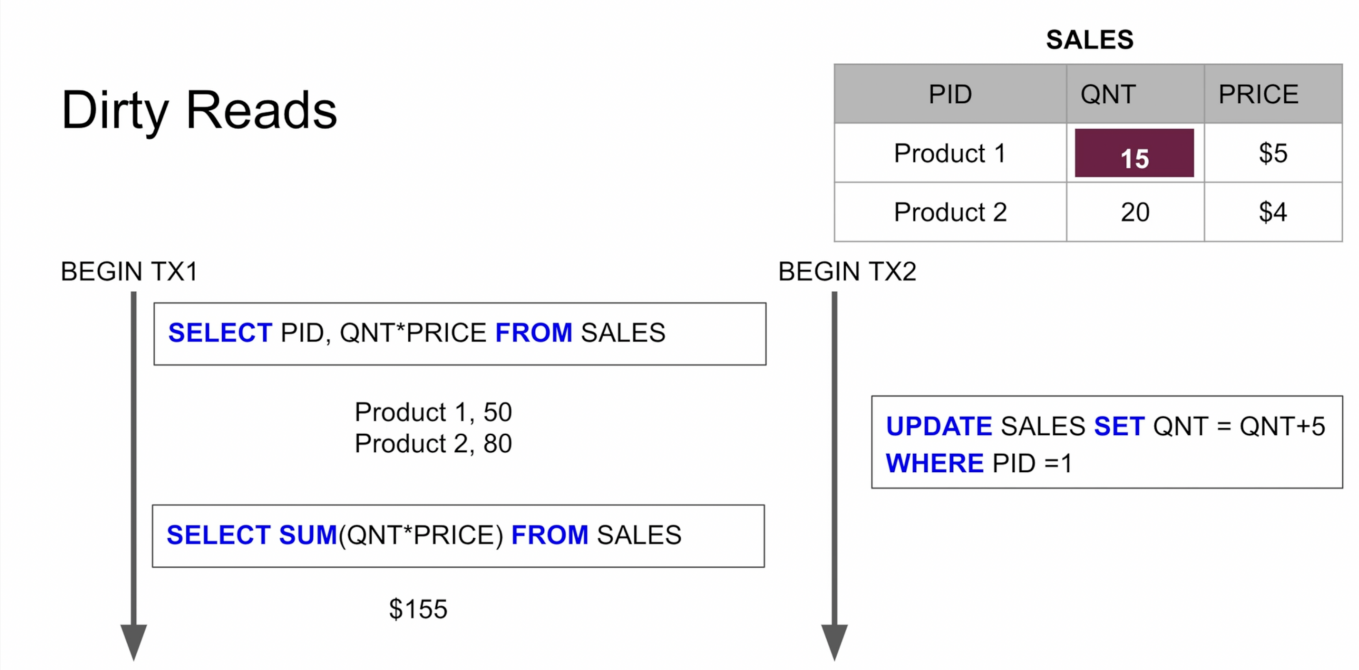
- in real lot of transactions will be taking place simultaneously
- there could be certain things that can go wrong, you might read an incorrect data, or write an incorrect one
- we have to make sure that even through transactions can run in parallel, their effects are isolated

1. Can my inflight transaction see changes made by the other transactions?
2. Read phenomena (without isolation or there is some level of isolation)
3. Isolation levels (to fix the problems)

What all things can go wrong?

Dirty Read

- reading an uncommitted change



Qnt for PID = 1 is 10 at the moment.

```
SELECT PID, SUM(QNT*PRICE) FROM SALES GROUP BY PID
```

Sales Report Product 1 -> 50 Product 2 -> 80

TX2 started and updated Qnt for PID = 1 => 10+5 = 15

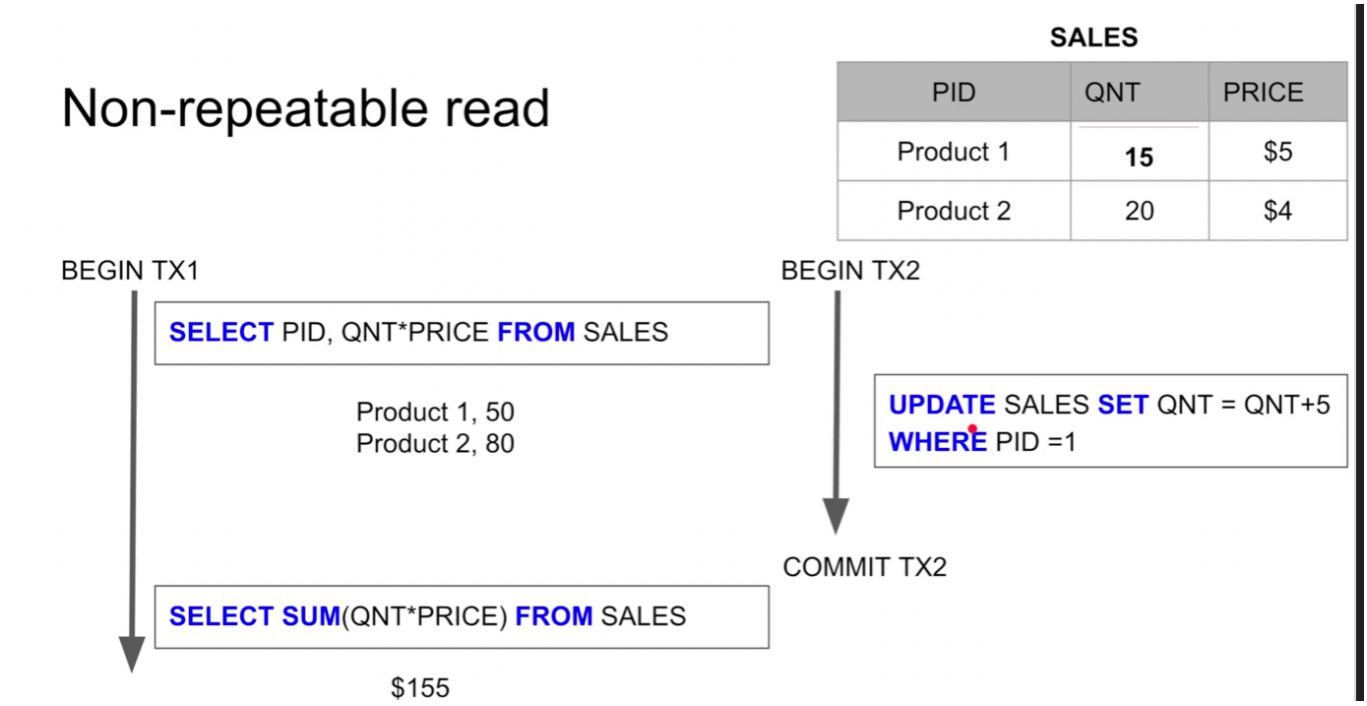
`SELECT SUM(QNT*PRICE) FROM SALES` [READING uncommitted data]

A => 50 + 80 = 130 B => 75 + 80 = 155

TX2 gets rolled back

Non - Repeatable Read

- reading a committed change



Sales Report Product 1 -> 50 Product 2 -> 80

In read uncommitted mode

1. Outside transaction => will NOT be able to see uncommitted changes
2. But inside transaction => will be able to see uncommitted changes => DIRTY READ PROBLEM

Isolation level

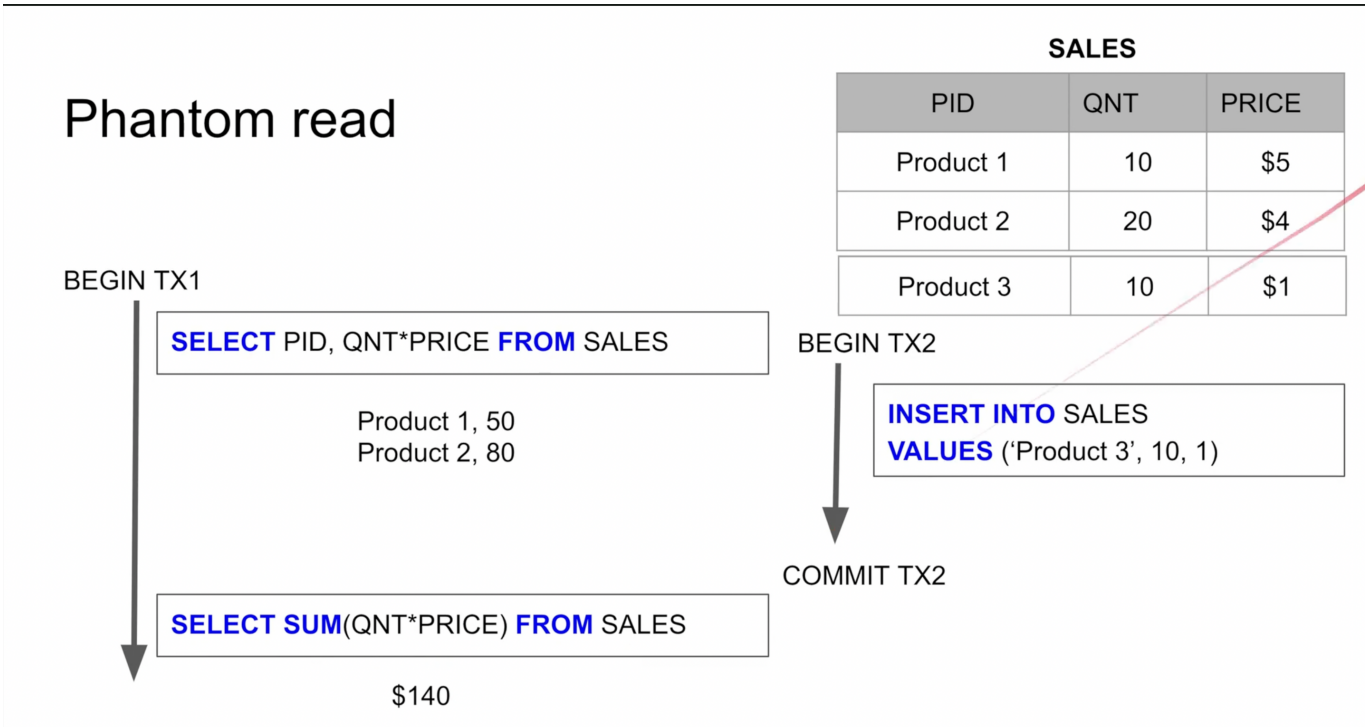
- read uncommitted
  - no isolation => all the read phonomena can occur
  - any change from the other transaction is visible in the rest of the transactions happening at the same time
- read committed
  - each query in a transaction only see the committed stuff (at the time of the query)
  - dirty read can't occur because you're not reading any dirty change
- repeatable-read
  - each query in a transaction ONLY SEES the committed changes at the beginning of the transaction

    - implemented using
        - versioning
        - locking mechanism
    - solve the non-repeatable read phonomena
    - non-repeatable read and dirty reads both do not occur
- serialisable
    - all the transactions are bound to happen without any concurrency
    - you are not allowed to have parallel transactions

Isolation levels vs read phenomena [edit]

Isolation level	Dirty reads	Lost updates	Non-repeatable reads	Phantoms
Read Uncommitted	may occur	may occur	may occur	may occur
Read Committed	don't occur	may occur	may occur	may occur
Repeatable Read	don't occur	don't occur	don't occur	may occur
Serializable	don't occur	don't occur	don't occur	don't occur

Phantom read problem



TX1's step1 took place => Product 1 -> 50, Product 2 -> 80 TX2's step1 occurred TX2 got committed

In middle of TX1 product 3 came into the sale

TX1's step2 => Total sale amt => 140

Consistency

- atomicity and isolation leads to consistency in some way
- types
    - in the actual data
        - lack of atomicity => inconsistent data
        - lack of isolation => inconsistent results
        - their lack can lead to inconsistent actual
        - we need atomicity and isolation
        - ALSO DEFINED BY USERS => business constraints
        - constraints also make sure data is consistent
        - foreign key constraints
        - picture and picture_likes table
        - in case of redundancy we need to be responsible to maintain it consistently

Consistency in Data

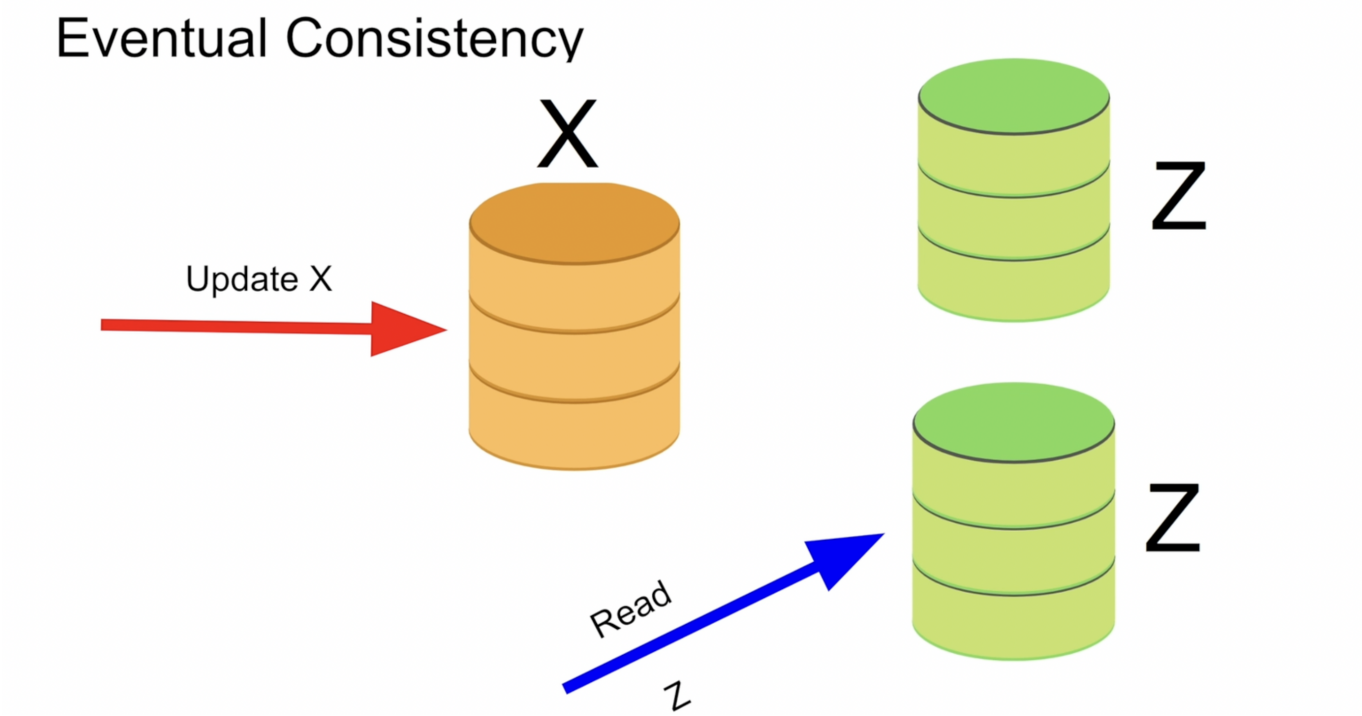
Pictures

ID **(PK)**	BLOB	LIKES
1	xx	2
2	xx	1

Picture_Likes

USER **(PK)**	PICTURE_ID **(PK)(FK)**
Jon	1
Edmond	1
Jon	2

    - in reading the data
        - if I update likes to be 100 in the pictures table for id = 1
        - Am I ALWAYS going to read the value 100?
        - If I update a value am I ALWAYS GOING TO READ THE UPDATED VALUE? `UPDATE Pictures SET LIKES = 100 WHERE ID = 1`
        - change has been committed




```
    * problems arises when we have multiple copies of your database
    * and when you have caches
    * eventually the data will get updated at all servers
```

Durability

- power failure happens after you commit a transaction
  - it gets updated in the primary memory => RAM
  - then power failure occurs
  - will it be good if the change is not reflected in the secondary memory (hard disk) where your DB is actually stored???
  - you want these changes to be persisted in the database
- committed changes to be reflected to your actual storage no matter what happens
- implemented using some form of logging mechanism
