Transactions2:

Transactions in Distributed environment.

Isolation means every transaction should work independently, work in series of operations.

How much isolation?
- should Transaction ‘T2’ read only committed data from ‘T1’
- can it read uncommitted data also
- should we in a Wait state or should we create Locks for our resources + stream. If one transaction creates streams, other transaction will wait for sometime.

Anomalies / Problems: 3 major database issues which we are going to solve with Isolation
1. Dirty read → Reading uncommitted changes

Transaction T1	Transaction T2
Write changing value W(A-->B)	
	Read(B) which is an uncommitted value
RolledBack	

Transaction T1 is making a change, it changed value from A to B. Only when transactions are committed, we put them into the database so data remains in the proper state.

2. NonRepeatable Read:
- same row is showing different values

Transaction T1	Transaction T2
Read(empID=1)	
	Write(change name for empID=1)
Read(empID=1) returns a new name	

Inside single Transaction T1, when we do Read operations twice, we get different values (names) this is issue number 2, which is non-repeatable reads. changing the same record

3. PhatomRead:
- same query will lead us to different results.

Transaction T1	Transaction T2
Read(price > 100), give me products price > 100	
	Write(NewProduct => 200). add new product
Read(price > 100), give me products price > 100	

When Transaction T1 Re-reads within the same transaction: give me products with price > 100 then it gives a different response. Adding a new row

Transaction is not a single operation, it is a group of operations. It could be multiple queries also

According to the issues, we will apply our Isolation level.

Isolation levels:
1. Read uncommitted changes – we want the Read operation to read the uncommitted changes as well.
- Allowing current transaction to read data from other ongoing transactions, which are not committed.

Problems solved:
- will it solve Dirty read? Or does dirty read still exist? => Not solved we will still have Dirty read issue
- Non-Repeatable reads? => Not solved
- Phantom read? => Not solved

It is implemented in MySQL. If it doesn't solve any of our issues, what does it do? What it ensures is, it will have a better reading experience on non-critical data. For example, number of likes on a YouTube video, an approximate count is good enough. We are not interested in a too accurate count.

2. Read committed:
- Only Read values when other transaction is committed or rolledback. We will read only committed values

Problems solved:
- Dirty read: Yes solved, we are not reading from any uncommitted transactions
- Non-Repeatable read: Not solved
- PhatomRead: Not solved

3. Repeatable Read Isolation Level:
If we are performing a Read operation on a particular row and even if that row's value gets changed by any other transactions, we will not read that.
- if a transaction is reading a row, it will basically not see any updates on that row during the transaction.

Transaction T1	ID	Name	Value	Transaction T2
Read this row	1	ABC	123	Update Name to 'BCD'

Say Transaction T1 is reading Row 1 and even if Name value changes to BCD from ABC by Transaction T2, still T1 will read only Name as ABC not BCD. We are not stopping value update by any other transactions. What we are stopping is, stopping to see the updated values. If Transaction T1 is reading a data, it will keep the same data even if any other transactions are updating it.

Problems solved:
- Dirty read: Yes solved
- NonRepeatable read: Yes (even if we update the data, T1 will still see the old data only)
- Phantom read: Not solved (if new row is inserted, it will see the new row)

However, the downside is, Transaction T1 will read stale data, which is not updated. That means, we will encounter a new problem that's Read skew (Reading unupdated data)
Postgres uses this Isolation. Postgres suggests that it is almost perfect (correct enough). It is giving responsibility to the Developers. Relying on Developers for atomic updates. We cant still use it for Banking application.

4. Serializable Isolation Level:
- strongest isolation level
- provides complete isolation
- it will consider every transaction in sequence

Problems solved:
- Dirty read: Yes solved
- NonRepeatable read: Yes solved
- Phantom read: Yes solved

Pros:
- Consistency

Cons:
- Blocking
- Deadlock
- Lower throughput

Only this is the strong isolation level, all others are weak isolation levels

Different application scenarios:
MoneyTransfer → Serializable Isolation Level
InventoryManagement → Serializable Isolation Level
BookingApplication → Serializable Isolation Level (otherwise we will book same seats for different people)
Dashboards → Read committed (We don't want Write operations to slowdown like Serializable)
SocialFeeds → Low isolation level
Analytics → Low isolation level (We don't want Write operations to slowdown as in Serializable)

MySQL uses Read Uncommitted Level. MySQL is a server, as a server, it is allowing Read uncommitted. With MySQL, it comes with an engine, InnoDB ensures that no transactions can read from uncommitted data changes. When we have a combination like this, then only we are able to use MySQL for our Banking applications. Read Uncommitted is the default setting for most applications. MySQL is the server and InnoDB is the storage engine. Storage engine makes sure that no read operations will be performed through uncommitted change. Even if MySQL server allows reading uncommitted changes, underlying InnoDB makes sure it doesn't happen.