Objectives

In this Lecture you will learn:

- Serializability and the purpose of its implementation.
- What is locking?
- Two types of locking
- 2PL algorithm.

1. Introduction

- To avoid data anomalies, the transaction log should linearize transactions so that there is no interleaving of reads and writes belonging to different transactions. To avoid conflicts, we must not interleave transactions.
- While the Serial execution avoids transaction interleaving since every transaction has exclusive access to the database, there is one way we could achieve the same goal without sacrificing parallelism. And that solution is called Serializability. Unlike Serial execution, Serializability allows multiple concurrent transitions to run, with one catch. The outcome needs to be equivalent to a Serial execution.
- Therefore, if both Alice and Bob are running two concurrent transactions, there are only two possible Serial execution outcomes:

  Alice followed by Bob
  Bob followed by Alice

If the statements in the transaction log follow this pattern, the outcome is said to be Serializable. In case there are three concurrent transactions, A, B, and C, there are 3! = 6 possible Serial execution outcomes. The order doesn't really matter for Serializability to be achieved. The only constraint is to get a Serial execution outcome.

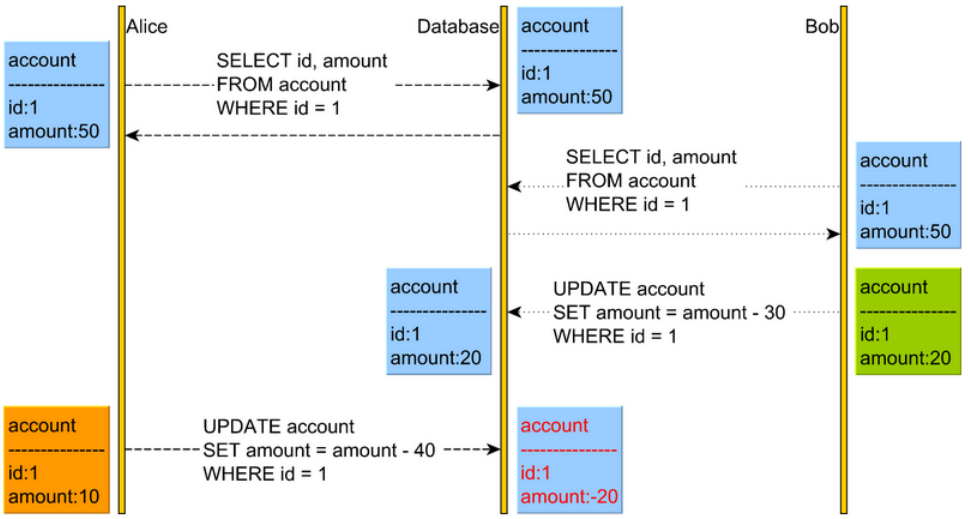


Fig1. A Concurrency Conflict Example.

2. Serializability

There are two possible ways to implement Serializability:

- 2PL (Two-Phase Locking), and this is what SQL Server and MySQL use to implement the Serializable isolation level.
- Serializable Snapshot Isolation, implemented by PostgreSQL.

2.1 Lock-based Protocols

We can define a lock-based protocol in DBMS as a mechanism that is responsible for preventing a transaction from reading or writing data until the necessary lock is obtained.

When multiple users or processes try to access or modify the same data concurrently, there is a risk of data inconsistency or corruption. Locks help prevent such issues by ensuring that only one user or process can access or modify a particular piece of data at a time.

- A lock-based protocol in DBMS prevents a transaction from reading or writing data until the necessary lock is obtained.
- The concurrency problem can be solved by securing or locking a transaction to a specific user.
- The lock is a variable that specifies which activities are allowed on a certain data item.

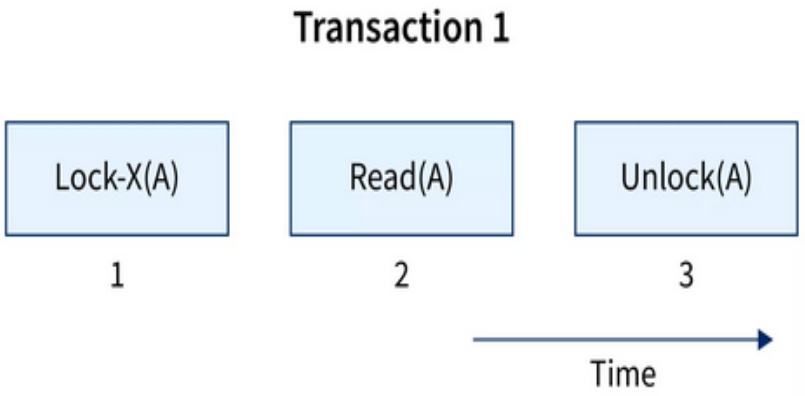


Fig2. General Timeline of a Lock-based protocol.

Example: Read after write problem.

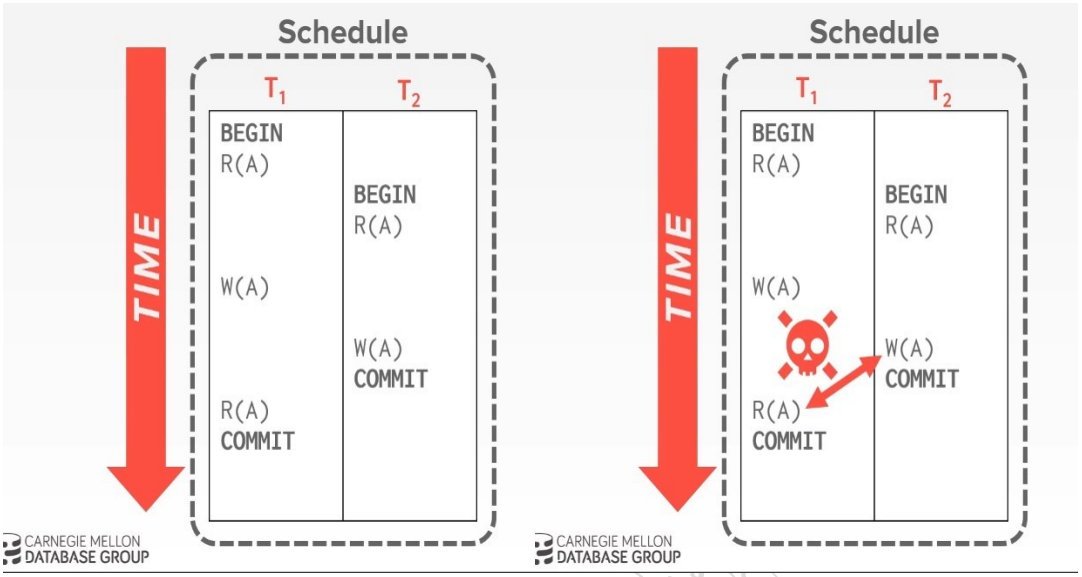


Fig3. Read after writing Conflict.

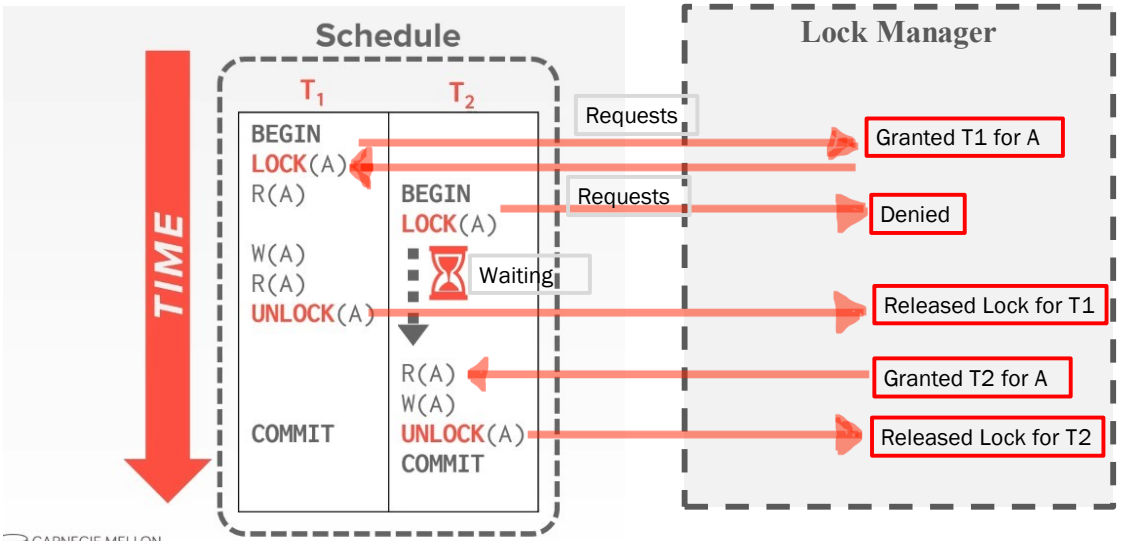


Fig4. Locking Tecknique.

2.2 Lock Types

Shared Locks (S-Locks):(for Read) Allow multiple transactions to read a resource simultaneously but prevent them from writing to it. This lock is acquired using the LOCK SHARED statement.

Exclusive Locks (X-Locks): (for Write) Ensure that only one transaction can modify a resource at a time, blocking other transactions from both reading and writing to it. This lock is acquired using the LOCK EXCLUSIVE statement.

	S-Lock	X-Lock
S-Lock	√	×
X-Lock	×	×

Fig. 5. Lock Compatibility Matrix.

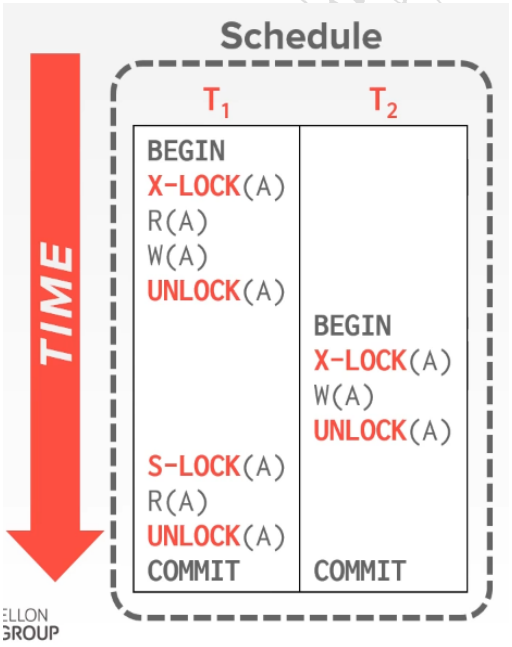


Fig. 6. Read after Write problem still exists after implementing 2 types of Lockings.

2.2 Two-phase Locking Protocol 2PL

- 2PL is a concurrency control strategy that determines how locks are being acquired and released because this also has an impact on transaction interleaving.
- If Locking as well as Unlocking can be performed in 2 phases, a transaction is considered to follow the Two-Phase Locking protocol. The two phases are known as the growing and shrinking phases:

- **Growing Phase**: acquires new locks, but none of them can be released.
- Shrinking phase: releases the existing locks, but no new locks can obtained.

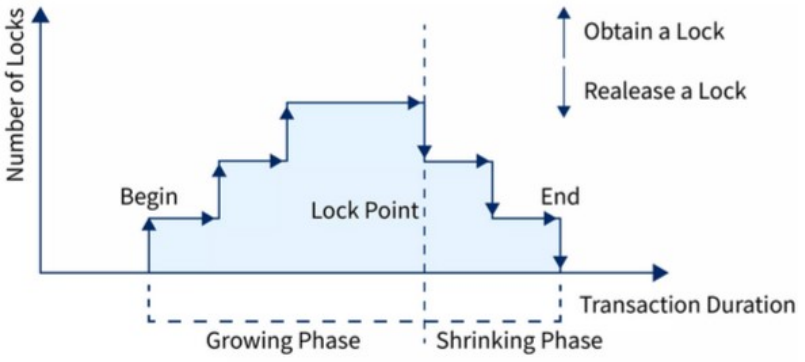


Fig. 7. 2PL Protocol.

Two-phase locking helps to reduce the amount of concurrency in a schedule but just like the two sides of a coin two-phase locking has a few cons too. The likelihood of establishing deadlocks is one bad result.

Example:
In the figure below:

- Both Alice and Bob acquire a read lock on a given post record via a SELECT FOR SHARE PostgreSQL clause.
- When Bob attempts to execute an UPDATE statement on the post entry, his statement is blocked by the Lock Manager because the UPDATE statement needs to acquire a write lock on the post row while Alice is still holding a read lock on this database record.
- Only after Alice's transaction ends and all her locks are released, Bob can resume his UPDATE operation.
- Bob's UPDATE statement will generate a lock upgrade, so his previously acquired read lock is replaced by an exclusive lock, which will prevent other transactions from acquiring a read or write lock on the same post record.
- Alice starts a new transaction and issues a SELECT FOR SHARE query with a read lock acquisition request for the same post entry, but the statement is blocked by the Lock Manager since Bob owns an exclusive lock on this record.
- After Bob's transaction is committed, all his locks are released, and Alice's SELECT query can be resumed.

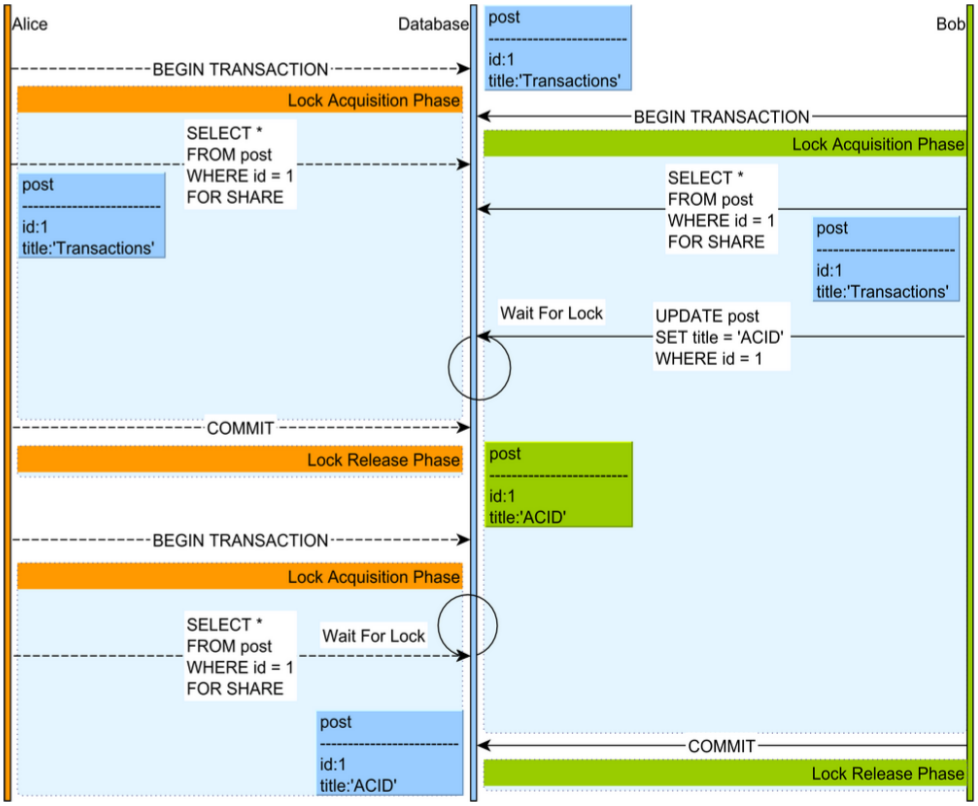


Fig. 8. 2PL Protocol.

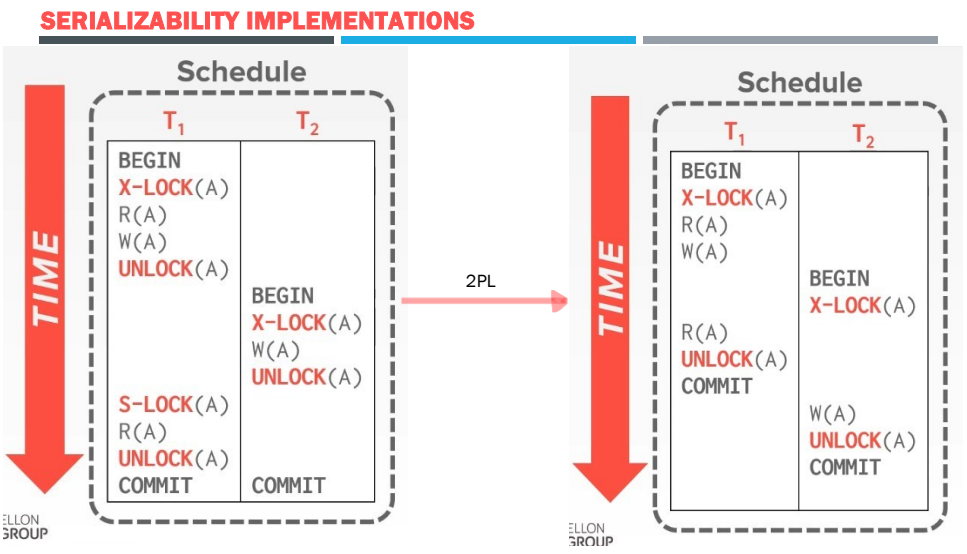
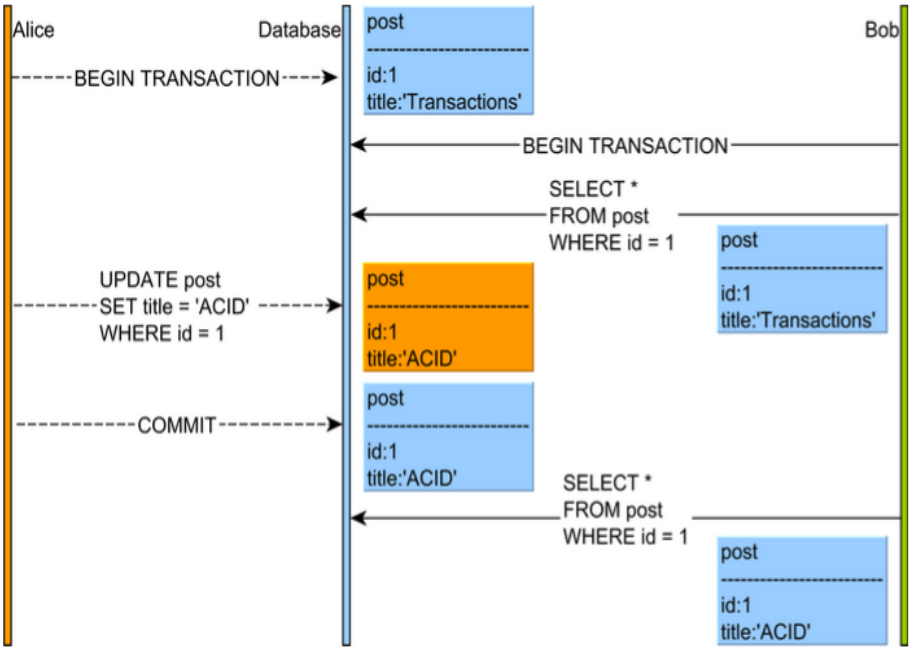


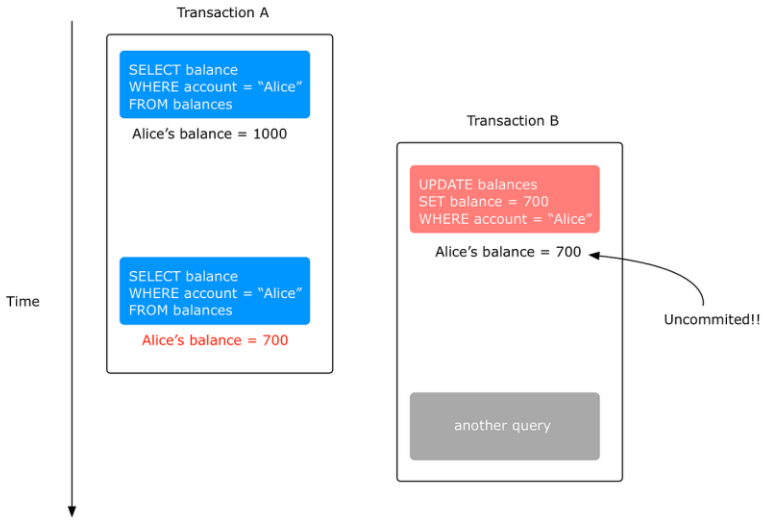
Fig. 9. 2PL Protocol to solve the read after write problem.

Appendix: Some Data Anomalies in Concurrent Transactions.

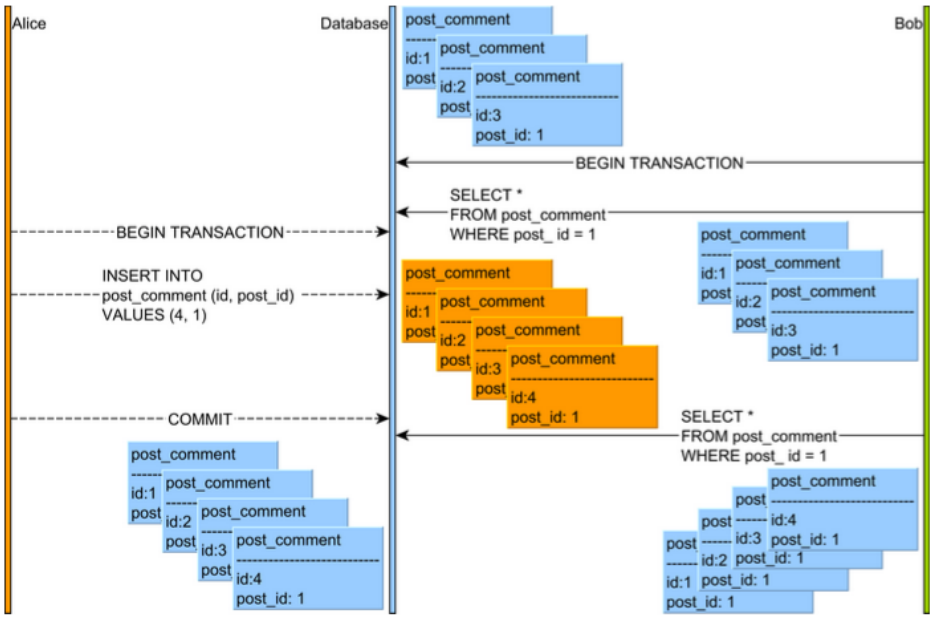
A1: **Non-repeatable Read** anomaly



1. Alice and Bob start two database transactions.
2. Bob's reads the post record and title column value is Transactions.
3. Alice modifies the title of a given post record to the value of ACID.
4. Alice commits her database transaction.
5. If Bob's re-reads the post record, he will observe a different version of this table row.



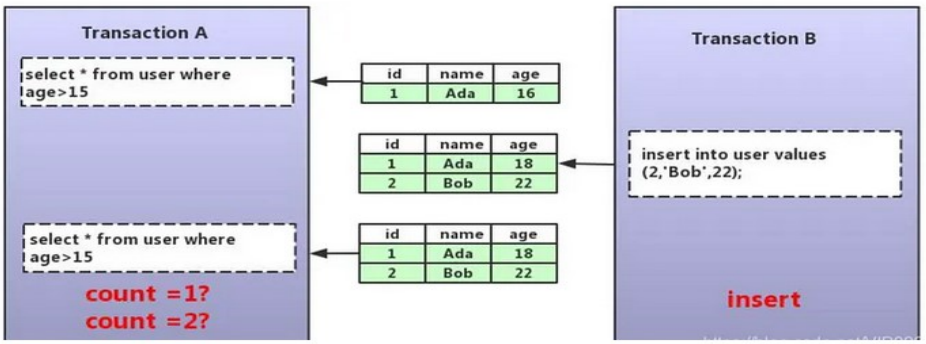
A2: **The Phantom Read** anomaly



1. Alice and Bob start two database transactions.
2. Bob's reads all the post_comment records associated with the post row with the identifier value of 1.
3. Alice adds a new post_comment record which is associated with the post row having the identifier value of 1.
4. Alice commits her database transaction.
5. If Bob's re-reads the post_comment records having the post_id column value equal to 1, he will observe a different version of this result set.

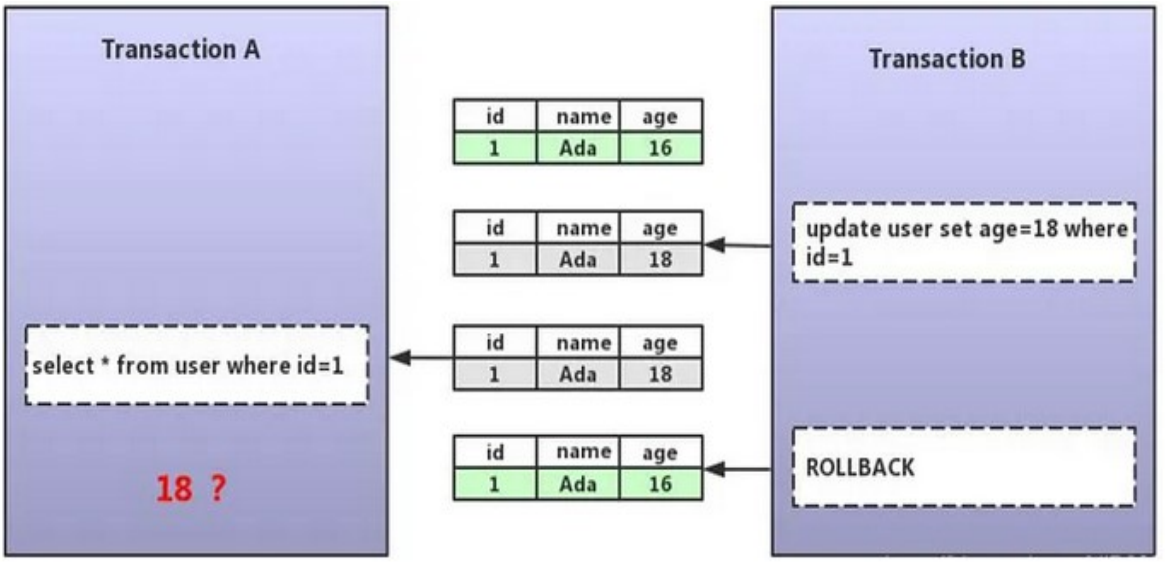
Phantom read

When the user reads records, another transaction inserts or deletes rows to the records being read. When the user reads the same rows again, a new "phantom" row will be found.



Transaction B inserts a new row where transaction A reads, then transaction A finds the total count of the result changes to 2

A3: **The dirty Read** anomaly



Transaction B is rolled back at this time, then the second transaction A reads dirty data which age is 18