

Concurrency control problem in more detail with examples:

1. Lost Updates:

- Problem: When two or more transactions try to update the same data simultaneously, one transaction's updates may be lost if they are overwritten by another transaction's updates.
- Example: Consider two bank transactions:
 - Transaction 1: Transfer \$100 from Account A to Account B.
 - Transaction 2: Transfer \$150 from Account A to Account B.
- If both transactions run concurrently and read the balance of Account A (e.g., \$500) at the same time, they may both calculate the new balance incorrectly. Without proper concurrency control, one of these transactions might update the balance to \$350, and the other may update it to \$400, causing a lost update.

2. Dirty Reads:

- Problem: A dirty read occurs when one transaction reads data that has been modified by another transaction but not yet committed.
- Example: Transaction 1 updates a customer's address:
 - Transaction 1: Update Customer A's address to "New Address."
 - Transaction 2 reads Customer A's data before Transaction 1 commits.
- If Transaction 1 rolls back for some reason after Transaction 2 has read the "New Address," Transaction 2 will have read incorrect and uncommitted data.

3. Non-Repeatable Reads:

- Problem: Inconsistent data can be read during multiple reads by a transaction if another transaction modifies the data in between.
- Example: Transaction 1 reads the price of a product:
 - Transaction 1: Read Product X's price (e.g., \$100).
 - Transaction 2 updates Product X's price to \$120.
 - Transaction 1 reads Product X's price again.
- Transaction 1 will observe a different price during the second read, resulting in non-repeatable reads.

4. Phantom Reads:

- Problem: A transaction reads a set of rows that satisfy a condition, but before the transaction can complete, another transaction inserts, updates, or deletes rows that match the condition.
- Example: Transaction 1 queries all products with a price less than \$50:
 - Transaction 1: `SELECT FROM Products WHERE Price < $50;`
 - Transaction 2 inserts a new product with a price of \$40.
 - Transaction 1 re-executes the same query.
- Transaction 1 will see a different set of rows in the second query, including the newly inserted product, causing phantom reads.

5. Deadlocks:

- Problem: Deadlocks occur when two or more transactions are waiting for each other to release locks on resources, leading to a system stall.
- Example: Consider two transactions:
 - Transaction 1: Locks Resource A, requests Resource B.
 - Transaction 2: Locks Resource B, requests Resource A.
- If both transactions acquire their initial locks and then request the opposite resource, they will enter a deadlock state, as neither can proceed without the other releasing its lock.

To mitigate these problems, DBMSs use various concurrency control mechanisms such as locking, timestamps, and isolation levels (e.g., Read Committed, Serializable). These mechanisms ensure that transactions maintain data consistency and integrity while allowing for concurrent execution. Properly configuring these mechanisms and testing your application's behavior under concurrent conditions is crucial for avoiding these concurrency control problems.