## Why Concurrency Control is Needed in DBMS?

Concurrency control is essential in a Database Management System (DBMS) to manage the simultaneous execution of multiple transactions in a multi-user environment. Without it, several problems can arise that compromise data integrity and consistency.

**Need for Concurrency Control:**

- **Maintain Data Consistency:** Concurrent transactions accessing and modifying the same data can lead to inconsistent database states if their operations are not properly synchronized. Concurrency control mechanisms ensure that the database remains in a consistent state after the execution of concurrent transactions, as if they were executed serially.
- **Prevent Data Anomalies:** Uncontrolled concurrency can result in various data anomalies, such as lost updates, dirty reads, and incorrect summaries, which can lead to incorrect information and flawed decision-making.
- **Ensure Transaction Isolation:** Concurrency control aims to provide transaction isolation, meaning that each transaction should appear to execute in isolation from other concurrently running transactions. This prevents one transaction from being affected by the uncommitted changes of another.
- **Improve Resource Utilization:** By allowing multiple transactions to run concurrently, the DBMS can improve resource utilization (CPU, memory, disk I/O) and increase throughput, leading to better overall system performance.

**Data Anomalies due to Lack of Concurrency Control:**

Here are three common data anomalies that can occur without proper concurrency control:

- **Lost Update Problem:**
  - **Description:** Occurs when two or more transactions access the same data item, and their operations interleave in a way that the changes made by one transaction are overwritten by another transaction, leading to the loss of the first transaction's update.
  - **Example:**
    1. Transaction T1 reads the balance of account A (say, X=$100).
    2. Transaction T2 reads the balance of account A (X=$100).
    3. T1 deducts $20 from the balance and updates A to $80.
    4. T2 adds $50 to the original balance it read ($100) and updates A to $150.
    5. The update by T1 (deducting $20) is lost because T2 overwrites it with its

own update based on the initially read value. The final balance should ideally be $130 ($100 - $20 + $50), but it becomes $150.

- **Dirty Read Problem (Uncommitted Dependency):**
  - **Description:** Occurs when one transaction reads data that has been modified by another transaction but not yet committed. If the modifying transaction[1] subsequently aborts, the first transaction has read inconsistent and potentially incorrect data.
  - **Example:**
    1. Transaction T1 updates the balance of account B from $200 to $250. (This update is not yet committed).
    2. Transaction T2 reads the uncommitted balance of account B ($250) and proceeds with its operations based on this value.
    3. Transaction T1 encounters an error and aborts, rolling back the balance of account B to $200.
    4. Transaction T2 has now used a value ($250) that was never actually committed to the database, leading to potential inconsistencies in its own operations and the final database state.
- **Incorrect Summary Problem (Non-Repeatable Read in a Summary Context):**
  - **Description:** Occurs when one transaction is performing an aggregate function (like sum or average) over a set of data items while other transactions are concurrently updating those same data items. The result of the aggregate function might be incorrect because it reflects some values before the updates and some values after the updates.
  - **Example:**
    1. Transaction T1 calculates the total balance of all accounts by reading each account balance and summing them up.
    2. While T1 is in the process of reading and summing, transaction T2 transfers $100 from account C to account D and commits its changes.
    3. The total calculated by T1 might include the balance of account C before the transfer and the balance of account D after the transfer, resulting in an incorrect total balance that never actually existed in the database at any single point in time.

In conclusion, concurrency control mechanisms are vital for ensuring the reliability, consistency, and integrity of database systems in the presence of concurrent transactions. They prevent data anomalies and guarantee that the outcome of concurrent execution is equivalent to some serial execution.