Online Banking System

Database Management System(DBMS)

By Janaki Neupane

BCSIT—III

Online Banking System

Scenario Overview:

You are managing the database system for an online banking platform. The system supports multiple users who can perform transactions simultaneously, such as transferring money between accounts, checking, balances, and updating personal details. To ensure the integrity and consistency of the data, it’s important to control concurrency and handle transactions effectively, especially when multiple users perform operations at the same time.

Questions :

1.What are the ACID properties of a transaction, and how do they contribute to ensuring the integrity of banking data?

The **ACID** properties are a set of principles that ensure reliable processing of database transactions. They stand for **Atomicity**, **Consistency**, **Isolation**, and **Durability**, and they play a crucial role in maintaining the integrity of data in an online banking system:

1. Atomicity

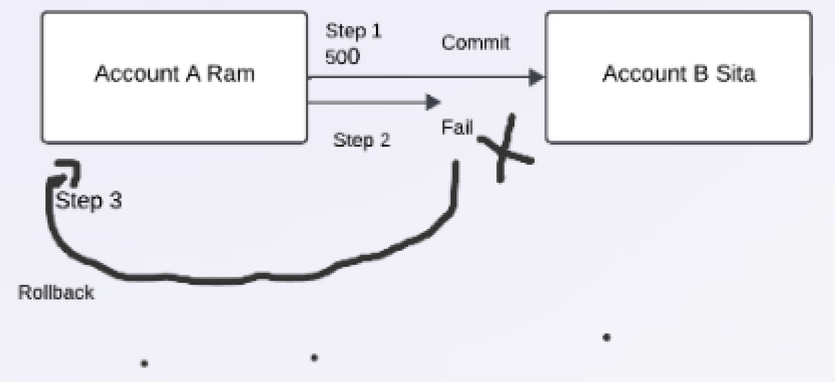
- **Definition:** A transaction must be treated as a single, indivisible unit of work. Either all the operations within the transaction are executed successfully, or none of them are applied.
- **Contribution:**
    - Ensures that partial transactions (e.g., only debiting an account but failing to credit another in a transfer) do not occur.
    - Example: When transferring money between two accounts, if the debit operation succeeds but the credit operation fails, the entire transaction will be rolled back to maintain balance integrity.

2.Consistency

Definition: A transaction must ensure that the database moves from one valid state to another, maintaining all integrity constraints.

- **Contribution:**
    - Prevents invalid states, such as negative balances in accounts or violation of business rules.
    - Example: If a bank’s rule requires that the sum of all accounts must remain constant during a transfer, the transaction will ensure this consistency.

Atomicity

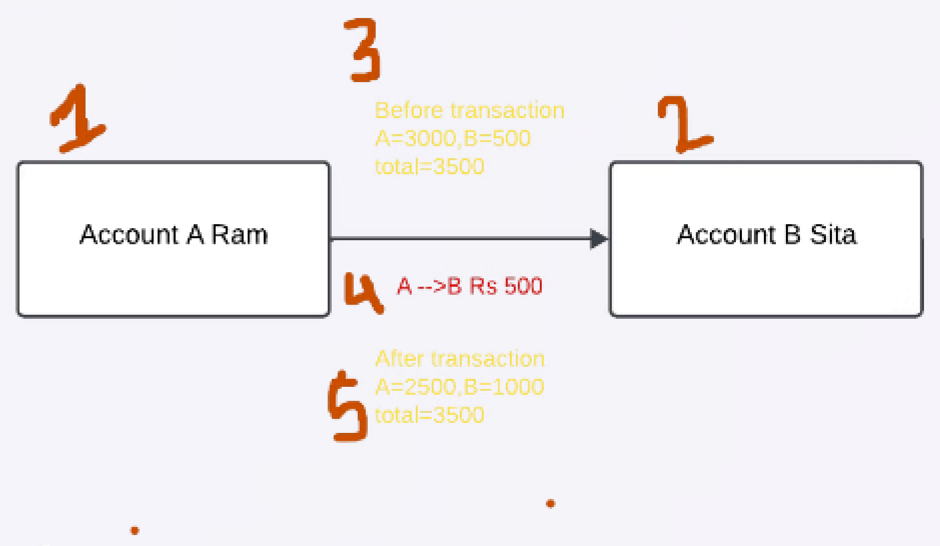


Example: Imagine transferring ₹500 from Account A to Account B.

Step 1: Debit ₹500 from Account A.

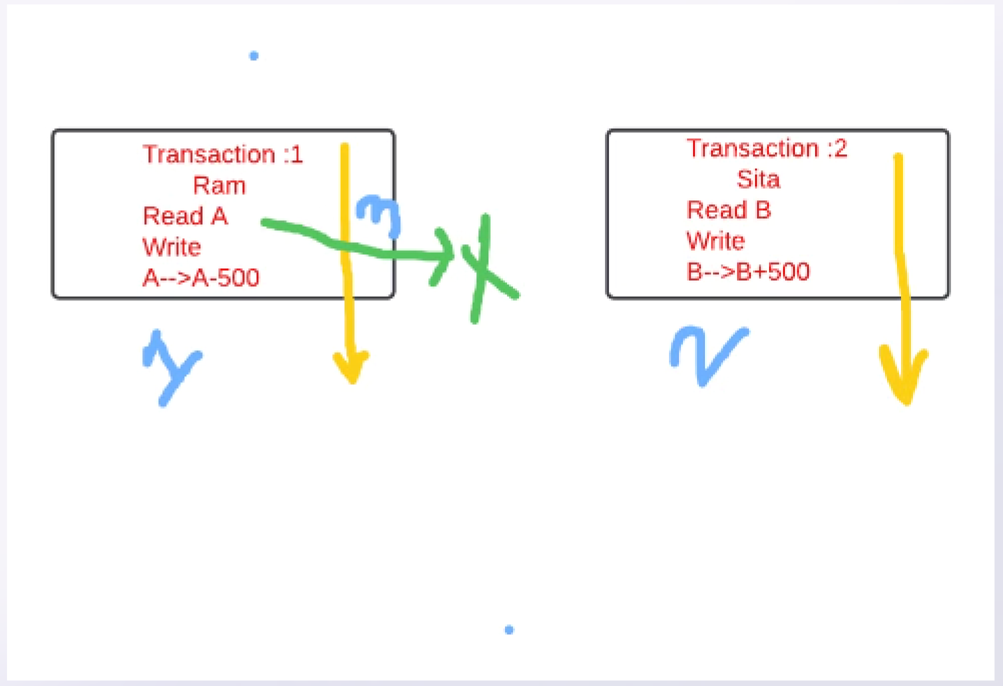
Step 2: Credit ₹500 to Account B. If Step 2 fails (e.g., due to a system crash), Step 1 should be rolled back to maintain the system's original state

Consistency



Example: In the same transfer example, if the total balance of Account A and Account B before the transaction is ₹3500, the sum must still be ₹3500 after the transaction. Any violation of constraints (e.g., overdrawing Account A beyond its limit) would prevent the transaction from committing.

Isolation



Example: Suppose two transactions are occurring simultaneously.

Transaction 1: Transfers ₹500 from Account A to Account B. Transaction 2: Reads the balance of Account B. Transaction 2 should not see the intermediate state of Account B while Transaction 1 is still in progress

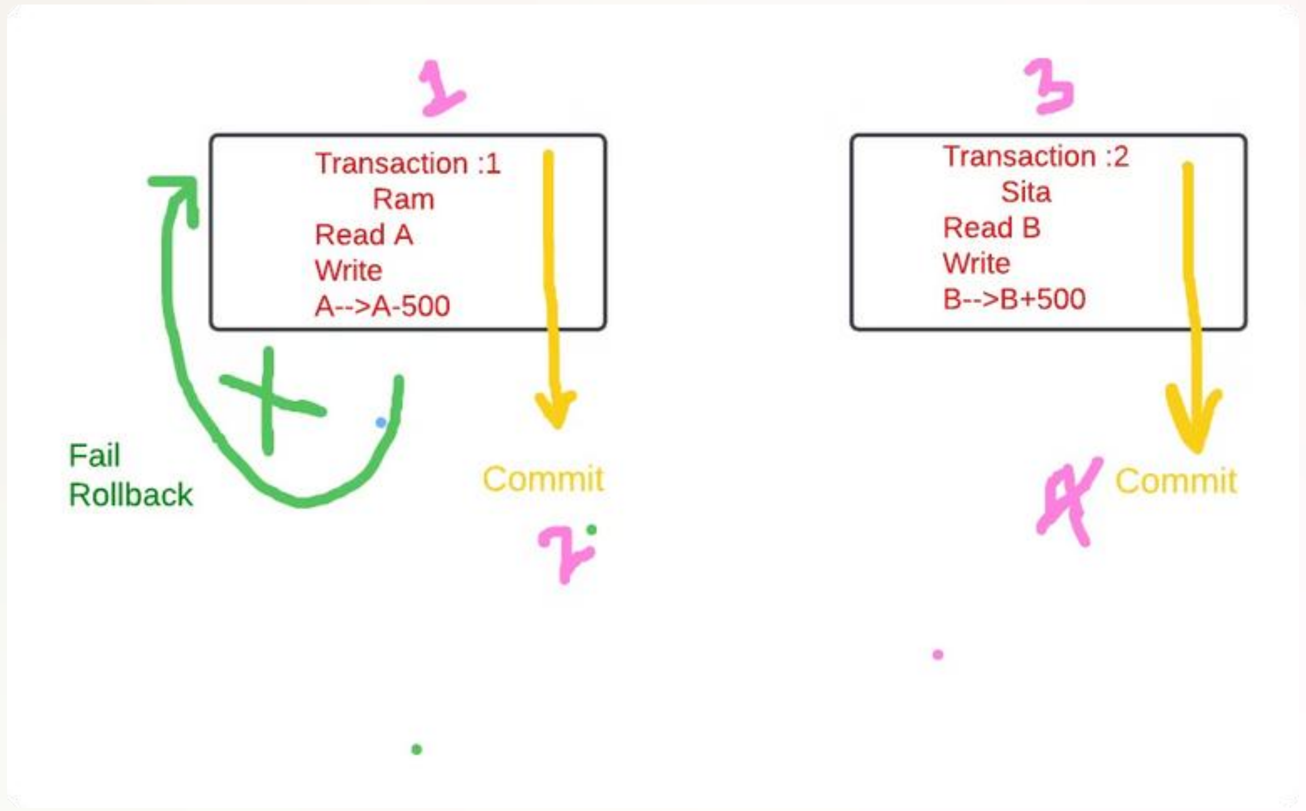
3. Isolation

- **Definition:** Transactions must be executed independently, as if they are the only transaction running in the system, even when multiple transactions occur concurrently.
- **Contribution:**
  - Prevents problems like dirty reads, lost updates, and phantom reads that could occur when users simultaneously access or modify data.
  - Example: If one user is transferring funds while another checks their balance, the balance inquiry should reflect either the state before or after the transfer, but not a partially updated state.

4. Durability

- **Definition:** Once a transaction is committed, its changes are permanent and must survive system failures, such as power outages or crashes.
- **Contribution:**
  - Ensures that users can trust the system to retain their transaction data reliably.
  - Example: After a fund transfer is confirmed, the changes (debit and credit) will persist even if the server crashes immediately afterward.

Example **Durability**: After the transfer of ₹500 from Account A to Account B is committed, the changes are permanently saved. Even if the system crashes right after the commit, the updated balances in both accounts should be preserved.



2.Can you give an example of a situation where two concurrent transactions might violate serializability in an online banking system?

Example: Violation of Serializability in an Online Banking System

Scenario: Consider a bank with two accounts, **Account A** and **Account B**. Two concurrent transactions are initiated:

1. **Transaction 1 (T1):** Transfer $100 from **Account A** to **Account B**.
1. **Transaction 2 (T2):** Check the total balance of **Account A** and **Account B**.

Initial State:

- Balance of **Account A**: $500
- Balance of **Account B**: $300
- Total Balance: $800

Steps of Concurrent Execution (Non-Serializable):

1. **Step 1 (T2 reads initial balances):**
   - Transaction T2 reads the balance of **Account A**: $500.
   - Transaction T2 reads the balance of **Account B**: $300.
   - Total balance calculated by T2: $800 (correct so far).
2. **Step 2 (T1 debits Account A):**
   - Transaction T1 deducts $100 from **Account A**.
   - Balance of **Account A** becomes $400
3. **Step 3 (T2 reads modified balance of Account A):**
   - Transaction T2 reads the updated balance of **Account A**: $400.
   - Transaction T2 reads the unchanged balance of **Account B**: $300.
   - Total balance calculated by T2: $700 (incorrect because T2 missed the $100 being transferred to **Account B**).
4. **Step 4 (T1 credits Account B):**
   - Transaction T1 adds $100 to **Account B**.
   - Balance of **Account B** becomes $400.

Issue:

- Transaction T2 calculated the total balance as **$700**, which violates the integrity of the system since the actual total should always be **$800**.
- This discrepancy arises because T2 observed a partial state of T1, breaking **serializability**.

Ensuring Serializability:

To prevent such issues, the system can employ **concurrency control mechanisms**, such as:

- **Two-Phase Locking (2PL):** Ensures that transactions lock resources in a way that no other transaction can access them until the locks are released.

3.How do locks (shared and exclusive) help maintain consistency when multiple users access bank accounts simultaneously?

Locks are essential in maintaining **consistency** in a database system when multiple users access or modify data simultaneously. In the context of an online banking system, **shared locks** and **exclusive locks** help ensure that transactions adhere to the ACID properties and prevent conflicts or inconsistencies.

Types of lock:

Shared Lock (S-lock):

- **Purpose:** Allows multiple transactions to **read** a resource (e.g., account balance) simultaneously but prevents any transaction from modifying the resource while it is locked.
- **Use Case:** Suitable for read-only operations where consistency must be maintained without the risk of data modification.

Exclusive Lock (X-lock):

- **Purpose:** Allows a single transaction to **modify** a resource while preventing other transactions from reading or modifying the resource until the lock is released.
- **Use Case:** Suitable for write operations where isolation is required to prevent data inconsistencies.

How Locks Maintain Consistency:

Scenario: Two users, **User A** and **User B**, access **Account X** simultaneously:

- **User A**: Transfers $200 from **Account X** to **Account Y**.
- **User B**: Checks the balance of **Account X**.

Without Locks:

- User A initiates a transfer and debits $200 from **Account X**.
- Before User A completes the transfer (e.g., crediting **Account Y**), User B reads the balance of **Account X**.
- User B sees an **inconsistent state**:
  - □ The balance reflects the debit operation but not the completion of the transfer.
- **Result:**
- User B might see an incorrect or incomplete balance, leading to **data inconsistency**.

With Locks:

1.**Step 1: User A** initiates the transfer and acquires an **exclusive lock** on **Account X**.

- □ No other transaction can read or write to **Account X** until **User A** completes the transfer and releases the lock.

1.**Step 2: User B** attempts to read the balance of **Account X**.

- □ Since **User A** holds an exclusive lock, **User B** is forced to wait until the lock is released.

1.**Step 3: User A** completes the transfer and releases the lock.

- □ **User B** acquires a **shared lock** to read the balance, ensuring a consistent view of the data.

Aspect	Without Locks	With Locks
Transaction Initiation	User A initiates a transfer from **Account X** to **Account Y**.	User A initiates a transfer from **Account X** to **Account Y**.
Lock Mechanism	No locks are acquired.	User A acquires an **exclusive lock** on **Account X**.
User B's Action	User B attempts to read the balance of **Account X** while User A's transaction is incomplete.	User B attempts to read the balance of **Account X**, but is **blocked** until the lock is released.
Data Access by User B	User B reads an **inconsistent balance**: reflects the debit but not the credit.	User B reads the balance only after User A completes the transfer, ensuring **consistency**.
Transaction Interference	User B's read operation interferes with User A's transaction, leading to **dirty reads** or partial updates.	User B's operation is isolated from User A's transaction, preventing interference.
Consistency of Data	**Violated**: User B sees an incorrect balance due to incomplete transaction.	**Ensured**: User B sees the correct and final balance after the transaction is committed.
System Behavior	High risk of anomalies, such as dirty reads or inconsistent data views.	Enforces **isolation** and prevents anomalies, ensuring data integrity.
Performance Impact	No delay for User B, but at the cost of **data inconsistency**.	User B experiences a slight delay but receives **accurate and consistent data**.

Benefits of Locks in an Online Banking System:

1. **Prevents Dirty Reads:**
    - Shared locks ensure that transactions only read committed data, avoiding intermediate states of other transactions.
1. **Prevents Lost Updates:**
    - Exclusive locks prevent simultaneous write operations that could overwrite each other's changes.
1. **Maintains Isolation:**
    - Locks enforce transaction boundaries, ensuring that concurrent operations do not interfere with each other.
1. **Ensures Atomicity:**
    - Locks guarantee that transactions complete fully or not at all, avoiding partial updates

Transaction	**Operation**	**Lock Type**	**Effect**
T1	Read balance of Account X	Shared (S)	Multiple transactions can read simultaneously, ensuring consistent views.
T2	Update balance of Account X	Exclusive (X)	Only one transaction can write; no reads or writes allowed until committed.

4.What are the differences between shared locks and exclusive locks in the context of a banking system’s transaction management?

Aspect	Shared Lock(S-Lock)	Exclusive Lock(X-Lock)
Concurrency	Enables concurrent **read operations** by multiple transactions.	Prevents both **read** and **write** by other transactions.
Use Case	For **read-only operations**, such as checking account balances.	For **write operations**, such as transferring funds or updating details.
Blocking Behavior	- Other transactions can acquire shared locks simultaneously. - Blocks exclusive locks until the shared lock is released.	- Prevents both shared and exclusive locks until the exclusive lock is released.
Impact on Performance	Increases concurrency by allowing multiple reads without delays.	Reduces concurrency as it prevents other operations on the locked resource.
Example in Banking	- A user checks the balance of **Account A**, acquiring a shared lock. - Other users can also check the balance simultaneously.	- A user transfers money from **Account A**, acquiring an exclusive lock. - No other transaction can read or modify **Account A** until the transfer is complete.
Data Consistency	Ensures consistency by preventing **uncommitted writes** from being read.	Ensures consistency by preventing simultaneous **writes** or **reads** during a modification.
Lock Duration	Typically held for the duration of the read operation.	Typically held until the transaction is committed or rolled back.