Case Study 1: ATM Withdrawals and ACID Properties

Scenario:
A user withdraws ₹10,000 from an ATM. The bank deducts the amount and updates the user's account balance. The transaction must reflect in the database only if all steps succeed.

Questions:

1. Which ACID properties are important here?
2. What happens if a system failure occurs before updating the balance?

Answers:

1. All four: **Atomicity**, **Consistency**, **Isolation**, **Durability**
2. If failure happens mid-way, **Atomicity** ensures rollback. The transaction won't be partially completed.

Explanation:
Banking systems use **log-based recovery** to roll back incomplete transactions and maintain consistency. Atomicity guarantees no partial deduction.

Case Study 2: Two Users Transferring Money (Lost Update Problem)

Scenario:
User A and User B are transferring money to the same account at the same time. Both read the same old balance and try to update it concurrently.

Questions:

1. What concurrency issue may arise here?
2. How to prevent it?

Answers:

1. **Lost Update Problem**
2. Use **locking** (Exclusive Lock) or **2PL (Two-Phase Locking)** to serialize access.

Explanation:
If both users read ₹5,000 and each deposits ₹1,000, the final balance should be ₹7,000. Without locking, one update might overwrite the other → final result = ₹6,000. Using locks ensures serialized access.

Case Study 3: Airline Ticket Booking (Dirty Read Issue)

Scenario:
User A books a flight ticket. Before the transaction is committed, User B checks seat availability. B sees the uncommitted changes.

Questions:

1. What concurrency problem is seen here?
2. What isolation level prevents this?

Answers:

1. **Dirty Read**
2. **Read Committed** or **Repeatable Read**

Explanation:
User B read a value that might be rolled back. To avoid this, DBMS enforces **isolation levels**—ensuring only committed changes are visible to other users.

Case Study 4: Online Shopping Cart (Serializability)

Scenario:
User A adds an item to a cart. User B updates the inventory stock simultaneously. Both transactions affect the same product.

Questions:

1. What is required to ensure correctness?
2. What is the use of serializability?

Answers:

1. **Serializable Schedule**
2. It ensures the outcome is equivalent to transactions running one after another.

Explanation:
Without serialization, if two transactions affect the same data, the result may be incorrect. DBMS checks **conflict serializability** to validate safe execution order.

Case Study 5: Power Failure During Transaction (Log-Based Recovery)

Scenario:
A power failure occurs while a bank is transferring ₹5,000 from one account to another. The debit is done but credit is pending.

Questions:

1. What technique helps restore the DB?
2. What kind of log record is needed?

Answers:

1. **Log-Based Recovery** (Using undo/redo logs)
2. **Before and after** image of data must be recorded in log files

Explanation:
If system crashes mid-transaction, logs help to either **undo** incomplete transactions or **redo** committed ones after reboot.