Lecture-6

Database Management system: *Transactions, Concurrency & Recovery*

- **Topics:**
  - ACID properties, Transaction States.
  - Concurrency issues: Lost update, dirty read, etc.
  - Serializability, locking technique (2pl, Shared & Exclusive). (With Real Time case Studies)
  - Recovery techniques: Log-based, Shadow Paging.

What is a Transaction?

In DBMS, a transaction is a sequence of one or more operations that are executed as a single logical unit of work.

Example:

Transferring ₹1000 from Account A to B:
1. Deduct ₹1000 from A
2. Add ₹1000 to B

BEGIN;
UPDATE accounts SET balance = balance - 1000 WHERE id = 1;
UPDATE accounts SET balance = balance + 1000 WHERE id = 2;
COMMIT;

ACID Properties & Transaction States

ACID Properties:

Atomicity – All or nothing
Consistency – Valid state transitions
Isolation – Concurrent execution doesn’t cause issues
Durability – Changes are permanent

ACID properties in DBMS

ACID

A = Atomicity
All or nothing transactions

C = Consistency
Valid state maintenance

I = Isolation
Transactions do not affcet eachother

D = Durability
Permanent changes after commitment

States of a Transaction

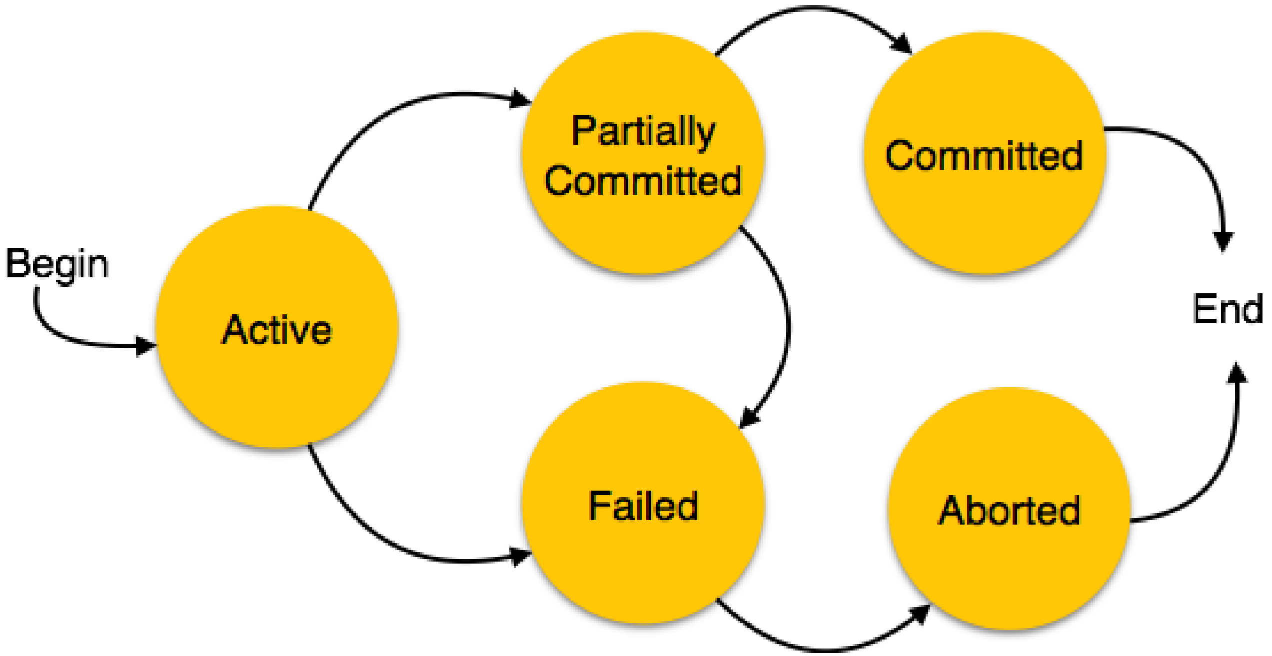


Figure 1: Transaction states

Concurrency

Concurrency in a DBMS refers to the execution of multiple transactions at the same time.

Improves

i)throughput and

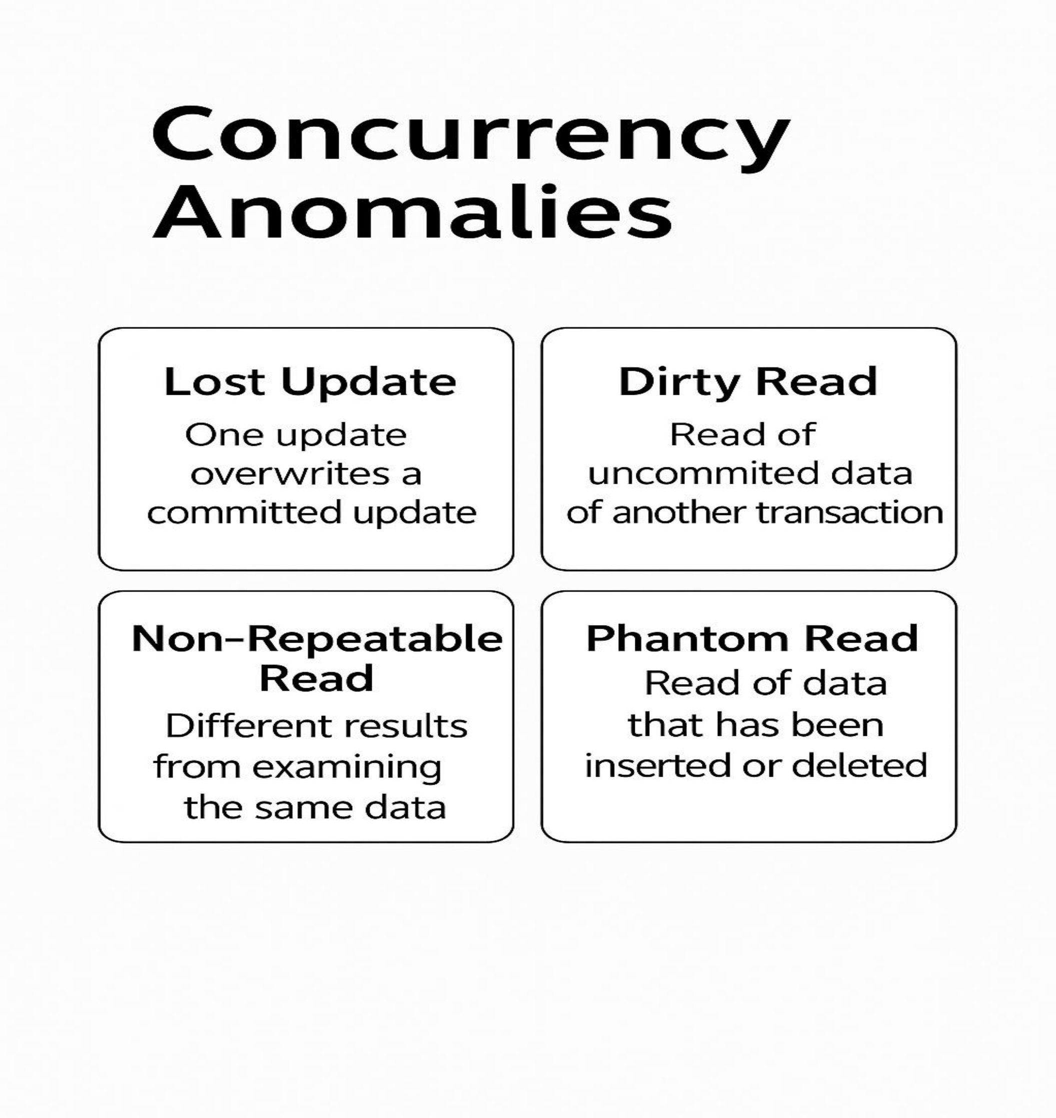
ii)resource utilisation,

But it also risks transactions interfering with each other’s reads and writes.

Concurrency Issues in Transactions

Lost Update – Two transactions overwrite each other
Dirty Read – Reading uncommitted data from another transaction
Non-repeatable Read – Same query gives different results
Phantom Read – Rows appear/disappear in repeated queries

Concurrency Anomalies



Concurrency Control

Helps manage multiple transactions at once.

Example:
T1 reads balance ₹5000 and T2 updates to ₹4000.
Without control: T1 overwrites T2’s changes.

How Concurrency Control Prevents Errors

- **Two-Phase Locking (2PL)**
  - *Growing phase*: acquire all needed locks (shared/exclusive)
  - *Shrinking phase*: release locks
- **Timestamp Ordering**
  - Assign each transaction a unique timestamp
  - Enforce that older transactions' reads/writes occur "as if" in timestamp order

Two-Phase Locking (2PL)

Shared & Exclusive Locks

Shared Lock (S):
- Allows multiple transactions to read a data item.
- No transaction can write while shared lock is held.

Exclusive Lock (X):
- Only one transaction can hold an exclusive lock.
- Allows both read and write access.
- Prevents other transactions from reading or writing.

Two-Phase Locking (2PL)

Ensures conflict serializability.
Prevents dirty reads and lost updates.
Drawbacks: May cause deadlocks if transactions wait for each other’s locks.

Serializability

Serializability ensures the outcome of transactions is the same as if executed serially.

Types:

- Conflict Serializability: Swap non-conflicting operations to achieve serial order.
- View Serializability: Final state and read operations match a serial schedule.

Conflicts: Read-Write, Write-Read, Write-Write between transactions on same data item.

Goal: Maintain database consistency during concurrent transaction execution.

Recovery Techniques

- Log-Based Recovery – Maintain log records for undo/redo
- Deferred and Immediate Update methods
- Shadow Paging – Maintain two-page tables (current & shadow)
- Used to restore consistent DB state after crash

Log-Based Recovery

Log records every change made to the database.
Types:
- Undo Logging: Before-image is stored. Rollback restores previous state.
- Redo Logging: After-image is stored. Redo reapplies operations on recovery.

Checkpointing used to minimize recovery scope.
Essential for both committed and uncommitted transaction recovery.

Shadow Paging

Shadow Paging avoids logs by maintaining two page tables:
- Shadow Page Table (original)
- Current Page Table (updated during transaction)

During commit, updated pages are made permanent by replacing shadow table.

Efficient for small transactions.

Drawbacks: Difficult to handle concurrent transactions and page deallocation.

Case Study: Concurrent Transactions in Banking

Multiple users transferring money simultaneously.

Ensuring no balance inconsistencies occur (Lost update, Dirty read).

Using locks on account rows during debit/credit.

Log entries to recover from crashes.

Problem Statement

Multiple users transferring money simultaneously

Issues to address:

- Lost updates (e.g., race conditions on account balances)
- Dirty reads (uncommitted data access)
- Crash recovery (partial transactions, disk failure)

Solution Overview

1. Use ACID-compliant transactions
2. Apply row-level locking (2PL with Shared & Exclusive Locks)
3. Maintain log-based recovery (Write-Ahead Logging)
4. Use appropriate isolation levels (Serializable or Repeatable Read)
5. Implement checkpoints for recovery

Transaction Execution Flow

1. User A initiates transfer to User B
2. Start transaction and lock both account rows (exclusive lock)
3. Debit from A, credit to B
4. Write changes to log (WAL)
5. If successful: COMMIT; On failure: ROLLBACK using logs

Result & Benefits

✓ Prevents dirty reads and lost updates
✓ Ensures atomic, isolated transaction execution
✓ Crash recovery via logs (undo/redo)
✓ Maintains data consistency in concurrent environment

Discussion

- How do modern banking systems manage concurrency?
- What mechanisms are used to avoid anomalies?
- Can you relate ACID to banking software you use?
- Which locking mechanism would best suit ATM withdrawals?
- How would you design recovery after a server crash?

Thank You