Lecture Notes on Transaction Management and Recovery

1. System Recovery

System recovery in DBMS ensures that a database remains consistent and operational after system failures such as crashes, power failures, or disk errors. Recovery mechanisms help restore the database to a consistent state.

Types of Failures

1. **Transaction Failures** - Due to logical errors or system constraints.

2. **System Crashes** - Unexpected shutdowns due to hardware/software failure.

3. **Disk Failures** - Corruptions in storage media.

Recovery Techniques

1. **Immediate Update** - Updates are written to the database immediately but logs are maintained.

2. **Deferred Update** - Updates are stored in logs and applied to the database only if the transaction commits.

3. **Checkpoints** - Periodic saving of database states to minimize recovery time.

2. Two-Phase Commit (2PC) Protocol

Two-Phase Commit (2PC) ensures atomicity in distributed databases. It guarantees that a transaction either commits or aborts across multiple sites.

Phases of 2PC

1. **Prepare Phase**:
    - Coordinator sends a "Prepare" request to all participants.
    - Participants respond with "Yes" (ready) or "No" (abort).
2. **Commit Phase**:
    - If all participants respond "Yes," the coordinator sends a "Commit" message.
    - If any participant responds "No," the coordinator sends an "Abort" message.

Example

A bank transfer from Account A (Bank1) to Account B (Bank2):

- Bank1 locks A’s balance and prepares to deduct funds.
- Bank2 prepares to add the amount to B.
- If both banks agree, the transaction commits; otherwise, it aborts.

3. Recovery and Atomicity

Atomicity ensures that a transaction is either fully completed or not executed at all.

Atomicity Recovery Methods

1. **Undo/Redo Logging** - Maintains records to rollback or reapply changes.

2. **Shadow Paging** - A backup page table is maintained, and changes are applied only after a successful commit.

---

4. Log-Based Recovery

Logs record all modifications for recovery purposes.

Types of Logging

1. **Undo Logging -** Records changes before execution (Rollback possible).

2. **Redo Logging** - Records changes after execution (Reapply changes if needed).

3. **Undo/Redo Logging** - Combination of both for complete recovery.

Example

A transaction modifying a bank balance:

- **Before Update**: Balance = $500 (Stored in log).

- **After Update**: Balance = $600.

- If failure occurs before commit, rollback restores balance to $500.

5. Concurrent Executions of Transactions and Related Problems

When multiple transactions run simultaneously, they may cause issues like inconsistency and deadlocks.

Problems in Concurrent Transactions

1. **Dirty Read** - A transaction reads uncommitted data of another transaction.
2. **Lost Update** - One transaction overwrites the update of another transaction.
3. **Non-Repeatable Read** - A repeated read yields different results.
4. **Phantom Read** - A transaction reads a dataset that changes before it completes.

Concurrency Control Techniques

1. **Lock-Based Protocols** - Ensure transactions lock resources before access.
2. **Timestamp-Based Protocols** - Assign timestamps to transactions to order execution.
3. **Multiversion Concurrency Control (MVCC)** - Maintains multiple versions of data to avoid conflicts.

Conclusion: These recovery and concurrency control mechanisms ensure the integrity, consistency, and durability of a database even in the presence of failures and concurrent transactions. Implementing proper logging and recovery methods is crucial for robust database management.