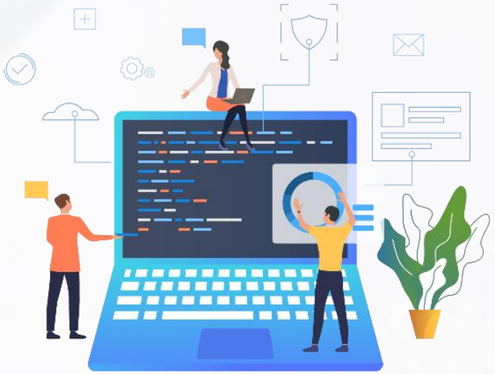
25CS305
Database Management Systems (DBMS)

Unit-5

Transaction Management



CSE(Cyber Security), NRCM

Department of CSE(Cyber Security)

Narsimha Reddy Engineering College (NRCM)

Transaction Management

Concurrency Control & Recovery

Transaction Concept · Serializability · Isolation

Lock Protocols · Log-based Recovery · Shadow Paging

Contents Overview

1

Transaction Mgmt

Concept, States, ACID, Serializability, Recoverability, Isolation

2

Concurrency Control

Lock-based, Timestamp-based, Validation-based Protocols

3

Recovery System

Log-based Recovery, Shadow Paging, Buffer Management

This unit covers the **fundamental mechanisms** that ensure database transactions are processed reliably in multi-user environments. We explore how DBMS maintains **ACID properties** through concurrency control and recovery techniques.

PART 01

Transaction Concept, States, ACID Properties & Serializability

Transaction Management

Understanding transactions, their states, ACID properties, and how to ensure correct concurrent execution through serializability

** This chapter covers the theoretical foundation for reliable database transaction processing*

Transaction Concept & Transaction State

- **Definition**: A transaction is a logical unit of DB processing
- **ACID Properties**: Atomicity, Consistency, Isolation, Durability
- **Operations**: read(X), write(X), begin, commit, abort

ACID Properties Breakdown

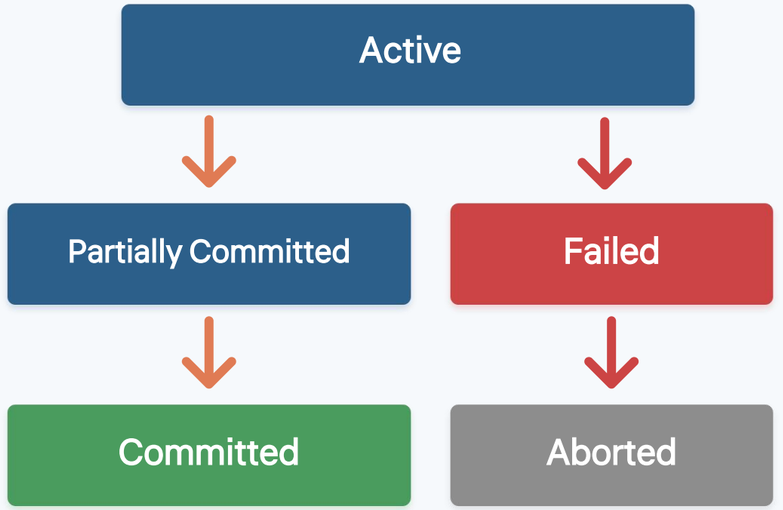
Atomicity — All operations complete, or none

Consistency — Preserves database integrity

Isolation — Concurrent txs don't interfere

Durability — Committed changes persist

Transaction State Diagram



After abort: transaction may be **restarted** or **killed**

Key Insight: A transaction moves through states from Active to either Committed (success) or Aborted (failure). The DBMS must handle all intermediate states correctly to maintain data integrity.

Implementing Atomicity & Durability

Atomicity

All or Nothing — No Partial Results

- Either **all operations** complete successfully, or **none** do
- No partial results left in the database
- Achieved via **recovery mechanisms** that undo incomplete effects
- Uses **undo logs** to roll back changes on failure
- On abort: all writes by the transaction are reversed

Durability

Committed Changes Persist Forever

- Once committed, changes survive **all failures**
- Updates must reach **stable storage** before commit ack
- Uses **redo logs** to reapply changes after crash
- **Shadow paging**: switch pointers on commit
- Checkpoints reduce recovery time

Concurrent Executions & Serializability

- **Schedule**: sequence of operations from multiple txs
- **Serial Schedule**: txs execute one after another
- **Serializable**: concurrent execution = some serial schedule

Two Types of Serializability

Conflict Serializability: same conflicting op order as a serial schedule. Tested via **precedence graph**.

View Serializability: broader condition based on read-from and final-write relationships.

Precedence Graph Test

Create a node for each transaction. Add edge Ti → Tj if Ti has a conflicting operation before Tj.

No cycle in graph → **Conflict-Serializable**

Cycle detected → **Not Serializable**

Key Point: Conflict serializability is stricter than view serializability. Every conflict-serializable schedule is also view-serializable, but not vice versa.

Recoverability & Implementation of Isolation

Recoverable Schedule

If Tj reads data written by Ti, then Ti must commit **before** Tj commits

Cascading Rollback

When one tx aborts, other txs that read its data must also abort — a chain reaction

Cascadeless Schedule

Txs only read data from **already-committed** txs — prevents cascading rollbacks

Strict Schedule

Transactions neither read nor write data written by **uncommitted** transactions. Strictest and most desirable — simplifies recovery significantly.

Implementing Isolation

Isolation is implemented using concurrency control protocols: Lock-based, Timestamp-based, and Validation-based. Testing for serializability: construct precedence graph and check for cycles.

PART 02

Lock-based, Timestamp-based & Validation-based Protocols

Concurrency Control

Protocols that manage concurrent transaction execution to ensure correctness while maximizing performance

** This chapter covers the three major families of concurrency control mechanisms*

Lock-Based Protocols

	S Lock	X Lock
S Lock	Compatible	Conflict
X Lock	Conflict	Conflict

Lock Compatibility Matrix

Two-Phase Locking (2PL)

Growing Phase: Acquire all needed locks

Shrinking Phase: Release all locks

- Guarantees **conflict serializability**
- **Strict 2PL**: release X-locks only after commit
- **Rigorous 2PL**: release all locks after commit

Lock Modes

Shared (S) Lock
For reading data only

Exclusive (X) Lock
For reading and writing

Deadlock Handling

Wait-Die (older waits for younger, or dies)

Wound-Wait (older wounds younger, or waits)

Both are timestamp-based deadlock prevention schemes that avoid cyclic waits.

Timestamp-Based Protocols

Basic Concept

- Each transaction gets a **unique timestamp** on entry
- TS(Ti) < TS(Tj) → Ti appears before Tj
- For each data item Q, maintain:

W-timestamp(Q) and **R-timestamp(Q)**

Thomas' Write Rule

If a transaction tries to write a value that is already **outdated**, the write can be **safely ignored** rather than causing a rollback.

This reduces unnecessary aborts compared to basic timestamp ordering.

Timestamp Ordering Rules

Read Operation (read(Q)):

If TS(Ti) < W-timestamp(Q): reject read (Ti too old), rollback Ti

Else: allow read, update R-timestamp(Q) = max(R-ts, TS(Ti))

Write Operation (write(Q)):

If TS(Ti) < R-timestamp(Q): reject write, rollback Ti

If TS(Ti) < W-timestamp(Q): ignore write (outdated)

Else: allow write, update W-timestamp(Q) = TS(Ti)

Advantage: No deadlocks possible — no locking!

Validation-Based Protocols

Also known as Optimistic Concurrency Control

1 Read Phase

Execute all reads and writes using **local copies** of data items. No actual database modifications yet.

2 Validation Phase

Check for **conflicts** with other transactions. Validate serializability order matches timestamp order.

3 Write Phase

If validation passes, **commit changes** to the database. If not, rollback and restart.

Validation Condition

For transaction Tj with TS(Tj), validate against all Ti where TS(Ti) < TS(Tj):

- Ti must finish writing **before** Tj starts its validation, OR
- If Ti writes items Tj reads, Ti must complete **before** Tj's validation starts

Advantages

No locking overhead, no deadlocks, excellent for low-conflict environments with mostly read operations

Disadvantages

Higher rollback rate in high-conflict scenarios, validation bottleneck, wasted work on rollbacks

PART 03

Log-based Recovery, Shadow Paging & Buffer Management

Recovery System

Mechanisms to ensure atomicity and durability despite system crashes, transaction failures, and media failures

** This chapter covers how DBMS recovers from failures while maintaining data consistency*

Recovery & Atomicity, Log-Based Recovery

Recovery Manager

Ensures atomicity and durability despite **transaction**, **system**, and **media** failures.

Log Records

Record Type	Description
<Ti, start>	Transaction Ti has started
<Ti, X, old, new>	Ti updated item X
<Ti, commit>	Ti has committed
<Ti, abort>	Ti has aborted

Deferred Modification

- Changes applied **only after** commit
- Log contains redo records only
- No undo needed — simpler recovery

Immediate Modification

- Changes applied **before** commit
- Log needs both undo and redo
- More flexible, most DBMS use this

Checkpoints

Periodic snapshots to reduce recovery time. At a checkpoint, the system:

● Suspends all transactions temporarily ● Writes all modified buffer pages to disk ● Writes a <checkpoint> record to the log

All committed transactions before the checkpoint are guaranteed durable — no recovery needed for them.

Shadow Paging & Recovery with Concurrent Txs

Shadow Paging

- Maintain a **page table** (shadow) pointing to current pages
- On commit: **swap** current and shadow page tables
- No undo needed — old pages stay intact
- **Disadvantages**: page table bottleneck, hard to extend to concurrent txs, garbage collection needed

Recovery with Concurrent Txs

- Log records include **transaction IDs**
- Undo and redo must handle **interleaved** txs
- Checkpoints record all **active** transactions
- Only txs **active at last checkpoint** need recovery

ARIES Recovery Algorithm

1. Analysis → 2. Redo → 3. Undo

Analysis: Determine active transactions at crash. **Redo**: Reapply all updates (committed and uncommitted). **Undo**: Roll back transactions that were active at crash.

Buffer Management

The **buffer manager** handles the transfer of pages between disk and memory, using policies that affect recovery requirements.

Steal vs No-Steal

Steal: Uncommitted data may be written to disk

→ Requires **undo** capability

No-Steal: Uncommitted data never written

→ No undo needed, more memory

Force vs No-Force

Force: All modified pages written at commit

→ No **redo** needed, slower commits

No-Force: Pages may stay in buffer

→ Requires **redo** capability

Most DBMS: Steal + No-Force

This combination offers the **most flexibility** and best I/O performance:

- **Steal** allows replacing dirty pages when buffer is full (better memory utilization)
- **No-Force** avoids synchronous disk writes at commit (faster commits)
- Recovery handles both undo and redo via **write-ahead logging (WAL)**

Thank You

Transaction Mgmt · Concurrency Control · Recovery System