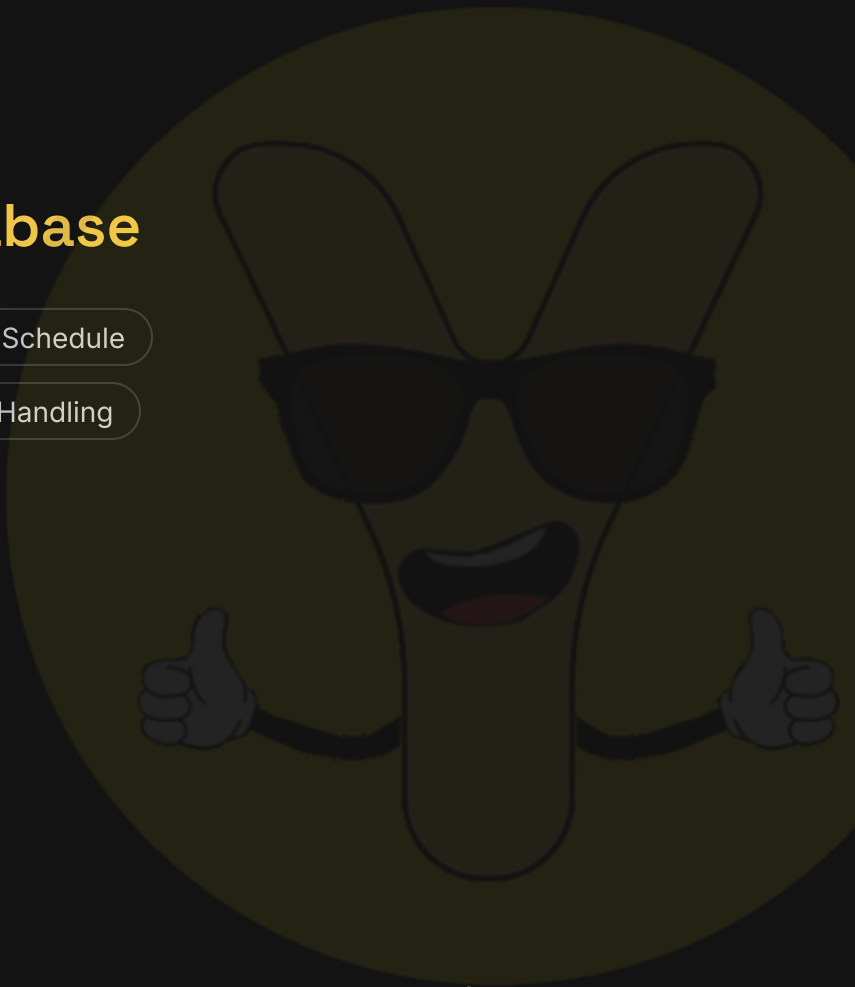
AKTU · 3RD YEAR · SEMESTER 5 · BCS501

Database Management System

Unit 4 — Transaction Processing and Distributed Database

- Transaction System
- Testing of Serializability
- Serializability of Schedules
- Conflict & View Serializable Schedule
- Recoverability
- Recovery from Transaction Failures
- Log Based Recovery
- Checkpoints
- Deadlock Handling
- Distributed Data Storage
- Concurrency Control in Distributed Databases
- Directory System



DECODED BY

Chirag Jain

Founder, Ycotes · Gold Medalist in Artificial Intelligence, IIT Madras · taught 10,000+ students

theycotes.com

Transaction System and ACID Properties

TRANSACTION

A logical unit of database processing that includes one or more database operations such as insert, update, or delete.

- Atomicity: All operations succeed completely or none take effect ('All or Nothing').
- Consistency: Database transitions from one valid state to another, preserving integrity constraints.
- Isolation: Concurrent execution of transactions yields results as if they were executed serially.
- Durability: Committed changes persist permanently in stable storage, surviving system crashes.

Transaction States and Transitions



State transition lifecycle of a database transaction from initiation to completion or failure.

Testing Serializability via Precedence Graph

Test whether Schedule S is conflict serializable: S: R1(X); W1(X); R2(X); W2(X); R1(Y); W1(Y);

1. Step 1: Identify transactions present in schedule: T1 and T2.
2. Step 2: Identify conflicting operations on same data item X: W1(X) before R2(X), and W1(X) before W2(X).
3. Step 3: Draw directed edges in Precedence Graph: Edge T1 → T2 due to write-read and write-write conflicts.
4. Step 4: Check for cycles in the constructed Precedence Graph: The graph contains no cycle.

Schedule S is Conflict Serializable because its Precedence Graph has no cycles.

Serializability of Schedules

Serializability ensures correctness in concurrent transaction execution.

- A Schedule S defines the execution order of operations across multiple transactions.
- Types of Schedules
  - Serial Schedule: Operations of one transaction complete entirely before another starts.
  - Non-Serial Schedule: Operations of concurrent transactions are interleaved.
- Serializable Schedule: A non-serial schedule that produces the same output state as a serial schedule.
- Ensures consistency without requiring sequential execution of transactions.
- Reduces waiting time and improves CPU and I/O resource utilization.

Conflict and View Serializable Schedules

Two main techniques determine whether a non-serial schedule is serializable.

- Conflict Operations
  - Operations belong to different transactions.
  - Operations access the same data item.
  - At least one operation is a Write operation.
- Conflict Serializable: Schedule can be transformed into a serial schedule by swapping non-conflicting operations.
- View Serializable: Schedule is view equivalent to a serial schedule by preserving read and write relationships.
- If a schedule is Conflict Serializable, it is guaranteed to be View Serializable.
- Schedules with 'blind writes' (write without prior read) can be View Serializable without being Conflict Serializable.

Comparison: Conflict vs View Serializability

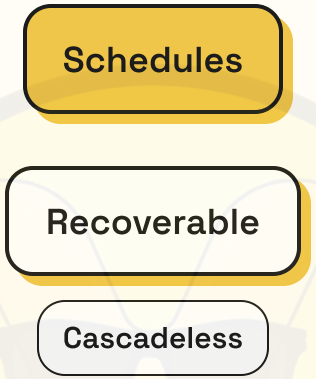
Parameter	Conflict Serializability	View Serializability
Conflict Checking	Based on swapping non-conflicting operations	Based on initial read, updated read, final write
Computational Complexity	Poly-time complexity $O(V + E)$	NP-Complete problem
Applicability	Checked using Precedence Graph	Checked using View Equivalence rules
Blind Writes	Does not allow blind write flexibility	Handles blind writes effectively
Strictness	More restrictive subset	Less restrictive, broader subset

Recoverability of Schedules

Ensures database consistency when transactions abort during concurrent execution.

- Unrecoverable Schedule: Reads uncommitted data from a transaction that later aborts, leaving dirty data.
- Recoverable Schedule: Transaction T_j commits only after transaction T_i commits, where T_j reads data written by T_i.
- Cascadeless Schedule: Prevents cascading rollbacks by allowing reads only on committed data items.
- Strict Schedule: A transaction cannot read or write an item until the previous transaction modifying it commits or aborts.
- Strict schedules simplify system recovery and rollback operations.

Hierarchy of Recoverable Schedules



Strict schedules are a subset of Cascadeless, which are a subset of Recoverable schedules.

Recovery from Transaction Failures

Restores database to a consistent state following operational or system failures.

- Types of Failures
  - Transaction Failures: Logical errors or system aborts.
  - System Crashes: Power failures or hardware faults causing RAM loss.
  - Disk Failures: Hard drive corruption or head crash.
- Write-Ahead Logging (WAL): Log record must be written to disk before corresponding data block is written.
- Undo Operation: Reverts changes made by uncommitted or failed transactions.
- Redo Operation: Re-applies changes made by committed transactions not yet persisted to disk.

Log-Based Recovery: Deferred vs Immediate Update

Feature	Deferred Modification	Immediate Modification
Database Write	Occurs after transaction commits	Occurs dynamically during execution
Log Record	Created before physical write	Created before physical write
Rollback Overhead	No UNDO required, only REDO	Requires both UNDO and REDO
Crash Recovery	Applies REDO to committed transactions	Applies UNDO to uncommitted, REDO to committed
I/O Efficiency	Lower concurrent I/O overhead	Higher disk write overhead

Checkpoints in Log-Based Recovery

CHECKPOINT

A snapshot point saved on disk where all dirty database log buffers and modified data blocks are flushed to stable storage.

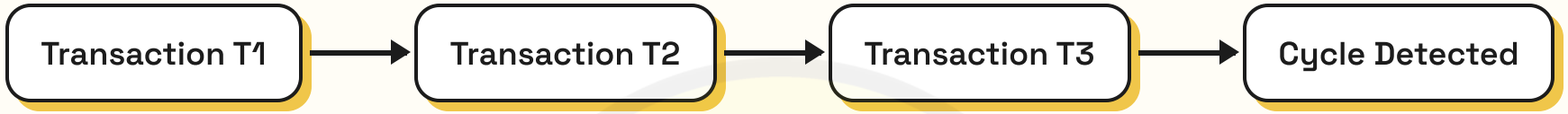
- Eliminates the need to scan the entire log file from the beginning during recovery.
- During recovery, the system scans backward in log up to the latest checkpoint.
- Transactions committed before checkpoint are ignored as their data is safely on disk.
- Transactions active during or started after checkpoint require UNDO or REDO actions.

Deadlock Handling Strategies

Deadlock occurs when two or more transactions wait indefinitely for locks held by each other.

- Deadlock Prevention
  - Wait-Die Scheme: Non-preemptive, older transaction waits, younger transaction dies.
  - Wound-Wait Scheme: Preemptive, older transaction wounds younger, younger waits.
- Deadlock Detection: System periodically constructs and checks a Wait-For Graph (WFG) for cycles.
- Victim Selection: Choosing minimum cost transaction to abort upon detecting a deadlock cycle.
- Rollback Strategy: Total rollback (abort transaction) or partial rollback to safe savepoint.

Deadlock Detection via Wait-For Graph



Wait-For Graph showing circular dependency causing a deadlock situation.

Distributed Data Storage

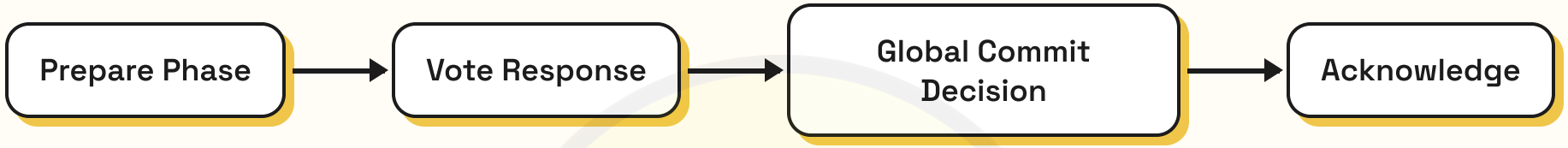
Distributes database relation fragments across multiple interconnected physical nodes.

- Horizontal Fragmentation: Splits relation rows using selection operation σ.
- Vertical Fragmentation: Splits relation columns using projection operation π with primary key.
- Mixed Fragmentation: Combines horizontal and vertical partitioning strategies.
- Replication: Stores identical copies of data fragments across multiple distributed network nodes.
- Full Replication: Complete database copied to every site, maximizing read performance.

Comparison of Data Fragmentation Types

Criterion	Horizontal Fragmentation	Vertical Fragmentation
Partition Basis	Rows (Tuples)	Columns (Attributes)
Relational Operator	Selection σ	Projection π
Reconstruction	Union operation $\cup$	Natural Join operation $\bowtie$
Primary Key	Retained in each fragment	Must be replicated in all fragments
Use Case	Region-based access control	Feature/Department specific separation

Two-Phase Commit (2PC) Protocol



Execution flow of Two-Phase Commit protocol for distributed transaction atomicity.

Directory Systems in Distributed DBMS

Maintains metadata and location information for all distributed database objects.

- Global Directory: Stores mapping between logical relation names and physical fragment locations.
- Location Transparency: Users execute queries without knowing physical site locations of data.
- Centralized Directory: Single site holds entire catalog; simple but single point of failure.
- Distributed Directory: Directory partition or replication across multiple system sites.
- LDAP (Lightweight Directory Access Protocol): Specialized protocol used for fast read-heavy directory queries.

Exam Corner: AKTU Important Questions

- **Q1** Define ACID properties. Explain transaction state transitions with a neat diagram.
- **Q2** What is serializability? Test if schedule S: R1(X), W1(X), R2(X), W2(X) is conflict serializable using a precedence graph.
- **Q3** Differentiate between Conflict Serializable and View Serializable schedules with an example.
- **Q4** Explain recoverability of schedules. Differentiate between Cascadeless and Strict schedules.
- **Q5** Compare Deferred Database Modification and Immediate Database Modification techniques in log-based recovery.
- **Q6** Describe the significance of Checkpoints in DBMS recovery mechanism.
- **Q7** Explain Wait-Die and Wound-Wait deadlock prevention algorithms with suitable scenarios.
- **Q8** Describe Horizontal and Vertical fragmentation in Distributed Databases with example schemas.
- **Q9** Explain the working of Two-Phase Commit (2PC) protocol in distributed concurrency control.

Unit 4 done. You are exam-ready.

Revise the Exam corner once more, then open Unit 5 on theycotes.com. Every AKTU subject, unit by unit, free. Notes by Chirag Jain, Ycotes.

