Lesson: Deadlock in DBMS

Ensuring Performance and Reliability in Multi-User Database Environments

Jiby John

May 27, 2025

Objective

Upon completion of this presentation, participants will be able to:

- Define deadlock in the context of Database Management Systems (DBMS).
- Identify the four necessary characteristics for a deadlock to occur.
- Explain the causes and potential impacts of deadlocks on DBMS performance and reliability.
- Differentiate between deadlock avoidance, detection, and prevention techniques.
- Describe common strategies and schemes (e.g., Wait-for-graph, Wait-Die, Wound-Wait) used to manage deadlocks in a database environment.

Prerequisite Knowledge

Understanding Transactions

What are Locks?

Concurrency Control

Understanding Transactions:

- A logical unit of work performed on a database.
- **ACID Properties (Briefly):** Atomicity, Consistency, Isolation, Durability.
- Importance of Isolation for concurrent operations.

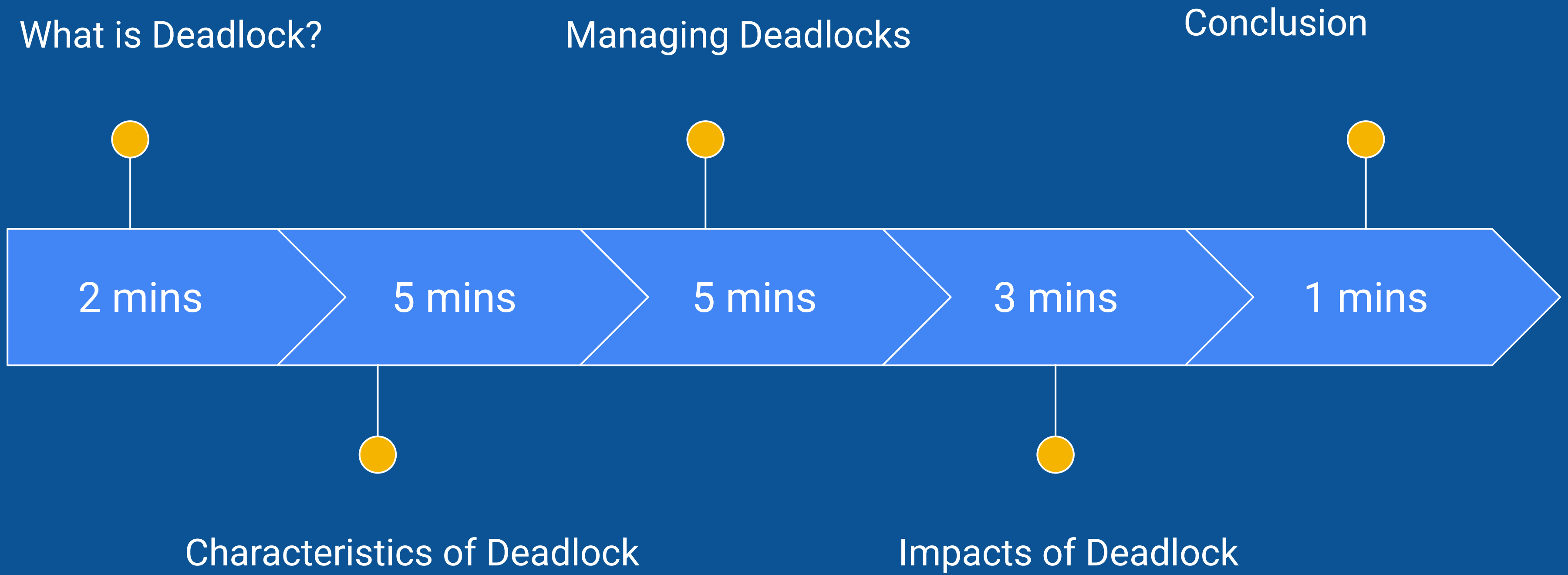
What are Locks?

- Mechanisms to control concurrent access to data.
- **Shared Locks (Read Locks):** Allow multiple transactions to read concurrently.
- **Exclusive Locks (Write Locks):** Only one transaction can hold an exclusive lock, preventing others from reading or writing.

Concurrency Control:

- Techniques used to manage simultaneous access to data in a multi-user environment.
- Ensures data integrity and consistency.

Procedure


What is Deadlock?
Managing Deadlocks
Conclusion
2 mins
5 mins
5 mins
3 mins
1 mins
Characteristics of Deadlock
Impacts of Deadlock

What is Deadlock?

- A deadlock occurs when two or more transactions are unable to proceed because each transaction is waiting for the other to release locks on resources.
- This situation creates a cycle of dependencies where no transaction can continue, leading to a standstill in the system.

Characteristics of Deadlock

The Four Necessary Conditions: (All must be present for a deadlock to occur)

1. Mutual Exclusion
2. Hold and Wait
3. No Preemption
4. Circular Wait

Characteristics of Deadlock

The Four Necessary Conditions: (All must be present for a deadlock to occur)

1. Mutual Exclusion:
   - Only one transaction can hold a particular resource (e.g., a lock on a row) at a time.
   - Example: If T1 has a lock on Item A, T2 cannot get it until T1 releases.

Characteristics of Deadlock

The Four Necessary Conditions: (All must be present for a deadlock to occur)

2. Hold and Wait:
   - A transaction holding one or more resources may request additional resources that are currently held by other transactions.
   - Example: T1 holds A, requests B (held by T2).

Characteristics of Deadlock

The Four Necessary Conditions: (All must be present for a deadlock to occur)

3. No Preemption:
    - Resources cannot be forcibly taken away from the transaction holding them. The transaction must voluntarily release them.
    - Example: You can't just snatch a lock from T1.

Characteristics of Deadlock

The Four Necessary Conditions: (All must be present for a deadlock to occur)

4. **Circular Wait**:
   - A cycle of transactions exists where each transaction is waiting for a resource held by the next transaction in the cycle.
   - Example: T1 waits for T2, T2 waits for T3, and T3 waits for T1. This is the heart of the problem!

Deadlock Example

Scenario:

- Transaction T1 holds a lock on rows in Students table.
- Transaction T2 holds a lock on rows in Grades table (which T1 needs).
- T1 needs to update Grades (held by T2).
- T2 needs to update Students (held by T1).

The Problem:

- T1 waits for T2 to release Grades.
- T2 waits for T1 to release Students.

Managing Deadlocks

Three Main Approaches:

1. **Deadlock Avoidance:** Prevent deadlocks from ever occurring.
2. **Deadlock Detection:** Identify deadlocks once they have occurred.
3. **Deadlock Prevention:** Design the system to make deadlocks impossible.

Deadlock Avoidance

Application-Consistent Logic:

- Always acquire locks on resources (e.g., tables) in the *same predefined order* across all transactions.
- *Example (T1/T2):* If both T1 and T2 always try to lock Students then Grades, T1 would get Students, then wait for T2 to finish Grades (or vice-versa), preventing the circular wait.

Deadlock Detection

Wait-for-Graph

- **How it works:**
  - Nodes represent transactions.
  - An edge from Transaction A to Transaction B means A is waiting for a resource held by B.
- **Deadlock Condition:** If the graph created has a closed loop or a cycle, then a deadlock exists.
  - T1->T2, T2->T1

Deadlock Prevention Schemes

Wait-Die Scheme (Non-Preemptive):

- If an **older** transaction (T1) requests a resource held by a **younger** transaction (T2), T1 **waits** for T2 to release it.
- If a **younger** transaction (T2) requests a resource held by an **older** transaction (T1), T2 is **killed** (aborted) and restarted later with the same timestamp.

Deadlock Prevention Schemes

Wound-Wait Scheme (Preemptive):

- If an **older** transaction (T1) requests a resource held by a **younger** transaction (T2), T1 **wounds** (forces T2 to abort) T2 and takes the resource. T2 is restarted later.

- If a **younger** transaction (T2) requests a resource held by an **older** transaction (T1), T2 **waits** for T1 to release it.

Impacts & Disadvantages of Deadlock

Performance & Reliability Degradation:

- **Delayed Transactions:** Slower response times, longer wait times.
- **Lost/Aborted Transactions:** Data inconsistencies, need for rollbacks.
- **Reduced Concurrency:** Fewer transactions can run simultaneously, leading to slower processing and reduced throughput.
- **Increased Resource Usage:** Blocked transactions still consume system resources.
- **System Downtime:** In severe cases, can halt the entire system.

Impacts & Disadvantages of Deadlock

User & Operational Impact:

- **Reduced User Satisfaction:** Perception of poor system performance.
- **Complex Resolution:** Requires manual intervention from administrators, increasing overhead.

Conclusion

Recap: Deadlock is like a traffic jam in a database where transactions are stopped because they are waiting for each other to move.

Key Takeaway: To maintain a smooth-running database, we need:

- Careful transaction design.
- Smart strategies for detection.
- Effective plans for resolution.

Goal: Keep your database running smoothly and avoid traffic jams!