Fakultas Informatika
School of Computing
Telkom University

Database System

09 | Concurrency Control

Tahun Ajar Ganjil 2024/2025

Oleh:
Tim Dosen

Goals of the Meeting

01

Students understand how the working of protocol locking

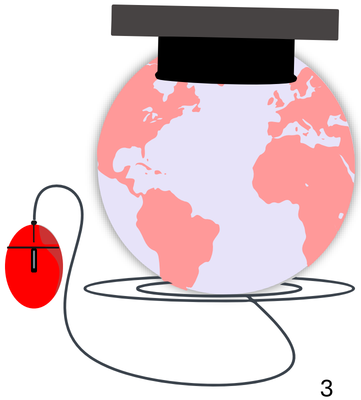
02

Students understand how to handling deadlock problem

Fakultas Informatika
School of Computing
Telkom University

OUTLINES

- Lock-Based Protocols
- Deadlock Handling



LOCK-BASED PROTOCOL

Fakultas Informatika
School of Computing
Telkom University

INTRODUCTION

- The fundamental properties of a transaction is **isolation**.
- When several transactions execute concurrently in the database, however, the isolation property may no longer be preserved.
- To ensure that it is, the system must control the interaction among the concurrent transactions; this control is achieved through one of a variety of mechanisms called **concurrency-control** schemes.
- There are a variety of concurrency-control schemes. No one scheme is clearly the best; each one has advantages.
- In practice, the most frequently used schemes are **two-phase locking** and **snapshot isolation**.
- In this course we will learn the **lock-based protocols**.

LOCK-BASED PROTOCOLS

- One way to ensure isolation is to require that data items be accessed in a mutually exclusive manner;
- That is, while one transaction is accessing a data item, no other transaction can modify that data item.
- The most common method used to implement this requirement is to allow a transaction to access a data item only if it is currently holding a **lock** on that item.

LOCK-BASED PROTOCOLS (CONT.)

- A **locking protocol** is a set of rules followed by all transactions while requesting and releasing locks.
- Locking protocols enforce serializability by restricting the set of possible schedules.

LOCK-BASED PROTOCOLS (CONT.)

- Data items can be locked in two modes :

  1. **exclusive** *(X) mode*. Data item can be both read as well as written. X-lock is requested using **lock-X** instruction.
  2. **shared** *(S) mode*. Data item can only be read. S-lock is requested using **lock-S** instruction.

- Lock requests are made to concurrency-control manager.
- Transaction can proceed only after request is granted.

LOCK-BASED PROTOCOLS (CONT.)

- **Lock-compatibility matrix**

	S	X
S	true	false
X	false	false

- A transaction may be granted a lock on an item if the requested lock is compatible with locks already held on the item by other transactions
- Any number of transactions can hold shared locks on an item,
- But if any transaction holds an exclusive on the item no other transaction may hold any lock on the item.

LOCK-BASED PROTOCOLS (CONT.)

- Example of a transaction performing locking:

  T_2: **lock-S**(*A*);

  read (*A*);

  unlock(*A*);

  lock-S(*B*);

  read (*B*);

  unlock(*B*);

  display(*A+B*)

- Locking as above is *not sufficient* to guarantee serializability

SCHEDULE WITH LOCK GRANTS

- This schedule is not serializable (why?)

Note:

- Grants omitted in rest of chapter. Assume grant happens just before the next instruction following lock request

T_1	T_2	concurrency-control manager
lock-X(B)		
		grant-X(B, T_1)
read(B) $B := B - 50$ write(B) unlock(B)		
	lock-S(A)	
		grant-S(A, T_2)
	read(A) unlock(A) lock-S(B)	
		grant-S(B, T_2)
	read(B) unlock(B) display($A + B$)	
lock-X(A)		
		grant-X(A, T_1)
read(A) $A := A + 50$ write(A) unlock(A)		

SCHEDULE WITH LOCK GRANTS (CONT.)

- Suppose that the values of accounts A and B are \$100 and \$200, respectively.
- If these two transactions are executed serially, either in the order T1, T2 or the order T2, T1, then transaction T2 will display the value \$300.
- If, however, these transactions are executed concurrently, then schedule 1, is possible.
- In this case, transaction T2 displays \$250, which is incorrect.
- The reason for this mistake is that the transaction T1 unlocked data item B too early, as a result of which T2 saw an **inconsistent state**.

T_1	T_2
lock-X(B)	
read(B)	
$B := B - 50$	
write(B)	
unlock(B)	
	lock-S(A)
	read(A)
	unlock(A)
	lock-S(B)
	read(B)
	unlock(B)
	display($A + B$)
lock-X(A)	
read(A)	
$A := A + 50$	
write(A)	
unlock(A)	

Schedule 1

SCHEDULE WITH LOCK GRANTS (CONT.)

- Suppose now that unlocking is delayed to the end of the transaction.
  Transaction T3 corresponds to T1 with unlocking delayed.
- Transaction T4 corresponds to T2 with unlocking delayed.
- T4 will not print out an inconsistent result in any of them.

T_3: lock-X(B);
read(B);
$B := B - 50$;
write(B);
lock-X(A);
read(A);
$A := A + 50$;
write(A);
unlock(B);
unlock(A).

T_4: lock-S(A);
read(A);
lock-S(B);
read(B);
display($A + B$);
unlock(A);
unlock(B).

DEADLOCK

- Consider the partial schedule

T_3	T_4
lock-X(B)	
read(B)	
$B := B - 50$	
write(B)	
	lock-S(A)
	read(A)
	lock-S(B)
lock-X(A)	

- Neither T_3 nor T_4 can make progress — executing **lock-S**(B) causes T_4 to wait for T_3 to release its lock on B, while executing **lock-X**(A) causes T_3 to wait for T_4 to release its lock on A.
- Such a situation is called a **deadlock**.
  - To handle a deadlock one of T_3 or T_4 must be rolled back and its locks released.

DEADLOCK (CONT.)

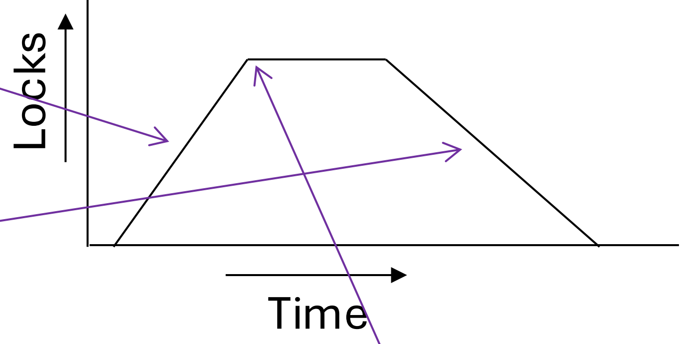
- The potential for deadlock exists in most locking protocols.
- Deadlocks are a necessary evil associated with locking, if we want to avoid inconsistent states.
- Deadlocks are definitely preferable to inconsistent states, since they can be handled by rolling back transactions, whereas inconsistent states may lead to real-world problems that cannot be handled by the database system.

STARVATION

- **Starvation** is also possible if concurrency control manager is badly designed.
- For example:
  - A transaction may be waiting for an X-lock on an item, while a sequence of other transactions request and are granted an S-lock on the same item.
  - The same transaction is repeatedly rolled back due to deadlocks.
- Concurrency control manager can be designed to prevent starvation.

THE TWO-PHASE LOCKING PROTOCOL

- A protocol which ensures conflict-serializable schedules.
- Phase 1: **Growing Phase**
  - Transaction may obtain locks
  - Transaction may not release locks



- Phase 2: **Shrinking Phase**
  - Transaction may release locks
  - Transaction may not obtain locks
- The protocol assures serializability. It can be proved that the transactions can be serialized in the order of their **lock points** (i.e., the point where a transaction acquired its final lock).

THE TWO-PHASE LOCKING PROTOCOL (CONT.)

- Two-phase locking *does not* ensure freedom from deadlocks
- Extensions to basic two-phase locking needed to ensure recoverability of freedom from cascading roll-back
  - **Strict two-phase locking:** a transaction must hold all its exclusive locks till it commits/aborts.
    - Ensures recoverability and avoids cascading roll-backs
  - **Rigorous two-phase locking**: a transaction must hold *all* locks till commit/abort.
    - Transactions can be serialized in the order in which they commit.
- Most databases implement rigorous two-phase locking, *but refer to it as simply two-phase locking*

REFERENCES

Silberschatz, Korth, and Sudarshan. *Database System Concepts* – 7th Edition. McGraw-Hill. 2019.

Slides adapted from Database System Concepts Slide.

Source: https://www.db-book.com/db7/slides-dir/index.html

Elmasri, Navathe, "Fundamental of Database Systems", Seventh Edition, Pearson, 2015.

Fakultas Informatika
School of Computing
Telkom University

ANY QUESTIONS?