



Dept. of Computer Science and Engineering
University of Rajshahi
www.ru.ac.bd

Dr. Shamim Ahmad



Chapter 16: Concurrency Control

- Lock-Based Protocols
- Timestamp-Based Protocols
- Validation-Based Protocols
- Multiple Granularity
- Multiversion Schemes
- Deadlock Handling
- Insert and Delete Operations
- Concurrency in Index Structures



Database System Concepts 3rd Edition

16.2

©Silberschatz, Korth and Sudarshan



Lock-Based Protocols

- A lock is a mechanism to control concurrent access to a data item
- 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.



Database System Concepts 3rd Edition

16.3

©Silberschatz, Korth and Sudarshan



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.
- If a lock cannot be granted, the requesting transaction is made to wait till all incompatible locks held by other transactions have been released. The lock is then granted.



Database System Concepts 3rd Edition

16.4

©Silberschatz, Korth and Sudarshan

Lock-Based Protocols (Cont.)

- Example of a transaction performing locking:

```
T2: 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 — if *A* and *B* get updated in-between the read of *A* and *B*, the displayed sum would be wrong.
- A **locking protocol** is a set of rules followed by all transactions while requesting and releasing locks. Locking protocols restrict the set of possible schedules.

```
T1: lock-X(B);
    read(B);
    B := B - 50;
    write(B);
    unlock(B);
    lock-X(A);
    read(A);
    A := A + 50;
    write(A);
    unlock(A).
```


Figure 16.2 Transaction *T*₁.

```
T2: lock-S(A);
    read(A);
    unlock(A);
    lock-S(B);
    read(B);
    unlock(B);
    display(A + B).
```

Figure 16.3 Transaction *T*₂.

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)	
	read(A)	grant-S(A, T ₂)
	unlock(A)	
	lock-S(B)	grant-S(B, T ₂)
	read(B)	
	unlock(B)	
	display(A + B)	
lock-X(A)		grant-X(A, T ₂)
read(A)		
A := A + 50		
write(A)		
unlock(A)		

Figure 16.4 Schedule 1.




```

T3: lock-X(B);
    read(B);
    B := B - 50;
    write(B);
    lock-X(A);
    read(A);
    A := A + 50;
    write(A);
    unlock(B);
    unlock(A).

```

Figure 16.5 Transaction T_3 .



```

T4: lock-S(A);
    read(A);
    lock-S(B);
    read(B);
    display(A + B);
    unlock(A);
    unlock(B).

```

Figure 16.6 Transaction T_4 .



Pitfalls of Lock-Based Protocols

- 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.