# Concurrency Control 1

**Instructor: Beom Heyn Kim**

beomheynkim@hanyang.ac.kr

Department of Computer Science

# Overview

- Lock-based Protocols
- Assignments

# 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. **shared** *(S) mode*.
     - Data item *Q* can only be read.
     - S-lock is requested using **lock-S** instruction. E.g., lock-S(*Q*)
  2. **exclusive** *(X) mode*.
     - Data item *Q* can be both read as well as written.
     - X-lock is requested using **lock-X** instruction. E.g., lock-X(*Q*)
- locks can be unlocked by calling unlock instruction. E.g., unlock(*Q*)
- A transaction makes lock requests to concurrency-control manager.
  - Transaction can proceed only after request is **granted**.

# Lock-Based Protocols (Cont.)

- A concurrency-control manager decides whether to grant or not a lock depending on lock compatibility.
- For the given lock modes *A* and *B*:
  - If a transaction can be granted a lock of mode *A* on *Q* immediately, in spite of the presence of the mode *B* lock, then mode *A* is **compatible** with mode *B*. (*A* and *B* can be either *S* or *X*)
- Compatibility relation between lock modes can be represented as a lock-compatibility matrix.
  - E.g. compatibility matrix between S-lock and X-lock

| | S | X |
|---|---|---|
| S | true | false |
| X | false | false |

Shared mode is compatible with shared mode, but not with exclusive mode

- A S-lock can be granted even if other transactions hold S-locks
- A S-lock cannot be granted if another transaction holds a X-lock
- A X-lock cannot be granted if another transaction holds a lock

# Schedule With Lock Grants

Consider the following transactions T1 and T2:

$T_1$: lock-X($B$);
read($B$);
$B := B - 50$;
write($B$);
unlock($B$);
lock-X($A$);
read($A$);
$A := A + 50$;
write($A$);
unlock($A$).

$T_2$: lock-S($A$);
read($A$);
unlock($A$);
lock-S($B$);
read($B$);
unlock($B$);
display($A + B$).

# Schedule With Lock Grants (Cont.)

Consider the following schedule:

| $T_1$ | $T_2$ | 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($A$) | | |

- In this schedule, $T_2$ sees the inconsistent state
  - Reading state where $50 has been subtracted from A but not added to B; then displaying the sum.
- The reason of having this schedule is because unlock(*B*) for X-lock on *B* was done too early.
  - Thus, $T_2$ could read inconsistent state of *B*
- Unlock instructions may be delayed to the end of each transaction

# Schedule With Lock Grants (Cont.)

Consider the following transactions T3 and T4:

$T_3$: 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($B$).

Problem in the previous slide is not possible.
However, there is another problem.

# Schedule With Lock Grants (Cont.)

Consider the following transactions T3 and T4:

$T_3$: 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($B$).

1

Like before, T4 gets executed after write($B$) instruction in T3

2

This instruction blocks and waits for T3 to release the *X*-lock on *B*

3

This instruction blocks and waits for T4 to release the *S*-lock on *A*

Is it possible to make any further progress?

# Deadlock

- Consider the partial schedule (more focused view on the example schedule in the previous slide which leads to a deadlock)

| $T_3$ | $T_4$ |
|---|---|
| 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. (More on deadlock handling in Ch18.2)

# Locking Protocol

- A **locking protocol** is a set of rules followed by all transactions while requesting and releasing locks.
  - It indicates when a transaction may lock and unlock each data item.
- Locking protocols restricts possible schedules
  - All possible schedules enforced by a locking protocol are serializable schedules
    - We shall see some locking protocols allowing only conflict-serializable schedules.
- The potential for deadlock exists in most locking protocols.
  - Deadlocks are necessary devil associated with locking.
    - Deadlocks are not desirable but still better than inconsistent states.

# Starvation

- **Starvation** is also possible if concurrency control manager is badly designed. For example:
  - A transaction T1 attempts to hold a X-lock on an item Q.
  - A transaction T2 already holds a S-lock on an item Q.
  - Before T2 releases the S-lock, T3 requests for a S-lock on Q which is immediately granted; T1 should wait not only T2 releases the S-lock but T3 releases the S-lock
  - There may be a long sequence of more transactions requesting S-lock on Q before all previous transactions release their S-locks.
  - The same transaction T1 cannot get a X-lock and starves.
- Concurrency control manager can be designed to prevent starvation.
  - Trivially, there should be no other transaction holding a lock in non-compatible mode
  - Additionally, only grant the lock when there is no other transaction that already made a request

# Overview

- Lock-based Protocols
- Assignments

# Assignments

- Reading: Ch 18.1.1, 18.1.2
- Practice Excercises: n/a

# The End