# Tutorial 10: Concurrency Control

## CS3402 Database Systems

# Question 1

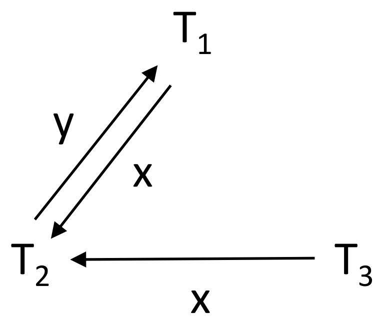
- Consider the following arrival order of operations to the scheduler. (a) If the scheduler adopts a serial execution method for concurrency control, define the serial schedule if the arrival order of operations remains the same as those shown in the table. (b) If the scheduler uses strict two-phase locking to schedule the operations, modify the above table to show the new schedule.

| **$T_1$** | **$T_2$** | **$T_3$** |
|---|---|---|
| | write(x) | |
| read(y) | | |
| | read(z) | |
| | | read(x) |
| | write(y) | |
| write(x) | | |
| | read(x) | |
| | commit | |
| | | write(z) |
| commit | | |
| | | commit |

# Question 1 (Answer)

- (a) If the serial execution method for concurrency control is used, the serial schedule is : $T_2$, $T_1$, $T_3$
- (b) If the strict two-phase locking is used to schedule the operations

| **$T_1$** | **$T_2$** | **$T_3$** |
|---|---|---|
| | write_lock(x); write(x) | |
| read_lock(y); read(y) | | |
| | read_lock(z); read(z) | |
| | | read_lock(x); → blocked |
| | write_lock(y); → blocked | |
| write_lock(x); → blocked | | |



wait-for-graph

- There is a deadlock; (a cycle in the wait-for-graph: $T_2 \rightarrow T_1 \rightarrow T_2$)

# Question 2

- Consider the following schedule at a single server system.

| **$T_1$** | **$T_2$** |
|---|---|
| read(a) | |
| | read(a) |
| write(a) | |
| | write(a) |

a) Add lock and unlock operations to the schedule if Conservative 2PL is adopted.

b) Add lock and unlock operations to the schedule if Strict 2PL is adopted.

c) Which one (S2PL or C2PL) will you choose for scheduling the two transactions?

# Question 2(a) (Answer)

a) Add lock and unlock operations to the schedule if Conservative 2PL is adopted.

| **$T_1$** | **$T_2$** |
|---|---|
| write_lock(a) | |
| read(a) | |
| write(a) | |
| unlock(a) | |
| | write_lock(a) |
| | read(a) |
| | write(a) |
| | unlock(a) |

# Question 2(b) (Answer)

b) Add lock and unlock operations to the schedule if Strict 2PL is adopted.

| **$T_1$** | **$T_2$** |
|---|---|
| read_lock(a) | |
| read(a) | |
| | read_lock(a) |
| | read(a) |
| write_lock(a) → blocked | |
| | write_lock(a) → blocked |

# Question 2(c) (Answer)

c) Which one (S2PL or C2PL) will you choose for scheduling the two transactions?

C2PL since it does not have the deadlock problem and the transactions are short.

# Question 3

- The following table shows the schedule for transactions $T_1$ and $T_2$ with $T_1$ having an “older” time-stamp than $T_2$.

| $T_1$ | $T_2$ |
|---|---|
| read(a) | |
| | read(b) |
| write(b) | |
| | write(a) |

a) Strict Two-Phase Locking is used for concurrency control. Define the wait-for-graph.

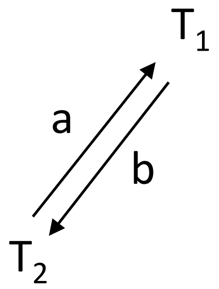
b) Show the new schedule if the wait-die method is used.

c) Show the new schedule if the wound-wait method is used.

# Question 3(a) Answer

a) Strict Two-Phase Locking is used for concurrency control. Define the wait-for-graph at each server.

| $T_1$ | $T_2$ |
|---|---|
| read_lock(a) | |
| read(a) | |
| | read_lock(b) |
| | read(b) |
| write_lock(b) → blocked | |
| | write_lock(a) → blocked |



wait-for-graph

# Question 3(b) Answer

b) Show the new schedule if the wait-die method is used.

- Wait-die: If TS($T_i$) < TS($T_j$), $T_i$ waits else $T_i$ dies
- Thus, write(a) from $T_2$ will make it to abort and release the read lock on data item b.
- Thus, the final schedule will be $T_1$ and then $T_2$.

| **$T_1$** | **$T_2$** |
|---|---|
| read_lock(a); read(a); | |
| | read_lock(b); read(b); |
| write_lock(b) → blocked | |
| | write_lock(a) → restarts because it is younger than $T_1$ and $T_2$ releases its read lock on b before it restarts |
| write(b); | |
| release_lock($T_1$); | |
| | read_lock(b); read(b); |
| | write_lock(a); write(a); |
| | release_lock($T_2$); |

# Question 3(c) Answer

c) Show the new schedule if the wound-wait method is used.

- Wound-wait: If $TS(T_i) < TS(T_j)$, $T_j$ wounds else $T_i$ waits
- When the write(b) from $T_1$ arrives, $T_2$ is aborted.
- Thus, the final schedule will also be $T_1$ and then $T_2$.

| **$T_1$** | **$T_2$** |
|---|---|
| read_lock(a); read(a); | |
| | read_lock(b); read(b); |
| write_lock(b); write(b);<br>($T_2$ is restarted by $T_1$ because $T_2$ is younger than $T_1$. The write lock on b is granted to $T_1$ after $T_2$ has released its read lock on b) | |
| release_lock($T_1$); | |
| | read_lock(b); read(b); |
| | write_lock(a); write(a); |
| | release_lock($T_2$); |