IF3140 – Sistem Basis Data

Concurrency Control:

- Validation-based Protocol

KNOWLEDGE & SOFTWARE ENGINEERING

Optimistic Concurrency Control

Validation-Based Protocol

- Idea: can we use commit time as serialization order?
- To do so:
  - Postpone writes to end of transaction
  - Keep track of data items read/written by transaction
  - **Validation** performed at commit time, detect any out-of-serialization order reads/writes
- Also called as **optimistic concurrency control** since transaction executes fully in the hope that all will go well during validation

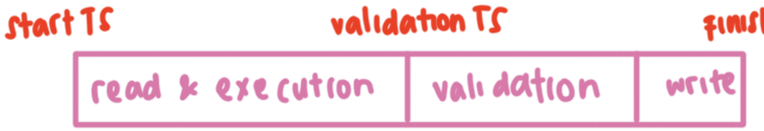
write tdk pernah dilakukan sblm mendekati commit

KNOWLEDGE & SOFTWARE ENGINEERING

Validation-Based Protocol

- Execution of transaction T_i is done in three phases.
  1. **Read and execution phase**: Transaction T_i writes only to temporary local variables → tidak ada write ke database
  2. **Validation phase**: Transaction T_i performs a "validation test" to determine if local variables can be written without violating serializability.
  3. **Write phase**: If T_i is validated, the updates are applied to the database; otherwise, T_i is rolled back.
- The three phases of concurrently executing transactions can be interleaved, but each transaction must go through the three phases in that order.
  - We assume for simplicity that the validation and write phase occur together, atomically and serially → harus validation duku baru write
    - I.e., only one transaction executes validation/write at a time.

Validation-Based Protocol (Cont.)



- Each transaction T_i has 3 timestamps
  - **StartTS**(T_i) : the time when T_i started its execution
  - **ValidationTS**(T_i): the time when T_i entered its validation phase
  - **FinishTS**(T_i) : the time when T_i finished its write phase
- Validation tests use above timestamps and read/write sets to ensure that serializability order is determined by validation time
  - Thus, $TS(T_i) = ValidationTS(T_i)$
- Validation-based protocol has been found to give greater degree of concurrency than locking/TSO if probability of conflicts is low.

KNOWLEDGE & SOFTWARE ENGINEERING

Validation Test for Transaction T_j

- If for all T_i with TS (T_i) < TS (T_j) either one of the following condition holds:
  - **finishTS**(T_i) < **startTS**(T_j)
  - **startTS**(T_j) < **finishTS**(T_i) < **validationTS**(T_j) **and** the set of data items written by T_i does not intersect with the set of data items read by T_j.

  then validation succeeds and T_j can be committed.
- Otherwise, validation fails and T_j is aborted.
- Justification:
  - First condition applies when execution is not concurrent
    - The writes of T_i do not affect reads of T_j since they occur before T_j starts its reads.
  - If the second condition holds, execution is concurrent and T_j does not read any item written by T_i.

KNOWLEDGE & SOFTWARE ENGINEERING

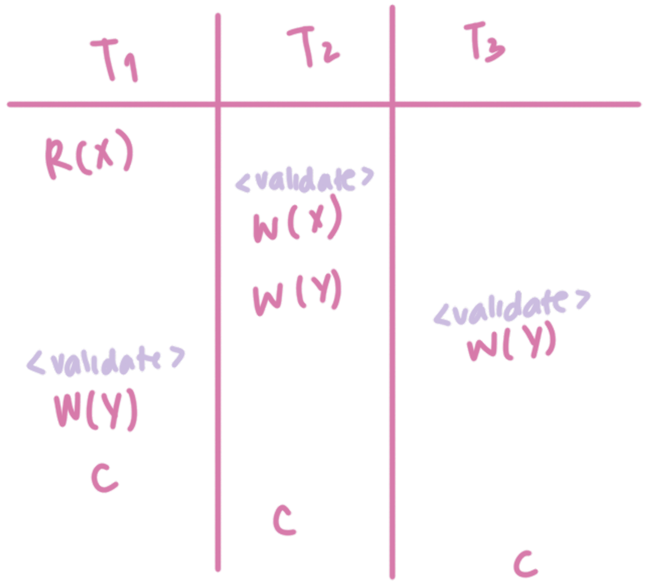
Schedule Produced by Validation

- Example of schedule produced using validation

T_{25}	T_{26}
read(B)	
	read(B)
	$B := B - 50$
	read(A)
	$A := A + 50$
read(A)	
<validate>	
display($A + B$)	
	<validate>
	write(B)
	write(A)

T_{25} bisa divalidasi krn tdk ada T lain yg selesai saat T_{25} masuk di read & execute phase

T_{26} bisa divalidasi krn di T_{25} tdk ada proses write



Latihan Soal

Periksalah apakah schedule

S: R1(X); W2(X); W2(Y); W3(Y); W1(Y); C1; C2; C3;

dapat dihasilkan dengan menggunakan ***Validation-based Protocol***. Timestamp transaksi Ti adalah i dan sebelum S dieksekusi timestamp semua item data adalah 0.
Jelaskan jawaban Anda.

- T1 memiliki timestamp 1 dan tidak ada transaksi konkuren yang lebih awal. Validasi T1 berhasil dan bisa commit.
- T2 memiliki timestamp 2 dan harus divalidasi terhadap T1. Karena T2 tidak melakukan read (terhadap item yang di-write T1) maka validasi T2 berhasil.
- T3 memiliki timestamp 3 dan harus divalidasi terhadap T1 dan T2. Karena T3 tidak melakukan read (terhadap item yang di-write T1 dan T2) maka validasi T3 berhasil.

∴S dapat dihasilkan dengan validation-based protocol

KNOWLEDGE & SOFTWARE ENGINEERING