Concurrency Control Protocol (Bag. 2)

SOAL 1

Periksalah apakah schedule *S: R1(X); W2(X); W2(Y); W3(Y); W1(Y); C1; C2; C3;* dapat dihasilkan dengan menggunakan protokol-protokol berikut ini. *Timestamp* transaksi *Ti* adalah *i* dan sebelum *S* dieksekusi *timestamp* semua item data adalah 0. **Jelaskan** jawaban Anda.

1. Timestamp ordering
2. Timestamp ordering with Thomas' Write Rule
3. Validation based

1. Timestamp Ordering (TO)
(menggunakan aturan: read allowed jika TS(T) ≥ WTS, write allowed jika TS(T) ≥ RTS & TS(T) ≥ WTS)

1) R1(X)
TS(T1)=1 ≥ WTS(X)=0 → RTS(X) = 1

2) W2(X)
TS(T2)=2 ≥ RTS(X)=1
TS(T2)=2 ≥ WTS(X)=0 → update: WTS(X)=2

3) W2(Y)
TS(T2)=2 ≥ RTS(Y)=0
TS(T2)=2 ≥ WTS(Y)=0 → WTS(Y)=2

4) W3(Y)
TS(T3)=3 ≥ RTS(Y)=0
TS(T3)=3 ≥ WTS(Y)=2 → WTS(Y)=3

5) W1(Y)
TS(T1)=1 < WTS(Y)=3 → VIOLATE

T1 harus ABORT, karena ingin menulis Y tetapi versi Y sudah ditulis oleh transaksi lebih baru (T3).

Maka, schedule S tidak dapat dihasilkan oleh Timestamp Ordering. Karena operasi W1(Y) melanggar aturan TS(T1) ≥ WTS(Y).

2. Timestamp Ordering with Thomas' Write Rule (TWR)

Aturan TWR:
Jika suatu transaksi melakukan write tetapi
TS(T) < WTS(item) → write diabaikan (di-drop) tanpa abort

Bagian awal sama seperti TO:

R1(X)
W2(X)
W2(Y)
W3(Y) → WTS(Y)=3

TS(T1)=1 < WTS(Y)=3 → write obsolete → DROPPED, tidak menyebabkan abort → T1 tetap bisa commit.

Schedule tetap berjalan:
C1
C2
C3

Maka, schedule S dapat dihasilkan oleh Timestamp Ordering dengan Thomas' Write Rule karena write T1 pada Y hanya diabaikan, bukan menyebabkan abort.

3. Validation-Based

Aturan validasi:
Sebelum commit, transaksi Ti divalidasi terhadap transaksi Tj sebelumnya:

Ti valid jika
(WriteSet(Tj) ∩ ReadSet(Ti) = ∅) atau Tj selesai sebelum Ti mulai

Cek commit order di schedule:
C1 → C2 → C3

1) Validasi T1
T1 commit paling awal → selalu valid

2) Validasi T2 terhadap T1
T2 write: {X, Y}
T1 read: {X}

Cek:
T2 writes X
T1 reads X
Namun T1 commit sebelum T2 mulai fase write

→ maka valid (tipe conflict 1 tetapi T1 < T2 di waktu commit)

3) Validasi T3 terhadap T1 & T2

Terhadap T1
T3 write Y
T1 read X
→ tidak ada konflik (valid)

Terhadap T2
T3 write Y
T2 write Y
→ konflik write-write

Aturan validasi:
Konflik write-write diperbolehkan asalkan T2 selesai sebelum T3 mulai validation phase.
Karena commit order adalah C2 → C3, maka T2 selesai sebelum T3 divalidasi (valid)

Maka, schedule S valid karena setiap transaksi lolos fase validasi tanpa ada conflict yang melanggar aturan.

SOAL 2

Diberikan urutan kedatangan instruksi transaksi T1, T2, dan T3 ke DBMS berikut. Transaksi dimulai tepat sebelum instruksi pertama pada transaksi tersebut.
Keterangan: R(Q) adalah read data Q dan W(Q) adalah write pada data Q untuk transaksi terkait.

Tuliskan langkah-langkah untuk mengeksekusi jadwal di atas sampai semua transaksi tuntas dieksekusi dengan **multiversion timestamp ordering protocol** dan jelaskan setiap tahapan eksekusi, termasuk versi data yang dihasilkan. Sebutkan dan jelaskan apa yang terjadi pada tiap transaksi: apakah abort atau commit. Jika terjadi abort, jelaskan bagaimana proses rollback dilakukan. Jika dibutuhkan, timestamp (Tx,Ty,Tz) = (1,2,3).
Asumsikan: Jika terjadi abort yang menyebabkan sebuah transaksi di-rollback, DBMS akan memprioritaskan eksekusi ulang dari instruksi transaksi tersebut hingga bagian instruksi yang menyebabkan abort.

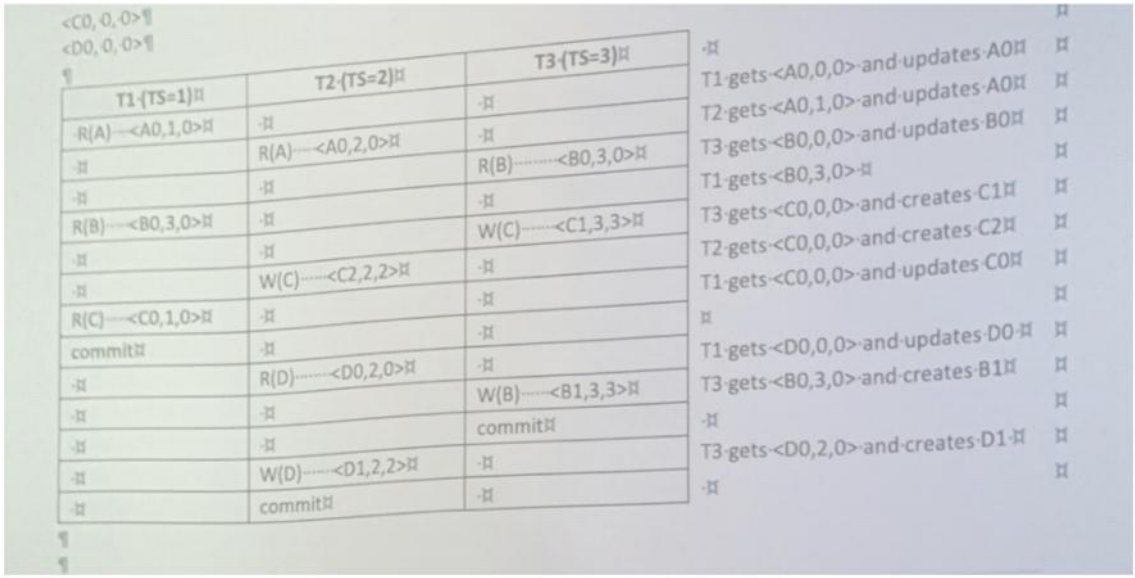
Versi data saat jadwal dimulai:
<A0, 0, 0>
<B0, 0, 0>
<C0, 0, 0>
<D0, 0, 0>

T1(TS=1)	T2(TS=2)	T3(TS=3)
R(A) <A0, 1, 0>	R(A) <A0, 2, 0>	
		R(B) <B0, 3, 0>
R(B) <B0, 3, 0>		
		W(C) <C1, 3, 3>
	W(C) <C2, 2, 2>	
R(C) <C0, 1, 0>		
commit		
	R(D) <D0, 2, 0)	
		W(B) <B1, 3, 3>
		commit
	W(D) <D1, 2, 2>	
	commit	

T1 get <A0, 0, 0> and update A0
T2 get <A0, 1, 0> and update A0
T3 get <B0, 0, 0> and update B0
T1 get <B0, 3, 0>
T3 get <C0, 0, 0> and create C1
T2 get <C0, 0, 0> and create C2
T1 get <C0, 0, 0> and update C0

T1 get <D0, 0, 0> and update D0
T3 get <B0, 3, 0> and create D1

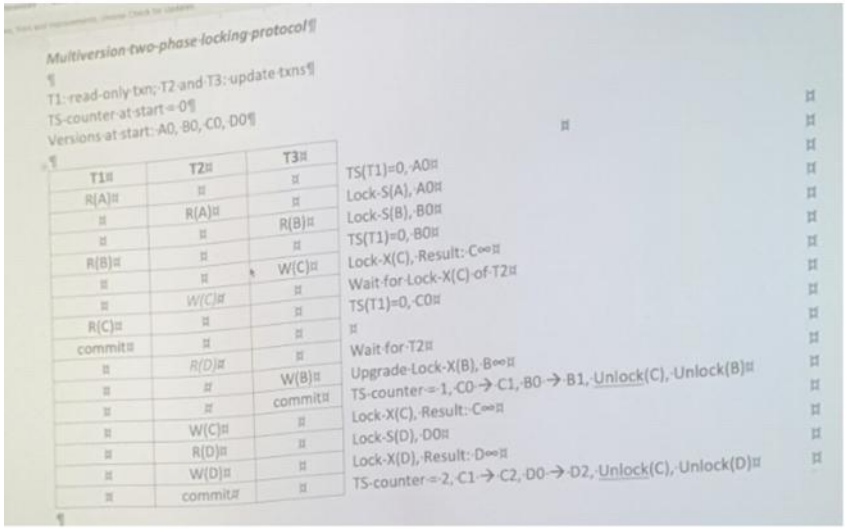
T3 get <D0, 2, 0> and create D1



Diberikan urutan kedatangan instruksi transaksi T1, T2, dan T3 ke DBMS berikut. Transaksi dimulai tepat sebelum instruksi pertama pada transaksi tersebut.
Keterangan: R(Q) adalah read data Q dan W(Q) adalah write pada data Q untuk transaksi terkait.

T1	T2	T3
R(A)		
	R(A)	
		R(B)
R(B)		
		W(C)
	W(C)	
R(C)		
commit		
	R(D)	
		W(B)
		commit
	W(D)	
	commit	

Tuliskan langkah-langkah untuk mengeksekusi jadwal di atas sampai semua transaksi tuntas dieksekusi dengan ***multiversion two-phase locking protocol*** dan jelaskan setiap tahapan eksekusi, termasuk versi data yang dihasilkan.



Diberikan urutan kedatangan instruksi transaksi T1, T2, dan T3 ke DBMS berikut. Transaksi dimulai tepat sebelum instruksi pertama pada transaksi tersebut.
Keterangan: R(Q) adalah read data Q dan W(Q) adalah write pada data Q untuk transaksi terkait.
Asumsikan nilai dari tiap data di awal A=10; B=20; C=0; D=30

T1	T2	T3
R(A)		
	R(A)	
		R(B)
R(B)		
		C:=50
		W(C)
	C:=35	
	W(C)	
R(C)		
commit		
	R(D)	
		B:=B-0.1*C
		W(B)
		commit
	D:=0.2*D+A	
	W(D)	
	commit	

Tuliskan langkah-langkah untuk mengeksekusi jadwal di atas sampai semua transaksi tuntas dieksekusi dengan **snapshot isolation (dengan first-committer wins)** dan jelaskan setiap tahapan eksekusi, termasuk versi data yang dihasilkan dan nilai dari tiap data. Sebutkan dan jelaskan apa yang terjadi pada tiap transaksi: apakah abort atau commit. Jika terjadi abort, jelaskan bagaimana proses rollback dilakukan. Sebutkan pula versi nilai data di akhir seluruh transaksi.

Snapshot Isolation (First-Committer Wins)

Content saat start:
A0 = 10
B0 = 20
C0 = 0
D0 = 30

T1	T2	T3	
R(A)			A0 = 10

	R(A)		A0 = 10
		R(B)	B0 = 20
R(B)			B0 = 20
		C := 50	
		W(C)	C3 = 50
	C := 35		
	W(C)		C2 = 35
R(C)			C0 = 0
commit			commit sukses
	R(D)		D0 = 30
		B := B - 0.1 * C	
		W(B)	B3 = 20 - 0.1 * 50 = 25
		commit	commit sukses, C3 = 50, B3 = 25
	D := 0.2 * D + A		D2 = 0.2 * 30 + 10 = 16
	W(D)		D2 = 10
	commit		commit gagal, serialization error → T2 abort dan rollback
	R(A)		A0 = 10
	C := 35		

	W(C)		C2 = 35
	R(D)		D0 = 30