Berikut ini adalah urutan kedatangan operasi-operasi dari 3 buah transaksi yang berjalan secara konkuren. Ry(A) dan Wy(A) menyatakan operasi yang dilakukan oleh transaksi Ty terhadap item data A, Cy menyatakan commit transaksi Ty.

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

Apabila protokol konkurensi yang digunakan adalah *two phase locking*, **tuliskan *schedule*** yang dihasilkan dari eksekusi urutan operasi di atas. *Schedule* harus memuat urutan pemberian *lock* (termasuk jenis *lock* yang diperoleh – SL atau XL), eksekusi operasi (R atau W atau C), dan pembebasan *lock* (UL). **Beri penjelasan** untuk jawaban Anda.

Asumsikan bahwa *lock* diminta saat pertama kali transaksi akan mengakses sebuah item data, dengan jenis *lock* yang disesuaikan berdasarkan keperluan akses item tersebut oleh transaksi (*exclusive lock* jika nilainya akan diubah oleh transaksi atau *shared lock* jika tidak akan diubah).

Secara umum *transaction manager* akan memproses operasi berdasarkan urutan kedatangan. Namun, jika sebuah transaksi sedang terblok (karena menunggu *lock*), semua operasi dari transaksi tersebut yang diterima oleh *transaction manager* akan diantrikan hingga transaksi bisa jalan kembali. *Transaction manager* akan melanjutkan eksekusi operasi berikutnya yang diterima. Pada saat sebuah transaksi yang tadinya terblok dapat berjalan kembali, semua operasi transaksi tersebut yang berada di antrian akan mendapat prioritas untuk dieksekusi sebelum *transaction manager* kembali melayani operasi berikutnya yang diterima.

Asumsi tidak automatic

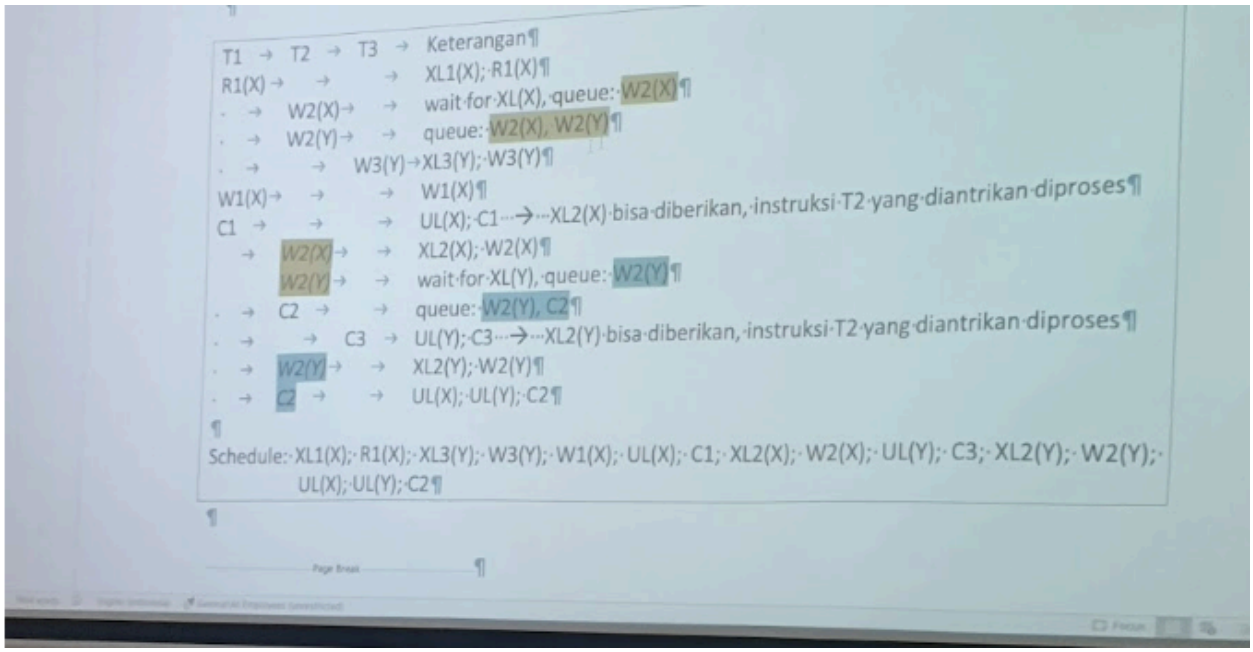
T1	T2	T3	CC Manager
R1(X)			XL1(X)
	W2(X)		waiting L(X) queue: W2(X)
	W2(Y)		queue: W2(X), W2(Y)
		W3(Y)	XL3(Y)
W1(X)			W1(X)
C1			UL1(X)
	W2(X)		XL2(X)
	W2(Y)		waiting L(Y) queue: W2(Y)
	C2		queue: W2(Y), C2
		C3	UL3(Y)
	W2(Y)		XL2(Y)
	C2		UL2(X), UL2(Y)

Schedule:

XL1(X), R1(X), XL3(Y), W3(Y), W1(X), UL1(X), C1, XL2(X), W2(X), UL3(Y), C3, XL2(Y), W2(Y), UL2(Y), C2

Penjelasan:

T1 membaca data X (R1(X)) dan memperoleh shared lock (SL) pada X. Kemudian, T1 meng-upgrade lock tersebut ke exclusive lock (XL) untuk menulis ke X (W1(X)), dan setelah selesai, T1 melakukan commit dan melepaskan lock (UL1(X)).
T2 berusaha menulis ke X (W2(X)) dan Y (W2(Y)). Untuk X, T2 harus menunggu hingga T1 melepaskan lock-nya karena konflik lock. Untuk Y, T2 langsung mendapatkan XL karena belum ada lock lain dan berhasil menulis. Setelah T1 melepaskan lock pada X, T2 melanjutkan dengan operasi penulisannya ke X. T2 menyelesaikan operasi-operasinya dan melakukan commit, melepaskan semua lock yang dipegangnya pada X dan Y (UL2(X), UL2(Y)).
T3 ingin menulis ke Y (W3(Y)), tetapi terhalang karena T2 telah memegang XL pada Y. T3 harus menunggu hingga T2 melepaskan lock pada Y. Setelah lock dilepaskan oleh T2, T3 memperoleh XL, menulis ke Y, melakukan commit, dan melepaskan lock (UL3(Y)).



T1	T2	T3	Keterangan
R1(X)			XL1(X); R1(X)
	W2(X)		wait for XL(X), queue: W2(X)
	W2(Y)		queue: W2(X), W2(Y)
		W3(Y)	XL3(Y); W3(Y)
W1(X)			W1(X)
C1			UL(X); C1 → XL2(X) bisa diberikan, instruksi T2 yang diantrikan diproses
	W2(X)		XL2(X); W2(X)
	W2(Y)		wait for XL(Y), queue: W2(Y)
	C2		queue: W2(Y), C2
		C3	UL(Y); C3 → XL2(Y) bisa diberikan, instruksi T2 yang diantrikan diproses
	W2(Y)		XL2(Y); W2(Y)
	C2		UL(X); UL(Y); C2

Schedule: XL1(X); R1(X); XL3(Y); W3(Y); W1(X); UL(X); C1; XL2(X); W2(X); UL(Y); C3; XL2(Y); W2(Y); UL(X); UL(Y); C2

Diberikan 2 (dua) buah transaksi berikut ini.

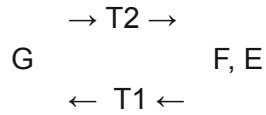
T1: SL(E); R(E); XL(F); R(F); XL(G); R(G); W(F); W(G); UL(E); UL(F); UL(G);
T2: SL(G); R(G); SL(F); R(F); XL(E); R(E); W(E); UL(G); UL(F); UL(E);

1. **Perlihatkan sebuah contoh *schedule* konkuren** dari eksekusi kedua transaksi tersebut dengan menggunakan protokol *two phase locking* yang memiliki kondisi *deadlock*. **Perlihatkan *wait-for graph*** pada saat terjadi *deadlock*.
2. **Jelaskan proses *deadlock recovery*** yang dapat dilakukan pada kondisi *deadlock* pada poin 1.
3. Dengan menggunakan *schedule* yang Anda hasilkan pada poin 1, **jelaskan 2 (dua) strategi *deadlock prevention*** yang dapat dilakukan sehingga dapat mencegah terjadinya *deadlock*.

1. Contoh Schedule Deadlock

T1: SL(E)
T1: R(E)
T2: SL(G)
T2: R(G)
T1: XL(F)
T1: R(F)
T2: SL(F) → T2 menunggu karena T1 memegang XL pada F
T1: XL(G) → T1 menunggu karena T2 memegang SL pada G
T2: XL(E) → T2 menunggu karena T1 memegang SL pada E

Wait-for-graph



2. Proses Deadlock Recovery

Deadlock recovery dapat dilakukan dengan menghentikan (abort) salah satu transaksi untuk memecah siklus. Contoh, abort T2 dan rollback semua operasi yang telah dilakukannya, lalu ulangi transaksi setelah T1 selesai. Jika T2 di-rollback maka SL2(G) dapat dilepas sehingga T1 dapat mendapat XL1(G).

3. Strategi Deadlock Prevention

a. Wait-die scheme
   Dengan asumsi T1 datang lebih awal dari T2, maka saat T2 meminta SL2(F) dan T1 masih memegang XL1(F), maka saat itu T2 harus melakukan rollback/die. Ini memastikan bahwa transaksi yang memulai lebih dulu lebih mungkin untuk menyelesaikan tanpa tertunda oleh transaksi yang dimulai kemudian, sehingga mengurangi peluang terjadinya deadlock.
b. Ordering Resources
   Menentukan urutan tetap untuk semua jenis lock yang mungkin diperlukan oleh semua transaksi. Misalnya, semua transaksi harus mengunci E, F, dan G dalam urutan tersebut. Ini memastikan tidak ada siklus tunggu yang terbentuk.

Diberikan 2 (dua) buah transaksi berikut ini.

T1: R(E); R(F); R(G); W(F); W(G);
T2: R(G); R(F); R(E); W(E);

Tuliskan instruksi *lock* dan *unlock* pada kedua transaksi tersebut sehingga mengikuti *Tree Protocol* dengan *partial ordering* terhadap item seperti pada gambar berikut. Urutan perintah *read* dan *write* diasumsikan tidak berubah.

T1: XL1(B), XL1(E), R1(E), UL1(E), XL1(F), R1(F), XL1(D), XL1(G), R1(G), W1(F), UL1(F), W1(G), UL1(G), UL1(D), UL1(B)
T2: XL2(B), XL2(D), XL2(G), R2(G), UL2(G), UL2(D), XL2(F), R2(F), Ul2(F), XL2(E), R2(E), W2(E), UL2(E), UL2(B)

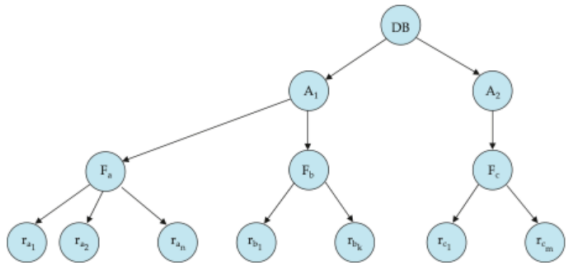
Jawaban Tjhia:

T1: XL(B); XL(E); R(E); UL(E); XL(F); R(F); XL(D); UL(B); XL(G); R(G); W(F); UL(F); W(G); C; UL(G); UL(D);
T2: XL(D); XL(G); R(G); UL(G); UL(D); XL(B); XL(F); R(F); UL(F); XL(E); R(E); W(E); C; UL(E); UL(B);

S: XL1(B); XL1(E); R1(E); UL1(E); XL1(F); R1(F); XL2(D); XL2(G); R2(G); UL2(G); UL2(D); XL1(D); UL1(B); XL1(G); R1(G); W1(F); UL1(F); W1(G); C1; UL1(G); UL1(D); XL2(B); XL2(F); R2(F); UL2(F); XL2(E); R2(E); W2(E); C2; UL2(E); UL2(B);

→ Kalau bisa unlock, segera unlock secepatnya.

Hirarki granularitas item pada sebuah basis data didefinisikan seperti pada gambar berikut.



1. Apabila T1 sedang melakukan penulisan untuk ra2 dan rb1, tuliskan daftar lock yang dimiliki oleh T1. Apakah pada saat ini T2 dapat melakukan penulisan untuk rb3? Tuliskan daftar lock yang diperoleh T2 hingga berhasil melakukan operasinya atau harus menunggu lock.
2. Apabila T1 sedang melakukan pembacaan untuk hampir semua record pada file Fb dan melakukan penulisan untuk rb3 dan rb4, , tuliskan daftar lock yang dimiliki oleh T1. Apakah pada saat ini T2 dapat melakukan pembacaan untuk rb1? Bagaimana dengan T3 yang ingin melakukan penulisan terhadap rb5? Tuliskan daftar lock yang diperoleh T2 dan T3 hingga berhasil melakukan operasinya atau harus menunggu lock.
3. Apabila T1 sedang melakukan operasi seperti pada poin b, apakah pada saat ini T2 dapat melakukan pembacaan untuk seluruh record pada Fb? Tuliskan daftar lock yang diperoleh T2 hingga berhasil melakukan operasinya atau harus menunggu lock.

1. **T1 menulis ra2 dan rb1**
   - T1: IXL1(DB), IXL1(A1), IXL1(Fa), XL1(ra2), W1(ra2), IXL1(Fb), XL1(rb1), W1(rb1)
   - T2: IXL2(DB), IXL2(A1), IXL2(Fb), XL2(rb3), W(rb3)

   T2 berhasil melakukan penulisan untuk R3 karena T2 mengunci rb3 dan tidak ada transaksi lain yang memiliki lock eksklusif pada Fb atau rb3 saat itu, T2 berhasil melakukan penulisan pada rb3. Intent locks dan exclusive locks yang dipakai oleh T1 pada record lain (seperti rb1) tidak memblokir T2 untuk mengakses rb3.

2. **T1 membaca hampir semua record pada Fb, menulis rb3 dan rb4**
   - T1: W1(rb4)
   - T2: ISL2(DB), ISL2(A1), ISL2(Fb), SL2(rb1), R(rb1)
   - T3: IXL3(DB), IXL3(A1), **IXL3(Fb)**

   T2 berhasil membaca rb1, namun T3 harus menunggu T1 terlebih dahulu untuk membuka akses Fb. Hal ini karena lock SIXL1(Fb) tidak compatible dengan IXL3(Fb)

3. **T2 ingin membaca seluruh record pada Fb**
   - T1: IXL1(DB), IXL1(A1), SIXL(Fb), XL1(rb3), W1(rb3), XL1(rb4), W1(rb4)
   - T2: ISL2(DB), ISL2(A1), **SL2(Fb)**

   T2 harus menunggu untuk membaca seluruh record Fb. Hal ini karena lock SIXL1(Fb) tidak *compatible* dengan SL2(Fb)

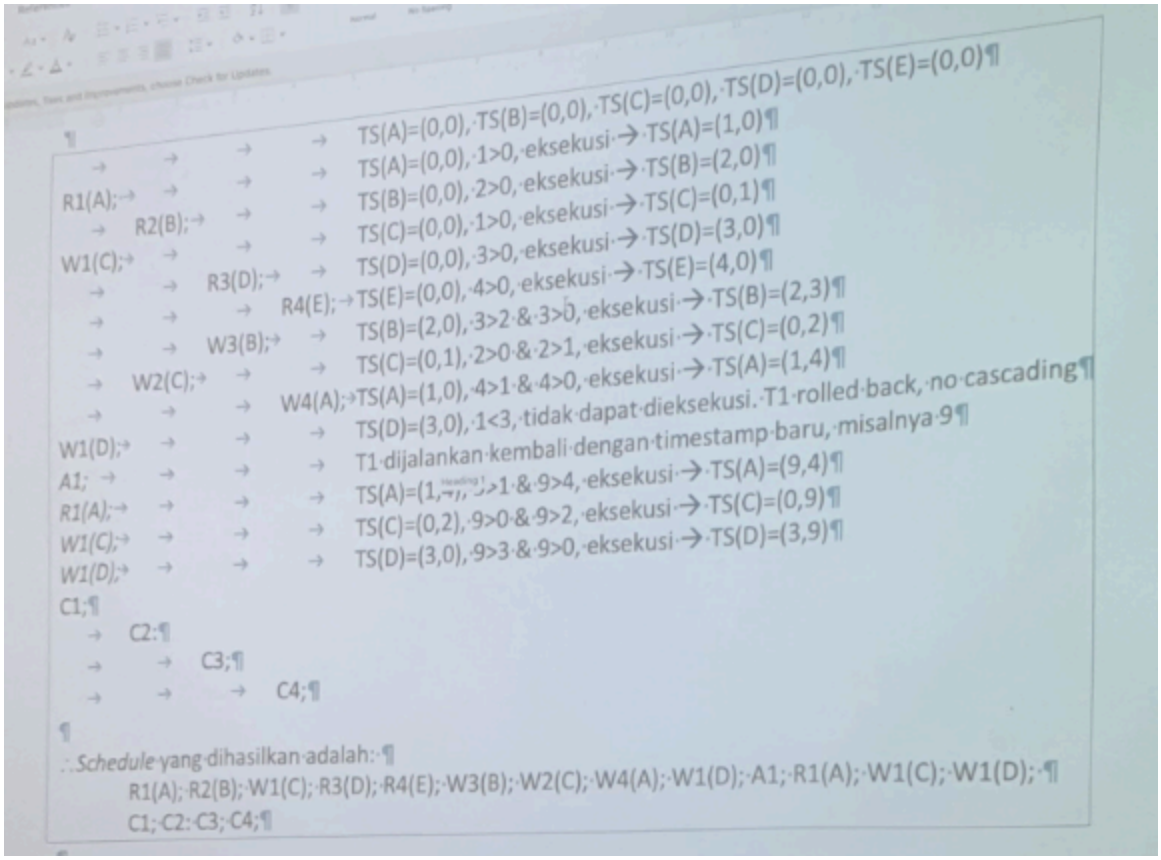
Berikut ini adalah urutan kedatangan operasi-operasi dari 4 buah transaksi yang berjalan secara konkuren ke *Transaction Manager*. *Timestamp* transaksi *Ti* adalah *i* dan sebelum *S* dieksekusi *timestamp* semua item data adalah 0.

R1(A); R2(B); W1(C); R3(D); R4(E); W3(B); W2(C); W4(A); W1(D); C1; C2: C3; C4;

Apabila protokol konkurensi yang digunakan adalah *timestamp ordering protocol*, **tuliskan *schedule* yang dihasilkan** dari eksekusi urutan operasi di atas. *Schedule* harus memuat eksekusi operasi (R atau W atau C) dan A apabila ada sebuah transaksi yang harus *rollback*. Pada saat sebuah transaksi *rollback* semua instruksi transaksi tersebut hingga saat terjadi *rollback* akan diprioritaskan untuk dieksekusi kembali. **Jelaskan** jawaban Anda.

Langsung rollback kalo misal ada rollback, jadi ga nunggu commit

T1	T2	T3	T4	Keterangan (TS(X) = (R, W))
R1(A)				TS(A) = (0,0) → TS(A) = (1,0)
	R2(B)			TS(B) = (0,0) → TS(B) = (2,0)
W1(C)				TS(C) = (0,0) → TS(C) = (0,1)
		R3(D)		TS(D) = (0,0) → TS(D) = (3,0)
			R4(E)	TS(E) = (0,0) → TS(E) = (4,0)
		W3(B)		TS(B) = (2,0) → TS(B) = (2,3), karena 3 lebih besar dari 0 dan 2
	W2(C)			TS(C) = (0,1) → TS(C) = (0,2), karena 2 lebih besar dari 0 dan 1
			W4(A)	TS(A) = (1,0) → TS(A) = (1,4), karena 4 lebih besar dari 1 dan 0
W1(D)				TS(D) = (3,0) → karena 1<3 maka TS(D) tidak dapat dieksekusi dan terjadi rollback untuk T1
A1				T1 dijalankan dengan timestamp baru, misal 5
R1(A)				TS(A) = (1,4) → TS(A) = (5,4)
W1(C)				TS(C) = (0,2) → TS(A) = (0,5)
W1(D)				TS(D) = (3,0) → TS(D) = (3,5)
C1				
	C2			
		C3		
			C4	



				TS(A)=(0,0), TS(B)=(0,0), TS(C)=(0,0), TS(D)=(0,0), TS(E)=(0,0)
				TS(A)=(0,0), 1>0, eksekusi → TS(A)=(1,0)
R1(A);				TS(B)=(0,0), 2>0, eksekusi → TS(B)=(2,0)
	R2(B);			TS(C)=(0,0), 1>0, eksekusi → TS(C)=(0,1)
W1(C);				TS(D)=(0,0), 3>0, eksekusi → TS(D)=(3,0)
		R3(D);		TS(E)=(0,0), 4>0, eksekusi → TS(E)=(4,0)
			R4(E);	TS(B)=(2,0), 3>2 & 3>0, eksekusi → TS(B)=(2,3)
		W3(B);		TS(C)=(0,1), 2>0 & 2>1, eksekusi → TS(C)=(0,2)
	W2(C);			TS(A)=(1,0), 4>1 & 4>0, eksekusi → TS(A)=(1,4)
			W4(A);	TS(D)=(3,0), 1<3, tidak dapat dieksekusi. T1 rolled back, no cascading
W1(D);				T1 dijalankan kembali dengan timestamp baru, misalnya 9
A1;				TS(A)=(1,4), 9>1 & 9>4, eksekusi → TS(A)=(9,4)
R1(A);				TS(C)=(0,2), 9>0 & 9>2, eksekusi → TS(C)=(0,9)
W1(C);				TS(D)=(3,0), 9>3 & 9>0, eksekusi → TS(D)=(3,9)
W1(D);				
C1;				
	C2;			
		C3;		
			C4;	

∴ *Schedule* yang dihasilkan adalah:

R1(A); R2(B); W1(C); R3(D); R4(E); W3(B); W2(C); W4(A); W1(D); A1; R1(A); W1(C); W1(D);

C1; C2; C3; C4;

Periksalah apakah schedule

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

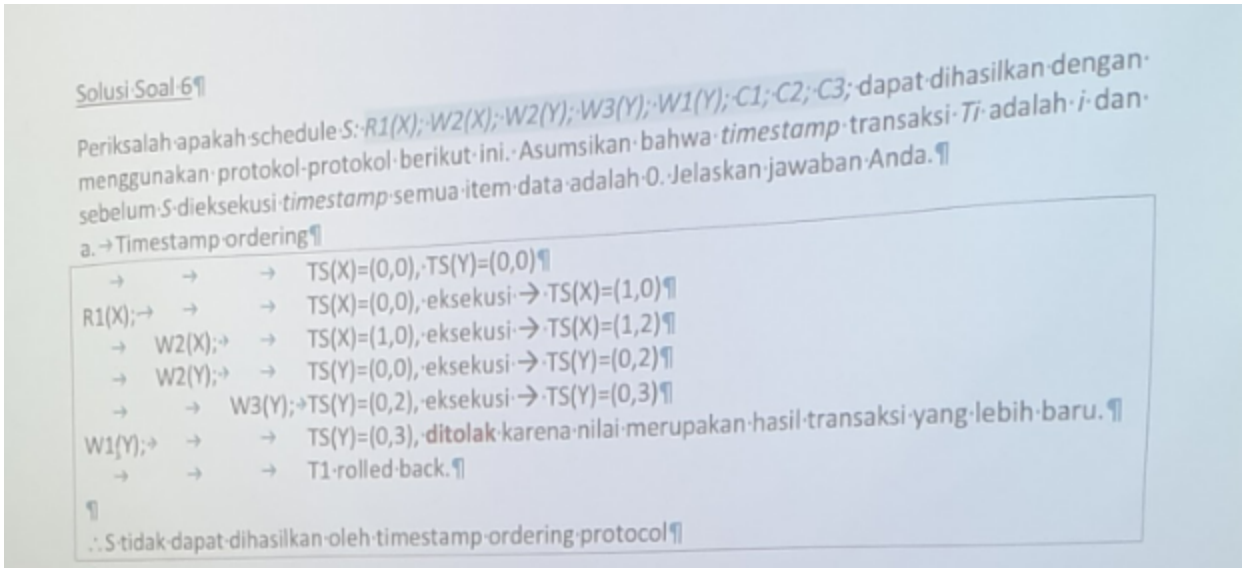
dapat dihasilkan dengan menggunakan protokol-protokol berikut ini. Asumsikan bahwa *timestamp* transaksi *Ti* adalah *i* dan sebelum *S* dieksekusi *timestamp* semua item data adalah 0. **Jelaskan** jawaban Anda.

1. Timestamp ordering
2. Timestamp ordering with Thomas' Write Rule

1. Timestamp Ordering

T1	T2	T3	Keterangan
R1(X)			TS(X) = (0,0) → TS(X) = (1,0)
	W2(X)		TS(X) = (1,0) → TS(X) = (1,2)
	W2(Y)		TS(Y) = (0,0) → TS(Y) = (0,2)
		W3(Y)	TS(Y) = (0,2) → TS(Y) = (0,3)
W1(Y)			TS(Y) = (0,3) → tidak dapat dieksekusi karena 1<3
A1			T1 dijalankan di timestamp baru, misal 5
R1(X)			TS(X) = (1,2) → TS(X) = (5,2)
W1(Y)			TS(Y) = (0,3) → TS(Y) = (0,5)
C1			
	C2		
		C3	

Berdasarkan tabel tersebut, disimpulkan bahwa schedule dapat dihasilkan dengan timestamp ordering.



Solusi Soal 6¶

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. Asumsikan bahwa *timestamp* transaksi *Ti* adalah *i* dan sebelum *S* dieksekusi *timestamp* semua item data adalah 0. Jelaskan jawaban Anda.¶

a. → Timestamp ordering¶

			TS(X)=(0,0), TS(Y)=(0,0)¶
R1(X);	→	→	TS(X)=(0,0), eksekusi → TS(X)=(1,0)¶
→	W2(X);	→	TS(X)=(1,0), eksekusi → TS(X)=(1,2)¶
→	W2(Y);	→	TS(Y)=(0,0), eksekusi → TS(Y)=(0,2)¶
→	→	W3(Y);	TS(Y)=(0,2), eksekusi → TS(Y)=(0,3)¶
$W1_1(Y)$;	→	→	TS(Y)=(0,3), ditolak karena nilai merupakan hasil transaksi yang lebih baru.¶
→	→	→	T1 rolled back.¶

¶

∴ S tidak dapat dihasilkan oleh timestamp ordering protocol¶

2. Timestamp Ordering with Thomas' Write Rule

T1	T2	T3	Keterangan
R1(X)			TS(X) = (0,0) → TS(X) = (1,0)
	W2(X)		TS(X) = (1,0) → TS(X) = (1,2)
	W2(Y)		TS(Y) = (0,0) → TS(Y) = (0,2)
		W3(Y)	TS(Y) = (0,2) → TS(Y) = (0,3)
W1(Y)			TS(Y) = (0,3) → W1(Y) diabaikan karena diasumsikan sudah dieksekusi sebelum W3(Y)
C1			
	C2		
		C3	

Berdasarkan tabel tersebut, disimpulkan bahwa schedule dapat dihasilkan dengan timestamp ordering with Thomas' Write Rule.

¶

b. → Timestamp ordering with Thomas' Write Rule¶

→	→	→	TS(X)=(0,0), TS(Y)=(0,0)¶
R1(X); →	→	→	TS(X)=(0,0), eksekusi → TS(X)=(1,0)¶
→	W2(X); →	→	TS(X)=(1,0), eksekusi → TS(X)=(1,2)¶
→	W2(Y); →	→	TS(Y)=(0,0), eksekusi → TS(Y)=(0,2)¶
→	→	W3(Y); →	TS(Y)=(0,2), eksekusi → TS(Y)=(0,3)¶
W1(Y); →	→	→	TS(Y)=(0,3), W1(Y) diabaikan (diasumsikan sudah dieksekusi sebelum W3(Y))¶
C1;¶			
→	C2;¶		
→	→	C3;¶	

∴ S dapat dihasilkan dengan timestamp ordering with Thomas' write rule protocol¶