

Berikut ini adalah urutan kedatangan operasi-operasi dari 3 buah transaksi yang berjalan secara konkuren ke *Transaction Manager*. 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.

Schedule yang dihasilkan menggunakan protokol two phase locking:

| T1 | T2 | T3 | CC Manager |
|-------|-------|-------|--------------------------------|
| RI(X) | | | XL1(X); R1(X) |
| | W2(X) | | waiting: 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 |
| | W2(X) | | XL2(X); W2(X) |
| | W2(Y) | | waiting: XL(Y) queue: W2(Y) |
| | C2 | | queue: W2(Y), C2 |
| | | C3 | UL(Y); C3 |
| | W2(Y) | | XL2(Y); W2(Y) |
| | C2 | | 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(Y), C2

Penjelasan:

T1 membaca data X ($R1(X)$) dan memperoleh $XL1(X)$ untuk melakukan operasi baca dan tulis. Setelah selesai menulis ($W1(X)$), T1 melakukan C1 dan melepaskan lock-nya ($UL(X)$). Saat T2 berusaha menulis ke X ($W2(X)$), T2 harus menunggu karena X masih dikunci oleh T1. Selain itu, T2 juga ingin menulis ke Y ($W2(Y)$), tetapi operasi ini juga harus menunggu karena ada operasi T2 yang masih *waiting*. Setelah T1 selesai dan melepaskan lock pada X, CC Manager memberikan $XL2(X)$ kepada T2 sehingga T2 dapat melanjutkan penulisan pada X. Namun, T2 kembali tertunda saat meminta $XL2(Y)$ sampai T3 melakukan commit (C3) dan melepaskan lock-nya ($UL(Y)$). Setelah itu, T2 mendapatkan $XL2(Y)$, menulis pada Y, melakukan commit (C2), dan melepaskan seluruh lock yang dipegang ($UL(X)$ dan $UL(Y)$).

JAWABAN DOSEN:

| $R1(X); W2(X); W2(Y); W3(Y); W1(X); C1; C2; C3;$ | | | |
|---|---------|---------|--|
| 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 | | | $UL(X); C1 \rightarrow XL2(X)$ bisa diberikan, instruksi T2 yang diantrikan diproses |
| | $W2(X)$ | | $XL2(X); W2(X)$ |
| | $W2(Y)$ | | wait for $XL(Y)$, queue: $W2(Y)$ |
| | C2 | | queue: $W2(Y), C$ |
| | | C3 | $UL(Y); C3 \rightarrow XL2(Y)$ 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(F); UL(G);

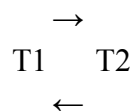
T2: SL(G); R(G); SL(F); R(F); XL(E); R(E); W(E); UL(G); UL(F); UL(E);

1. Perhatikan **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* konkuren menggunakan 2PL yang memiliki kondisi *deadlock*:

| T1 | T2 | CC Manager |
|-------|-------|--------------------------------|
| R1(E) | | SL1(E); R1(E) |
| R1(F) | | XL1(F); R1(F) |
| | R2(G) | XL2(G); R2(G) |
| | R2(F) | waiting: XL(F) queue: R2(F) |
| R1(G) | | waiting: XL(G) queue: R1(G) |

wait-for-graph:



2. Proses *deadlock recovery*:

Deadlock dapat diatasi dengan menghentikan salah satu transaksi untuk memutus siklus pada *wait-for-graph*. Sebagai contoh, apabila T2 di-*rollback*, maka seluruh operasi yang telah dilakukan T2 akan dibatalkan dan *lock* yang dipegangnya (SL(G)) dilepas. Dengan demikian, T1 dapat memperoleh XL1(G) dan melanjutkan eksekusi hingga selesai. Setelah T1 selesai dan seluruh *lock* dilepas, T2 dapat dijalankan kembali dari awal tanpa menyebabkan konflik.

3. Dua strategi *deadlock prevention*:

a. Strategi *Global Lock Ordering*

Pada strategi ini, setiap transaksi diwajibkan untuk meminta kunci (*lock*) terhadap item data dalam urutan global yang sama. Misalnya, urutan yang ditetapkan adalah $E \rightarrow F \rightarrow G$, maka seluruh transaksi harus memperoleh *lock* mengikuti urutan tersebut. Dengan adanya urutan global yang konsisten, tidak akan ada kemungkinan *circular wait* karena tidak ada transaksi yang dapat menunggu item dengan urutan lebih rendah yang sedang dipegang transaksi lain.

b. Strategi *Wait-Die* dan *Wound-Wait*

Strategi ini menggunakan waktu kedatangan transaksi (*timestamp*) untuk menentukan prioritas dalam memperoleh *lock*. Pada skema *wait-die*, jika transaksi yang lebih lama meminta *lock* yang dipegang oleh transaksi yang lebih baru, maka transaksi lama diizinkan menunggu. Namun jika transaksi yang lebih baru meminta *lock* milik transaksi yang lebih lama, transaksi lama tersebut langsung dibatalkan (*abort*). Sebaliknya, pada skema *wound-wait*, transaksi yang lebih lama dapat “melukai” transaksi yang lebih baru dengan memaksanya *rollback*, sedangkan transaksi yang lebih baru akan menunggu jika *lock* dipegang transaksi yang lebih lama. Kedua skema ini memastikan arah tunggu selalu satu arah (berdasarkan usia transaksi), sehingga siklus pada *wait-for-graph* tidak pernah terbentuk dan *deadlock* dapat dicegah.

JAWABAN DOSEN:

Diberikan 2 (dua) buah transaksi berikut ini.

$T_1: SL(E); R(E); XL(F); R(F); XL(G); R(G); W(F); W(G); UL(E); UL(F); UL(G);$
 $T_2: SL(G); R(G); SL(F); R(F); XL(E); R(E); W(E); UL(G); UL(F); UL(E);$

a. Perhatikan sebuah contoh *schedule* konkuren dari eksekusi kedua transaksi tersebut dengan menggunakan protokol *two phase locking* yang memiliki kondisi *deadlock*. Perhatikan *wait-for graph* pada saat terjadi *deadlock*.

Contoh *schedule* dengan *deadlock*: $SL_1(E); R_1(E); SL_2(G); R_2(G); XL_1(F); R_1(F);$ [] []
Pada titik ini terjadi *deadlock* karena T_2 menunggu T_1 untuk mendapatkan $SL(F)$ dan T_1 menunggu T_2 untuk mendapatkan $XL(G)$.

Wait for graph:

b. Jelaskan proses *deadlock recovery* yang dapat dilakukan pada kondisi *deadlock* pada poin a.

Deadlock recovery dapat dilakukan dengan cara me-rollback salah satu transaksi.

Misalnya T_2 dirollback, maka SL terhadap G akan dilepas. Dengan demikian XL(G) oleh T_1 akan berhasil.

c. Dengan menggunakan *schedule* yang Anda hasilkan pada poin a, jelaskan 2 (dua) strategi *deadlock prevention* yang dapat dilakukan sehingga dapat mencegah terjadinya *deadlock*.

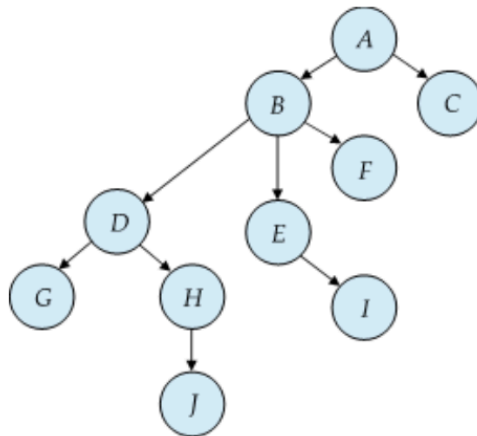
- Membuat partial ordering terhadap data, sehingga semua transaksi harus melakukan lock sesuai urutan itu. Dengan demikian, salah satu transaksi ditulis ulang sehingga urutan permintaan lock pada kedua transaksi menjadi serupa.
- Wait-die scheme (dengan asumsi bahwa T_1 datang lebih awal dari T_2): saat T_2 meminta SL(F), karena saat itu lock terhadap F sedang dipegang oleh transaksi yang datang lebih awal maka T_2 die (harus rollback).
- Wound-wait scheme (asumsi seperti sebelumnya): saat T_1 meminta XL(G), maka dia akan memaksa T_2 yang saat itu sedang memegang lock terhadap G untuk melepas lock tersebut (rollback).
- Time out based scheme dengan memberikan maksimal durasi menunggu lock. Jika sudah menunggu lebih dari durasi tersebut maka transaksi rollback.

Diberikan 2 (dua) buah transaksi berikut ini.

T_1 : R(E); R(F); R(G); W(F); W(G);

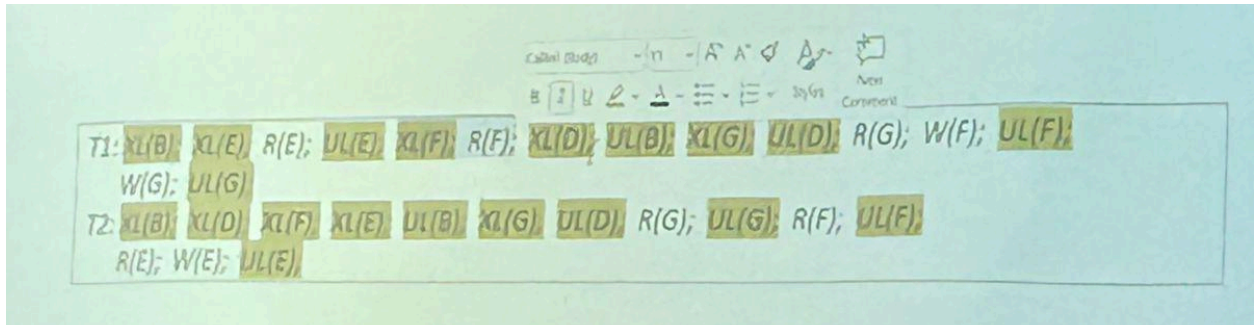
T_2 : R(G); R(F); R(E); W(E);

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

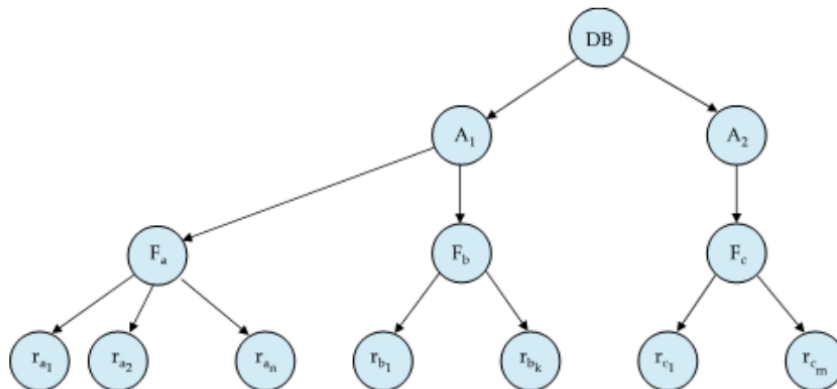


T_1 : XL1(B), XL1(E), R1(E), UL(E), XL1(F), R1(F), XL1(D), XL1(G), R1(G), W1(F), W1(G), UL(G), UL(D), UL(F), UL(B)

T_2 : XL2(B), XL2(D), XL2(G), R2(G), UL(G), UL(D), XL2(F), R2(F), UL(F), XL2(E), R2(E), W2(E), UL(E), UL(B)



Hirarki granularitas item pada sebuah basis data didefinisikan sebagai 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 soal 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), W2(rb3)

T1 memiliki IX pada level DB, A1, Fa, dan Fb karena akan melakukan penulisan terhadap *record* ra2 dan rb1. T2 juga memegang IX pada level yang sama untuk menulis rb3. Kedua transaksi tidak saling mengganggu karena *record* yang diakses berbeda (rb1 dan rb3), dan tidak ada *exclusive lock* pada level Fb itu sendiri. Dengan demikian, T2 dapat menulis pada rb3 tanpa harus menunggu T1, karena IX-IX kompatibel di level yang sama dan *exclusive lock* hanya diberikan pada *record* yang spesifik.

2. T1 membaca hampir semua record pada Fb dan menulis rb3 dan rb4

T1: IXL1(DB), IXL1(A1), SIXL1(Fb), XL1(rb3), W1(rb3), XL1(rb4), W1(rb4)

T2: ISL2(DB), ISL2(A1), ISL2(Fb), SL2(rb1), R2(rb1)

T3: IXL3(DB), IXL3(A1), IXL3(Fb), XL3(rb5), W3(rb5)

T1 memegang SIX lock pada Fb, karena membaca hampir semua *record* dan menulis sebagian (rb3, rb4). T2 dapat membaca record lain (rb1), karena SIX(Fb) masih kompatibel dengan IS(Fb). Namun, T3 yang ingin menulis *record* lain pada file yang sama (rb5) tidak dapat memperoleh *lock*, karena SIX(Fb) dan IX(Fb) tidak kompatibel. Oleh karena itu, T3 harus menunggu sampai T1 melepaskan *lock* pada Fb.

3. T2 ingin membaca seluruh record pada Fb

T1: IXL1(DB), IXL1(A1), SIXL1(Fb), XL1(rb3), W1(rb3), XL1(rb4), W1(rb4)

T2: ISL2(DB), ISL2(A1), SL2(Fb)

T2 bermaksud membaca seluruh *record* pada file Fb, sehingga membutuhkan S-lock (SL) pada level Fb. Namun, T1 telah mengunci Fb dengan SIX *lock*, yang tidak kompatibel dengan S-lock, karena T1 berpotensi melakukan penulisan di file tersebut. Akibatnya, T2 harus menunggu hingga T1 menyelesaikan transaksinya dan melepaskan *lock* Fb. Konflik ini terjadi karena SIX(Fb) menghalangi *lock* lain pada level yang sama selain IS.

JAWABAN DOSEN:

Solusi soal 4

Hirarki granularitas item pada sebuah basis data didefinisikan sebagai berikut.

```

graph TD
    DB((DB)) --> A1((A1))
    DB --> A2((A2))
    A1 --> F1((F1))
    A1 --> F2((F2))
    A2 --> F3((F3))
    F1 --> r1((r1))
    F1 --> r2((r2))
    F1 --> r3((r3))
    F2 --> r4((r4))
    F2 --> r5((r5))
    F3 --> r6((r6))
    F3 --> r7((r7))
  
```

a. Apabila T_1 sedang melakukan penulisan untuk r_{a2} dan r_{b1} , tuliskan daftar lock yang dimiliki oleh T_1 . Apakah pada saat ini T_2 dapat melakukan penulisan untuk r_{b3} ? Tuliskan daftar lock yang diperoleh T_2 hingga berhasil melakukan operasinya atau harus menunggu lock.

T_1 : IXL(DB); IXL(A_1); IXL(F_1); XL(r_{a2}); W(r_{a2}); IXL(F_2); XL(r_{b1}); W(r_{b1});
 T_2 : IXL(DB); IXL(A_1); IXL(F_1); XL(r_{b3}); W(r_{b3});
 Dapat dilihat bahwa T_2 berhasil melakukan penulisan untuk r_{b3} .

b. Apabila T_1 sedang melakukan pembacaan untuk hampir semua record pada file F_b dan melakukan penulisan untuk r_{b3} dan r_{b5} , tuliskan daftar lock yang dimiliki oleh T_1 . Apakah pada saat ini T_2 dapat melakukan pembacaan untuk r_{b1} ? Bagaimana dengan T_3 yang ingin melakukan penulisan terhadap r_{b5} ? Tuliskan daftar lock yang diperoleh T_2 dan T_3 hingga berhasil melakukan operasinya atau harus menunggu lock.

T_1 : IXL(DB); IXL(A_1); SIXL(F_b); XL(r_{b3}); W(r_{b3}); XL(r_{b4}); W(r_{b4});
 T_2 : ISL(DB); ISL(A_1); ISL(F_b); SL(r_{b3}); R(r_{b3});
 T_3 : IXL(DB); IXL(A_1); lock ini tidak kompatibel dengan SIXL(F_b) yang dimiliki $T_1 \rightarrow$ wait.
 Dapat dilihat bahwa T_2 berhasil melakukan pembacaan untuk r_{b3} sedangkan T_3 masih harus menunggu.

Berikut ini adalah urutan kedatangan operasi-operasi dari 4 buah transaksi yang berjalan secara konkuren ke *Transaction Manager*. *Timestamp* transaksi T_i 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.

| T1 | T2 | T3 | T4 | 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) |
| | W2(C) | | | TS(C) = (0, 1) → TS(C) = (0, 2) |
| | | | W4(A) | TS(A) = (1, 0) → TS(A) = (1, 4) |
| W1(D) | | | | TS(D) = (3, 0) tidak dapat dieksekusi, T1 <i>rollback</i> |
| A1 | | | | T1 dijalankan kembali dengan <i>timestamp</i> baru, misalkan 5 |
| R1(A) | | | | TS(A) = (1, 4) → TS(A) = (5, 4) |
| W1(C) | | | | TS(C) = (0, 2) → TS(C) = (0, 5) |
| W1(D) | | | | TS(D) = (3, 0) → TS(D) = (3, 5) |
| C1 | | | | |
| | C2 | | | |
| | | C3 | | |

| | | | | |
|--|--|--|----|--|
| | | | C4 | |
|--|--|--|----|--|

Sehingga, *schedule* yang dihasilkan:

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), A1, R1(A), W1(C), W1(D), C1, C1, C3, C4

JAWABAN DOSEN:

semua instruksi transaksi tersebut hingga selesai
kembali. Jelaskan jawaban Anda

$TS(X) = (RTS, WTS)$

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

| |
|---|
| C1; |
| C2; |
| C3; |
| C4; |
| 1 |
| 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. *Timestamp* transaksi T_i 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 | TS(X) = (R, W) |
|-------|-------|-------|---|
| 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) ditolak, nilai merupakan hasil transaksi yang lebih baru |
| | | | T1 <i>rollback</i> |

Berdasarkan tabel tersebut, dapat disimpulkan bahwa S tidak dapat dihasilkan oleh *timestamp ordering protocol*

2. *Timestamp ordering with Thomas' Write Rule*

| T1 | T2 | T3 | TS(X) = (R, W) |
|-------|-------|-------|--|
| 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 asumsi sudah eksekusi sebelum W3(Y) |
| | | | T1 <i>rollback</i> |
| C1 | | | |
| | C2 | | |
| | | C3 | |

Berdasarkan tabel tersebut, dapat disimpulkan bahwa S dapat dihasilkan oleh *timestamp ordering with Thomas' write rule*