Latihan Soal Concurrency Control Protocol (Bag. 1)

Soal 1

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.

Jawaban

Hasil Schedule

SL1(X); R1(X); XL3(Y); W3(Y); XL1(X); W1(X); C1; UL1(X); XL2(X); W2(X); C3; UL3(Y); XL2(Y); W2(Y); C2; UL2(X); UL2(Y);

Penjelasan

Status Awal:

- Lock table (LT) : {}
- Blocking Queue (BQ) : {}

Operasi: R1(X)

- T1 meminta SL1(X) (Shared Lock untuk Read).
- Lock Table untuk X kosong. Diberikan.

- T1 mengeksekusi R1(X).
- Schedule: SL1(X); R1(X);
- LT: { T1: SL(X) }

Operasi: W2(X)

- T2 meminta XL2(X) (Exclusive Lock untuk Write).
- Konflik: T1 sedang memegang SL(X). XL tidak kompatibel dengan SL.
- T2 diblok. Operasi W2(X) masuk ke antrian T2.
- Schedule: (Tidak berubah)
- BQ: { T2 [W2(X)] }

Operasi: W2(Y)

- Operasi ini milik T2, namun T2 sedang diblok (menunggu X).
- Operasi W2(Y) ditambahkan ke antrian T2.
- BQ: { T2 [W2(X), W2(Y)] }

Operasi: W3(Y)

- T3 meminta XL3(Y).
- Lock Table untuk Y kosong. Diberikan.
- T3 mengeksekusi W3(Y).
- Schedule: ... R1(X); XL3(Y); W3(Y);
- LT: { T1: SL(X), T3: XL(Y) }

Operasi: W1(X)

T1 meminta XL1(X) (untuk Write).

- T1 sudah memegang SL(X). Sebagai permintaan Upgrade Lock.
- Cek Konflik: Apakah ada transaksi lain yang memegang lock di X? Tidak ada.
- Lock T1 di-upgrade dari SL(X) menjadi XL(X). Diberikan.
- T1 mengeksekusi W1(X).
- Schedule: ... W3(Y); XL1(X); W1(X);
- LT: { T1: XL(X), T3: XL(Y) }

Operasi: C1 (Commit T1)

- T1 mengeksekusi Commit.
- Sesuai aturan 2PL (fase shrinking), T1 melepaskan semua lock yang dipegangnya.
- T1 melepaskan lock X. UL1(X).
- Schedule: ... W1(X); C1; UL1(X);
- LT: { T3: XL(Y) }

Cek Blocking Queue (BQ):

- Pelepasan UL1(X) membuat T2 (yang menunggu X) dapat berjalan.
- T2 tidak diblok lagi.
- Sesuai aturan, T2 mendapat prioritas untuk menjalankan antrian operasinya [W2(X), W2(Y)] sebelum TM memproses C2 dan C3.

Eksekusi Prioritas T2: W2(X)

- T2 meminta XL2(X) (dari antriannya).
- Lock Table untuk X kosong. Diberikan.
- T2 mengeksekusi W2(X).
- Schedule: ... UL1(X); XL2(X); W2(X);
- LT: { T3: XL(Y), T2: XL(X) }

Eksekusi Prioritas T2: W2(Y)

- T2 meminta XL2(Y) (dari antriannya).
- Konflik: T3 sedang memegang XL(Y).
- T2 diblok (lagi), kali ini menunggu Y. W2(Y) tetap di antrian T2.
- Prioritas T2 selesai (karena terblok). TM kembali ke urutan kedatangan.
- BQ: { T2 [W2(Y)] }

Operasi: C2 (dari urutan kedatangan)

- Operasi ini milik T2, namun T2 sedang diblok (menunggu Y).
- Operasi C2 ditambahkan ke antrian T2.
- BQ: { T2 [W2(Y), C2] }

Operasi: C3 (dari urutan kedatangan)

- T3 mengeksekusi Commit.
- T3 melepaskan semua lock yang dipegangnya.
- T3 melepaskan lock Y. UL3(Y).
- Schedule: ... W2(X); C3; UL3(Y);
- LT: { T2: XL(X) }

Cek Blocking Queue (BQ):

- Pelepasan UL3(Y) membuat T2 (yang menunggu Y) dapat berjalan.
- T2 tidak diblok lagi.
- T2 mendapat prioritas untuk menjalankan antrian operasinya [W2(Y), C2].

Eksekusi Prioritas T2: W2(Y)

- T2 meminta XL2(Y).

- Lock Table untuk Y kosong. Diberikan.
- T2 mengeksekusi W2(Y).
- Schedule: ... UL3(Y); XL2(Y); W2(Y);
- T: { T2: XL(X), T2: XL(Y) }

Eksekusi Prioritas T2: C2

- T2 mengeksekusi Commit.
- T2 melepaskan semua lock: UL2(X) dan UL2(Y).
- Schedule: ... W2(Y); C2; UL2(X); UL2(Y);
- LT: {}
- BQ: {}

Selesai. Semua operasi telah dieksekusi.

Soal 2

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. 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*.

Jawaban

No 1

Deadlock terjadi ketika dua atau lebih transaksi saling menunggu sumber daya (dalam hal ini, lock) yang dipegang oleh transaksi lainnya, membentuk siklus tunggu.
Misal ada schedule di mana T1 mendapatkan lock pada beberapa item (misalnya E dan F) dan T2 mendapatkan lock pada item lain (misalnya G), kemudian keduanya mencoba meminta lock pada item yang dipegang oleh transaksi lainnya.

Waktu	Transaksi T1	Transaksi T2	Lock Table (Item: Mode (Pemegang))	Keterangan
1	SL(E) (Grant)		{ E: SL (T1) }	T1 mendapat Shared Lock (SL) di E.

2	R(E)		{ E: SL (T1) }	T1 membaca E.
3		SL(G) (Grant)	{ E: SL (T1), G: SL (T2) }	T2 mendapat SL di G.
4		R(G)	{ E: SL (T1), G: SL (T2) }	T2 membaca G.
5	XL(F) (Grant)		{ E: SL (T1), G: SL (T2), F: XL (T1) }	T1 mendapat Exclusive Lock (XL) di F.
6	R(F)		{ E: SL (T1), G: SL (T2), F: XL (T1) }	T1 membaca F.
7		SL(G) (Block)	{ E: SL (T1), G: SL (T2), F: XL (T1) }	T2 meminta SL di F. Konflik (T1 memegang XL). T2 diblok menunggu T1.
8	XL(G) (Block)		{ E: SL (T1), G: SL (T2), F: XL (T1) }	T1 meminta XL di G. Konflik (T2 memegang SL). T1 diblok menunggu T2.

Pada Waktu 8, terjadi Deadlock.

- T1 menunggu T2 melepaskan lock G.
- T2 menunggu T1 melepaskan lock F.

Wait-For Graph (WFG)

Wait-For Graph memvisualisasikan siapa menunggu siapa. Pada saat deadlock (Waktu 8), WFG-nya adalah:

```
T1  ----(Menunggu F)---->  T2
^                          |
|                          |
+------(Menunggu G)-------+
```

Grafik ini menunjukkan siklus (T1 → T2 → T1), yang merupakan kondisi deadlock.

No 2

Deadlock recovery adalah proses untuk memecah siklus deadlock agar transaksi dapat berjalan kembali. Proses ini dilakukan dengan mengorbankan (memilih victim) salah satu transaksi dalam siklus.

Gunakan schedule dari no1:

1. Deteksi: Sistem mendeteksi siklus T1 → T2 → T1 melalui WFG.
2. Pemilihan Korban (Victim Selection): Sistem harus memilih transaksi mana yang akan di-abort (dibatalkan). Faktor pemilihan bisa bervariasi (misalnya, transaksi yang paling muda, transaksi

yang memegang lock paling sedikit, atau transaksi yang paling sedikit progresnya). Mari kita asumsikan sistem memilih T2 sebagai korban.
3. Rollback: Transaksi T2 di-roll back. Semua perubahannya dibatalkan.
4. Pelepasan Lock: T2 melepaskan semua lock yang dipegangnya (yaitu, SL(G)).
5. Pemecahan Siklus: Karena T2 melepaskan SL(G), T1 (yang menunggu G) tidak lagi diblok. T1 sekarang bisa mendapatkan XL(G) yang dimintanya.
6. Eksekusi Lanjutan: T1 melanjutkan eksekusinya (R(G), W(F), W(G), dst.) hingga selesai (Commit/Abort) dan melepaskan semua lock-nya.
7. Restart Transaksi Korban: Setelah T1 selesai, T2 (yang tadi di-abort) dapat dimulai ulang (restart) dari awal.

No 3

Deadlock prevention (pencegahan) adalah strategi di mana sistem memastikan kondisi deadlock (siklus WFG) tidak akan pernah terjadi sejak awal.

Berdasarkan schedule dari no1, deadlock terjadi karena T1 lock F lalu G, sementara T2 lock G lalu F.

Berikut dua strategi untuk mencegahnya:

Strategi 1: Resource Ordering (Pengurutan Sumber Daya)

Strategi ini mencegah deadlock dengan memaksa semua transaksi meminta lock dalam urutan yang sama (misalnya, urutan alfabetis).

Aturan: Tetapkan urutan global untuk semua item data, misalnya: E → F → G.

Penerapan:

- T1: Meminta lock E, lalu F, lalu G. Ini sesuai dengan urutan.
- T2: Ingin meminta lock G, lalu F, lalu E. Ini melanggar urutan.

Cara Mencegah Deadlock: Dengan strategi ini, T2 tidak diizinkan meminta SL(G) sebelum mendapatkan lock untuk E dan F. T2 harus meminta XL(E) terlebih dahulu.

- S1: SL(E) (Grant)
- S2: XL(E) (Blok) → T2 langsung diblok menunggu T1.
- T1 melanjutkan eksekusi (mendapat F, mendapat G) lalu selesai.
- Setelah T1 selesai dan melepas lock E, T2 baru bisa berjalan.

Hasil: T2 menunggu T1, tetapi T1 tidak pernah menunggu T2. Siklus tidak terbentuk, sehingga deadlock dicegah.

Strategi 2: Timestamp Ordering (Wait-Die atau Wound-Wait)

Strategi ini menggunakan timestamp (penanda waktu) untuk menentukan transaksi mana yang harus menunggu atau di-abort ketika terjadi konflik. Asumsikan T1 lebih tua dari T2 (TS(T1) < TS(T2)).

Gunakan skema Wait-Die:

Aturan: Jika T(i) meminta lock yang dipegang T(j):

- Jika T(i) lebih tua (TS(i) < TS(j)): T(i) (Tua) Menunggu T(j) (Muda).
- Jika T(i) lebih muda (TS(i) > TS(j)): T(i) (Muda) "Dies" (di-abort dan restart).

Penerapan pada Schedule Deadlock:

- ... (T1 memegang F, T2 memegang G) ...
- S2: SL(F) (T2 meminta lock F yang dipegang T1).
- Cek Timestamp: TS(T2) > TS(T1) (T2/Muda meminta lock dari T1/Tua).
- Keputusan (Wait-Die): T2 "Dies" (di-abort).

Hasil: T2 langsung di-abort saat meminta lock F. T2 tidak pernah masuk ke status menunggu. Karena T2 tidak menunggu, T1 juga tidak akan pernah menunggu T2 (karena T2 akan melepaskan lock G saat di-abort). Siklus tidak terbentuk, deadlock dicegah.

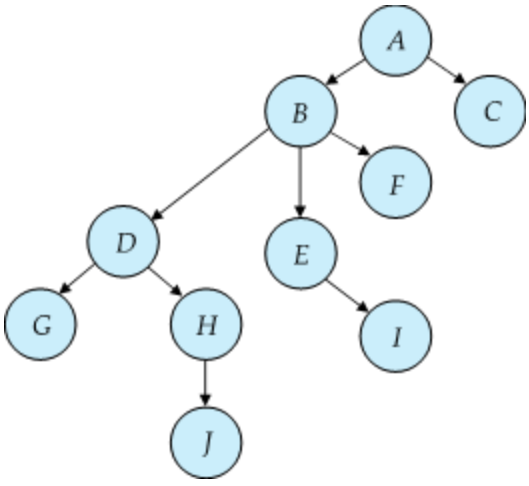
Soal 3

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 berikut ini. Urutan perintah read dan write diasumsikan tidak berubah.



Jawaban

Transaksi T1 dan T2 mengakses item {E, F, G}.

Analisis Jalur (Path Analysis):

- Untuk mengakses G: Jalurnya adalah A → B → D → G
- Untuk mengakses E: Jalurnya adalah A → B → E
- Untuk mengakses F: Jalurnya adalah A → B → F

Node Kunci (Key Node):

- Least Common Ancestor (LCA) untuk {E, F, G} adalah node B.
- Menurut Tree Protocol, lock pertama bisa di node mana saja. Untuk efisiensi, kedua transaksi bisa memulai lock pertama mereka di B karena LCAnya B.

Berikut adalah instruksi lock dan unlock berdasarkan aturan tersebut.

Untuk T1

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

F,G punya aksi R/W, sedangkan E hanya R, maka T1 perlu SL(E), XL(F), dan XL(G). Semuanya adalah turunan dari B.

```
T1:
SL(B);   // Lock pertama di LCA (B)
SL(E);   // Lock E (P(E)=B), butuh R
R(E);
XL(F);   // Lock F (P(F)=B), butuh R/W
R(F);
SL(D);   // Lock D (P(D)=B), untuk jalur ke G
XL(G);   // Lock G (P(G)=D), butuh R/W
R(G);
W(F);
W(G);
UL(B);   // Fase unlock
UL(E);
UL(F);
UL(D);
UL(G);
```

Penjelasan T1:

- T1 memulai dengan locking B.
- Dari B, locking E dan F.
- Untuk mendapatkan G, harus lock D (child B) terlebih dahulu, baru kemudian G (child D).

Sehingga urutan R/W internal transaksi tetap terjaga.

Untuk T2

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

F,G punya aksi R, sedangkan E punya R/W, maka T2 perlu SL(G), SL(F), dan XL(E). Semuanya juga turunan dari B.

```
T2:
SL(B);   // Lock pertama di LCA (B)
SL(D);   // Lock D (P(D)=B), untuk jalur ke G
SL(G);   // Lock G (P(G)=D), butuh R
R(G);
SL(F);   // Lock F (P(F)=B), butuh R
R(F);
XL(E);   // Lock E (P(E)=B), butuh R/W
R(E);
W(E);
UL(B);   // Fase unlock
UL(D);
UL(G);
UL(F);
UL(E);
```

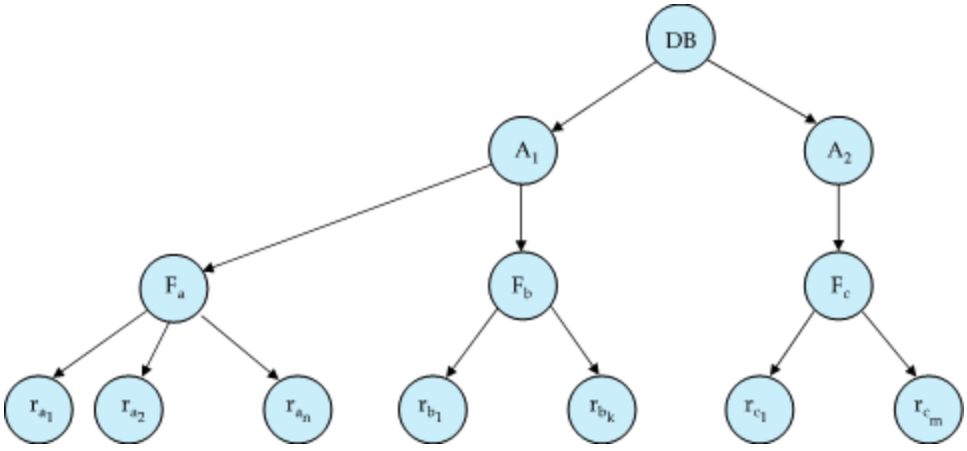
Penjelasan T2:

- T2 memulai dengan locking B.
- Untuk mendapatkan G, lock D lalu G, sehingga bisa langsung R(G).
- Kemudian kembali ke B (yang masih di-lock) untuk lock F dan E.

Sehingga urutan R/W internal transaksi tetap terjaga.

Soal 4

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*.

Jawaban

Glossary

- IS (Intention Shared): Niat untuk me-lock S pada child
- IX (Intention Exclusive): Niat untuk me-lock X (atau S) pada child
- SIX (Shared Intention Exclusive): Me-lock S pada node ini dan niat untuk me-lock X pada child

No 1 (T1 write ke ra2 dan rb1)

Agar T1 dapat menulis (membutuhkan X lock) pada record ra2 dan rb1, T1 harus mendapatkan IX lock pada semua ancestor dari record tersebut.

- Jalur ke ra2: DB → A1 → Fa → ra2
- Jalur ke rb1: DB → A1 → Fb → rb1

Daftar lock yang dimiliki T1:

- IX(DB)
- IX(A1) (LCA dari Fa dan Fb)
- IX(Fa)
- X(ra2)
- IX(Fb)
- X(rb1)

T2 Menulis rb3, T2 ingin melakukan penulisan (X lock) pada rb3. Jalurnya adalah DB → A1 → Fb → rb3.

- T2 meminta IX(DB). Kompatibel dengan IX(DB) milik T1. Diberikan.
- T2 meminta IX(A1). Kompatibel dengan IX(A1) milik T1. Diberikan.
- T2 meminta IX(Fb). Kompatibel dengan IX(Fb) milik T1. Diberikan.
- T2 meminta X(rb3). T1 tidak memegang lock apa pun pada rb3. Diberikan.

Kesimpulan: Ya, T2 dapat melakukan penulisan.

Daftar lock yang diperoleh T2: IX(DB), IX(A1), IX(Fb), X(rb3).

No 2 (T1 Membaca (Hampir Semua) Fb dan Menulis rb3, rb4)

T1 ingin membaca seluruh file Fb (membutuhkan S lock pada Fb) dan juga menulis ke beberapa record di dalamnya (membutuhkan X lock pada rb3 dan rb4). Berikut adalah skenario ideal untuk lock SIX (Shared Intention Exclusive).

- Jalur ke Fb: DB → A1 → Fb

Daftar lock yang dimiliki T1:

- IX(DB) (Diperlukan karena parent dari A1, yang akan membutuhkan IX)
- IX(A1) (Diperlukan karena parent dari Fb, yang akan membutuhkan SIX)
- SIX(Fb) (Lock S pada Fb, dan Intention X pada turunannya)
- X(rb3)
- X(rb4)

T2 Membaca rb1, T2 ingin membaca (S lock) pada rb1. Jalurnya adalah DB → A1 → Fb → rb1. T2 harus mendapatkan IS lock pada semua ancestor.

- T2 meminta IS(DB). Kompatibel dengan IX(DB) milik T1. Diberikan.
- T2 meminta IS(A1). Kompatibel dengan IX(A1) milik T1. Diberikan.
- T2 meminta IS(Fb). Tidak Kompatibel dengan SIX(Fb) milik T1.

Kesimpulan (T2): T2 harus menunggu.

Daftar lock yang diperoleh T2: IS(DB), IS(A1). (T2 menunggu untuk mendapatkan IS(Fb)).

T3 Menulis rb5, T3 ingin menulis (X lock) pada rb5. Jalurnya adalah DB → A1 → Fb → rb5. T3 harus mendapatkan IX lock pada semua ancestor.

- T3 meminta IX(DB). Kompatibel dengan IX(DB) milik T1. Diberikan.
- T3 meminta IX(A1). Kompatibel dengan IX(A1) milik T1. Diberikan.
- T3 meminta IX(Fb). Tidak Kompatibel dengan SIX(Fb) milik T1.

Kesimpulan (T3): T3 harus menunggu.

Daftar lock yang diperoleh T3: IX(DB), IX(A1). (T3 menunggu untuk mendapatkan IX(Fb)).

No 3 (T1 (Sama seperti 2) vs T2 Membaca Seluruh Fb)

Status T1: Memegang IX(DB), IX(A1), SIX(Fb), X(rb3), X(rb4).

T2 Membaca Seluruh Record pada Fb T2 ingin membaca seluruh file Fb, yang berarti T2 akan meminta S lock pada node Fb.

- Jalur ke Fb: DB → A1 → FbT2 meminta IS(DB). Kompatibel dengan IX(DB) milik T1. Diberikan.
- T2 meminta IS(A1). Kompatibel dengan IX(A1) milik T1. Diberikan.
- T2 meminta S(Fb). Tidak Kompatibel dengan SIX(Fb) milik T1.

Kesimpulan: T2 harus menunggu.

Daftar lock yang diperoleh T2: IS(DB), IS(A1). (T2 menunggu untuk mendapatkan S(Fb)).

Soal 5

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.

Jawaban

Schedule Hasil

R1(A); R2(B); W1(C); R3(D); R4(E); W3(B); W2(C); W4(A); A1; R1(A); A1; C2; C3; C4;

Penjelasan

Kondisi Awal:

- TS(T1)=1, TS(T2)=2, TS(T3)=3, TS(T4)=4
- R-TS dan W-TS untuk semua item (A, B, C, D, E) = 0.

Operasi	Transaksi	Aturan Timestamp Ordering (TO)	Keputusan	Schedule	Perubahan Timestamp Item
R1(A)	T1	TS(T1) ≥ W-TS(A) ? (1 ≥ 0) → Ya.	Execute	R1(A);	R-TS(A) = max(0, 1) = 1
R2(B)	T2	TS(T2) ≥ W-TS(B) ? (2 ≥ 0) → Ya.	Execute	... R2(B);	R-TS(B) = max(0, 2) = 2
W1(C)	T1	TS(T1) ≥ R-TS(C) (1 ≥ 0) ? Ya. TS(T1) ≥ W-TS(C) (1 ≥ 0) ? Ya.	Execute	... W1(C);	W-TS(C) = 1
R3(D)	T3	TS(T3) ≥ W-TS(D) ? (3 ≥ 0) → Ya.	Execute	... R3(D);	R-TS(D) = max(0, 3) = 3
R4(E)	T4	TS(T4) ≥ W-TS(E) ? (4 ≥ 0) → Ya.	Execute	... R4(E)	R-TS(E) = max(0, 4) = 4

W3(B)	T3	TS(T3) ≥ R-TS(B) (3 ≥ 2) ? Ya. TS(T3) ≥ W-TS(B) (3 ≥ 0) ? Ya.	Execute	... W3(B);	W-TS(B) = 3
W2(C)	T2	TS(T2) ≥ R-TS(C) (2 ≥ 0) ? Ya. TS(T2) ≥ W-TS(C) (2 ≥ 1) ? Ya.	Execute	... W2(C);	W-TS(C) = 2
W4(A)	T4	TS(T4) ≥ R-TS(A) (4 ≥ 1) ? Ya. TS(T4) ≥ W-TS(A) (4 ≥ 0) ? Ya.	Execute	... W4(A);	W-TS(A) = 4
W1(D)	T1	TS(T1) ≥ R-TS(D) ? (1 ≥ 3) → Tidak.	Abort T1	... W4(A); A1;	(Tidak ada perubahan)
Antrian Prioritas (Rollback T1)	T1	Sesuai aturan, T1 di-rollback dan operasinya (R1(A), W1(C), W1(D)) diprioritaskan untuk dieksekusi kembali.			
R1(A)	T1	(Eksekusi ulang) TS(T1) ≥ W-TS(A) ? (1 ≥ 4) → Tidak.	Abort T1 (Lagi)	... A1; R1(A); A1;	(Tidak ada perubahan)
Akhir T1	T1	T1 gagal saat restart dan akan selalu gagal karena TS-nya (1) lebih kecil dari W-TS(A) (4). Transaksi T1 diaborsi secara permanen. Operasi C1 tidak pernah dieksekusi.			
C2	T2	(Melanjutkan antrian utama)	Execute	... A1; C2;	(Tidak ada perubahan)
C3	T3		Execute	... C2; C3;	(Tidak ada perubahan)
C4	T4		Execute	... C3; C4;	(Tidak ada perubahan)

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. *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

Jawaban

Kondisi Awal:

- TS(T1)=1, TS(T2)=2, TS(T3)=3
- R-TS(X)=0, W-TS(X)=0
- R-TS(Y)=0, W-TS(Y)=0

Timestamp Ordering

Protokol TO standar mengharuskan rollback jika transaksi mencoba R/W data yang telah diakses (R/W) oleh transaksi dengan timestamp yang lebih baru.

R1(X) (TS=1)

- TS(T1) ≥ W-TS(X) ? (1 ≥ 0) → Ya.
- Eksekusi.
- R-TS(X) = max(0, 1) = 1.

W2(X) (TS=2)

- TS(T2) ≥ R-TS(X) ? (2 ≥ 1) → Ya.
- TS(T2) ≥ W-TS(X) ? (2 ≥ 0) → Ya.
- Eksekusi.
- W-TS(X) = 2.

W2(Y) (TS=2)

- TS(T2) ≥ R-TS(Y) ? (2 ≥ 0) → Ya.
- TS(T2) ≥ W-TS(Y) ? (2 ≥ 0) → Ya.
- Eksekusi.
- W-TS(Y) = 2.

W3(Y) (TS=3)

- TS(T3) ≥ R-TS(Y) ? (3 ≥ 0) → Ya.
- TS(T3) ≥ W-TS(Y) ? (3 ≥ 2) → Ya.
- Eksekusi.
- W-TS(Y) = 3.

W1(Y) (TS=1)

- TS(T1) ≥ R-TS(Y) ? (1 ≥ 0) → Ya.
- TS(T1) ≥ W-TS(Y) ? (1 ≥ 3) → Tidak.
- Aturan Write dilanggar.
- Operasi ditolak dan T1 harus Abort (Rollback).

Kesimpulan: Schedule S tidak dapat dihasilkan.

Penjelasan: Timestamp Ordering standar tidak mengizinkan T1 (TS=1) untuk menulis ke Y, karena Y sudah ditulis oleh T3 (TS=3) yang memiliki timestamp lebih besar. Operasi T1 dianggap "terlambat" (obsolete) dan T1 akan dipaksa untuk rollback.

Timestamp Ordering with Thomas' Write Rule

Protokol TWR memodifikasi aturan Write. Jika operasi Write terlambat (TS(Ti) < W-TS(X)), operasi tersebut diabaikan (ignored) alih-alih menyebabkan rollback.

R1(X) (TS=1)

- TS(T1) ≥ W-TS(X) ? (1 ≥ 0) → Ya.
- Eksekusi.
- R-TS(X) = 1.

W2(X) (TS=2)

- TS(T2) ≥ R-TS(X) ? (2 ≥ 1) → Ya.
- TS(T2) ≥ W-TS(X) ? (2 ≥ 0) → Ya.
- Eksekusi.
- W-TS(X) = 2.

W2(Y) (TS=2)

- TS(T2) ≥ R-TS(Y) ? (2 ≥ 0) → Ya.
- TS(T2) ≥ W-TS(Y) ? (2 ≥ 0) → Ya.
- Eksekusi.
- W-TS(Y) = 2.

W3(Y) (TS=3)

- TS(T3) ≥ R-TS(Y) ? (3 ≥ 0) → Ya.
- TS(T3) ≥ W-TS(Y) ? (3 ≥ 2) → Ya.
- Eksekusi.
- W-TS(Y) = 3.

W1(Y) (TS=1)

- TS(T1) ≥R-TS(Y) ? (1 ≥ 0) → Ya.
- TS(T1) < W-TS(Y) ? (1 < 3) → Ya.
- Terapkan Thomas' Write Rule: Operasi W1(Y) diabaikan.
- Transaksi T1 tidak rollback.

C1; C2; C3;

- Karena tidak ada transaksi yang rollback, semua commit dapat dieksekusi.

Kesimpulan: Schedule S dapat dihasilkan.

Penjelasan: Thomas' Write Rule (TWR) secara spesifik dirancang untuk menangani skenario ini. Ketika W1(Y) tiba, protokol melihat bahwa write tersebut sudah usang (karena W-TS(Y)=3). Daripada memaksa T1 rollback, TWR mengizinkan schedule untuk melanjutkan dengan mengabaikan W1(Y). Karena tidak ada rollback, urutan operasi dalam schedule S dapat berjalan hingga selesai.