09

Desain Aplikasi

by: Ahmad Syauqi Ahsan

Pengendalian Konkurensi

- Protokol berbasis-penguncian
- Protokol berbasis-pembatasan waktu
- Protokol berbasis-validasi
- Multiple Granularity
- Skema multiversi
- Penanganan Deadlock
- Operasi Insert dan Delete
- Konkurensi dalam struktur indeks

Protokol berbasis penguncian

- Penguncian adalah salah satu mekanisme pengendalian akses konkonkuren terhadap sebuah item data
- Item data dapat dikunci dengan dua cara :

  1. *exclusive (X) mode*. Item data dapat dibaca (read) dan diubah(write) dengan sama baik. Penguncian tergadap data x membutuhkan instruksi **lock-X**.

  2. *shared (S) mode*. Item datahanay dapat dibaca (read). Untuk men*share* kan data digunakan perintah **lock-S**.

- Penguncian dibutuhkan untuk mengelola proses konkuren. Transaksi dapat diperoses setelah ada jaminan.

- Tabel kemungkinan penguncian

	S	X
S	true	false
X	false	false

- Sebuah transaksi terkadang membutuhkan jaminan penguncian pada saat mengakses item data supaya tertutup terhadap transaksi yang lain
- Beberapa transaksi dapat men-share sebuah item, tetapi jika beberapa transaksi menahan secara eksklusif pada sebuah item maka tidak ada transaksi lain yang dapat melakukan penguncian pada item tersebut.
- Jika sebuah penguncian tidak diperoleh, transaksi yang diminta akan dibuat menunggu sampai penguncian yang dilakukan transaksi lain dilepas.

- Contoh penerapan penguncian pada pentransferan dana dari B ke A :

T1: **lock-X***(B)*;
read *(B)*;
B← B – 500
write *(B)*
unlock*(B)*;
lock-X*(A)*;
read *(A)*;
A ← A + 500
wtite (A)
unlock*(A)*;

- Transaksi T2 yang akan menampilkan total saldo kedua rekening:

T_2: **lock-S**(*A*);
read (*A*);
unlock(*A*);
lock-X(*B*);
read (*B*);
write (B)
unlock(*B*);
display(*A*+*B*)

- Sebuah **locking protocol** adalah sekumpulan aturan dalam sebuah transaksi yang memanggil dan melepas penguncian. Protokol locking akan membatasi penjadwalan yang ada.

Kemungkinan pada protokol-penguncian

- Sehubungan dengan sebagian jadwal

T_3	T_4
lock-X(B)	
read(B)	
$B := B - 50$	
write(B)	
	lock-S(A)
	read(A)
	lock-S(B)
lock-X(A)	

- Baik T_3 maupun T_4 tidak dapat maju lagi — ekesekusi **lock-S**(*B*) mengakibatkan T_4 menunggu T_3 untuk melepaskan penguncian terhadap *B*, sementara eksekusi **lock-X**(*A*) mengakibatkan T_3 menunggu T_4 melepaskan penguncian terhadap *A*.
- Kondisi ini disebut **deadlock**.
  - Untuk mengatasi masalah ini T_3 atau T_4 harus di roll back dan melepaskan kuncian.

- Deadlock selalu mungkin terjadi dalam protokol lock.
- **Starvation** juga mungkin terjadi jika pengendalian akses konkuren tidak baik. Contoh :
  - Sebuah transaksi mungkin dapat menunggu X-lock pada sebuah item, sementara transaksi lain pada urutan membutuhkan S-lock pada item yang sama.
  - Transaksi lain yang sama berulang-ulang melakukan roll back sampai dengan deadlock.
- Pengelolaan konkurensi dapat dirancang untuk menghindari starvation.

Tahapan penguncian

- Aturan ini menjamin terjadinya conflict-serializable .
- Phase 1: Fase bertumbuh (Growing Phase)
  - Transaksi dapat melakukan sejumlah penguncian, tetapi belum melepaskan satupun penguncian
- Phase 2: Fase pelepasan (Shrinking Phase)
  - Transakssi mungkin melepas kunci
  - Transaksi belum melakukan penguncian yang baru
- Titik dalam schedule dimana transaksi tersebut telah mendapatkan penguncian akhir disebut lockpoint transaksi.

- Locking dua fase *tidak menjamin* terjadinya deadlock
- **strict two-phase locking**. Dengan mekanisme ini dikehendaki bahwa semua penguncian dengan mode *exclusive* dari sebuah transakasi harus tetap dipegang hingga transaksi berada dalam status berhasil sempurna (commiteed).
- **Rigorous two-phase locking** yang menghendaki semua penguncian (exclusive maupun share) tetap diterapkan hingga transksaksi committed.

Konversi penguncian

- Contoh :

  T_5 : read (a_1)
  read (a_2)
  …
  read (a_n)
  write (a_1)

  T_6 : read (a_1)
  read (a_2)
  write (a_1 +a_2)

- Jika menerapkan Locking Dua fase, maka T_5 harus mengunci a_1 dalam mode exclusive. Akibatnya semua eksekusi konkuren dari kedua transaksi menjadi eksekusi serial.
- Jika T_5 melakukan penguncian dengan mode exclusive di saat penulisan a_1, maka kondisi konkurensi akan lebih baik, karena T5 dan T_6 dapat mengakses a_1 dan a_2 secara simultan
- Peningkatan penguncian dari share menjadi exclusive disebut **upgrade** dan sebaliknya disebut **downgrade**

Konversi Penguncian

T_5	T_6
Lock-S (a1)	
Lock-S (a2)	Lock-S (a1)
Lock-S (a3) Lock-S (a4)	Lock-S (a2)
	Unlock (a1) Unlock (a2)
Lock-S (an) Upgdrade (a1)	

Akuisisi otomatis dari penguncian

- Transaksi T_i menjalankan operasi read/write standar tanpa ada prosedur penguncian.
- Operasi **read**(*D*) akan dijalankan :

```
if Ti sudah mengunci D
  then
        read(D)
  else
      begin
          if diperlukan tunggu s/d tidak ada
              transaksi lain me- lock-X pada D
          lakukan Ti  lock-S pada D;
          read(D)
      end
```

- Proses **write**(*D*) :

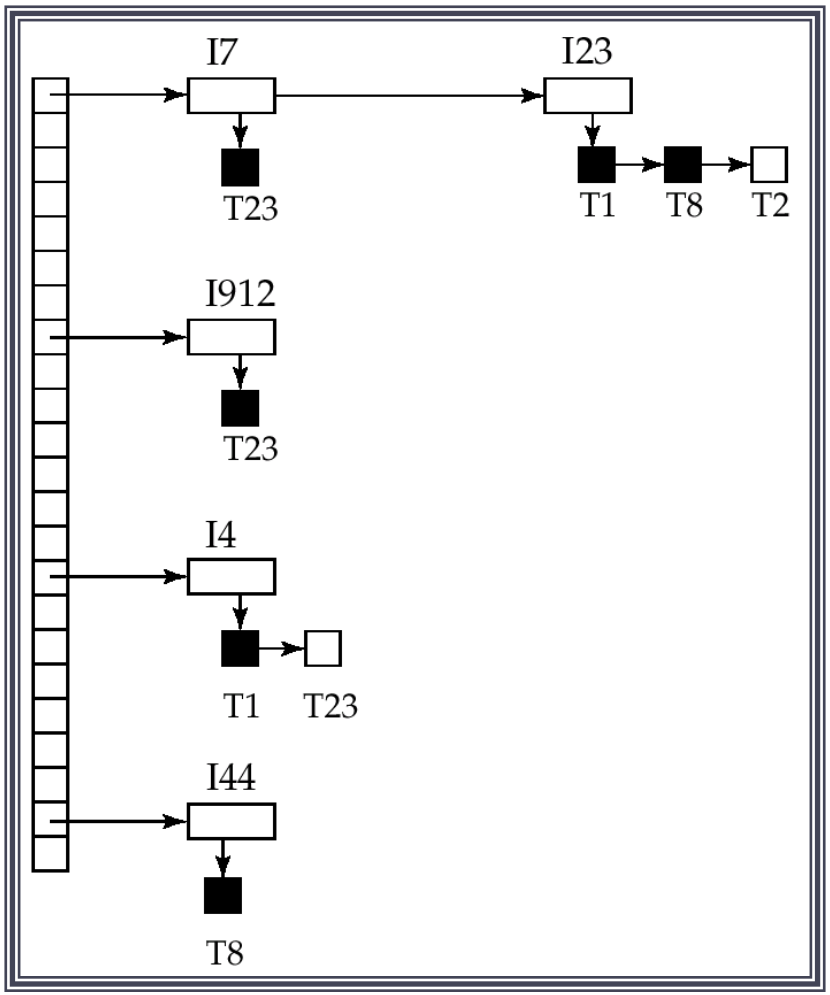
  if T_i telah **lock-X** pada *D*
  then
  write(*D*)
  else
  begin
  jika perlu tunggu s.d tidak ada transaksi lain lock pada *D*,
  jika T_i telah **lock-S** pada *D*
  then
  upgrade lock pada *D* ke **lock-X**
  else
  perintahkan T_i me- **lock-X** pada *D*
  write(*D*)
  end;

- Semua penguncian akan dilepas setelah transaksi *committed* atau *abort*

Penerapan Penguncian

- **Sebuah Lock manager** dapat diterapkan sebagai sebagian dari proses yang melayani permintaan lock dan unlock
- lock manager menjawab permintaan lock dengan mengirimkan pesan penguncian (atau pesan melakukan roll back dalam kasus deadlock)
- Transaksi yang minta akan mennggu sampai dijawab
- lock manager merawat struktur data yang disebut **lock table** untuk menjamin penguncian record dan menunda permintaan
- lock table selalu diterapkan sebagai tabel indeks yang ada di memory pada nama data yang di lock

Lock Table

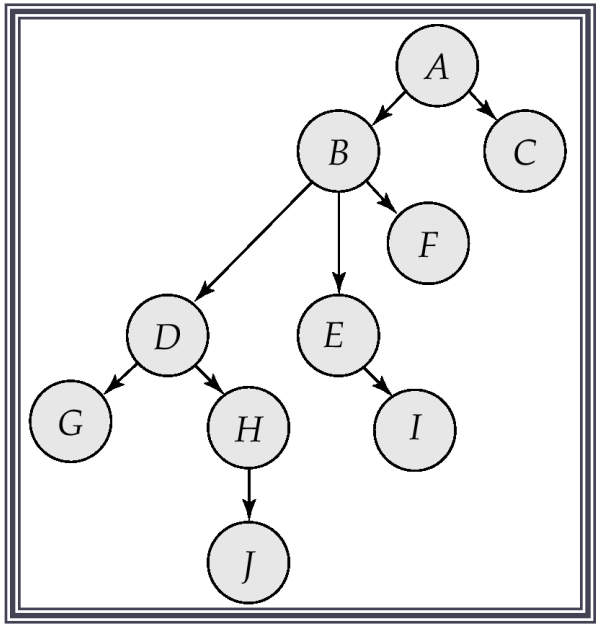


- Kotak hitam tandanya sedang mengunci, sedang yang putih menunggu permintaan
- Lock table juga mencatat jenis penguncian
- Permintaan barau ditambahkan diakhir antrian permintaan untuk item data, dan menjadi jaminan terhadap semua penguncian terakhir
- Permintaan Unlock akan menghapus lock, dan kemudian permintaan akan memeriksa apakah bisa dilakukan sekarang
- Jika transaksi batal, semua proses tunggu dihapus
  - lock manager akan menjaga daftar kejadian lock setiap transaksi secara efisien

Protokol berbasis Graph

- Protokol berbasis Graph adalah alternatif dalam two-phase locking
- Memberikan sebagian permintaan $\rightarrow$ pada himpunan **D** = $\{d_1, d_2, ..., d_h\}$ semua item data.
  - Jika $d_i \rightarrow d_j$ maka semua transaksi yang mengakses d_i dan d_j harus mengakses d_i lebih dahulu sebelum mengakses d_j.
  - Akibatnya himpunan **D** dapat dilipandang sebagai *database graph*.
- *Tree-protocol* dalam graph protocol.

Tree Protocol



- Hanya mengijinkan exclusive lock.
- Penguncian pertama oleh T_i mungkin terhadap beberapa item data. Setelah itu, sebuah data Q dapat dikunci oleh T_i hanya jika parent dari Q saat ini di kunci oleh T_i.
- Item data mungkin di-unlocked beberapa kali.

- tree protocol menjamin conflict serializability dengan membebaskan dari deadlock.
- Unlocking terjadi lebih cepat diakhir tree-locking protocol dibanding two-phase locking protocol.
  - Waktu tunggu lebih pendek, dan meningkat dalam konkurensi
  - protocol bebas deadlock, tidak perlu rollback
  - Pembatalan transaksi dapat mengakibatkan penumpukan rollback.
- Bagaimanapun, dalam penguncian dengan protokol tree dapat terjadi , sebuah transaksi mengunci item data yang tidak diakses.
  - memperkuat locking, dan menambah waktu tunggu
  - bisa berkurang dalam konkurensi
- Penjadwalan yang tidak mungkin dibawah two-phase locking menjadi mungkin dibawah tree protocol.

Timestamp-Based Protocols

- setiap transaksi di tandai waktu kehadirannya dalam sistem. Jika transaksi yang lamaT_i mempunyai time-stamp TS(T_i), transaksi yang baru T_j diberi time-stamp TS(T_j) dimana TS(T_i) <TS(T_j).
- Skema ini menjamin serializability dengan memilih sebuah urutan diantara setiap pasangan transaksi.
- Untuk menerapkan skema ini, diterapkan dua nilai timestamp pada setiap item data Q :
  - **W-timestamp**(Q) yang menunjukkan nilai *timestamp* terbesar dari setiap transaksi yang berhasil menjalankan operasi **write**(Q).
  - **R-timestamp**(Q) yangmenunjukkan nilai timestamp terbesar dari setiap transaksi yang berhasil menjalankan operasi **read**(Q).

- Timestamp akan terus diperbarui ketika ada perintah baru **read** dan **write** yang dieksekusi.
- Untuk transaksi T_i yang menjalankan operasi **read**(Q)
  - Jika TS(T_i) ≤ **W**-timestamp(Q), maka T_i perlu membaca kembali nilai Q yang ditulis. Karena itu,operasi **read** ini akan ditolak dan T_i akan di rolled back.
  - Jika TS(T_i)≥ **W**-timestamp(Q), maka operasi **read** dieksekusi, dan R-timestamp(Q) diisi dengan nilai terbesar diantara R-timestamp(Q) dan TS(T_i).

Timestamp-Based Protocols (Cont.)

- Untuk transaksi T_i yang menjalankan operasi **write**(Q).
  - Jika TS(T_i) < R-timestamp(Q), maka nilai Q yang baru yang dihasilkan T_i adalah nilai yang tidak akan dimanfaatkan lagi, dan sistem berasumsi bahwa nilai tersebut tidak pernah dihasilkan. Karena itu, operasi **write** ditolak dan transaksi T_i di- roll back.
  - Jika TS(T_i) < W-timestamp(Q), maka berarti T_i sedang berusaha melakukan penulisan nilai Q yang kadaluwarsa. Karena itu, operasi **write** akan ditolak dan T_i di- roll back.
  - Kecuali itu, operasi **write** dieksekusi, dan W-timestamp(Q) diberi nilai baru yang sama dengan TS(T_i).

Contoh penggunaan Protocol

Sebagian jadwal item data dengan transaksi yang mempunyai timestamps 1, 2, 3, 4, 5

T_1	T_2	T_3	T_4	T_5
				read(X)
read(Y)	read(Y)			
		write(Y)		
		write(Z)		
				read(Z)
	read(X)			
	abort			
read(X)				
		write(Z)		
		abort		
				write(Y)
				write(Z)

Correctness of Timestamp-Ordering Protocol

- Protokol timestamp-ordering menjamin conflict serializability jika prosesnya mengikuti urutan:

- Protokol ini menjaminkonkurensi terbebas dari deadlock, karena tidak ada transaksi yang harus menunggu.

Validation-Based Protocol

- Eksekusi dari transaksi T_i selesai dalam tiga tahap.
  1. **Read dan eksekusi:** Transaksi T_i melakukan operasi ***write*** hanya pada variabel lokal temporer tanpa melakukanperubahan ke basis data aktual
  2. **Validasi:** Transaksi T_i membentuk uji validasi untukmenentukan apkah transaksi tersebutdapat melakukan penyalinan / pengubahan ke basis data dari variabel lokal temporere yang nilainya diperoleh dari operasi write tanpa menyebabkan pelanggaran serializability.
  3. **Write :** Jika fase validasi transaksi T_i berhasil, maka perubahan sesungguhnya dilakukan ke basis data. Jika validasi tidak berhasil, maka T_i akan di-roll back.
- Semua fase dalam eksekusi transaksi konkuren dapat terjadi pada waktu bersamaan.
- Disebut juga **optimistic concurrency control**

- Setiap transaksi T_i akanmemiliki 3 timestamp
  - **Start**(T_i) : wkatu dimana T_i memuliaieksekusinya
  - **Validation**(T_i): waktu dimana T_i, selesai melakukan Fase pembacaan dan memulai fase validasi
  - **Finish**(T_i) : waktu dimana T_i menyelesaikan fase penulisan
- Urutan serializability ditentukan dengan teknik pengurutan timestamp dengan menggunakan nilai timestamp validation (T_i), oleh karena itu nilai $TS(T_i)$ = **Validation**(T_i).

Uji validasi untuktransaksi T_i

- Jika untuk semua transaksi T_i dengan TS (T_i) < TS (T_j) salah satu dari dua kondisi berikut harus dapat dipenuhi :
  - **finish**(T_i) < **start**(T_j) **,** karena T_i menyelesaikan eksekusinya sebelum T_j dimulai
  - **start**(T_j) < **finish**(T_i) < **validation**(T_j) **dan** thimpunan item data yang ditulis T_i dtidak beririsan dengan himpunan item data yang dibaca oleh T_j.

  kemudian validasi T_j dikatakan berhasil, jika tidak validasi gagal dan T_j di batalkan.
- *Justification*: Either first condition is satisfied, and there is no overlapped execution, or second condition is satisfied and

1. operasi write oleh T_j jangan dilakukan sampaidengan operasi read dari T_i selesai.
2. operasi write dariT_i jangan mempengaruhi operasi reads T_j jikaT_j tidak melakukan operasi read terhadap operasi wite yamg dilakukan T_i.

Schedule Produced by Validation

- Contoh skedul yang menggunakan validation

T_{14}	T_{15}
read(*B*)	
	read(*B*)
	B:- B-50
	read(*A*)
	A:- A+50
read(*A*)	
(*validate*)	
display (*A*+*B*)	
	(*validate*)
	write (*B*)
	write (*A*)

Penanganan Deadlock

- Ada dua transkasi sebagai berikut :

T_1: write (X) T_2: write(Y)

write(Y) write(X)

- Penjadwalan dengan deadlock

T_1	T_2
lock-X on X	
write (X)	
	lock-X on Y
	write (X)
	wait for **lock-X** on X
wait for **lock-X** on Y	

- Sistem dikatak deadlock bilaman ada lebih dari satu transaksi berada dalamkeadaan saling tunggu untuk melakukan akses terhadap sebuah item data.
- ***Pencegahan Deadlock* dapat dilakukan dengan dua metode berikut :**
  - Transaksi harus mengunci semua item data sebelum memulai eksekusi.
  - Mengijinkan sistem memasuki kondisi deadlock dan kemudian berusaha untuk mengatasinya dengan memanfaatkan skema pendeteksian dan pemulihan deadlock.

Strategi pencegahan deadlock

- Ada dua skema pendekatan dalam mencegah terjadinya deadlock yangmenggunakn timestamp.
- **wait-die** — non-preemptive
  - Ketika transaksi T_i membutuhkan sebuah item data yang sedang dipegang oleh T_j, T_i dibolehkanmenunggu hanya jika ia memiliki timestamp yanglebih kecil dari T_j (T_i lebih dahulu dari T_j).Jika tidak, T_i akan dibatalkan.
- **wound-wait** — preemptive
  - Merupakan lawan dari skema pertama. Ketika transaksi T_i membutuhkan item data yang sedang dipegang oleh T_j, T_i diperbolehkan menunggu jika ia memiliki time stamp yang lebih besar dari pada T_j (Ti datang belakangan). Jika tidak, T_i akan dibatalkan

Tanya Jawab

Terima Kasih