Manajemen Transaksi
(Penjadwalan & Kontrol konkurensi)

Sistem Basis Data

Gentisya Tri Mardiani, S.Kom., M.Kom

Schedule (Penjadwalan)

- Urutan instruksi yang menspesifikasikan urutan kronologi instruksi dari transaksi yang dieksekusi.

- Sebuah jadwal harus menjaga urutan instruksi yang muncul di setiap transaksi.

Schedule 1

- A= $100, B= $100
- T1 transfer $50 dari A ke B
- T2 transfer 10% dari A ke B
- Schedule serial dimana T1 diikuti T2:

T1	T2
Read (A) A← A-50 Write(A) Read(B) B← B+50 Write(B)	
	Read (A) Temp← 0.1*A A← A-temp Write(A) Read(B) B← B+temp Write(B)

Schedule 2

- Penjadwalan tidak serial, tetapi ekuivalen dengan schedule 1

T1	T2
Read(A) A← A-50 Write(A)	
	Read (A) Temp← 0.1*A A← A-temp Write(A)
Read(B) B← B+50 Write(B)	
	Read(B) B← B+temp Write(B)

Schedule 3

- Penjadwalan tidak serial, dan hasilnya tidak konsisten.

T1	T2
Read(A) A← A-50	
	Read (A) Temp← 0.1*A A← A-temp Write(A) Read(B)
Write(A) Read(B) B← B+50 Write(B)	
	B← B+temp Write(B)

Serializability

- Penjadwalan serializable merupakan penjadwalan secara serial.
- Setiap transaksi harus tetap menjaga konsistensi database.
- Sistem basis data harus dapat mengontrol eksekusi konkurensi dari suatu transaksi untuk memastikan database tetap terjaga konsistensinya.

Serializability

- Setiap transaksi dengan sepenuhnya terisolasi sedemikian rupa sehingga transaksi bertindak seolah-olah mereka telah mengeksekusi berturutan, satu demi satu; berturut-turut.

- Dalam mencapai hal ini**, DBMS akan secara khusus mengunci setiap baris yang dibaca**, maka sesi lain tidak boleh memodifikasi data itu sampai transaksi telah selesai. Kunci dilepaskan ketika transaksi ***commit*** atau ***rollback***.

Teknik Pengontrolan Konkurensi

- Metode locking
- Metode timestamp

- Metode locking dan timestamp dapat menyebabkan penundaan transaksi jika terjadi konflik dengan transaksi lainnya pada waktu yang sama.

Locking

- Metode locking merupakan pendekatan yang paling banyak digunakan untuk memastikan serializability.

- Apabila suatu transaksi mengakses suatu data maka suatu lock (kunci) dapat mencegah pengaksesan oleh transaksi lain.

Locking

- Secara umum, transaksi harus menegaskan penguncian (*lock*) *shared* (*read*) atau *exclusive* (*write*) terhadap data item sebelum pembacaan (*read*) atau penulisan (*write*).

- Aturan dasar penguncian (locking):
- ***Shared Lock***, maka transaksi dapat melakukan pembacaan tetapi tidak melakukan perubahan.
- ***Exclusive Lock,*** maka transaksi dapat melakukan pembacaan dan perubahan terhadap data item tersebut.

Locking

Cara kerja dari kunci :

- Kita asumsikan terdapat 2 (dua) macam kunci :
- **Kunci X** (kunci eksklusif) dan **kunci S** (kunci yang digunakan bersama-sama/ shared)

- Jika **transaksi A** menggunakan **kunci X** pada record R, maka permintaan dari **transaksi B harus menunggu** sampai nanti transaksi A melepaskan kunci
- Jika **transaksi A** menggunakan **kunci S** pada record R, maka :
  - Bila **transaksi B** ingin menggunakan **kunci X**, maka B harus **menunggu** sampai A melepaskan kunci tersebut.
  - Bila **transaksi B** ingin menggunakan **kunci S**, maka B bisa **menggunakan** kunci S **bersama** A

Locking

- Kunci X dan kunci S akan dilepaskan pada saat ***Synchpoint*** (*synchronization point*).

- Bila synchpoint ditetapkan maka:
  - semua modifikasi program menjalankan operasi COMMIT atau ROLLBACK
  - semua kunci dari record dilepaskan

Matriks Locking

A \ B	x	s	-
x	N	N	Y
s	N	Y	Y
-	Y	Y	Y

- Untuk menjamin serializability, membutuhkan protokol tambahan mengenai posisi dari operasi penguncian dan pelepasan kunci dalam setiap transaksi.

Two-Phase Locking (2PL)

- Suatu transaksi menggunakan protokol 2PL jika seluruh operasi penguncian (locking) mendahului operasi pelepasan kunci (unlock) dalam transaksi.

- Terdapat dua fase untuk transaksi yang harus dilalui, yaitu :
- *Growing phase* – mendapatkan seluruh kunci tetapi tidak dapat melepaskan kunci.
- *Shrinking phase* – melepaskan kunci tetapi tidak mendapatkan kunci baru.

Two-Phase Locking (2PL)

- Intinya, suatu transaksi jangan pernah melepaskan kunci sebelum operasi selesai, dengan aturan:

- Satu transaksi harus meminta/ menetapkan sebuah kunci sebelum melaksanakan operasi pada transaksi tersebut. Kunci yang diminta dapat berupa *write lock* (*exclusive*) maupun *read lock* (*shared*) , sesuai kebutuhan.
- Sekali transaksi melepaskan kunci, maka transaksi tersebut tidak dapat meminta kunci yang baru.

Lost Update Problem

- Update yang dilakukan oleh user pertama diubah oleh user yang lain.

Time	T_1	T_2	bal_x
t_1		begin_transaction	100
t_2	begin_transaction	read(bal_x)	100
t_3	read(bal_x)	$bal_x = bal_x + 100$	100
t_4	$bal_x = bal_x - 10$	write(bal_x)	200
t_5	write(bal_x)	commit	90
t_6	commit		90

- Kehilangan modifikasi ini dapat diatasi dengan mencegah T1 melakukan pembacaan data sebelum perubahan T2 selesai dilaksanakan.

Penyelesaian masalah dengan Locking

- Lost update problem

Time	T_1	T_2	bal_x
t_1		begin_transaction	100
t_2	begin_transaction	write_lock(bal_x)	100
t_3	write_lock(bal_x)	read(bal_x)	100
t_4	WAIT	$bal_x = bal_x + 100$	100
t_5	WAIT	write(bal_x)	200
t_6	WAIT	commit/unlock(bal_x)	200
t_7	read(bal_x)		200
t_8	$bal_x = bal_x - 10$		200
t_9	write(bal_x)		190
t_{10}	commit/unlock(bal_x)		190

Uncommitted Dependency Problem

- Contoh transaksi T4 merubah balx menjadi $200 tetapi digagalkan, sehingga balx harus dikembalikan ke nilai awal sebelum transaksi yaitu $100. Sedangkan transaksi T3 membaca nilai hasil modifikasi tadi yaitu, balx ($200) dan menguranginya dengan $10, sehingga memperoleh nilai akhir $190, yang seharusnya $90.
- Masalah tersebut dapat dihindari Problem dengan mencegah T3 membaca balx sebelum T4 dinyatakan *committed* atau *abort*.

Time	T_3	T_4	bal_x
t_1		begin_transaction	100
t_2		read(bal_x)	100
t_3		$bal_x = bal_x + 100$	100
t_4	begin_transaction	write(bal_x)	200
t_5	read(bal_x)	⋮	200
t_6	$bal_x = bal_x - 10$	rollback	100
t_7	write(bal_x)		190
t_8	commit		190

Penyelesaian masalah dengan Locking

- Uncommitted Dependency Problem

Time	T_3	T_4	bal_x
t_1		begin_transaction	100
t_2		write_lock(bal_x)	100
t_3		read(bal_x)	100
t_4	begin_transaction	$bal_x = bal_x + 100$	100
t_5	write_lock(bal_x)	write(bal_x)	200
t_6	WAIT	rollback/unlock(bal_x)	100
t_7	read(bal_x)		100
t_8	$bal_x = bal_x - 10$		100
t_9	write(bal_x)		90
t_{10}	commit/unlock(bal_x)		90

Penyelesaian masalah dengan Locking

- Inconsistent Analysis Problem

Time	T_5	T_6	bal_x	bal_y	bal_z	sum
t_1		begin_transaction	100	50	25	
t_2	begin_transaction	sum = 0	100	50	25	0
t_3	read(bal_x)	read(bal_x)	100	50	25	0
t_4	$bal_x = bal_x - 10$	sum = sum + bal_x	100	50	25	100
t_5	write(bal_x)	read(bal_y)	90	50	25	100
t_6	read(bal_z)	sum = sum + bal_y	90	50	25	150
t_7	$bal_z = bal_z + 10$		90	50	25	150
t_8	write(bal_z)		90	50	35	150
t_9	commit	read(bal_z)	90	50	35	150
t_{10}		sum = sum + bal_z	90	50	35	185
t_{11}		commit	90	50	35	185

Penyelesaian masalah dengan Locking

- Latihan!
- Inconsistent Analysis Problem

Transaksi A	Waktu	Transaksi B
-	↓	-
Baca nilai 1(40) juml = 40	t1	-
-	↓	-
Baca nilai 2 (50) juml = 90	t2	-
-	↓	-
-	t3	Baca nilai 3 (30)
-	↓	-
-	t4	Modifikasi nilai 3 30 → 20
-	↓	-
-	t5	Baca nilai 1 (40)
	↓	
-	t6	Modifikasi nilai 1 40 → 50
-	↓	
-	t7	Commit
-	↓	
Baca nilai 3 (20) juml = 110 (bukan 120)	t8	
-	↓	

Nilai 1 = 40

Nilai 2 = 50

Nilai 3 = 30

Transaksi A menjumlahkan nilai 1, 2 dan 3

Transaksi B nilai3 dikurangi 10 dan nilai1 ditambah 10

Deadlock

- Deadlock merupakan kebuntuan (impasse) yang mungkin dihasilkan ketika dua atau lebih transaksi saling menunggu kunci yang disimpan oleh transaksi lain agar dilepaskan.

Deadlock

Time	T_{17}	T_{18}
t_1	begin_transaction	
t_2	write_lock($\mathbf{bal_x}$)	begin_transaction
t_3	read($\mathbf{bal_x}$)	write_lock($\mathbf{bal_y}$)
t_4	$\mathbf{bal_x} = \mathbf{bal_x} - 10$	read($\mathbf{bal_y}$)
t_5	write($\mathbf{bal_x}$)	$\mathbf{bal_y} = \mathbf{bal_y} + 100$
t_6	write_lock($\mathbf{bal_y}$)	write($\mathbf{bal_y}$)
t_7	WAIT	write_lock($\mathbf{bal_x}$)
t_8	WAIT	WAIT
t_9	WAIT	WAIT
t_{10}	⋮	WAIT
t_{11}	⋮	⋮

- Teknik untuk mengatasi deadlock…