Manajemen Transaksi

Sistem Basis Data

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

Konsep Transaksi

- Transaksi adalah sebuah aksi atau serangkaian aksi, yang dilakukan oleh user atau aplikasi yang mengakses atau mengubah isi dari database.

- Dua hasil transaksi adalah ***commit*** atau ***rollback***.

- Jika transaksi berjalan sukses maka dikatakan *commit*, sebaliknya jika transaksi tidak berjalan sukses maka transaksi dibatalkan dan kembali ke keadaan semula dikatakan *rollback*.

Konsep Transaksi

- DBMS harus dapat menjamin bahwa setiap transaksi harus dapat dikerjakan secara utuh atau tidak sama sekali

- Pengubahan, penambahan, penghapusan suatu tabel berdampak pada isi tabel tersebut. Pada saat eksekusi transaksi terjadi kagagalan, maka terjadi ketidakkonsistenan integrasi antar tabel.

- Namun pada saat transaksi sampai pada level commited, maka databasenya harus konsisten.

Konsep Transaksi

- Untuk memastikan integritas data tetap terjaga dan transaksi dapat berjalan dengan baik, maka database harus menjaga properti yang terdapat dalam transaksi.

- Properti dalam transaksi dikenal dengan istilah properti ACID (*Atomicity, Consistency, Isolation, Durability*)

Konsep Transaksi

- *Atomicity*(keutuhan)

  Transaksi merupakan unit yang tidak terlihat yang harus dilakukan secara keseluruhan atau tidak sama sekali.

- *Consistency* (Ketetapan)

  Transaksi harus membuat database tetap konsisten.

- *Isolation* (Pemisahan)

  Transaksi dieksekusi secara terpisah dari yang satu dengan yang lainnya.

- *Durability* (Daya tahan)

  Secara permanen direkam kedalam database dan tidak akan hilang dikarenakan kegagalan berikutnya.

Konsep Transaksi

- Transaksi mengakses data dengan operasi:

- read(x)

  membaca nilai x dari database (dalam disk) dan menampungnya pada sebuah lokasi (buffer) di memori

- write(x)

  mengambil nilai x dari sebuah lokasi (buffer) di memori kemudian menuliskannya ke database

Contoh Transaksi

- Transaksi transfer uang sebesar $50 dari rekening A ke rekening B, didefinisikan sbg berikut:

1. read (A)
2. A← A – 50
3. write (A)
4. read (B)
5. B←B + 50
6. write (B)

- Atomicity

  jika transaksi gagal di tahap ke 3 atau tahap ke 6, maka perubahan tidak akan disimpan dalam database atau akan terjadi inkonsistensi data. Sehingga selesaikan transaksi atau tidak sama sekali.

- Consistency

  total jumlah rekening A+B harus tetap, tidak berubah setelah proses transaksi.

- Isolation

  jika di antara tahap ke 3 atau tahap ke 6 disisipkan transaksi lain, maka akan terjadi inkonsistensi database.

- Durability

  harus dipastikan bahwa tidak ada kesalahan sistem yang menyebabkan hilangnya data pada proses transfer tersebut.

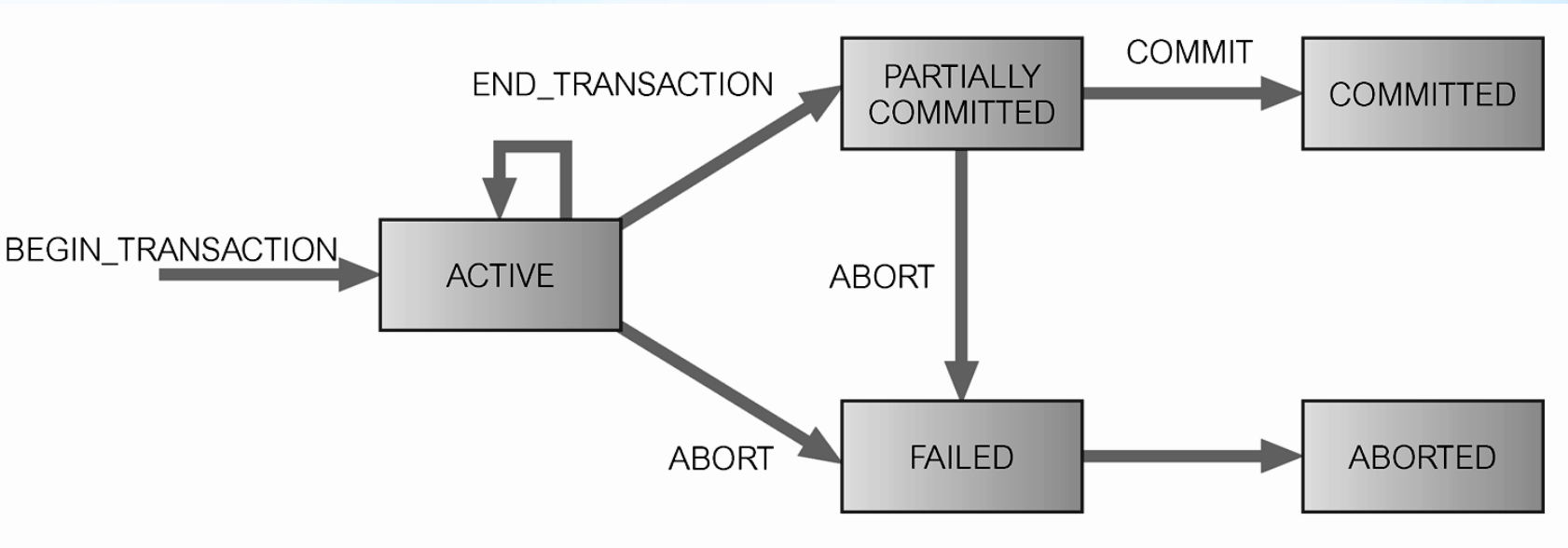
Output Transaksi

- Sukses – transaksi dikatakan *commited* dan database mencapai state berikutnya.

- Gagal – transaksi dikatakan *aborted,* dan database harus dikembalikan ke state sebelum dilakukannya transaksi. Transaksi seperti ini disebut *roll back* atau *undone.*

- Transaksi yang *committed* tidak dapat digagalkan. Transaksi yang digagalkan akan dilakukan *rollback* yang dapat diproses ulang (*restarted*) diwaktu mendatang.

Diagram State Transaksi



State Transaksi

- Transaksi dimulai dalam keadaan state active. Pada saat menyelesaikan statement terakhirnya, transaksi masuk ke kondisi state partially commited.
- Dalam keadaan state tersebut, transaksi telah selesai dieksekusi, tapi masih mungkin untuk dibatalkan (aborted) hanya jika transaksi memasuki state aborted, karena output yang sesungguhnya masih berada di tempat penyimpanan sementara/main memory, dan kesalahan pada hardware dapat mempengaruhi kesuksesan dari penyelesaian transaksi
- Sistem basis data kemudian menuliskan informasi yang dibutuhkan (write) ke dalam disk, bahwa perubahan yang dilakukan oleh transaksi dapat dibuat kembali pada saat sistem restart jika terjadi kegagalan pada sistem
- Pada saat informasi terakhir dituliskan, maka transaksi masuk ke kondisi state commited.

State Transaksi

- Sebuah transaksi akan memasuki kondisi state failed jika setelah sistem melakukan pemeriksaan, transaksi tidak dapat diproses dengan eksekusi normal, misal karena kerusakan hardware atau kesalahan logika
- Jika berada dalam kondisi tersebut, maka transaksi harus di rolled back, dan selanjutnya memasuki kondisi state aborted (pembatalan transaksi)
- Pada titik ini, sistem memiliki dua opsi, ulangi transaksi (restart transaction) dan hapus transaksi (kill the transaction)

State Transaksi

- Opsi pada state aborted:
- Sistem dapat mengulang transaksi (restart the transaction), tapi hanya jika transaksi dibatalkan karena adanya kerusakan software atau hardware yang tidak diciptakan melalui logika internal dari transaksinya.
- Sistem dapat menghapus transaksi (kill the transaction), biasanya terjadi karena adanya kesalahan logika internal yang dapat diperbaiki hanya dengan menulis kembali program aplikasinya, atau karena inputan yang tidak baik, atau karena data yang diinginkan tidak ada di database.

CONCURENCY (KONKURENSI)

Kontrol Konkurensi

Proses untuk menjamin transaksi-transaksi yang dilakukan bersamaan berjalan sukses tanpa adanya tumpang tindih maka dibutuhkan mekanisme pengontrolan konkurensi, jika tidak timbul permasalahan:

- Lost update
- Uncommited dependency
- Inconsistent analysis

Lost Update Problem

- Permasalahan timbul pada saat operasi update berjalan sukses kemudian diubah oleh operasi update lain yang dilakukan oleh user lain.

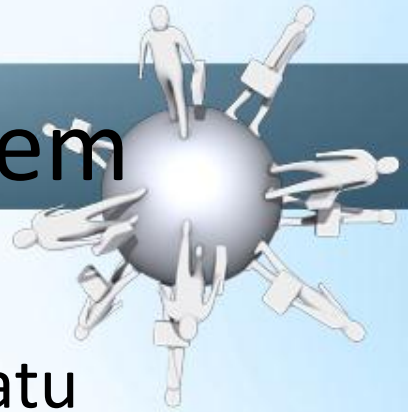
Lost Update Problem

- 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.

Uncommitted Dependency Problem



- Masalah modifikasi sementara terjadi jika satu transaksi (transaksi pertama) membaca hasil dari transaksi lainnya (transaksi kedua) sebelum transaksi kedua dinyatakan *committed,* terdapat kemungkinan jika transaksi tersebut dapat dibatalkan (rollback).

- Biasa dikenal dengan *dirty read problem*.

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

Inconsistent Analysis Problem

- Terjadi ketika transaksi pertama membaca beberapa nilai tetapi transaksi kedua melakukan perubahan terhadap nilai tersebut selama eksekusi transaksi pertama berlangsung.

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
-	↓ t8	
Baca nilai 3 (20) juml = 110 (bukan 120)		
-	↓	

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

Masalah tersebut dapat dihindari dengan mencegah transaksi B membaca nilai 3 sebelum transaksi A selesai (commit)

- Materi selanjutnya:

  Penjadwalan (Schedule)

~Terima Kasih~

☺