Sistem Basis Data

MANAJEMEN TRANSAKSI

Alif Finandhita, S.Kom

MANAJEMEN TRANSAKSI

- Konsep Transaksi
- State Transaksi
- Implementasi Atomik dan Durabilitas
- Eksekusi Konkuren
- Serializability
- Recoverability
- Implementasi Isolasi
- Definisi Transaksi di SQL
- Tes Serializability

Konsep Transaksi

- Transaksi adalah sebuah unit eksekusi dari program yang mengakses dan memungkinkan update berbagai macam tipe data.
- Biasanya suatu transaksi diinisialisasikan oleh program user yang ditulis dalam bahasa pemrograman atau manipulasi data tingkat tinggi (sebagai contoh, SQL, C/C++), yang dibatasi oleh statement (pemanggilan fungsi) dalam bentuk **begin transaction** dan **end transaction**.
- Transaksi terdiri dari semua operasi yang dieksekusi diantara **begin transaction** dan **end transaction**.

Konsep Transaksi (2)

- Pada saat eksekusi transaksi, database bisa saja menjadi tidak konsisten. Namun pada saat transaksi sampai pada level *commited*, maka databasenya harus konsisten.

- Dua hal utama yang mungkin akan dihadapi pada saat melakukan transaksi :
  - Terjadinya berbagai macam kegagalan, yang bisa disebabkan karena kegagalan hardware, system crash, dll
  - Eksekusi konkuren (secara bersama) yang melibatkan banyak transaksi

Konsep Transaksi (3)

- Untuk memastikan integritas data tetap terjaga dan transaksi dapat berjalan dengan baik, maka sistem database harus menjaga properti – properti yang terdapat di dalam transaksi.
- Properti – properti di dalam transaksi ini dikenal dengan istilah Properti ACID (*Atomicity, Consistency, Isolation, Durability).*
- Properti ACID memastikan perilaku yang dapat diprediksi dan menguatkan peran transaksi sebagai konsep *all or nothing* yang didesain untuk mengurangi manajemen *load* ketika ada banyak variabel.

Konsep Transaksi (4)

Properti ACID :

- **Atomicity.** Transaksi dilakukan sekali dan sifatnya atomik, artinya merupakan satu kesatuan tunggal yang tidak dapat dipisah – laksanakan pekerjaannya sampai selesai atau tidak sama sekali
- **Consistency.** Jika basis data awalnya dalam keadaan konsisten maka pelaksanaan transaksi sendirinya juga harus meninggalkan basis data tetap dalam status konsisten
- **Isolation.** Isolasi memastikan bahwa secara bersamaan eksekusi transaksi terisolasi dari yang lain
- **Durability.** Begitu transaksi telah dilaksanakan (di-*commit*) maka perubahan yang dilakukan tidak akan hilang dan tetap terjaga (*durable*), sekalipun ada kegagalan sistem.

Konsep Transaksi (4)

Transaksi mengakses data dengan operasi :

- **read(X)**, mentransfer data item X dari database ke local buffer yang dimiliki oleh transaksi yang mengeksekusi operasi pembacaan (*read*).
- **write(X)**, mentransfer data item X dari local buffer dari aksi transaksi yang mengeksekusi perintah penulisan kembali ke database (*write*)

Konsep Transaksi (5)

- Contoh implementasi transaksi, misalkan transaksi transfer uang sebesar $50 dari rekening A ke rekening B, maka transaksi tersebut dapat didefinisikan sebagai berikut :

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

Konsep Transaksi (6)

Berdasarkan contoh, ditinjau dari kebutuhan properti ACID-nya, maka :

- Kebutuhan Konsistensi (*Consistency Requierements*) :
  Total jumlah rekening A + B harus tetap, tidak berubah setelah proses eksekusi transaksi.

- Kebutuhan Atomik (*Atomicity Requeirements*) :
  Jika transaksi gagal diantara tahap ke-3 dan tahap ke-6, maka sistem harus memastikan bahwa perubahan yang terjadi tidak disimpan ke database, atau akan terjadi inkonsistensi data. Dengan kata lain, selesaikan transaksi atau tidak sama sekali.

Konsep Transaksi (7)

- Kebutuhan Durabilitas *(Durability)* :
  Pada saat eksekusi transaksi selesai dilaksanakan, dan user yang melakukan transaksi sudah diberitahu bahwa transfer yang dilakukannya sukses, maka harus dipastikan bahwa tidak ada kesalahan sistem yang akan terjadi yang menyebabkan hilangnya data yang berkaitan dengan proses transfer tersebut .

- Kebutuhan Isolasi *(Isolation)* :
  Jika diantara tahap ke-3 dan tahap ke-6 ada transaksi lain yang disisipkan, maka akan dapat menyebabkan inkonsistensi terhadap database (jumlah rekening A+B bisa jadi berkurang dari yang seharusnya). Untuk menghindari hal itu, maka transaksi bisa dieksekusi secara serial.
  Tapi walaubagaimanapun, eksekusi banyak transaksi secara bersama – sama (konkuren) memiliki banyak keuntungan.

State Transaksi

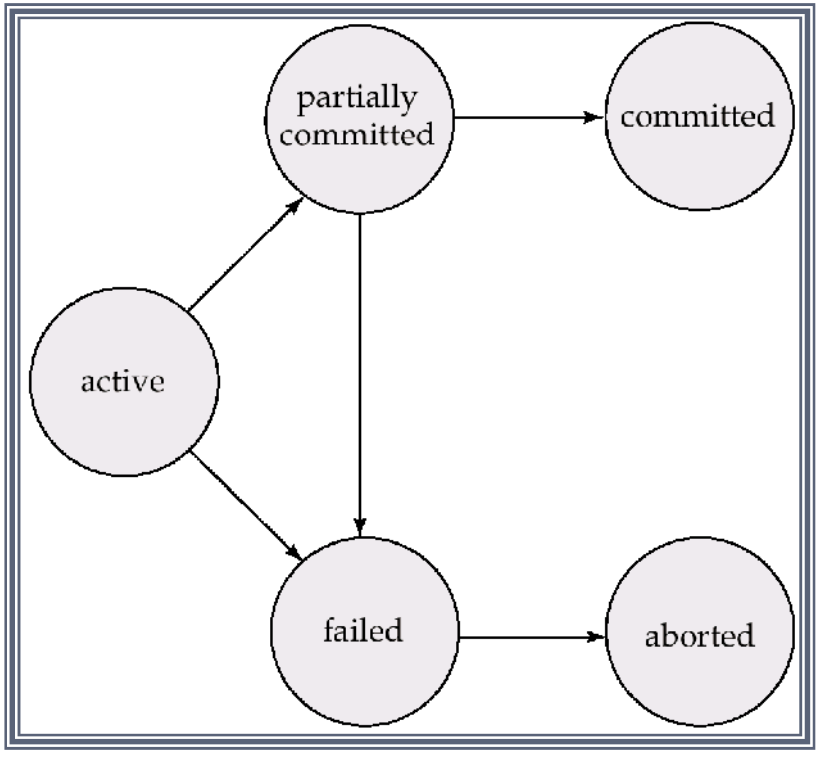
- Supaya transaksi benar – benar sukses dipenuhi (*successfully completion*), maka sebuah transaksi harus berada di dalam salah satu state sebagai berikut :
  - **Active**
  - **Partially committed**
  - **Failed**
  - **Aborted**
  - **Commited**
- State tersebut dapat direpresentasikan dalam model transaksi abstrak yang sederhana

State Transaksi (2)

- **Active**
  merupakan initial state, transaksi tetap berada pada state ini pada saat proses eksekusi
- **Partially Committed**
  Setelah statement final telah dieksekusi
- **Failed**
  Setelah ditelusuri bahwa eksekusi normal tidak dapat diproses kembali
- **Aborted**
  Setelah transaksi di-rolled back dan database direstore ke kondisi awal sebelum transaksi dimulai
- **Committed**
  Setelah transaksi sukses dipenuhi

State Transaksi (3)

- Diagram state yang menggambarkan proses transaksi :



State Transaksi (4)

- 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 karenanya 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 (5)

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

State Transaksi (6)

- Opsi pada state *aborted* :
  - Sistem dapat mengulang transaksi (**restart** the transaction), tapi hanya jika transaksi dibatalkan yang bisa terjadi 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 kesalah 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.

Implementasi Atomik *(Atomicity)* dan Durabilitas *(Durability)*

- Komponen manajemen recovery dari sistem basis data dapat mendukung atomisitas dan durabilitas dengan berbagai macam skema.
- Salah satu skema yang dikenal adalah shadow copy,
- *Shadow copy* adalah salah satu skema yang digunakan untuk mendukung atomisitas dan durabilitas pada transaksi dengan membuat salinan / copy dari database yang ada.

Implementasi Atomik *(Atomicity)* dan Durabilitas *(Durability)* – (2)

Skema *shadow-database :*

- Mengasumsikan bahwa hanya satu transaksi yang aktif dalam waktu bersamaan (simpel tapi tidak efisien).
- Mengasumsikan bahwa database secara sederhana merupakan sebuah file di dalam disk.
- Sebuah pointer yang bernama *db-pointer* digunakan di dalam disk yang selalu mengarah ke salinan konsisten database tersebut
- Sebelum transaksi mengupdate database, salinan untuk database tersebut dibuat terlebih dahulu sepenuhnya

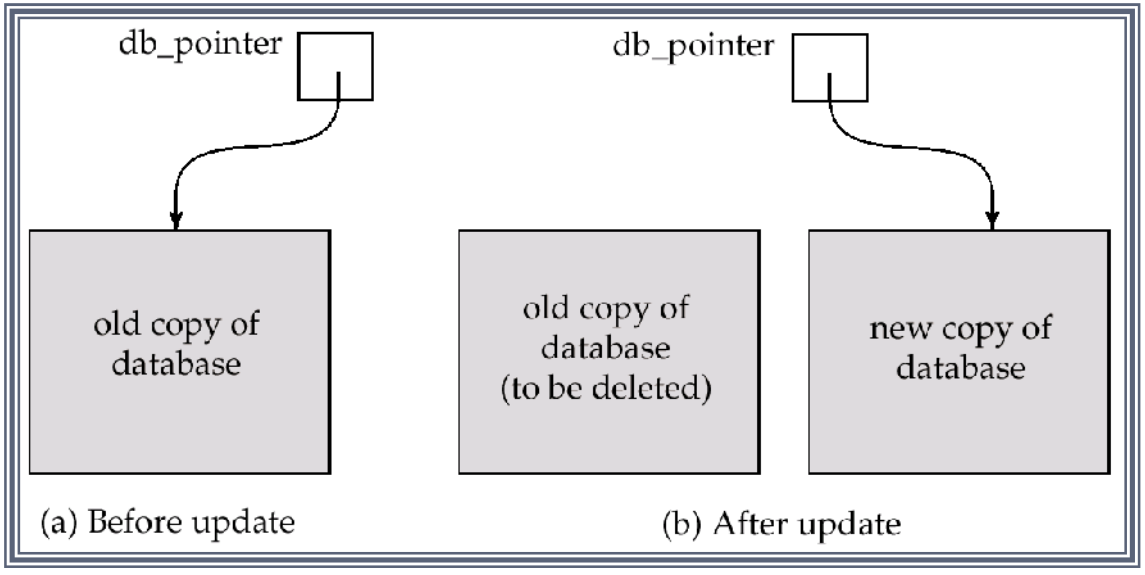
Implementasi Atomik *(Atomicity)* dan Durabilitas *(Durability)* – (3)

Skema *shadow-database :*

- Semua update dilakukan di salinan database yang baru, dan *db_pointer* akan mengarah ke salinan baru tersebut dengan catatan semua transaksi telah mencapai state *partial committed* dan semua update pagesnya telah di-*flush* ke dalam disk.
- Salinan database tersebut kemudian menjadi database utama, dan database yang lama dapat dihapus.
- Jika transaksi gagal, maka *db_pointer* akan kembali mengarah ke database lama, dan salinan dari database yang telah dibuat tersebut dapat dihapus.

Implementasi Atomik *(Atomicity)* dan Durabilitas *(Durability)* – (4)

Skema *shadow-database :*



- Asumsi transaksi tidak gagal
- Berguna untuk teks editor, tapi tidak efisien untuk database berukuran besar