Database Recovery

Pertemuan 9

Pokok Bahasan/Materi :

- Database Recovery
- Transaction Recovery
- Commit Protocol
- Shadow Paging
- Recovery With Concurrent Transactions
- Buffer Management

Database Recovery

- ***Database Recovery*** adalah proses memulihkan basis data ke kondisi konsisten terbaru yang ada sebelum kegagalan.
- Pemulihan data penting dalam mengelola sistem basis data.
- Memiliki rencana yang baik akan membantu membuat pemulihan yang cepat dan sukses dari kegagalan.

Database Recovery (Cont.)

1. **Purpose of Database Recovery**
   - Untuk membawa database ke kondisi konsisten terakhir, yang ada sebelum kegagalan.
   - Untuk mempertahankan properti transaksi (Atomicity, Consistency, Isolation and Durability).
   - **Example**:
     - Jika sistem macet sebelum transaksi transfer dana menyelesaikan eksekusi, maka salah satu atau kedua akun mungkin memiliki nilai yang salah. Dengan demikian, basis data harus dikembalikan ke keadaan sebelum transaksi mengubah salah satu akun.

Database Recovery (Cont.)

2. Types of Failure

- Basis data mungkin menjadi tidak tersedia untuk digunakan karena
  - **Transaction failure**: Transaksi mungkin gagal karena input yang salah, deadlock, sinkronisasi salah.
  - **System failure**: Sistem mungkin gagal karena mengatasi kesalahan, kesalahan aplikasi, kesalahan sistem operasi, kegagalan RAM, dll..
  - **Media failure**: Disk head crash,
  - Gangguan daya, dll.

Recovery Strategies

- Jenis update Transaksi
  - *Deferred Update*
    - Daftar semua perubahan dalam urutan komitmennya
    - 'Recovery and Roll Forward'
  - Shadow Update
    - Daftar semua perubahan dalam urutan terbalik
    - 'Roll Back'
    - Seringkali lebih efisien daripada pemulihan dan terus maju.

Deadlock

- Deadlock adalah keadaan dimana dua program memegang kontrol terhadap sumber daya yang dibutuhkan oleh program yang lain. Tidak ada yang dapat melanjutkan proses masing-masing sampai program yang lain memberikan sumber dayanya, tetapi tidak ada yang mengalah.
- Situasi deadlock dapat muncul jika semua kondisi berikut ini bertahan secara bersamaan dalam suatu sistem:
  1. ***Mutual Exclusion***
  2. ***Hol and Wait***
  3. ***No Preemption***
  4. ***Circular Wait***

1. ***Mutual Exclusion*** yaitu proses memiliki hak milik pribadi terhadap sumber daya yang sedang digunakannya. Jadi, hanya ada satu proses yang menggunakan suatu sumber daya.
2. ***Hol and Wait*** yaitu beberapa proses saling menunggu sambil menahan sumber daya yang dimilikinya. Suatu proses yang memiliki minimal satu buah sumber daya melakukan *request* lagi terhadap sumber daya.
3. ***No Preemption*** yaitu sebuah sumber daya hanya dapat dilepaskan oleh proses yang memilikinya secara sukarela setelah ia selesai menggunakannya.
4. ***Circular Wait*** yaitu kondisi membentuk siklus yang berisi proses-proses yang saling membutuhkan.

Penangan Deadlock & Recovery

Sebagian besar sistem operasi saat ini tidak dapat mencegah kebuntuan terjadi. Ketika kebuntuan terjadi, sistem operasi yang berbeda meresponsnya dengan cara yang berbeda. Pendekatan utama adalah sebagai berikut :

1. **Pengabaian.** Maksud dari pengabaian di sini adalah sistem mengabaikan terjadinya deadlock dan pura-pura tidak tahu kalau deadlock terjadi.
2. **Pencegahan.** Penanganan ini dengan cara mencegah terjadinya salah satu karakteristik deadlock. Penanganan ini dilaksanakan pada saat deadlock belum terjadi pada sistem.
3. **Penghindaran.** Menghindari keadaan deadlock. Bagian yang perlu diperhatikan adalah bahwa antara pencegahan dan penghindaran adalah dua hal yang berbeda.
4. **Pendeteksian dan Pemulihan**. Pada sistem yang sedang berada pada kondisi deadlock, tindakan yang harus diambil adalah tindakan yang bersifat represif. Tindakan tersebut adalah dengan mendeteksi adanya deadlock, kemudian memulihkan kembali sistem. Proses pendeteksian akan menghasilkan informasi apakah sistem sedang deadlock atau tidak serta proses mana yang mengalami deadlock.

Transaction Recovery

- Transaksi adalah tindakan, atau serangkaian tindakan, yang dilakukan oleh satu pengguna atau program aplikasi, yang membaca atau memperbarui konten basis data.
- Transaksi adalah 'unit kerja logis' pada basis data
  - Setiap transaksi melakukan sesuatu dalam database
- Transaksi adalah unit recovery, consistency, dan integrity
- ACID properties
  - Atomicity
  - Consistency
  - Isolation
  - Durability

Atomicity and Consistency

- Atomicity
  - Transaksi bersifat atomik - mereka tidak memiliki bagian (secara konseptual)
  - Dapat dieksekusi sebagian; tidak dapat dideteksi bahwa mereka berinteraksi dengan transaksi lain

- Konsistensi
  - Transaksi mengambil basis data dari satu kondisi yang konsisten ke kondisi lainnya
  - Di tengah transaksi database mungkin tidak konsisten

Isolation and Durability

- Isolasi
  - Efek dari suatu transaksi tidak terlihat oleh transaksi lain sampai selesai
  - Dari luar transaksi telah terjadi atau tidak
  - Bagi saya ini sebenarnya terdengar seperti konsekuensi dari atomisitas ...

- Durability
  - Setelah transaksi selesai, perubahannya dibuat permanen
  - Bahkan jika sistem crash, efek dari suatu transaksi harus tetap di tempatnya

Example of transaction

- Transfer $50 from account A to account B

```
Read(A)
A = A - 50
Write(A)
Read(B)
B = B+50
Write(B)
```

- Atomicity - tidak boleh mengambil uang dari A tanpa memberikannya kepada B
- Konsistensi - uang tidak hilang atau diperoleh
- Isolasi - pertanyaan lain tidak boleh melihat A atau B berubah sampai selesai
- Durability - uang tidak kembali ke A.

The Transaction Manager

- Mengelola transaksi memberlakukan properti ACID
  - Ini menjadwalkan operasi transaksi
  - COMMIT dan ROLLBACK digunakan untuk memastikan atomicity

- Locks atau timestamps digunakan untuk memastikan konsistensi dan isolasi untuk transaksi bersamaan (kuliah berikutnya).
- Sebuah log disimpan untuk memastikan daya tahan jika terjadi kegagalan system.

COMMIT and ROLLBACK

- COMMIT menandai keberhasilan akhir suatu transaksi
  - Setiap perubahan yang dilakukan oleh transaksi harus disimpan
  - Perubahan ini sekarang terlihat oleh transaksi lainnya

- ROLLBACK menandai akhir transaksi yang tidak berhasil
  - Setiap perubahan yang dilakukan oleh transaksi harus dibatalkan
  - Sekarang seolah-olah transaksi tidak pernah ada

Recovery

- Transaksi harus bandel, tetapi tidak dapat mencegah segala macam kegagalan:
  - Sistem macet
  - Listrik padam
  - Disk mogok
  - Kesalahan pengguna
  - Sabotase
  - Bencana alam

- Mencegah lebih baik daripada mengobati
  - Reliable OS
  - Security
  - UPS and surge protectors
  - RAID arrays
- Tidak dapat melindungi dari segalanya

Checkpointing

- Waktu ke waktu (secara acak atau berdasarkan beberapa kriteria) database mem-flush buffer-nya ke disk basis data untuk meminimalkan tugas pemulihan. Langkah-langkah berikut menentukan operasi checkpoint :
  1. Tangguhkan eksekusi transaksi sementara.
  2. Paksa menulis data buffer yang dimodifikasi ke disk.
  3. Tulis catatan [checkpoint] ke log, simpan log ke disk.
  4. Melanjutkan eksekusi transaksi normal.
- Diperlukan pemulihan atau pembatalan pemulihan untuk transaksi yang muncul setelah [checkpoint] dicatat.

The Transaction Log

- Log transaksi mencatat rincian semua transaksi
  - Setiap perubahan transaksi dilakukan ke database
  - Cara membatalkan perubahan ini
  - Kapan transaksi selesai dan bagaimana

- Log disimpan di disk, bukan di memori
  - Jika sistem macet itu dipertahankan
  - Tulis aturan log sebelumnya
  - Entri dalam log harus dibuat sebelum pemrosesan COMMIT dapat diselesaikan

System Failures

- Kegagalan sistem berarti semua transaksi yang berjalan terpengaruh
  - Perangkat lunak macet
  - Gangguan daya
- Media fisik (disk) tidak rusak

- Pada beberapa waktu, DBMS mengambil checkpoint
  - Semua transaksi yang dilakukan ditulis ke disk
  - Catatan dibuat (pada disk) dari transaksi yang sedang berjalan

System Recovery

- Setiap transaksi yang berjalan pada saat kegagalan harus dibatalkan dan dimulai kembali
- Setiap transaksi yang dilakukan sejak checkpoint terakhir harus diulang

- Transaksi tipe T_1 tidak perlu pemulihan
- Transaksi tipe T_3 atau T_5 harus dibatalkan dan dimulai kembali
- Transaksi tipe T_2 atau T_4 perlu diulang

Transaction Recovery

UNDO and REDO: lists of transactions

UNDO = all transactions running at the last checkpoint

REDO = empty

For each entry in the log, starting at the last checkpoint

If a BEGIN TRANSACTION entry is found for T

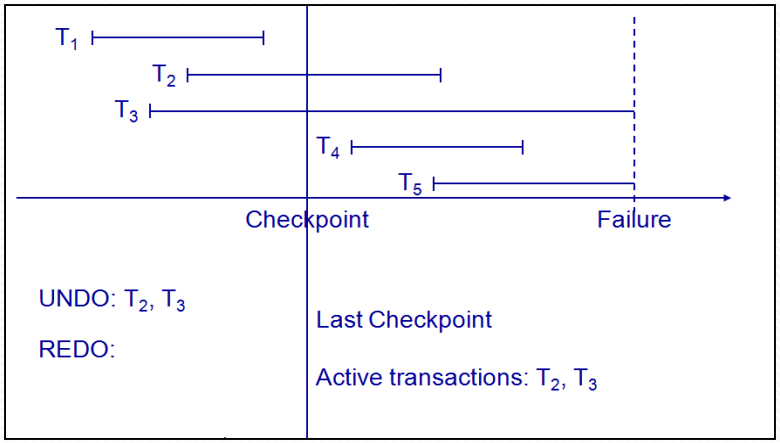
Add T to UNDO

If a COMMIT entry is found for T

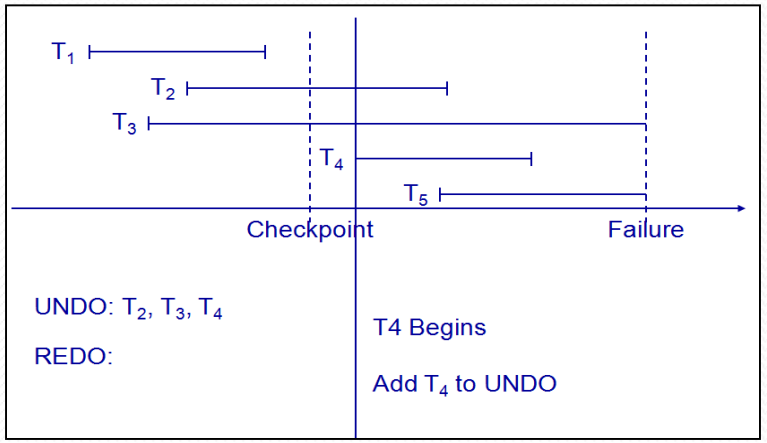
Move T from UNDO to REDO

Transaction Recovery (Cont.)

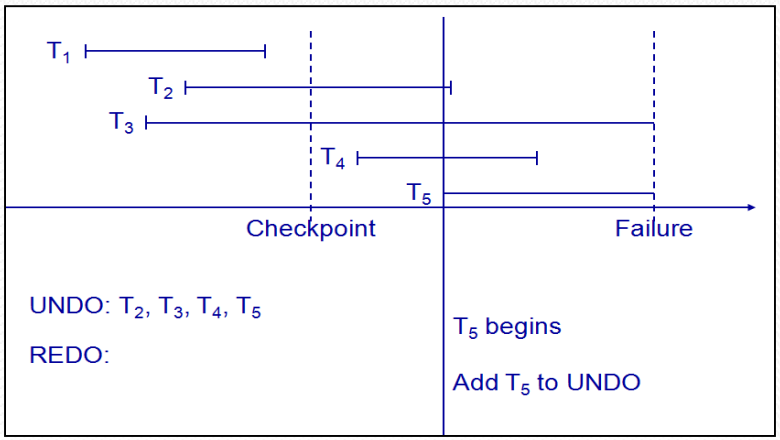
(1)



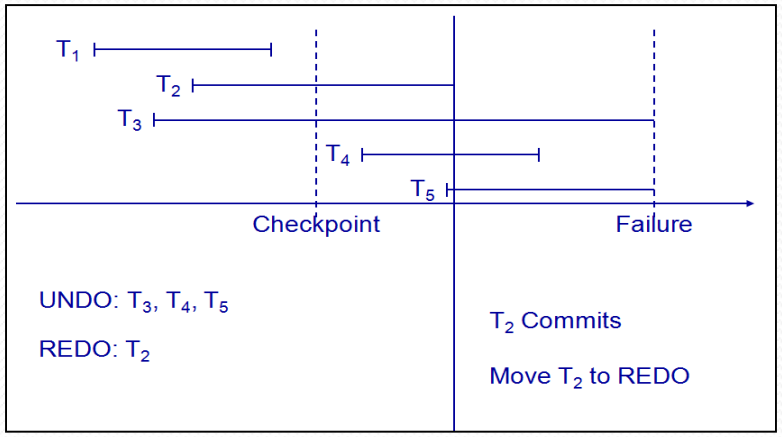
(2)



(3)

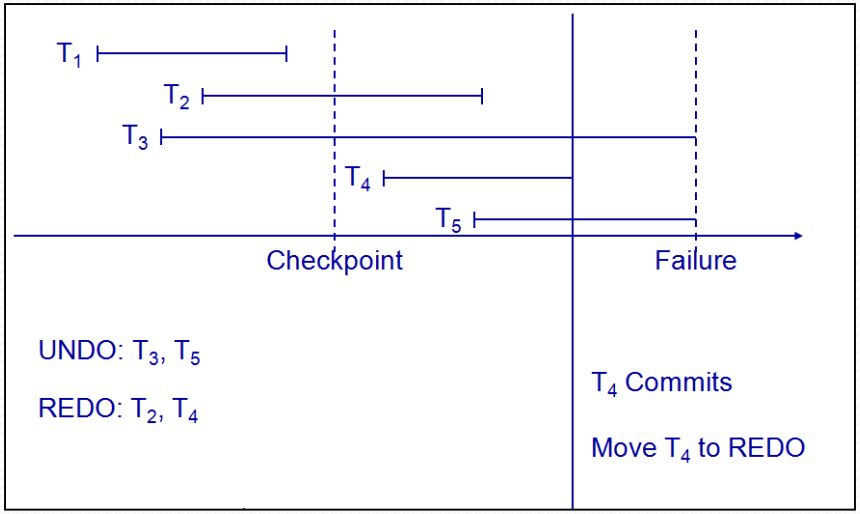


(4)



Transaction Recovery (Cont.)

(5)



Forwards and Backwards

- Backwards recovery
  - Perlu membatalkan beberapa transaksi
  - Bekerja mundur melalui log, membatalkan operasi apa pun oleh transaksi pada daftar UNDO
  - Ini mengembalikan database ke kondisi yang konsisten
- Forwards recovery
  - Beberapa transaksi perlu diperbaiki
  - Bekerja ke depan melalui log, mengulangi operasi apa pun dengan transaksi pada daftar REDO
  - Ini membuat database menjadi terbaru

Media Failures

- Kegagalan sistem tidak terlalu parah
  - Hanya informasi sejak checkpoint terakhir yang terpengaruh
  - Ini dapat dipulihkan dari log transaksi

- Kegagalan media (kerusakan disk, dll.) Lebih serius
  - Data yang disimpan ke disk rusak
  - Log transaksi itu sendiri mungkin rusak

Backups

- Backups diperlukan untuk memulihkan dari kegagalan media
  - Log transaksi dan seluruh isi basis data ditulis ke penyimpanan sekunder (sering direkam)
  - Memakan waktu, dan seringkali membutuhkan waktu henti
- Frekuensi Backup
  - Cukup sering sehingga sedikit informasi yang hilang
  - Tidak terlalu sering menyebabkan masalah
  - Setiap hari (malam) adalah hal biasa
- Backup storage

Recovery from Media Failure

- Kembalikan database dari cadangan terakhir
- Gunakan log transaksi untuk mengulangi setiap perubahan yang dibuat sejak backup terakhir

- Jika log transaksi rusak, Anda tidak dapat melakukan langkah 2
  - Simpan log pada perangkat fisik terpisah ke database
  - Risiko kehilangan keduanya kemudian dikurangi

Commit Protocol

- Commit Protocol
  - Memastikan integritas data untuk operasi pembaruan terdistribusi, manajer transaksi yang bekerja sama melaksanakan Commit Protocol
  - Memastikan bahwa transaksi global berhasil diselesaikan di semua situs atau dibatalkan
- Two-phase Commit (an algorithm)
  - Protokol yang paling banyak digunakan
  - Mengkoordinasikan pembaruan dalam lingkungan terdistribusi
  - Memastikan bahwa transaksi bersamaan di beberapa situs diproses seolah-olah mereka dieksekusi dalam urutan serial yang sama di semua situs
- Bagaimana cara kerjanya?

How does the protocol work?(1)

- Situs yang berasal dari transaksi global mengirimkan permintaan ke setiap situs yang akan memproses sebagian dari transaksi.
- Setiap situs memproses sub-transaksi tetapi tidak memasukkan hasilnya ke basis data (disimpan dalam file sementara).
- Setiap situs mengunci bagian basis datanya yang diperbarui.
- Setiap situs memberi tahu situs asal tentang penyelesaian sub-transaksi.
- Ketika semua situs merespons, situs asal memulai commit protocol dua fase
  - Preparation Phase
  - Final Commit Phase

How does the protocol work?(2)

- **Preparation Phase**
  - Pesan disiarkan ke setiap situs yang berpartisipasi, menanyakan apakah ia bersedia melakukan bagian transaksinya di situs itu
  - Setiap situs mengembalikan "OK" atau "tidak OK"
  - Situs yang berasal mengumpulkan semua pesan.

How does the protocol work?(3)

- **Final Commit Phase**
  - Jika semua "OK", itu menyiarkan pesan ke semua situs untuk melakukan bagian dari transaksi yang ditangani di setiap situs
  - Jika satu atau lebih situs merespons dengan “tidak baik-baik saja”, ia menyiarkan pesan ke semua situs untuk membatalkan transaksi
  - Transaksi dapat gagal selama commit phase
  - Transaksi semacam itu akan berada dalam limbo

How does the protocol work?(4)

- Transaksi limbo dapat diidentifikasi dengan polling atau dengan timeout. Dengan batas waktu (tidak ada konfirmasi komit untuk jangka waktu tertentu) tidak mungkin membedakan antara situs yang gagal atau sibuk.
- Polling panjang dalam hal memuat jaringan dan waktu pemrosesan
- Jika konfirmasi komit tidak diterima dari satu atau lebih situs, situs asal memaksa semua situs untuk membatalkan perubahan dengan membatalkan pesan

Strategi Peningkatan untuk *Two-Phase Commit*

- ***Two-phase commit protocol*** lambat karena keterlambatan yang disebabkan oleh koordinasi yang luas antar lokasi. Beberapa perbaikan yang dikembangkan adalah:
  - Read-only optimization

    Pendekatan ini mengidentifikasi bagian read-only dari suatu transaksi dan menghilangkan kebutuhan pesan konfirmasi pada mereka, misalnya, transaksi dapat mencakup pembacaan data sebelum memasukkan data baru. (cek saldo inventaris sebelum membuat pesanan baru, sehingga pembacaan data dapat terjadi tanpa konfirmasi panggilan balik)
  - Lazy commit optimization
  - Pendekatan ini memungkinkan situs-situs yang dapat memperbarui untuk melanjutkan untuk memperbarui dan yang tidak dapat memperbarui diizinkan untuk "mengejar ketinggalan" nanti
  - Linear commit optimization

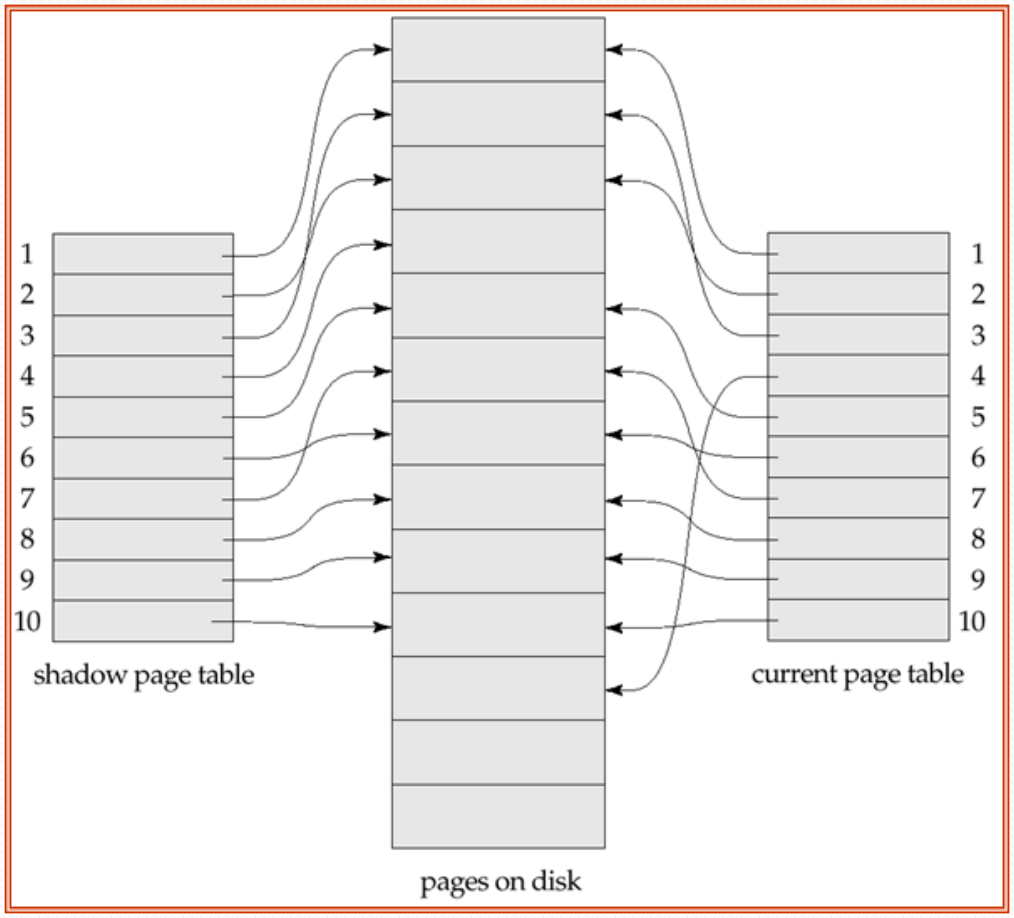
    Pendekatan ini memungkinkan setiap bagian dari transaksi (sub-transaksi) untuk dilakukan secara berurutan daripada menahan seluruh transaksi ketika bagian-bagian sub-transaksi tertunda untuk diproses

Shadow Paging

- ***Shadow paging*** adalah alternatif untuk pemulihan berbasis log; Skema ini berguna jika transaksi dijalankan secara serial
- Idea: memelihara dua tabel halaman selama masa transaksi - tabel halaman saat ini (**the current page table**), dan tabel halaman bayangan (**shadow page table**).
- Simpan tabel shadow page di penyimpanan non-volatil, sedemikian rupa sehingga keadaan database sebelum eksekusi transaksi dapat dipulihkan.
  - Shadow page table tidak pernah dimodifikasi selama eksekusi
- Untuk memulainya, kedua tabel halaman identik. Hanya tabel halaman saat ini digunakan untuk mengakses item data selama pelaksanaan transaksi.
- Setiap kali ada halaman yang akan ditulis untuk pertama kalinya
  - Salinan halaman ini dibuat ke halaman yang tidak digunakan.
  - Tabel halaman saat ini kemudian dibuat untuk menunjuk ke salinan
  - Pembaruan dilakukan pada salinan

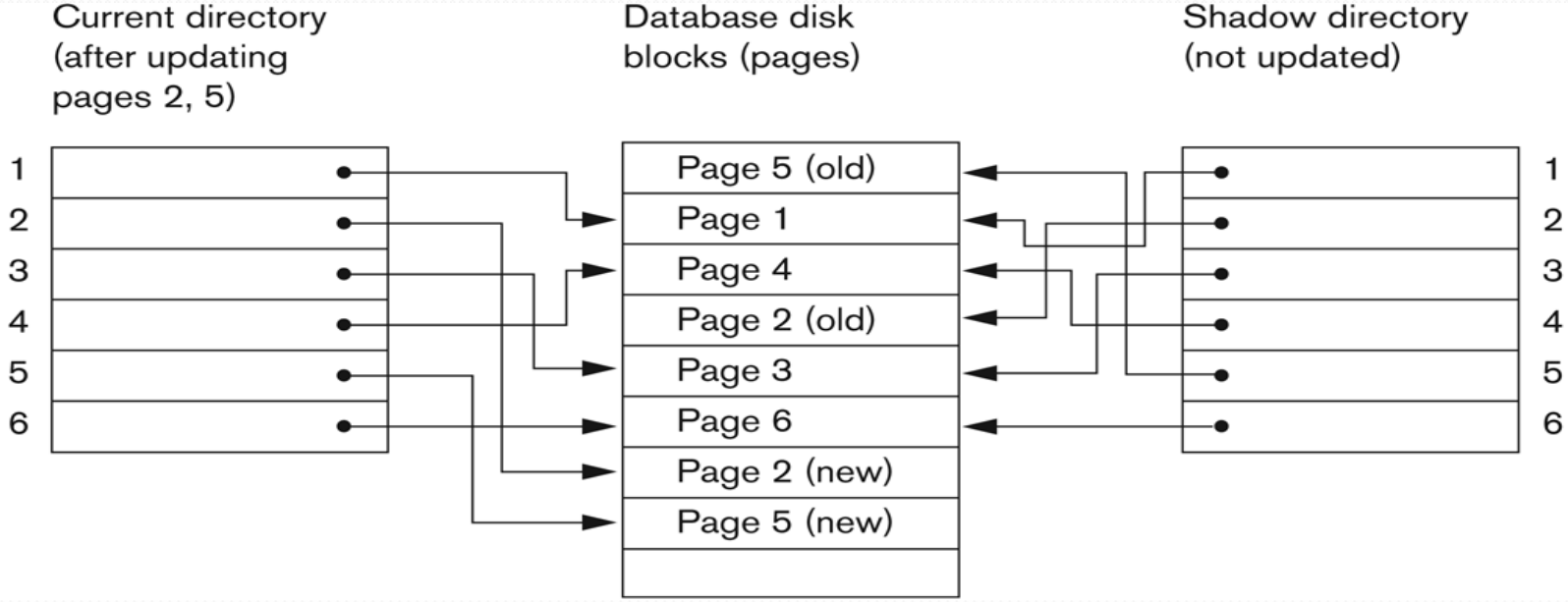
Shadow Paging (Cont.)

- **Example of Shadow Paging**
  - Shadow and current page tables after write to page 4



Shadow Paging (Cont.)

- Untuk mengelola akses item data dengan transaksi bersamaan dua direktori (*current and shadow*) digunakan.
  - Pengaturan direktori diilustrasikan di bawah ini. Di sini halaman adalah item data.



Shadow Paging (Cont.)

- To commit a transaction :
  1. Memindahkan semua halaman yang dimodifikasi dalam memori utama ke disk
  2. Keluarkan tabel halaman saat ini ke disk
  3. Jadikan tabel halaman saat ini sebagai tabel halaman bayangan baru, sebagai berikut:
     - Menyimpan pointer ke tabel halaman bayangan di lokasi tetap (dikenal) pada disk.
     - Untuk menjadikan tabel halaman saat ini sebagai tabel halaman bayangan baru, cukup perbarui pointer untuk menunjuk ke tabel halaman saat ini pada disk
- Setelah tabel pointer ke shadow page ditulis, transaksi dilakukan.
- Tidak diperlukan pemulihan setelah terjadi crash - transaksi baru dapat segera dimulai, menggunakan tabel halaman bayangan.
- Halaman yang tidak menunjuk dari tabel halaman saat ini / shadow harus dibebaskan (garbage collected /sampah dikumpulkan).

Shadow Paging (Cont.)

- **Keuntungan** *shadow-paging* dibandingkan skema berbasis log
  - Tidak ada overhead untuk menulis catatan log
  - Pemulihan itu mudah
- **Kerugian**:
  - Menyalin seluruh tabel halaman sangat panjang
    - Dapat dikurangi dengan menggunakan tabel halaman yang terstruktur seperti B + -tree
      - Tidak perlu menyalin seluruh pohon, hanya perlu menyalin jalur di pohon yang mengarah ke simpul daun yang diperbarui
  - Commit overhead tinggi bahkan dengan ekstensi di atas
    - Perlu memindahkan setiap halaman yang diperbarui, dan tabel halaman
  - Data terfragmentasi (halaman terkait terpisah pada disk)
  - Setelah setiap penyelesaian transaksi, halaman database yang berisi versi lama dari data yang dimodifikasi perlu dikumpulkan sebagai sampah
  - Sulit memperluas algoritma untuk memungkinkan transaksi berjalan secara bersamaan
    - Lebih mudah untuk memperluas skema berbasis log

Recovery With Concurrent Transactions

- Memodifikasi skema recovery berbasis log untuk memungkinkan beberapa transaksi dijalankan secara bersamaan..
  - Semua transaksi berbagi buffer disk tunggal dan satu log
  - Blok penyangga dapat memperbarui item data dengan satu transaksi atau lebih.
- Menganggap kontrol konkurensi menggunakan penguncian dua fase yang ketat;
  - mis. Pembaruan dari transaksi yang tidak di Commit tidak boleh terlihat oleh transaksi lain
    - Kalau tidak, bagaimana melakukan undo jika T_1 memperbarui A, lalu T_2 memperbarui A dan melakukan, dan akhirnya T_1 harus membatalkan?
  - Pembalakan/ Logging dilakukan seperti yang dijelaskan sebelumnya.
    - Catatan log dari berbagai transaksi dapat diselingi dalam log.
  - Teknik *checkpointing* dan tindakan yang diambil untuk pemulihan harus diubah
    - Karena beberapa transaksi mungkin aktif ketika checkpointing dilakukan.

Recovery With Concurrent Transactions (Cont.)

- Checkpoints dilakukan seperti sebelumnya, kecuali bahwa catatan log checkpoint sekarang dalam bentuk

  < **checkpoint** *L*>

  di mana L adalah daftar transaksi yang aktif pada saat checkpoint.
  - Menganggap tidak ada pembaruan yang sedang berlangsung saat checkpoint dilakukan (akan rileks ini nanti)
- Ketika sistem pulih dari kerusakan, pertama-tama sistem melakukan hal berikut:
  1. Inisialisasi undo-list dan redo-list menjadi kosong
  2. Scan log backwards dari ujung, berhenti ketika catatan <checkpoint L> pertama ditemukan.

     Untuk setiap catatan yang ditemukan selama *backward scan*:
     - if the record is <T_i **commit**>, add T_i to *redo-list*
     - if the record is <T_i **start**>, then if T_i is not in *redo-list*, add T_i to *undo-list*
  3. Untuk setiap Ti dalam L, jika Ti tidak ada dalam redo-list, tambahkan Ti ke undo-list

Recovery With Concurrent Transactions (Cont.)

- Pada titik ini, undo-list terdiri dari transaksi tidak lengkap yang harus dibatalkan, dan redo-list terdiri dari transaksi selesai yang harus diulang.
- Pemulihan sekarang berlanjut sebagai berikut:
  1. Scan log forwards dari sebagian besar catatan terbaru, berhenti ketika <Ti mulai> catatan telah ditemukan untuk setiap Ti di undo-list.
     - Selama scan, lakukan undo untuk setiap catatan log milik transaksi dalam undo-list.
  2. Temukan catatan <checkpoint L> terbaru.
  3. Scan ke depan log dari catatan <checkpoint L> hingga akhir log.
     - Selama Scan, lakukan pengulangan untuk setiap catatan log yang menjadi milik transaksi pada daftar ulang.

Example of Recovery

- Ikuti langkah-langkah algoritma pemulihan pada log berikut:

  <T_0 **start**>
  <T_0, A, 0, 10>
  <T_0 **commit**>
  <T_1 **start**> /* Scan at step 1 comes up to here */
  <T_1, B, 0, 10>
  <T_2 **start**>
  <T_2, C, 0, 10>
  <T_2, C, 10, 20>
  <checkpoint {T_1, T_2}>
  <T_3 **start**>
  <T_3, A, 10, 20>
  <T_3, D, 0, 10>
  <T_3 **commit**>

Log Record Buffering

- **Log record buffering** : catatan log disangga dalam memori utama, bukannya output langsung ke penyimpanan stabil.
  - Catatan log adalah output ke penyimpanan stabil ketika blok catatan log dalam buffer penuh, atau operasi **kekuatan log** (*Log force*) dieksekusi.
- Log force dilakukan untuk melakukan transaksi dengan memaksa semua catatan lognya (termasuk catatan commit) ke penyimpanan yang stabil.
- Dengan demikian, beberapa catatan log dapat berupa output menggunakan operasi keluaran tunggal, mengurangi cost I / O.

Database Buffering

- Basis data memelihara penyangga di dalam memori dari blok data
  - Ketika sebuah blok baru dibutuhkan, jika buffer penuh blok yang ada perlu dihapus dari buffer
  - Jika blok yang dipilih untuk dihapus telah diperbarui, itu harus menjadi output ke disk
- Sebagai akibat dari aturan write-ahead logging , jika blok dengan pembaruan yang tidak di Commit adalah output ke disk, catatan log dengan informasi undo untuk pembaruan adalah output ke log pada penyimpanan stabil terlebih dahulu.
- Tidak ada pembaruan yang sedang berlangsung di blok ketika itu output ke disk. Dapat dipastikan sebagai berikut..
  - Sebelum menulis item data, transaksi memperoleh kunci eksklusif di blok yang berisi item data.
  - Kunci dapat dilepaskan setelah penulisan selesai.
    - Kunci yang ditahan untuk jangka waktu pendek disebut kait (**latches**).
  - Sebelum blok adalah output ke disk, sistem memperoleh kait eksklusif pada blok
    - Pastikan tidak ada pembaruan yang sedang berlangsung di blok

Buffer Management (Cont.)

- Database buffer dapat diimplementasikan juga
  - di area memori utama nyata yang disediakan untuk database, atau
  - dalam memori virtual
- Menerapkan buffer dalam memori utama yang dicadangkan memiliki kelemahan:
  - Memori dipartisi sebelumnya antara buffer basis data dan aplikasi, membatasi fleksibilitas.
  - Kebutuhan dapat berubah, dan meskipun sistem operasi paling tahu bagaimana memori harus dibagi kapan saja, itu tidak dapat mengubah partisi memori..

Buffer Management (Cont.)

- Buffer database umumnya diimplementasikan dalam memori virtual meskipun ada beberapa kelemahan:
  - Ketika sistem operasi perlu memindahkan halaman yang telah dimodifikasi, untuk membuat ruang untuk halaman lain, halaman tersebut ditulis untuk menukar ruang pada disk.
  - Ketika database memutuskan untuk menulis halaman buffer ke disk, halaman buffer mungkin dalam ruang swap, dan mungkin harus dibaca dari ruang swap pada disk dan output ke database pada disk, sehingga menghasilkan tambahan I / O!
    - Dikenal sebagai masalah paging ganda (***dual paging***).
  - Idealnya saat mengganti halaman buffer basis data, sistem operasi harus memberikan kontrol ke basis data, yang pada gilirannya mengeluarkan halaman ke basis data alih-alih untuk menukar ruang (pastikan untuk mencetak catatan log terlebih dahulu)
    - Dengan demikian, ***dual paging*** dapat dihindari, tetapi sistem operasi umum tidak mendukung fungsi tersebut..

Failure with Loss of Nonvolatile Storage

- Sejauh ini mengasumsikan tidak ada kerugian penyimpanan non-volatile
- Teknik mirip dengan checkpoint yang digunakan untuk menangani kehilangan penyimpanan non-volatile
  - Secara berkala buang seluruh konten database ke penyimpanan yang stabil
  - Tidak ada transaksi yang aktif selama prosedur dump; prosedur yang mirip dengan checkpoint harus dilakukan
  - Secara berkala buang (dump) seluruh konten database ke penyimpanan yang stabil
  - Tidak ada transaksi yang aktif selama prosedur dump; prosedur yang mirip dengan checkpoint harus dilakukan.
    - Keluarkan semua catatan log yang saat ini berada di memori utama ke penyimpanan yang stabil.
    - Keluarkan semua blok buffer ke disk.
    - Salin konten database ke penyimpanan stabil.
    - Keluarkan catatan <dump> untuk masuk ke penyimpanan stabil.
  - Untuk memulihkan dari kegagalan disk
    - pulihkan database dari dump terbaru.
    - Konsultasikan log dan ulangi semua transaksi yang dilakukan setelah dump
- Dapat diperpanjang untuk memungkinkan transaksi menjadi aktif selama dump; dikenal sebagai **dump fuzzy** atau dump online
  - Will study fuzzy checkpointing later


TERIMA
KASIH
DAN
APA ADA
PERTANYAAN?