TEKNIK RECOVERY

(ref. Fundamentals of DB Systems, Elmasri, N)

Pengenalan Transaksi dan Pemrosesannya

Konsep transaksi menyediakan suatu mekanisme untuk menggambarkan unit logika dari proses database. Sistem pemrosesan transaksi merupakan sistem dengan database yang besar dan ribuan concurrent user yang mengeksekusi transaksi database.

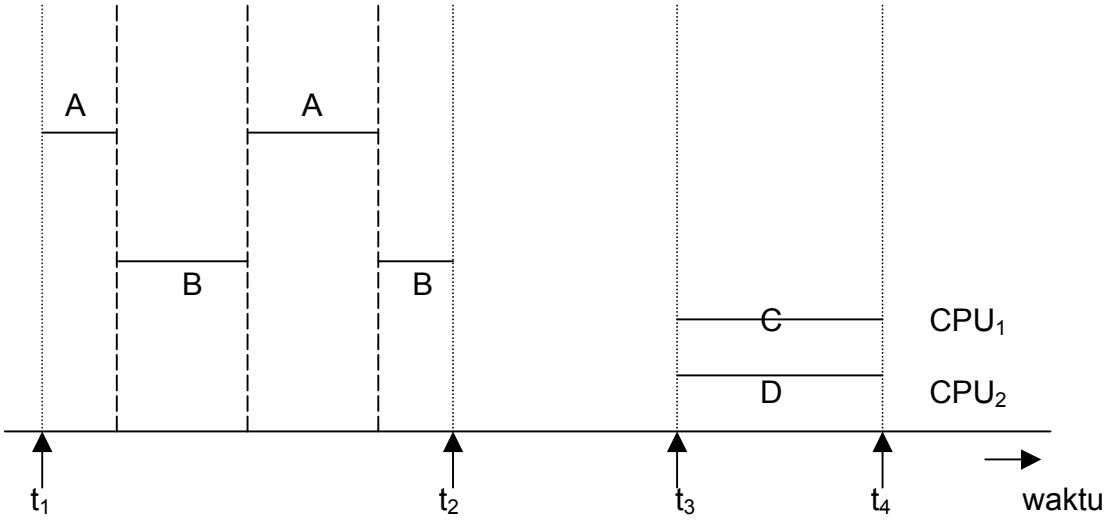
Sistem Single-user VS Multiuser

Satu kriteria untuk mengklasifikasikan suatu sistem database berdasarkan jumlah user yang menggunakan sistem secara konkuren – yaitu pada saat yang bersamaan.

DBMS adalah single-user jika paling banyak satu user yang dapat menggunakan sistem satu saat, dan multiuser jika banyak user dapat menggunakan sistem, dan kemudian mengakses database – bersamaan. Single-user DBMS kebanyakan terbatas pada beberapa sistem mikro komputer, selain itu multiuser.

Multiple user dapat mengakses database dan menggunakan sistem komputer secara simultan karena adanya konsep multiprogramming, dimana komputer dapat mengeksekusi multiple program – atau proses – pada saat yang bersamaan. Jika hanya ada sebuah CPU, berarti hanya dapat mengeksekusi satu proses satu saat. Sistem operasi multiprogramming mengeksekusi beberapa perintah dari satu proses kemudian menahan proses tsb dan mengeksekusi beberapa perintah dari proses selanjutnya, dst. Suatu proses dilanjutkan atau diteruskan dari tempat proses tsb ditahan pada saat gilirannya.

Gambar 19.1 (ref. Elmashri)



2 buah proses A dan B tereksekusi secara bergantian. Proses bergantian ini (interleaved) menyebabkan CPU tetap sibuk ketika sebuah proses membutuhkan operasi I/O misalnya baca blok dari disket. CPU diubah untuk mengeksekusi

proses lainnya daripada berada dalam keadaan idle selama waktu I/O. proses inteleave ini menghindari proses yang lama dari penundaan proses lain.

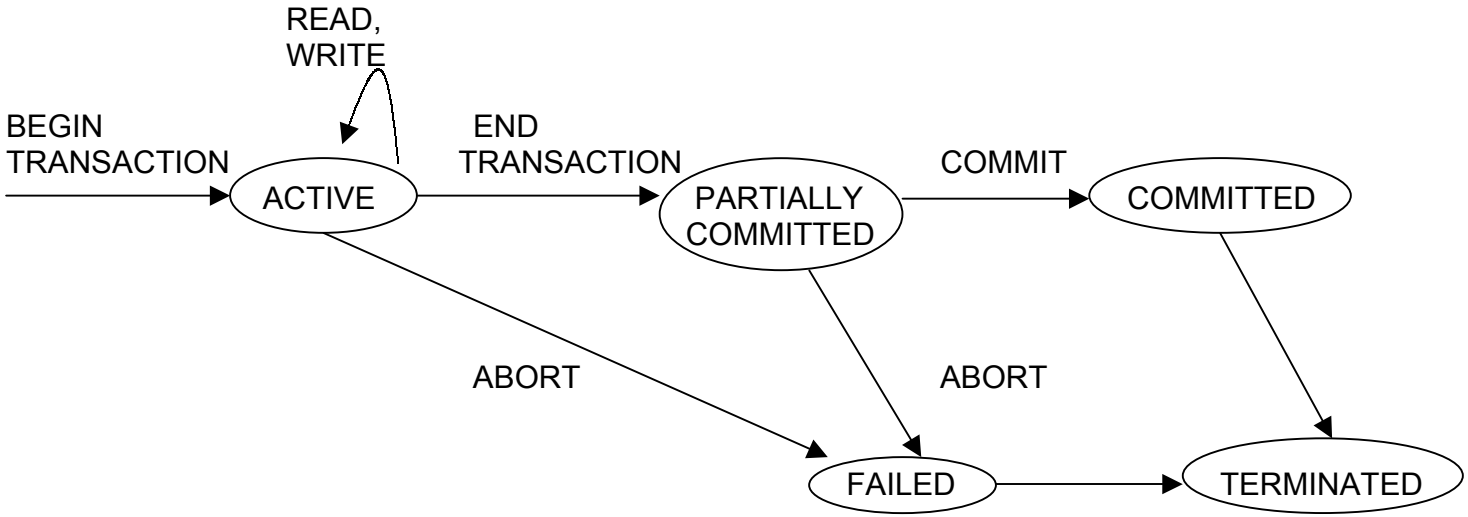
Jika sistem komputer memiliki prosesor hardware yang banyak, proses paralel dari proses yang multiple mungkin, seperti proses C dan D., istilah ini disebut dengan interleaved concurrency. Di DBMS yang multiuser, item data yang disimpan merupakan sumber utama yang dapat diakses secara konkuren oleh user yang interaktif atau program aplikasi, yang secara konstan mengambil informasi dari database dan memodifikasi DB.

Transaksi merupakan unit logika dari proses database yang mencakup satu atau lebih operasi akses database – meliputi insert, delete, modifikasi atau operasi retrieve. Operasi database ini dapat diembed dalam suatu program aplikasi atau dapat langsung dibuat interaktif dengan bahasa query level tinggi misal SQL. Satu cara untuk menspesifikasikan batasan transaksi adalah dengan membuat statemen **begin transaction** dan **end transaction** dalam program aplikasi; dalam kasus ini semua operasi akses database diantara statemen begin-end dianggap sebagai sebuah transaksi. Sebuah program aplikasi dapat berisi lebih dari sastu transaksi jika berisi beberapa batasan transaksi.
Jika operasi database dalam suatu transaksi tidak meng-update database tetapi hanya mengambil (retrieve) data, transaksinya disebut dengan **read-only transaction.**

Database direpresentasikan sebagai kumpulan dari item data yang dinamakan. Ukuran dari suatu item data disebut **granularity**, dan dapat berupa field dari beberapa record dalam database atau dapat merupakan unit yang lebih besar seperti record atau bahkan seluruh blok disk.

STATUS TRANSAKSI & OPERASI TAMBAHAN
Suatu transaksi adalah unit terkecil dari kerja yang dapat diselesaikan atau tidak dapat diselesaikan. Beberapa operasinya dengan diagram transisinya :

- BEGIN_TRANSACTION : memulai transaksi
- READ or WRITE : operasi baca atau tulis dari item database yang dieksekusi sebagai bagian dari transaksi
- END_TRANSACTION : operasi transaksi READ atau WRITE selesai dilakukan
- COMMIT_TRANSACTION : transaksi berakhir sukses sehingga semua perubahan (update) yang dilakukan melalui transaksi dapat dimasukkan ke database dan akan diselesaikan
- ROLLBACK (or ABORT) : transaksi berakhir dengan tidak sukses sehingga semua perubahan atau efek transaksi yang diaplikasikan ke database tidak dapat diselesaikan.


READ,
WRITE
BEGIN
TRANSACTION
ACTIVE
END
TRANSACTION
PARTIALLY
COMMITTED
COMMIT
COMMITTED
ABORT
ABORT
FAILED
TERMINATED

KONSEP RECOVERY
Recovery dari suatu kegagalan transaksi biasanya berarti database direstore ke status yang konsisten ke waktu sebelum terjadi kegagalan.

Untuk mengcover kesalahan atau kegagalan dari transaksi, sistem memaintain sebuah log untuk menjaga jalannya semua operasi yang mempengaruhi nilai dari item database. Informasi ini mungkin akan dibutuhkan untuk mengcover adanya kegagalan. LOG disimpan di storage, dan secara berkala diback-up ke storage lainnya untuk menjaga dari kerusakan yang fatal.

Beberapa strategi recovery :
1. Jika terjadi kerusakan ke sebagian besar database, misalnya disk crash, metode recovery merestore kopi sebelumnya dari database yang sudah diback-up ke storage khusus (biasanya tape) dan membangun kembali status dengan mengaplikasikan kembali atau redo operasi dari transaksi yang commit dari log backed-up sampai waktu dimana terjadi kegagalan
2. Ketika database tidak secara fisik rusak tetapi hanya menjadi tidak konsisten karena adanya kesalahan (system crash, system error, local error, concurrency control), strategi adalah membalik semua perubahan yang menyebabkan ketidak konsistenan yaitu dengan meng-undo beberapa operasi. Kemungkinan juga dengan melakukan redo beberapa operasi supaya dapat merestore status database yang konsisten.

Ada 2 teknik utama dalam melakukan recovery kesalahan transaksi :
1. Deferred update
2. Immediate update

Deferred update (update yang ditunda)
Update tidak dilakukan secara langsung pada database tetapi update dilakukan setelah transaksi mencapai titik commit. Sebelum mencapai commit, semua transaksi yang diupdate disimpan di buffer local. Selama commit, update pertama disimpan dalam log dan ditulis ke database. Jika transaksi gagal sebelum mencapai titik commit, database tidak akan berubah hingga UNDO dibutuhkan.

Ide dari protocol update yang tertunda :
1. Sebuah transaksi tidak dapat merubah database pada disk hingga mencapai titik point
2. Sebuah transaksi tidak dapat mencapai titik point hingga semua operasi update disimpan dalam log dan ditulis ke disk

Karena database tidak pernah ter-update pada disk hingga transaksi mencapai commit, operasi UNDO tidak diperlukan.
Operasi ini dikenal dengan algoritma recovery NO UNDO/ REDO. REDO dibutuhkan saat sistem gagal setelah transaksi mencapai commit tetapi sebelum perubahan disimpan pada database di disk.

Recovery dengan update tertunda pada single-user
PROCEDURE RDU_S : use 2 lists of transactions; the committed transactions since the last checkpoint, and the active transactions (at most of transaction will fall in this category, because the system is in single-user). Apply the REDO operation to all the write_item operations of the committed transactions from the log in the order in which they were written to the log. Restart the active transactions.
Recovery using Deferred Update in a Single-user environment = RDU_S

Prosedur REDO
REDO(WRITE_OP) : redoing a write_item operation WRITE_OP consists of examining its log entry [write_item, T, X, new_value] and setting the value of item X in the database to new_value, which is the after image (AFIM).

Operasi REDO mempunyai sifat idempoten, yaitu eksekusi yang terjadi berulang-ulang sama dengan sekali eksekusi saja. Ini disebabkan karena jika sistem gagal selama proses recovery, recovery selanjutnya adalah mencoba melakukan prosedur REDO write_item yang sudah dilakukan selama proses recovery awal. Hasil sistem crash selama recovery seharusnya sama dengan hasil recovery dari ketika tidak ada crash selama recovery.

Contoh

(a) operasi read dan write dari 2 buah transaksi

T1	T2
read_item(A)	read_item(B)
read_item(D)	write_item(B)
write_item(D)	read_item(D)
	write_item(D)

(b) log sistem saat terjadi crash

[start_transaction, T_1]
[write_item, T_1,D,20]
[commit, T_1]
[start_transaction, T_2]
[write_item, T_2,B,10]
[write_item, T_2,D,25] system crash

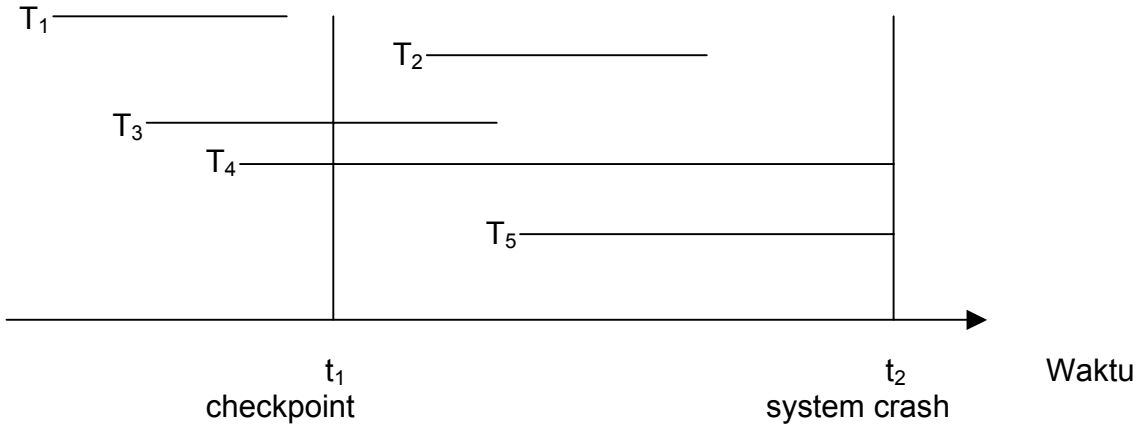
The [write_item, ⋯] operations of T_1 are redone.
T_2 log entries are ignored by the recovery process.

Kegagalan pertama terjadi pada eksekusi transaksi T_2 (gambar b). Proses recovery akan redo [write_item,T_1,D,20] pada log dengan me-reset nilai dari item D menjadi 20 (nilai barunya). Entri [write_item, T_2,...] pada log akan ditiadakan oleh proses recovery karena T_2 tidak commit. Jika kegagalan kedua terjadi selama recovery dari kegagalan pertama, proses recovery yang sama diulang dari awal hingga akhir dengan hasil yang sama.

Recovery update tertunda dengan eksekusi konkuren pada multi-user
Kontrol konkurensi dan proses recovery berhubungan. Secara umum, semakin besar tingkat konkurensi dicapai, semakin banyak waktu untuk recovery.

PROCEDURE RDU_M (WITH CHECKPOINT) : use 2 lists of transactions maintained by the system; the committed transactions T since the last checkpoint (commit list), and the active transactions T' (active list). REDO all the WRITE operations of the committed transactions from the log, *in the order in which they were written into the log*. The transactions that are active and did not commit are effectively canceled and must be resubmitted.
Recovery using Deferred Update in a Multi-user environment = RDU_M

Contoh



Transaksi T_1 mencapai commit saat t_1, dimana T_3 dan T_4 belum. Sebelum terjadi sistem crash pada t_2 , T_3 dan T_2 commit tetapi tidak untuk T_4 dan T_5. Berdasarkan prosedur RDU_M, tidak dibutuhkan operasi REDO write_item dari transaksi T_1 – atau transaksi commit lainnya sebelum waktu checkpoint yang terakhir dari t_1.

Operasi write_item dari T_2 dan T_3 harus diulang karena kedua transaksi mencapai titik commit setelah checkpoint yang terakhir.
Log bersifat force-written sebelum suatu transaksi commit.
Transaksi T_4 dan T_5 diabaikan, secara efektif ditunda atau rolled back karena operasi write_item disimpan dalam database karena deferred update.

Keuntungan dari metode atau algoritma NO-UNDO/REDO adalah operasi transaksi tidak pernah dibutuhkan untuk tidak jadi dilaksanakan, karena :

1. Transaksi tidak mencatat setiap perubahan dalam database pada disk sampai mencapai point commit, yaitu sampai menyelesaikan eksekusi secara lengkap. Sehingga transaksi tidak pernah dirolled back karena kesalahan selama eksekusi transaksi.
2. Transaksi tidak akan pernah membaca nilai yang ditulis oleh transaksi yang belum commit karena item tetap terkunci sampai transaksi mencapai titik commit.

Contoh

(c) operasi read dan write dari 4 buah transaksi

T1	T2	T3	T4
read_item(A)	read_item(B)	read_item(A)	read_item(B)
read_item(D)	write_item(B)	write_item(A)	write_item(B)
write_item(D)	read_item(D)	read_item(C)	read_item(A)
	write_item(D)	write_item(C)	write_item(A)

(d) log sistem saat terjadi crash

[start_transaction, T_1]
[write_item, T_1,D,20]
[commit, T_1]
[checkpoint]
[start_transaction, T_4]
[write_item, T_4,B,15]
[write_item, T_4,A,20]
[commit, T_4]
[start_transaction, T_2]
[write_item, T_2,B,12]
[start_transaction, T_3]
[write_item, T_3,A,30]
[write_item, T_2,D,25] system crash

T_2 dan T_3 are ignored because they did not reach their commit points.
T_4 is redone because its commit point is after the last system checkpoint.

Immediate update (update yang segera)

Di teknik ini, database akan diupdate oleh beberapa transaksi sebelum transaksi mencapai titik point. Operasi secara khusus disimpan di log pada disk dengan

force-writing sebelum diaplikasikan ke database. Jika transaksi gagal setelah menyimpan beberapa perubahan pada database tetapi mencapai titik commit, efek dari operasi pada database harus undone (lepas/ buka); harus rolled back. Jika transaksi mencapai commit sebelum semua perubahan ditulis ke database dikenal dengan algoritma UNDO/ REDO. Variasi dari algoritma ini adalah semua update disimpan dalam database sebelum sebuah transaksi mencapai commit (membutuhkan hanya UNDO), dikenal dengan algoritma UNDO/NO-REDO.

UNDO/ REDO Recovery based on immediate update in a single-user
Jika kegagalan terjadi, transaksi yang aktif pada saat terjadi kegagalan menyimpan beberapa perubahan pada database. Efek dari semua operasi ini harus dibuka (UNDONE).

Prosedur RIU_S (Recovery using Immediate Update in a Single-user environment)

1. Use 2 lists of transactions maintained by the system : the committed transactions since the last checkpoint and the active transactions (at most one transaction will fall in this category, because the system is single-user)
2. Undo all the `write_item` operations of the active transaction from the log, using the UNDO procedure described below.
3. Redo the `write_item` operations of the committed transactions from the log, in the order in which they were written in the log, using REDO procedure describe earlier.

UNDO(WRITE_OP) : Undoing a write_item operation write_op consists of examining its log entry [`write_item`, `T`, `X`, `old_value`, `new_value`] and setting the value of X in the database to `old_value` which is the before imaging (BFIM). Undoing a number of `write_item` operations from one or more transactions from the log must proceed in the reverse order from the order in which the operations were written in the log.

UNDO/ REDO Recovery based on immediate update with concurrent execution
Prosedur RIU_M

1. Use 2 lists of transactions maintained by the system : the committed transactions since the last checkpoint and the active transactions.
2. Undo all the `write_item` operations of the active (uncommitted) transactions, using the UNDO procedure. The operations should be undone in the reverse of the order in which they were written into the log.
3. Redo all the `write_item` operations of the committed transactions from the log, in the order in which they were written in the log.

SHADOW PAGING

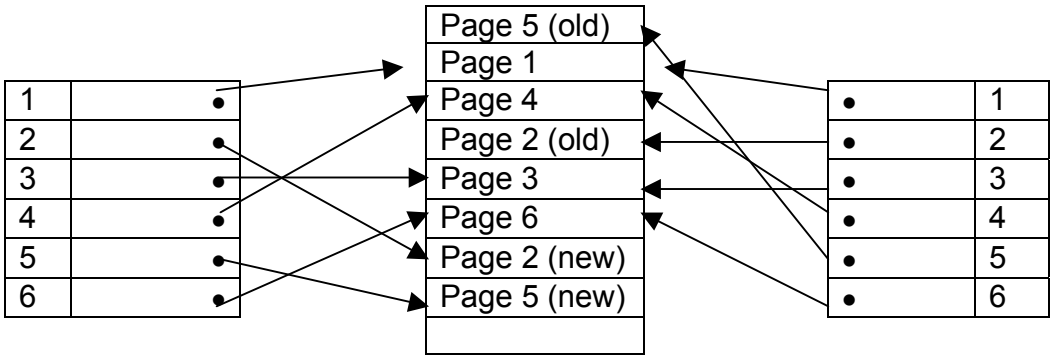
Skema recovery ini tidak membutuhkan penggunaan log pada single-user. Pada multiuser, sebuah log mungkin dibutuhkan untuk metode kontrol konkurensi.

Shadow paging mempertimbangkan database yang berasal dari sejumlah halaman disk berukuran tetap (fixed-size disk) atau blok disk, misalnya sebanyak n. sebuah direktori dengan n entri dibuat, dimana entri yang ke i menunjuk halaman database yang ke i pada disk. Direktori disimpan di memori utama jika tidak terlalu besar, dan semua referensi – baca atau tulis – ke halaman database pada disk.

Ketika transaksi memulai mengeksekusi, directori current – yang entrinya ke halaman database yang paling baru – dikopikan ke directori shadow. Direktori shadow ini kemudian disimpan pada disk ketika directori current digunakan untuk transaksi.

Selama eksekusi transaksi, directori shadow tidak pernah dimodifikasi. Ketika operasi write_item dilaksanakan, kopi baru dari halaman database yang dimodifikasi dibuat, tetapi kopian lama dari halaman tersebut tidak ditumpangi (overwritten). Selain itu, halaman baru ditulis di lain tempat – di beberapa blok disk yang tidak digunakan sebelumnya. Entri directori current dimodifikasi untuk menunjuk ke blok disk baru, dimana directori shadow tidak dimodifikasi dan berlanjut menunjuk ke blok disk lama yang tidak dimodifikasi.

Contoh.



Current directory (after updating page 2, 5) | Database disk blocks (pages) | shadow directory (not updated)

ada 2 versi. Versi yang lama direferensi oleh directori shadow, versi yang baru oleh directori current.

Untuk mengcover kegagalan selama eksekusi transaksi, cukup dengan membebaskan halaman database yang dimodifikasi dan menghapus directori current. Keadaan database sebelum eksekusi transaksi berada pada direktori shadow dan keadaan itu direcover dengan mengembalikan kembali direktori shadow. Database kemudian dikembalikan ke keadaan sebelumnya bagi transaksi yang dieksekusi ketika terjadi crash, dan semua halaman yang

dimodifikasi dihapus. Transaksi yang commit berhubungan dengan penghapusan direktori shadow sebelumnya. Teknik ini disebut dengan NO-UNDO/ NO-REDO.

Pada multiuser dengan transaksi yang concurrent, log dan checkpoint dihubungkan dengan teknik shadow paging. Satu keuntungan dari shadow paging adalah halaman database yang diupdate mengubah lokasi pada disk. Ini yang menyulitkan untuk menyimpan halaman database yang berhubungan dalam disk tanpa strategi manajemen penyimpanan yang kompleks. Jika direktori besar, penambahan penulisan direktori shadow ke disk sebagai transaksi yang commit menjadi berarti.