PENGONTROLAN BERBASIS KOMPUTER

1.Pengertian Transaksi

***Cat**: Transaksi adalah satu atau beberapa aksi program aplikasi yang mengakses/mengubah isi basis data.*

Transaksi merupakan bagian dari pengeksekusian sebuah program yang melakukan pengaksesan basis data dan bahkan juga melakukan serangkaian perubahan data. DBMS yang kita gunakan harus menjamin bahwa setiap transaksi harus dapat dikerjakan secara utuh atau tidak sama sekali. Tidak boleh ada transaksi yang hanya dikerjakan sebagian, karena dapat menyebabkan inkonsistensi basis data. Untuk itu transaksi selalu merubah basis data dari satu kondisi konsisten ke kondisi konsisten lain.

Sebuah transaksi berpeluang untuk 'mengganggu' integritas basis data yang dapat membuat kondisi/hubungan antar data tidak seperti seharusnya. Untuk menjamin agar integritas dapat tetap terpelihara maka setiap transaksi harus memiliki sifat-sifat:

1. **Atomik**, dimana semua operasi dalam transaksi dapat dikerjakan seluruhnya atau tidak sama sekali.
2. **Konsisten**, dimana eksekusi transaksi secara tunggal harus dapat menjamin data tetap konsisten setelah transaksi berakhir.
3. **Terisolasi**, jika pada sebuah sistem basis data terdapat sejumlah transaksi yang dilaksanakan secara bersamaan, maka semua transaksi yang dilaksanakan pada saat yang bersamaan tersebut harus dapat dimulai dan bisa berakhir.
4. **Bertahan**, dimana perubahan data yang terjadi setelah sebuah transaksi berakhir dengan baik, harus dapat bertahan bahkan jika seandainya sistem menjadi mati.

Terhentinya suatu transaksi tidak selalu diakibatkan oleh kegagalan insidental baik dari perangkat keras (crash) ataupun kemacetan sistem operasi (hang). Tapi lebih sering terjadi karena user sengaja menghentikan transaksi atau karena penghentian transaksi oleh DBMS akibat adanya kondisi tak diinginkan, seperti deadlock atau timeout.

Sebuah transaksi dapat menghasilkan dua kemungkinan:

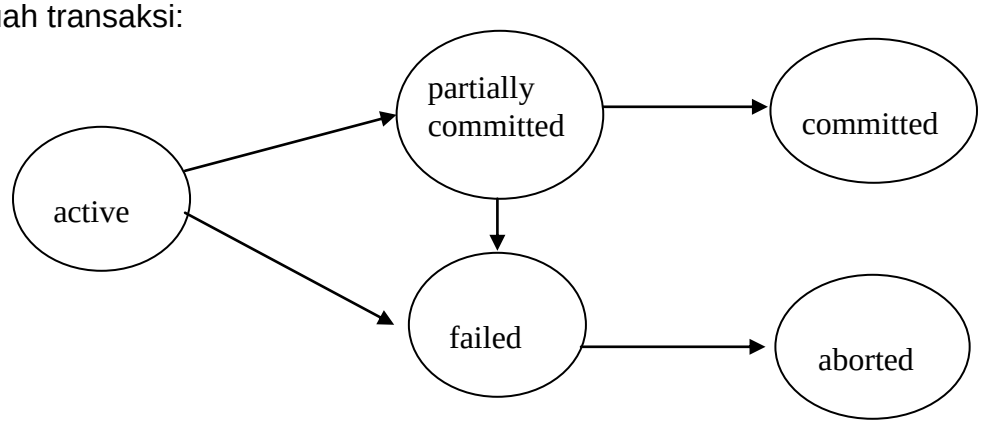
a. Jika dilaksanakan lengkap seluruhnya, transaksi tersebut telah di *commit* dan basis data mencapai keadaan konsisten baru.
b. Jika transaksi tidak sukses, maka transaksi dibatalkan dan basis data dikembalikan ke keadaan konsisten sebelumnya (*rollback*).

Transaksi yang sudah di commit tidak dapat dibatalkan lagi. Jika ada kesalahan, maka harus dilakukan transaksi lain yang membalik dampak transaksi sebelumnya.

Status-status yang dapat dicapai oleh sebuah transaksi sejak mulai dilaksanakan hingga selesai atau batal adalah:

1. **Aktif (Active)**, yang merupakan status awal (initial state) sebuah transaksi yang menunjukkan transaksi tersebut masih dieksekusi.
2. **Berhasil Sebagian (Partially Committed)**, yaitu keadaan yang dicapai transaksi tepat pada saat operasi terakhir dalam transaksi selesai dikerjakan.
3. **Gagal (Failed),** yang merupakan keadaan dimana sebuah transaksi terhenti pengeksekusiannya sebelum tuntas sama sekali.
4. **Batal (Aborted),** yaitu keadaan dimana sebuah transaksi dianggap tidak/belum dikerjakan yang tentu dengan terlebih dahulu diawali dengan mengembalikan semua data yang telah diubah ke nilai-nilai semula. (yang menjadi tanggung jawab DBMS).
5. **Berhasil Sempurna (Committed),** keadaan dimana transaksi telah dinyatakan berhasil dikerjakan seluruhnya dan basis data telah merefleksikan perubahan-perubahan yang memang diinginkan transaksi.

Diagram berikut ini menunjukkan aliran dan siklus peralihan status (state) dari sebuah transaksi:



Ketika sebuah transaksi mulai dikerjakan, maka transaksi itu berada dalam status **aktif.** Jika terjadi penghentian sebelum operasi berakhir, maka transaksi segera beralih ke status **gagal/failed**. Namun, bila keseluruhan transaksi selesai dikerjakan, maka transaksi itu berada pada status **berhasil sebagian/partially committed**, dimana perubahan-perubahan data masih berada di dalam memori utama yang bersifat *volatile*/tidak permanen. Transaksi dalam status ini masih mungkin untuk pindah ke status failed, karena ada pembatalan transaksi baik sengaja maupun tidak. Jika tidak beralih ke status failed, maka nilai-nilai data yang ada di memori utama akan direkam ke dalam disk yang bersifat permanen. Begitu proses perekaman selesai, maka transaksi beralih ke status **committed**. Sementara itu, transaksi yang berada pada status failed, maka DBMS harus menjalan proses *rollback*. Proses tersebut dapat berupa:

- Mengulangi pelaksanaan transaksi / *restart*, yang dilakukan pada transaksi yang failed akbiat kemacetan perangkat keras ataupun perangkat lunak dan bukannya penghentian transaksi secara sengaja oleh user.
- Mematikan transaksi / *kill*, yang dilakukan untuk transaksi yang dihentikan secara sengaja oleh user atau akibat adanya kesalahan lojik dalam penulisan aplikasi.

Begitu salah satu dari pilihan proses tersebut selesai dilakukan, maka transaksi berpindah ke status batal (*aborted*). Status berhasil sempurna/*committed* maupun batal/*aborted* merupakan status terminasi, yaitu status akhir dalam pelaksanaan transaksi.

2. Security Database

Adalah proteksi terhadap perusakan data oleh pemakai yang tidak berwenang (kerusakan yang disengaja)

Authorization
Pemberian hak akses yang mengizinkan sebuah subyek mempunyai akses secara legal terhadap sebuah sistem atau obyek.

Subyek : *user* atau program
Obyek : *database table*, *view*, *application*, *procedure*, atau obyek lainnya yang dibuat di dalam sebuah system

Jenis-jenis hak akses (*privileges*)

Penggunaan nama database yang spesifik
- *Select* (*retrieve*) data
- Membuat tabel (obyek lainnya)
- *Update* data, *delete* data, *insert* data (bisa untuk kolom-kolom tertentu)
- Menghasilkan *output* yang tidak terbatas dari operasi *query* (*user* tidak dibatasi untuk mengakses *record* tertentu)
- Menjalankan prosedur khusus dan utilitas program
- Membuat database
- Membuat (dan memodifikasi) DBMS *user identifiers* dan *authorized identifiers* jenis lainnya
- Anggota dari sebuah kelompok atau kelompok-kelompok user

Views (Subschemas)
Hasil yang dinamik dari satu atau lebih operasi relasi yang beroperasi pada relasi dasar untuk menghasilkan relasi lainnya. *View* merupakan *virtual relation* yang tidak secara nyata ada di dalam sebuah database, tetapi dihasilkan atas permintaan *user* secara khusus.

Backing Up
Proses yang secara periodik menyalin database dan menjurnal (dan memprogram) ke dalam media penyimpanan *offline*

Journaling
Proses penyimpanan dan pemeliharaan sebuah jurnal atau *log* seluruh perubahan terhadap database agar dapat me*recover* secara efektif jika terjadi kegagalan.

Checkpointing
Titik temu sinkronisasi antara database dan transaksi *log file*. Seluruh data yang disimpan di tempat sementara akan disimpan di media penyimpanan kedua.

Integrity
Pengontrolan integritas juga membantu memelihara sistem database yang aman dengan mencegah data dari invalid

Encryption
Penyandian (*encoding*) data dengan menggunakan algoritma khusus yang merubah data menjadi tidak dapat dibaca oleh program apapun tanpa mendeskripsikannya.

Security dilakukan pada beberapa tingkatan:

1. Fisik : lokasi tempat sistem computer harus aman dan terlindung dari orang-orang yang tidak berwenang.
2. Manusia : otoritas user harus dibatasi dan diberikan secara berhati-hati. Hal ini untuk mengurangi penyalahgunaan wewenang.
3. Sistem Operasi : bagaimanapun juga keamanan basis data tergantung dari pengamanan yang diterapkan oleh system operasi. Pengamanan yang longgar akan memperlemah aspek security.
4. Sistem Database : pemberian otoritas sebagian atau seluruhnya pada objek-objek basis data.
5. Jaringan : perlu mekanisme pengamanan software dan hardware jaringan, untuk mengatasi pembobolan jaringan.

Pada tingkatan system database, pada dasarnya ada 2 jenis otoritas untuk mengakses basis data bagi setiap user, yaitu:

1. Disabling permission / ketidakbolehan
2. Enabling permission / kebolehan

Keduanya diberikan pada objek-objek basis data sbb:

- Tabel / relasi : user boleh/tidak membuat table/relasi, atau mengakses langsung suatu table/relasi (lihat, tambah, hapus, ubah data)
- Indeks : user boleh/tidak membuat atau menghapus indeks table.
- View : user boleh/tidak mengakses kemunculan data di suatu view.

Note : View adalah objek basis data yang berisi perintah query ke basis data. Tiap kali view diaktifkan, user akan melihat hasil query-nya. Berbeda dengan table, data yang ditampilkan dalam view tidak bisa diubah.

Bentuk-bentuk otoritas/wewenang yang dapat dimiliki user pada setiap objek basis data:

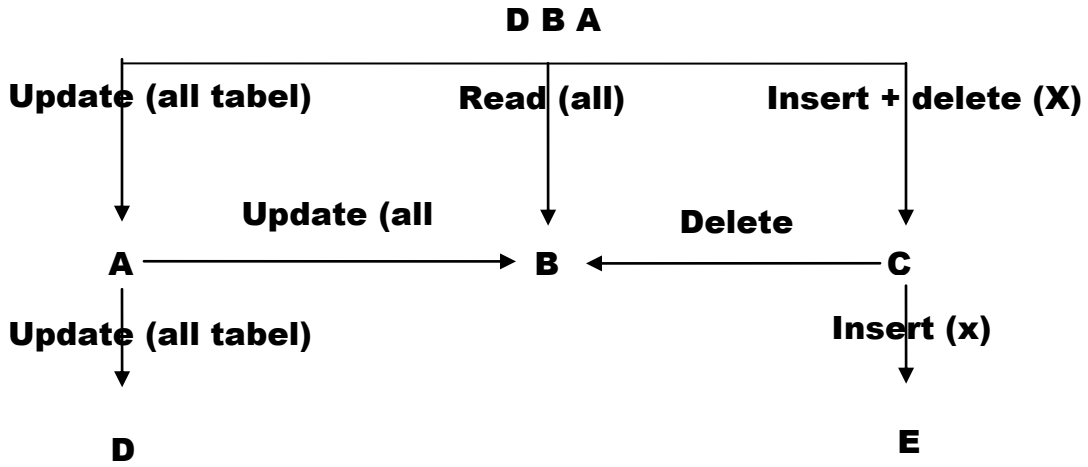
		Boleh	Tidak boleh
1. Read authorization	:	Baca	Hapus
2. Insert authorization	:	Tambah	Ubah
3. Update authorization	:	Ubah	Hapus
4. Delete authorization	:	Hapus	

Selain itu juga ada otoritas yang menyangkut pendefinisian & pengubahan skema basis data:

5. Index authorization	:	Boleh buat & hapus indeks
6. Resource authorization	:	Boleh buat & hapus table baru
7. Alteration authorization	:	Ubah skema tabel (tambah & hapus atribut)
8. Drop authorization	:	Hapus objek basis data.

PEMBERIAN OTORITAS

Pemberian otoritas pertama kali dilakukan oleh DBA. Selanjutnya user yang telah punya otoritas dari DBA dapat memberikan otoritas yang sama atau lebih rendah pada user lainnya. Pembatalan otoritas pada seorang user juga secara otomatis membatalkan pemberitan otoritas yang diberikan user tersebut ke user lain.



PERINTAH SQL UNTUK SECURITY BASIS DATA:

- GRANT <privilege list> ON <nama relasi/view> TO <pemakai/PUBLIC>
- REVOKE <privilege list> ON <nama relasi/view> FROM <pemakai>

GRANT: perintah untuk memberikan hak pemakai.
REVOKE: digunakan untuk mencabut hak pemakai.

Privelege list dapat berupa bentuk authoritas seperti: *READ, INSERT, DROP, DELETE, INDEX, ALTERATION* dan *RESOURCE.*

GRANT OPTION : memberikan fasilitas pada users untuk dapat melakukan GRANT pada users lain.
Contoh:

```
User U1     : GRANT SELECT ON S TO U2
                    WITH GRANT OPTION
User U2     : GRANT SELECT ON S TO U3
                    WITH GRANT OPTION
User U3     : GRANT SELECT ON S TO U4
                    WITH GRANT OPTION
```

Bila User U1 : REVOKE SELECT ON S FROM U2

Maka GRANT untuk U2 ke U3, GRANT U3 ke U4 akan di-REVOKE secara otomatis.

Pengontrolan atau fasilitas keamanan berbasis computer:

- Aturan otorisasi / authorization : mengidentifikasi user dan membatasi aksi-aksi user terhadap basis data
- View : yang membatasi basis data tertentu yang dapat dilihat dan dimodifikasi user.
- Back up, penjadwalan, penentuan check point.
- Enkripsi : yang merubah bentuk awal data menjadi bentuk lagin yang tidak dapat dikenali user (disandikan).

3. Concurrency

Concurrency : banyak transaksi yang dijalankan secara bersamaan.

Hampir semua DBMS adalah multiuser, sehingga berpeluang terjadinya inkonsistensi basis data. Maka perlu adanya pengendalian persaingan eksekusi transaksi (concurrency control).

Alasan mengapa transaksi yang konkuren banyak dipilih dibandingkan transaksi secara serial:

a. Idle time (waktu menganggur) kecil.
   Aktivitas transaksi terbagi 2, yaitu:
   - Aktivitas I / O, seperti pengaksesan disk, penulisan ke monitor.
   - Aktivitas CPU, seperti proses perhitungan, pembandingan.

   Operasi I / O dan CPU bisa dikerjakan secara paralel, dan bisa terjadi dari transaksi yang berbeda. Jika keparalelan ini bisa dioptimalkan, maka akan meningkatkan performansinya, atau dengan kata lain waktu pakai perangkat CPU dan I/O lebih berdaya guna, karena idle time-nya kecil.

b. Response time (waktu tanggap) lebih baik.
   Transaksi pada suatu sistem ada banyak/beragam. Ada yang singkat dan ringan, dan ada pula yang berat. Semua transaksi itu berbeda waktu prosesnya. Jika transaksi-transaksi itu dikerjakan secara serial maka dapat terjadi situasi dimana transaksi yang ringan dan butuh waktu singkat harus *menunggu* selesainya transaksi yang berat dan panjang, sehingga response time-nya menjadi rendah dan tidak dapat diprediksi.

Masalah umum yang terjadi pada sistem yang konkuren:

1. Masalah kehilangan modifikasi
2. Masalah kehilangan modifikasi sementara
3. Masalah analisa yang tidak konsisten

1. MASALAH KEHILANGAN MODIFIKASI

Transaksi A	Waktu	Transaksi B
=	|	=
Baca R	t1	=
=	|	=
=	t2	Baca R
=	|	=
Modifikasi R	t3	=
=	|	=
=	t4	Modifikasi R
=	|	=

- Transaksi A membaca R pada t1, transaksi B membaca R pada t2.
- Transaksi A memodifikasi R pada t3.
- Transaksi B memodifikasi record yang sama pada t4.
- Modifikasi dari transaksi A akan hilang karena transaksi B akan memodifikasi R tanpa memperhatikan modifikasi dari transaksi A pada t3.

2. MASALAH MODIFIKASI SEMENTARA

Masalah ini timbul jika transaksi membaca suatu record yang sudah dimodifikasi oleh transaksi lain tetapi belum terselesaikan (uncommited), terdapat kemungkinan kalau transaksi tersebut dibatalkan (rollback).

Transaksi A	Waktu	Transaksi B
=	|	=
=	t1	Modifikasi R
=	|	=
Baca R	t2	=
=	|	=
=	t3	rollback
=	|	=

- Transaksi B memodifikasi record R pada t1
- Transaksi A membaca R pada t2.
- Pada saat t3 transaksi B dibatalkan.
- Maka transaksi A akan membaca record yang salah.

Transaksi A	Waktu	Transaksi B
=	|	=
=	t1	Modifikasi R
=	|	=
Modifikasi R	t2	=
=	|	=
=	t3	rollback
=	|	=

- Pada waktu t2 transaksi A memodifikasi R
- Karena transaksi B dibatalkan pada waktu t3, maka transaksi A memodifikasi record yang salah.

3. MASALAH ANALISA YANG TIDAK KONSISTEN

Nilai 1 = 40 Nilai 2 = 50 Nilai 3 = 30

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	
=	|	

- Transaksi A menjumlah nilai 1, nilai 2 dan nilai 3.
- Transaksi B → nilai 1 + 10 ; nilai 3 - 10
- Pada waktu t8, transaksi A membaca nilai yang salah karena nilai 3 sudah dimodifikasi menjadi 20 (transaksi B sudah melakukan commit sebelum transaksi A membaca nilai 3).

Note:
- *Commit* adalah operasi yang menyatakan bahwa suatu transaksi sudah terselesaikan / sukses (*successful end-of-transaction*).
- *Rollback* adalah operasi yang menyatakan bahwa suatu transaksi dibatalkan (*unsuccessfull end-of-transaction*).

Dengan adanya masalah di atas, maka dibutuhkan suatu *concurrency control*.
Mekanisme *Concurrency Control*:
1. Locking.
2. Time Stamping

1. LOCKING

Locking adalah salah satu mekanisasi pengontrolan konkuren.
Konsep dasarnya adalah: ketika suatu transaksi memerlukan jaminan kalau record yang diinginkan tidak akan berubah secara mendadak, maka diperlukan kunci untuk record tersebut.
Fungsinya kunci (lock) adalah dengan menjaga record tersebut agar tidak dimodifikasi oleh transaksi lain.

Cara kerja dari kunci:
1. Pertama kita asumsikan terdapat 2 macam kunci:
   - Kunci X : kunci yang eksklusif.
   - Kunci S : kunci yang digunakan bersama-sama.

2. Jika transaksi A menggunakan kunci X pada record R, maka permintaan dari transaksi B untuk suatu kunci pada R ditunda, dan B harus menunggu sampai A melepaskan kunci tersebut.

3. 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 dapat menggunakan kunci S bersama A.

Tabel Kunci:

	X	**S**	**-**
X	N	N	Y
S	N	Y	Y
-	Y	Y	Y

X = kunci X
S = kunci S
N = No
Y = Yes

4. - Bila suatu transaksi hanya melakukan pembacaan saja, secara otomatis ia memerlukan kunci S → baca (S)
   - Bila transaksi tersebut ingin memodifikasi record maka secara otomatis ia memerlukan kunci S → memodifikasi (X)
   - Bila transaksi tersebut sudah menggunakan kunci S, setelah itu ia akan memodifikasi record, maka kunci S akan dinaikan ke level kunci X.

5. Kunci X dan kunci S akan dilepaskan pada saat synchpoint (synchronization point). Synchpoint menyatakan akhir dari suatu transaksi dimana basis data berada pada state yang konsisten. Bila synchpoint ditetapkan maka:
   - Semua modifikasi program menjalankan operasi commit atau rollback.
   - Semua kunci dari record dilepaskan.

MASALAH KEHILANGAN MODIFIKASI

Transaksi A	**Waktu**	**Transaksi B**
=	|	=
Baca R (kunci S)	t1	=
=	|	=
=	t2	Baca R (kunci S)
=	|	=
Modifikasi R (kunci S) tunggu	t3	=
=	|	=
=	t4	Modifikasi R (kunci X) tunggu
=	|	=
	|	=
tunggu	|	tunggu

- Pada waktu t3, transaksi A memerlukan kunci X, maka transaksi A harus menunggu sampai B melepaskan kunci S.
- Transaksi B juga harus menunggu pada t4.
- Maka tidak akan ada yang kehilangan modifikasi, tetapi terdapat keadaan baru yaitu *deadlock*.

MASALAH MODIFIKASI SEMENTARA

Transaksi A	Waktu	Transaksi B
=	|	=
=	t1	Modifikasi R
	|	(kunci X)
=	|	=
Baca R	t2	=
Kunci (S)	|	
tunggu	|	
=	|	=
=	t3	Synchpoint
	|	(kunci X dilepas)
tunggu	|	=
Baca R kembali	T4	=
(kunci S)		
	|	=

Transaksi A	Waktu	Transaksi B
=	|	=
=	t1	Modifikasi R
	|	(kunci X)
=	|	=
Modifikasi R	t2	=
(kunci X)	|	
tunggu		
=	|	=
=	t3	Synchpoint
tunggu	|	(kunci X dilepas)
Modifikasi R	|	=
(kunci X)	|	=

- Transaksi A pada t2 tidak dapat dijalankan langsung, tetapi harus menunggu sampai B melepas kunci X.
- Bila B sudah mencapai synchpoint, maka kunci X dilepaskan dan A dapat meneruskan prosesnya.
- Maka transaksi A tidak akan terjadi kesalahan dalam membaca, karena sudah mencapai synchpoint.

MASALAH ANALISA YANG TIDAK KONSISTEN

Nilai 1 = 40 Nilai 2 = 50 Nilai 3 = 30

Transaksi A	Waktu	Transaksi B
=	|	=
Baca nilai 1 (40)	t1	=
(kunci S)	|	=
Juml = 40	|	
=	|	=
Baca nilai 2 (50)	t2	=
(kunci S)	|	=
Juml = 90	|	=
=	|	=
=	t3	Baca nilai 3 (30)
	|	(kunci S)
=	|	=
=	t4	Modifikasi nilai 3
	|	(kunci X)
	|	30 → 20
=	|	=
	t5	Baca nilai 1 (40)
	|	(kunci S)
	|	=
	t6	Modifikasi nilai 1
	|	(kunci X)
	|	tunggu
	|	=
Baca nilai 3 (20)	t7	=
(kunci S)	|	=
tunggu	|	tunggu

- Transaksi B pada t6 tidak diijinkan, karena memerlukan kunci X maka B harus menunggu sampai A melepaskan kunci S kepada nilai 1.
- Pada t7 transaksi A juga tidak dapat langsung dilaksanakan, karena B menggunakan kunci X pada nilai 3. Maka A harus menunggu B melepaskan kunci X pada nilai 3.
- Transaksi A akan membaca nilai yang benar, tapi timbul masalah baru yaitu *deadlock*.

Dengan menggunakan locking, maka tidak ada transaksi yang akan kehilangan modifikasi. Tapi, terdapat keadaan / masalah baru yaitu ***Deadlock***, yaitu suatu kondisi dimana ke-2 transaksi dalam keadaan menunggu, sehingga keduanya tidak akan pernah selesai dieksekusi.

Jika pelepasan kunci terlalu cepat dilakukan, maka bisa terjadi inkonsistensi informasi. Tapi bila dilepas di akhir transaksi, bisa terjadi deadlock. Hal ini merupakan kondisi dilematis pada sebuah sistem konkuren yang memanfaatkan mekanisme locking.

2. TIME STAMPING

Salah satu alternatif concurrency control yang dapat menghilangkan deadlock adalah time stamping. Secara umum, timestamping (TS) adalah penanda waktu saat transaksi terjadi. Hal ini untuk mengurutkan eksekusi transaksi agar sama dengan eksekusi serial.

Time stamp dapat berupa:

a. waktu sistem saat transaksi dimulai, atau
b. penghitung logik (logical counter) yang terus bertambah nilainya tiap kali terjadi transaksi baru.

Jika timestamp transaksi a lebih kecil daripada timestamp transaksi b , atau
TS(Ta) < TS(Tb), maka transaksi a (Ta) selalu dilaksanakan sebelum transaksi b (Tb).

Contoh:
Misal rekaman pada basis data memuat TS 168, yang mengidentifikasikan transaksi dengn TS 168 adalah transaksi yang terkemudian yang sukses mengupdate rekaman yang bersangkutan. Maka jika ada transaksi dengan TS 170 mencoba mengupdate rekaman yang sama, maka update ini akan diijinkan, karena TS yang dimiliki lebih kemudian dari TS pada rekaman. Saat transaksi ini dilakukan, TS pada rekaman akan diatur menjadi 170. Sekaran, jika transaksi yang akan mengupdate rekaman tersebut memiliki TS 165, maka update *ditolak* karena TS-nya < TS di rekaman.

Selain transaksi, item data juga memiliki nilai time stamp. Untuk setiap item data Q, ada 2 nilai time stamp, yaitu:

a. Read time stamp atau R-timestamp(Q), yang menunjukkan nilai TS terbesar dari setiap transaksi yang berhasil menjalankan operasi read(Q).
b. Write time stamp atau W-timestamp(Q), yang menunjukkan nilai TS terbesar dari setiap transaksi yang berhasil menjalankan operasi write(Q).

Timestamp ini akan selalu diperbarui ketika ada perintah baru read(Q) atau write(Q) yang dijalankan.

Time-stamping Ordering Protocol

Protokol ini menjamin bahwa tiap operasi read dan write yang memiliki konflik dieksekusi sesuai urutan TS.

1. Untuk transaksi Ta yang menjalankan operasi read(Q)
   a. Jika TS(Ta) < W-TS(Q) maka transaksi Ta perlu membaca kembali nilai Q yang telah ditulis dan transaksi Ta akan dibatalkan (rollback).
   b. Jika TS(Ta) ≥ W-TS(Q) maka operasi read dieksekusi, dan R-TS(Q) diisi dengan nilai terbesar diantara TS(Ta) dan R-TS(Q).

2. Untuk transaksi Ta yang menjalankan operasi write(Q):
   a. jika TS(Ta) < R-TS(Q) maka nilai Q yang baru dihasilkan Ta tidak akan dimanfaatkan lagi, dan sistem berasumsi bahwa nilai tersebut tidak pernah dihasilkan. Karena itu operasi write ditolak, dan transaksi Ta di rollback.
   b. jika TS(Ta) < W-TS(Q) maka itu berarti transaksi Ta sedang berusaha melakukan penulisan nilai Q yang kadaluarsa. Maka operasi wrwite ini akan ditolak dan transaksi Ta akan di rollback.
   c. Di luar kondisi a dan b di atas, operasi write dieksekusi dan W-TS(Q) diberi nilai baru yang sama dengan TS(Ta).

Terhadap transaksi Ta yang di rollback, akan diberikan sebuah timestamp yang baru dan diulang kembali.

Properti timestamp:
1. Uniqueness : masing-masing timestamp suatu transaksi adalah unik.
2. Monotonicity : 2 timestamp yang dihasilkan transaksi yang sama meningkat secara monoton.

Cara pemberian nilai timestamp:
1. Global (systemwide) monotonically increasing number
2. Local (site) monotonically increasing number.

4. Recovery

Recovery Facilities

Sebuah DBMS sebaiknya menyediakan fasilitas-fasilitas berikut ini untuk membantu recovery :

• *Backup mechanism*
  Melakukan backup secara periodik terhadap database yang ada

• *Logging facilities*
  Mencatat transaksi-transaksi dan perubahan-perubahan yang terjadi terhadap

database. DBMS memelihara file khusus yang disebut Log (Journal) yang menyediakan informasi mengenai seluruh perubahan yang terjadi pada database.

• *Checkpoint facility*
Mengizinkan update terhadap database yang akan menjadi database yang Permanent

• *Recovery manager*
Mengizinkan sistem untuk *restore* database ke keadaan sebelum terjadi Kerusakkan

Recovery Techniques
Prosedur recovery yang digunakan tergantung dari kerusakkan yang terjadi pada database. Terdapat 2 kasus kerusakkan :

1. Jika database rusak secara fisik seperti : *disk head crash* dan menghancurkan database, maka yang terpenting adalah melakukan *restore backup database* yang terakhir dan mengaplikasikan kembali operasi-operasi *update* transaksi yang telah *commit* dengan menggunakan *log file*. Dengan asusmsi bahwa *log file*nya tidak rusak.
2. Jika database tidak rusak secara fisik tetapi menjadi tidak konsisten, sebagai contoh : sistem *crashed* sementara transaksi dieksekusi, maka yang perlu dilakukan adalah membatalkan perubahan-perubahan yang menyebabkan database tidak konsisten. Mengulang beberapa transaksi sangat diperlukan juga untuk meyakinkan bahwa perubahan-perubahan yang dilakukan telah disimpan di dalam *secondory storage*. Disini tidak perlu menggunakan salinan *backup* database, tetapi dapat me-*restore database* ke dalam keadaan yang konsisten dengan menggunakan *before*- dan *after-image* yang ditangani oleh *log file*.

Teknik *recover* berikut ini dilakukan terhadap situasi dimana database tidak rusak tetapi database dalam keadaan yang tidak konsisten.

1. Deferred Update
Update tidak dituliskan ke database sampai sebuah transaksi dalam keadaan *commit*. Jika transaksi gagal sebelum mencapai keadaan ini, transaksi ini tidak akan memodifikasi database dan juga tidak ada perubahan-perubahan yang perlu dilakukan.

Penulisan dilakukan secara initial hanya terhadap *log* dan *log record* yang digunakan untuk *actual update* terhadap database. Jika sistem gagal, sistem akan menguji log dan menentukan transaksi mana yang perlu dikerjakan ulang, tetapi tidak perlu membatalkan semua transaksi.

2. Immediate Update
Update diaplikasikan terhadap database tanpa harus menunggu transaksi dalam keadaan commit. Update dapat dilakukan terhadap database setiap saat setelah *log record* ditulis. Log dapat digunakan untuk membatalkan dan mengulang kembali transaksi pada saat terjadi kerusakkan.

Deadlock Detection & Recovery

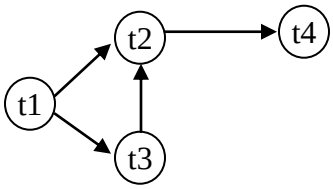
Bertanggung jawab terhadap pendeteksian kegagalan dan mengembalikan basis data ke keadaan konsisten.

Yang harus dilakukan sistem:

- mendapat informasi tentang pengalokasian data pada transaksi
- menggunakan algoritma yang menggunakan informasi tersebut untuk menentukan apakah sistem berada dalam keadaan deadlock.
- menghilangkan keadaan-keadaan deadlock jika algoritma mendeteksi adanya deadlock tersebut.

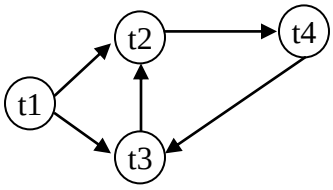
Deadlock dapat digambarkan dengan memanfaatkan graph, yang disebut *graph wait-for*.

- Simpul untuk menyatakan transaksi.
- Busur berarah untuk menunjukkan suatu transaksi Ta sedang menunggu transaksi Tb untuk melepaskan penguncian data.



t1 sedang menunggu t2 dan t3
t2 sedang menunggu t4
t3 sedang menunggu t2

Karena graph ini tidak memiliki siklus, maka sistem tidak berada dalam dalam kondisi deadlock.



t1 sedang menunggu t2 dan t3
t2 sedang menunggu t4
t3 sedang menunggu t2
t4 menunggu t3
t2, t3, t4 berada dalam kondisi deadlock

Kapankah harus menjalankan algoritma pendeteksian deadlock?
Tergantung 2 faktor:

a. seberapa sering deadlock terjadi
b. seberapa banyak transaksi yang *terkena dampak* deadlock

Jika sering terjadi, maka algoritma tersebut harus sering diaktifkan. Data yang dialokasi untuk transaksi-transaksi yang deadlock tidak akan diperoleh oleh transaksi lain, hingga kondisi deadlock tersebut diatasi. Dalam kondisi terburuk, dimana siklus bisa saja bertambah, maka kita dapat mengaktifkan algoritma pendeteksian deadlock ini setiap kali sebuah permintaan data tidak dapat dipenuhi dengan segera.

Recovery dari Deadlock

Cara umum yang digunakan adalah dengan proses rollback transaksi untuk lepas dari deadlock.

Tiga langkah yang harus dilakukan:

a. Selecting Victim / pemilihan korban
   Kita memilih transaksi mana saja yang akan di rollback, dan me-rollback transaksi yang memiliki minimum cost.
   Faktor yang menentukan cost:
   - berapa lama transaksi telah dilaksanakan dan berapa lama lagi transaksi selesai.
   - berapa banyak item data yang digunakan
   - berapa banyak item data lagi yang digunakan sampai selesai.
   - berapa banyak transaksi yang akan ikut di rollback.

b. Rollback
   Setelah menentukan transaksi mana yang di rollback, kita tentukan *sejauh mana* rollback dilakukan. Cara yang paling sederhana adalah *total rollback* kemudian restart. Namun lebih efektif me-rollback transaksi sejauh deadlock dapat dihilangkan.

c. Starvation
   Dalam selecting victim, bisa saja terjadi sebuah transaksi selalu dipilih sebagai korban. Akibatnya transaksi tersebut tidak pernah bisa selesai. Situasi ini disebut ***Starvation***.
   Untuk menghindarinya, sebuah transaksi boleh dipilih sebagai victim hanya beberapa kali saja, atau dalam jangka waktu terbatas. Caranya, dengan melibatkan jumlah rollback yang dialami sebuah transaksi sebagai suatu *biaya/cost*.