Mekanisme Penanganan Deadlock Dalam Pemrosesan Transaksi Oleh DBMS Menggunakan Algoritma Backtracking

Rizal Panji Islami (13510066)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
rizalpanjiislami@gmail.com

***Abstrak*—Database Management System (DBMS) merupakan perangkat lunak yang dikembangkan dan digunakan untuk mengelola database serta menjalankan operasi berupa transaksi yang diberikan oleh user. DBMS memiliki kemampuan untuk diakses secara bersamaan oleh banyak user (multiuser). Setiap user yang butuh mengakses data dalam database akan memberikan query yang selanjutnya akan dianggap sebagai transaksi yang harus diproses oleh database. Untuk meningkatkan *throughput*-nya, DBMS menerapkan mekanisme konkurensi yang memungkinkan dijalankannya lebih dari satu transaksi secara bersamaan. Permasalahan terjadi ketika transaksi-transaksi yang berjalan bersamaan tersebut mengalami keadaan yang disebut sebagai *deadlock*. Ketika *deadlock* terjadi, seluruh mekanisme proses dalam DBMS akan terhenti. Terdapat berbagai solusi untuk menangani keadaan ini. Dalam makalah ini, akan dilakukan salah satu mekanisme penanganan *deadlock* dengan memanfaatkan algoritma *backtracking*.**

***Kata Kunci*—DBMS, konkurensi, *deadlock*, *backtracking*.**

I. PENDAHULUAN

Suatu DBMS mengenali dan menangani perintah yang diberikan oleh user dalam bentuk transaki-transaksi. Transaksi merupakan suatu query yang diberikan oleh user dalam bahasa SQL. Query yang diberikan oleh user ini selanjutnya akan diproses oleh DBMS dengan berbagai tahapan. Misalkan saja terdapat query sebagai berikut:

SELECT DISTINCT(nama) FROM tb_mahasiswa INNER JOIN tb_nilai USING (nim) WHERE asalkota = ”bandung”;

Penanganan transaksi di dalam Database Management System (DBMS) haruslah memperhatikan terjaminnya empat sifat utama yang harus dimiliki oleh suatu DBMS, yaitu:

1. *Atomicity*: suatu transaksi yang terdiri dari beberapa sub transaksi merupakan satu kesatuan utuh dan harus dieksekusi secara keseluruhan atau tidak sama sekali.
2. *Consistency*: data yang tersimpan dalam database harus tetap konsisten pasca penanganan transaksi yang diberikan oleh user.
3. *Isolation*: suatu transaksi ditinjau terpisah dari transaksi yang lain.
4. *Durability*: jika suatu transaksi sudah selesai dieksekusi dengan sukses, maka efek transaksi tersebut bertahan meskipun sistem tersebut mengalami *crash*.

Suatu DBMS memiliki kemampuan untuk menangani query dari banyak user secara bersamaan. Query yang telah diterjemahkan dan dioptimalisasi oleh DBMS selanjutnya akan ditinjau sebagai suatu transaksi. Misalkan saja terdapat tiga transaksi yang diberikan oleh tiga user yang berbeda. Misalkan saja, transaksi tersebut disebut transaksi T1, T2 dan T3. (Untuk memberi kemudahan, setiap transaksi dalam makalah ini hanya ditinjau sebagai proses *read* dan *write* saja).

Normalnya, transaksi-transaksi yang diberikan ke dalam DBMS akan diproses secara serial (satu per satu). Hal ini berarti sebelum transaksi T1 selesai diproses, maka transaksi T2 dan T3 tidak dapat diproses. Hal ini akhirnya akan berdampak kurang baik terhadap *throughput* atau keluaran dari DBMS tersebut. Misalkan saja transaksi T1 membutuhkan 10 satuan waktu untuk selesai diproses, padahal sebetulnya transaksi T2 dan T3 dapat diselesaikan hanya dalam 1 satuan waktu. Hal ini tentu saja akan menghambat selesainya pemrosesan transaksi yang sebetulnya tidak membutuhkan waktu banyak.

Untuk menangani hal ini, DBMS sudah dilengkapi dengan kemampuan melakukan eksekusi transaksi secara konkuren. Penanganan konkuren memungkinkan terjadinya penanganan multi transaksi secara bersamaan. Definisi “bersamaan” dalam hal ini tidaklah diartikan benar-benar “bersamaan”, karena sebetulnya penanganan multi transaksi dilakukan dengan memenggal-menggal pemrosesan transaksi yang ada dan dilakukan secara bergantian (disebut sebagai kondisi *interleave*).

Misalkan saja terdapat dua buah transaksi T1 dan T2 sebagai berikut:

T1
Read(x) Write(x) Read(y) Write(y)

Tabel 1 : Transaksi T1

T2
Read(y) Write(y) Read(x) Write(x)

Tabel 2 : Transaksi T2

Eksekusi ini jika dilakukan secara serial dapat dilakukan dengan terlebih dahulu memproses T1 lalu memproses T2. Namun, jika dilakukan secara konkuren, maka tansaksi yang ada akan di-interleave untuk kemudian dieksekusi. Metoda interleave akan berbeda-beda, dan akan berakibat pada hasil yang berbeda pula. Sebagai contoh berikut penanganan proses T1 dan T2 yang dilakukan secara konkuren:

T1	T2
Read(x) Write(x)	
	Read(y) Write(y)
Read(y) Write(y)	
	Read(x) Write(x)

Tabel 3 : Transaksi Konkuren T1 dan T2

Penanganan konkuren haruslan menjamin sifat lain dari DBMS yaitu *serializability* yang mengakibatkan hasil pemrosesan secara konkuren haruslah sama dengan hasil pemrosesan secara serial. Hal inilah yang akhirnya akan berdampak terhadap munculnya berbagai paradigma penanganan konkurensi disertai dampak-dampaknya, termasuk *deadlock*.

II. Eksekusi Transaksi Konkuren

Seperti yang telah dibahas sebelumnya, penanganan terhadap transaksi konkuren dilakukan oleh DBMS dengan melakukan *interleaving* terhadap transaki-transaksi yang ada. Tidak ada jaminan pasti seperti apa *interleaving* yang dapat dilakukan terhadap multi transaksi yang akan diproses oleh DBMS. Perlakuan *interleaving* yang salah akan menyebabkan *seralizability* menjadi tidak terjamin.

Misalkan saja transaksi konkuren pada tabel 3 dapat ditangani dengan mekanisme *interleaving* yang berbeda seperti berikut:

T1	T2
Read(x)	
	Read(y)
Write(x) Read(y)	
	Write(y) Read(x) Write(x)
Write(y)	

Tabel 4 : Alternatif Transaksi Konkuren T1 dan T2

Penanganan dalam bentuk diatas mungkin saja mengakibatkan hasil akhir yang berbeda dengan penanganan secara serial (*serializability* tidak tercapai). Inilah yang menjadi alasan kenapa mekanisme *interleaving* sangatlah penting dalam penanganan transaksi secara konkuren.

Terdapat berbagai mekanisme yang ditawarkan oleh berbagai DBMS untuk menjamin *serializability* selalu tercapai dalam penanganan transaksi konkuren. Salah satunya dengan menerapkan mekanisme *lock* dengan menggunakan protokol *Strict Two-Phase Locking (Strict* 2PL).

Mekanisme *Strict* 2PL merupakan salah satu bentuk pemrosesan transaksi konkure untuk menjamin *serializability*. Mekanisme ini memiliki dua buah aturan, yaitu:

1. Jika transaksi T ingin membaca serta memodifikasi objek secara berturut-turut, maka transaksi tersebut pertama-tama meminta shared (secara berturut-turut exclusive) lock pada objek.
2. Semua lock yang dipunyai transaksi dilepas pada saat transaksi selesai.

Dampak dari dua buah aturan ini menyebabkan terjaminnya *serializability* dalam mekanisme penanganan konkuren. Protokol ini memproses konkurensi dengan memberikan kunci (*lock*) terhadap suatu *page* atau *sub page* yang merupakan lokasi penyimpanan data.

Terdapat dua jenis *lock* yang digunakan, yaitu *exclusive lock* dan *shared lock*. *Exclusive lock* memungkinkan suatu transaksi untuk membaca sekaligus memodifikasi data. Sementara *shared lock* hanya memungkinkan suatu transaksi untuk membaca data. Jika suatu data X tengah diberikan *shared lock* oleh transaksi T1, maka transaksi T2 juga dapat memberikan *shared lock* terhadap data X tersebut. Sementara jika T1 memberikan *exclusive lock* kepada X, maka T2 tidak dapat memberikan *lock* apapun terhadap X. Hal tersebut terangkum dalam tabel sebagai berikut:

	Exclusive Lock	Shared Lock
Exclusive Lock	Tidak	Tidak
Shared Lock	Tidak	Ya

Tabel 5 : Mekanisme Lock Strict 2PL

Sebagai contoh, transaksi pada tabel 3 jika ditangani oleh *strict* 2 PL akan menjadi seperti berikut:

T1	T2
Exclusive-Lock(x) Read(x) Write(x) Lepas-Kunci(x)	
	Exclusive-Lock(y) Read(y) Write(y) Lepas-Kunci(y)
Exclusive-Lock(y) Read(y) Write(y) Lepas-Kunci(y)	
	Exclusive-Lock(x) Read(x) Write(x) Lepas-Kunci(x)

Tabel 6 : Penanganan Transaksi Dengan Strict 2PL

Setiap transaksi yang telah selesai memproses data yang telah ditanganinya sebaiknya melepaskan kunci yang sudah diberikannya. Jika suatu transaksi meminta *exclusive lock* terhadap suatu data namun data tersebut tengah diberikan *exclusive lock* oleh transaksi yang lainnya, menyebabkan permintaan *exclusive lock* tersebut akan dimasukkan ke dalam queue (antrian) *lock*.

III. DEADLOCK

Penanganan transaksi dengan strict 2PL menyebabkan dapat terjadinya suatu keadaan yang disebut dengan *deadlock*. Fenomena deadlock terjadi jika terdapat dua transaksi yang saling mengantrikan permintaan *lock*-nya dan kedua transaksi tersebut saling menunggu. Sebagai contoh dalam kondisi berikut ini:

T1	T2
Exclusive-Lock(x) Read(x) Write(x)	
	Exclusive-Lock(y) Read(y) Write(y) Request-Lock(x)
Request-Lock(y)	
Next...	Next...

Tabel 7 Contoh Deadlock

Pada contoh diatas, transaksi T1 tengah memberikan *exclusive lock* pada x sementara T2 memberikan *exclusive lock* pada y. Namun, pada tahap selanjutnya, T1 meminta *exclusive lock* pada y dan T2 meminta *exclusive lock* pada x. Hal ini menyebabkan kedua transaksi akan saling menunggu yang disebut sebagai *deadlock*.

Deadlock ini dapat ditangani dengan melakukan *rollback* kepada transaksi-transaki tersebut, untuk mencari alternatif lain penanganan transaksi yang ada.

Mekanisme *rollback* ini dapat dilakukan dengan berbagai cara, salah satunya dengan memanfaatkan algoritma *backtracking* (runut balik).

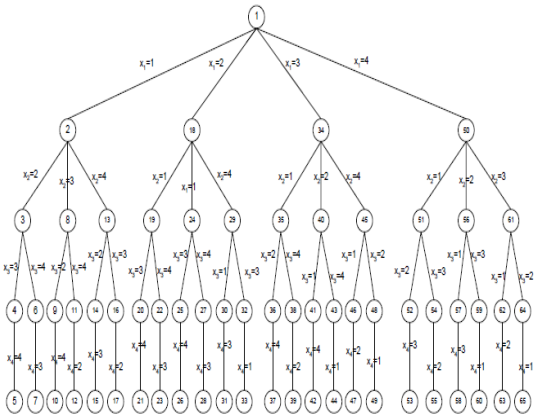
IV. ALGORITMA BACKTRACKING

Algoritma *backtracking* atau runut balik adalah algoritma yang berbasi pada DFS yang digunakan untuk mencari solusi persoalan secara lebih mangkus. Sebenarnya algoritma runut balik merupakan pengembangan dari algoritma *bruteforce* yang melakukan pemeriksaan pada seluruh kemungkinan solusi yang ada secara satu per satu. Dengan menggunakan algoritma runut balik, pencarian solusi dapat dihemat dengan hanya mempertimbangkan solusi yang mengarah kepada solusi.

Algoritma *backtracking* ini umumnya diterapkan dengan menggunakan skema rekursif sebagai berikut:

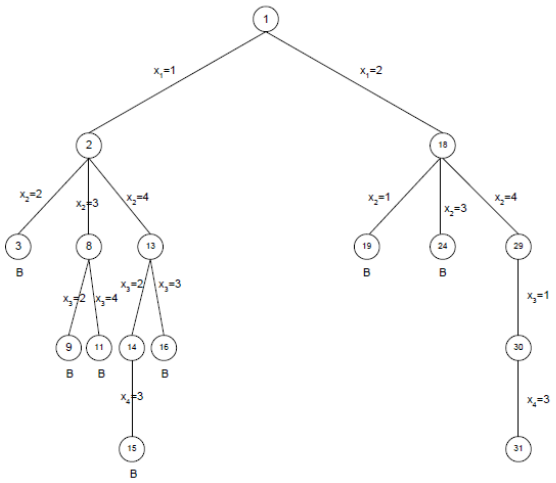
```
procedure RunutBalikR(input k:integer)
{Mencari semua solusi persoalan dengan
metode runut-balik; skema  rekursif
Masukan:  k,  yaitu  indeks  komponen
vektor solusi, x[k]
Keluaran: solusi x = (x[1], x[2], …,
x[n])
}

Algoritma:
for  tiap  x[k]  yang  belum  dicoba
sedemikian sehingga ( x[k]->T(k)) and
B(x[1], x[2], ... ,x[k])= true
do
  if  (x[1],  x[2],  ...  ,x[k])  adalah
   lintasan dari akar ke daun
  then
     CetakSolusi(x)
  endif

  RunutBalikR(k+1)  { tentukan  nilai
   untuk x[k+1]}
Endfor
```

Secara teori, algoritma ini dapat dilakukan dengan melakukan peninjauan memanfaatkan struktur pohon. Misalkan pada penanganan kasu 4 buah ratu, pohon yang digunakan adalah sebagai berikut:



Gambar 1 Pohon Ruang Status Penanganan 4 Ratu

Dengan menggunakan algoritma *backtracking*, proses pencarian solusi dapat dilakukan dengan memproses pohon secara DFS. Namun perbedaannya, setiap simpul yang dianggap tidak menuju kepada solusi akan dimatikan. Misalkan seperti pada keadaan berikut ini:



Gambar 2 Pohon Ruang Status Penanganan Backtracking Pada Persoalan 4 Ratu

V. Algoritma Backtracking Dalam Penanganan Deadlock Dalam DBMS

Penanganan *deadlock* dalam DBMS dapat ditangani dengan menggunakan algoritma runut balik. Berikut solusi penanganan yang penulis tawarkan dalam makalah kali ini:

1. Generate proses *interleaving* pada transaksi-transaksi yang akan diproses.
2. Generate solusi urutan penanganan operasi transaksi secara konkuren, dan bentuk pohon status untuk setiap kemungkinan tersebut.
3. Dengan menggunakan algoritma runut balik, mulai pemrosesan secara DFS, dan periksa pada setiap tahap apakah pemrosesan telah selesai ataukah terjadi *deadlock*.
4. Jika pemrosesan telah selesai, maka transaksi selesai.
5. Jika terjadi *deadlock* lakukan *rollback* untuk mencoba solusi alternatif lain.
6. Jika kondisi 4 dan 5 tidak terjadi, lanjutkan pemrosesan secara DFS.
7. Jika solusi 4 tidak terjadi dan pemrosesan telah sampai pada node yang terakhir, maka pemrosesan konkuren tidak dapat dilakukan dalam transaksi-transaksi tersebut.

Penulis membuat program dalam C++ untuk melakukan pengujian terhadap algoritma ini. Sebagai contoh, jika terdapat 2 transaksi yang akan diproses, maka terlebih dahulu transaki tersebut akan diproses untuk dipecah-pecah menjadi sub-sub transaksi yang ter-*interleave*. Selanjutnya, algoritma runut balik ini akan diterapkan.

VI. Hasil Implementasi Program

Setelah dilakukan implementasi dalam bahasa C++, dilakukan uji coba terhadap program untuk menangani kasus pada tabel 3. Berikut hasil eksekusi program tersebut:

1. Input awal disertai dengan pemrosesan interleaving
   Dalam tahapan ini user menginputkan transaksi-transaksi yang akan diproses disertai dengan rincian transaksi tersebut. Lalu secara otomatis program akan mengenerate *lock-lock* yang akan digunakan dalam pemrosesan. Lock berdasarkan pada proses *read-write* yang dilakukan.

```
Masukkan jumlah transaksi : 2

Masukkan transaksi 1 : (akhiri dengan 'end')
read(x)
write(x)
read(y)
write(y)
end

Masukkan transaksi 2 : (akhiri dengan 'end')
read(y)
write(y)
read(x)
write(x)
end

Hasil pemrosesan interleaving transaksi 1 :
exclusive-lock(x)
read(x)
write(x)
lepas-kunci(x)
exclusive-lock(y)
read(y)
write(y)
lepas-kunci(y)

Hasil pemrosesan interleaving transaksi 2 :
exclusive-lock(y)
read(y)
write(y)
lepas-kunci(y)
exclusive-lock(x)
read(x)
write(x)
lepas-kunci(x)
```

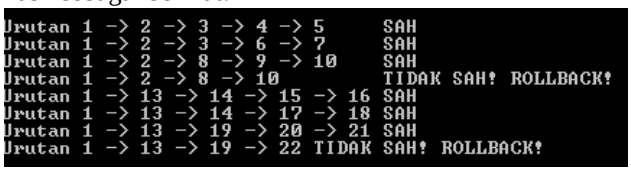
Gambar 3 Hasil Uji Coba Input dan Pemrosesan Interleaving

2. Pembentukan pohon
   Dalam tahapan ini, dibentuk pohon urutan pemrosesan yang mungkin untuk transaksi yang diberikan. Pohon dibentuk dengan berdasarkan pada sifat DFS.

```
1 Begin
        2 (ExLock(x) T1-1)
                3 (ExLock(y) T2-1)
                        4 (T1-2 LepasLock(x))
                                5 (T2-2 LepasLock(y))
                        6 (T2-2 LepasLock(y))
                                7 (T1-2 LepasLock(x))
                8 (T1-2 LepasLock(x))
                        9 (ExLock(y) T2-1)
                                0 (T2-2 LepasLock(y))
                        11 (T2-2 LepasLock(y))
                                12 (ExLock(x) T2-1)
        13 (ExLock(y) T2-1)
                14 (ExLock(x) T1-1)
                        15 (T2-2 LepasLock(y))
                                16 (T1-2 LepasLock(x))
                        17 (ExLock(x) T1-1)
                                18 (T2-2 LepasLock(y))
                19 (T2-2 LepasLock(y))
                        20 (ExLock(x) T1-1)
                                21 (T1-2 LepasLock(x))
                        22 (T1-2 LepasLock(x))
                                23 (ExLock(x) T1-1)
```

Gambar 4 Hasil Uji Coba Pembentukan Pohon

3. Proses Runut Balik Untuk Pencarian Solusi
   Setelah dilakukan uji coba pemrosesan terhadap setiap kemungkinan variasi konkurensi, diperoleh hasil sebagai berikut:



Gambar 5 Hasil Uji Coba Pemrosesan

Dapat dilihat dalam hasil uji coba diatas, pemrosesan akan dilakukan roll back jika terjadi *deadlock* ataupun pemrosesan yang tidak sah lainnya. Dalam program ini, *deadlock* akan dideteksi dengan menggunakan algoritma sebagai berikut:

1. Periksa apakah data yang diminta untuk di-*lock* sudah di-*lock* oleh transaksi lainnya atau tidak. Jika ya, lakukan proses antrian untuk *lock*.
2. Periksa apakah data selanjutnya yang dibutuhkan sudah di-*lock* oleh transaksi tersebut atau tidak. Jika tidak lanjutkan eksekusi program, jika ya (berarti *lock* tersebut saling menunggu) hentikan pemrosesan dan lakukan *rollback*.

Dalam DBMS yang sebenarnya ketika suatu proses yang "sah" sudah terbentuk maka proses transaksi akan dihentikan, namun dalam percobaan kali ini, dilakukan pengujian untuk semua kemungkinan untuk memeriksa ada berapa proses yang menyebabkan terjadinya *rollback*.

VII. KESIMPULAN

Setelah dilakukan uji coba dengan membuat program sederhana untuk mengatasi *deadlock* dalam DBMS, terbukti bahwa algoritma runut balik (*backtracking*) dapat digunakan dengan baik. Dari hasil eksekusi program terlihat bahwa ketika terjadi *deadlock* dalam eksekusi konkuren transaksi dalam DBMS, program akan melakukan *rollback* sehingga transaksi dapat terus dilanjutkan dan tidak terhenti.

Diperlukan pengembangan lebih lanjut untuk program ini, terutama dalam peningkatan kemangkusan algoritma runut balik yang digunakan.

REFERENSI

[1] Date, CJ, *Pengenalan Sistem Basis Data*. PT Indeks Kelompok Gramedia, Jakarta, 2004.
[2] Munir, Rinaldi, *Diktat Kuliah IF3051 Strategi Algoritma*. Program Studi Teknik Informatika ITB, 2012.
[3] Munir, Rinaldi, *Matematika Diskrit*. Informatika Bandung, Bandung, September 2010.
[4] Ramakrshinan, Raghu, *Sistem Manajemen Database*. Penerbit Andi, Yogyakarta, 2009.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 15 Desember 2012

Rizal Panji Islami
13510066