IMPLEMENTASI KENDALI KONKURENSI PADA BASISDATA ORACLE 10g

Nur Wijayaning Rahayu
Jurusan Teknik Informatika, Fakultas Teknologi Industri, Universitas Islam Indonesia, Yogyakarta
e-mail: nnur@fti.uii.ac.id

ABSTRAKSI

Seperti pada sistem operasi, transaksi-transaksi yang terjadi pada basisdata dijalankan secara konkuren, bukan serial. Meskipun pemrosesan serial menjamin setiap transaksi tetap bersifat ACID (atomicity, consistency, isolation, durability), kesenjangan antara response time CPU dan waktu pengiriman transaksi menjadikan konkurensi sebagai solusi terbaik dalam pemrosesan transaksi. Dengan metode ini, sejumlah transaksi bisa diproses bersamaan karena sistem operasi mampu mengerjakan instruksi-instruksi secara multiprogramming. Akan tetapi, konkurensi juga menimbulkan masalah penjadwalan jika urutan pemrosesan transaksi diserahkan pada sistem operasi. Oleh karena itu, diperlukan komponen pengendali konkurensi pada software basisdata agar sifat ACID untuk setiap transaksi tetap terjamin.

Terdapat dua pendekatan konkurensi, yakni optimistik dan pesimistik. Kendali konkurensi optimistik berawal dari asumsi bahwa transaksi-transaksi basisdata jarang bertabrakan, sehingga transaksi bisa langsung dieksekusi. Sedangkan pendekatan pesimistik adalah sebaliknya, sehingga setiap transaksi yang hendak memodifikasi harus mengunci (memperoleh lock) data tersebut sedari awal. Keduanya memiliki kelebihan dan kekurangan masing-masing. Pada tulisan ini dibahas implementasi konkurensi pada basisdata Oracle 10g.

Kata kunci: *konkurensi, transaksi, lock, basisdata Oracle 10g*

1. PENDAHULUAN

Makin banyak pengguna yang mengakses basisdata berarti makin besar peluang terjadinya sejumlah transaksi yang dikirim pada saat bersamaan. Terdapat dua pilihan bagi sistem operasi untuk memprosesnya: secara serial (bersesuaian dengan urutan terkirim transaksi) atau konkuren (beberapa transaksi sekaligus dikerjakan dalam 'waktu' yang sama). Sifat konkuren dalam basisdata ini bisa dianalogikan dengan pengaturan tugas secara *multiprogramming*[1] bagi sistem operasi. Meski sebenarnya prosesor hanya bisa mengerjakan satu tugas dalam satu waktu, perbedaan yang cukup signifikan antara kecepatan prosesor dan kecepatan manusia[2] menimbulkan kesan bahwa prosesor bisa mengerjakan banyak tugas sekaligus.

Akses ke basisdata bisa dilaksanakan dengan dua operasi:

- **read(X)**: mengirimkan data X dari basisdata ke *buffer* lokal milik transaksi yang mengeksekusi operasi *read* (secara fisik terjadi di RAM, *Read Only Memory*).
- **write (X)**: menyimpan data X dari *buffer* lokal milik transaksi yang mengeksekusi operasi *write* ke basisdata (biasanya berupa hardisk).

Transaksi dalam basisdata didefinisikan sebagai sejumlah instruksi atau operasi yang membentuk sebuah satuan kerja lojik. Misalnya dalam contoh kasus klasik perbankan: sebuah transaksi transfer uang sejumlah satu juta rupiah dari rekening A ke rekening B. Transaksi ini terdiri dari urutan operasi:

```
read (A)
A --> A – 1000000
write (A)
read (B)
B --> B + 1000000
write (B)
```

Jika keenam operasi (instruksi) diatas tidak dikerjakan secara keseluruhan dan berurutan, maka transaksi T1 dianggap gagal dan harus diulang karena setiap transaksi basisdata harus bersifat ACID (*Atomicity*, *Consistency*, *Isolation*, *Durability*). *Atomicity* berarti bahwa keenam operasi dianggap sebagai satu kesatuan; jika hanya satu atau lima operasi yang dieksekusi, maka dianggap bukan sebagai satu transaksi. *Consistency* berarti bahwa setelah transaksi dieksekusi, data yang tersimpan di basisdata harus tetap valid. Dalam contoh ini berarti jumlah saldo rekening A dan B harus tetap sama antara sebelum transaksi T1 dieksekusi dan sesudahnya. Sedangkan sifat *Isolation* berarti bahwa setiap transaksi dianggap sebagai satu unit terpisah dari transaksi lainnya, sehingga nilai data yang dieksekusi oleh T1 terisolasi dari operasi-operasi dari transaksi lain. *Durability* berarti data hasil modifikasi T1 tetap akan tersimpan meskipun terjadi kerusakan sistem. Data yang dimaksud adalah data yang berhasil disimpan di hardisk, bukan memori primer semacam RAM.

Jika dalam waktu yang sama, misal t1, terkirim dua transaksi T1 dan T2 yang berusaha memodifikasi data yang sama, maka software

[1] *Multiprogramming* adalah kemampuan sebuah prosesor untuk mengerjakan dua program atau lebih yang berasal dari banyak pengguna dalam waktu bersamaan.

[2] Waktu respon waktu respon kurang dari 0,1 detik tergolong sebagai reaksi manusia yang sangat cepat (instan). Bandingkan dengan komputer berprosesor 1 Gigahertz yang kini sudah umum digunakan. (1 Gigahertz berarti prosesor bisa mengerjakan 1 milyar instruksi dalam 1 detik).

basisdata harus mengatur urutan eksekusi operasi-operasinya agar sifat ACID tetap terjamin. Pengambilan keputusan tersebut tidak bisa diserahkan kepada sistem operasi, karena akan muncul banyak kemungkinan penjadwalan, termasuk penjadwalan yang akan menyebabkan basisdata tidak konsisten.[3] Misalnya jika saldo rekening B bertambah satu juta rupiah sebelum saldo rekening A berkurang. Dan inilah yang menjadi tugas bagi komponen pengendali konkurensi pada basisdata.

Terdapat dua macam metode kendali konkurensi, yaitu:

1. **Konkurensi pesimistik**: sinkronisasi eksekusi transaksi dilakukan mulai dari awal siklus eksekusi transaksi. Baris-baris data dikunci sebelum dimodifikasi, sehingga tidak ada transaksi lain yang bisa mengaksesnya, kecuali setelah kunci dibebaskan. Terdapat tiga jenis mekanisme pesimistik yang bisa digunakan, yaitu *locking*, *timestamp (row-version)*, dan gabungan keduanya (*hybrid*). Metode pesimistik ini menjamin perubahan data bisa dilaksanakan dengan aman. Kekurangan metode ini terletak pada munculnya *delay* jika transaksi berjalan lambat atau data hasil modifikasi transaksinya tidak segera disimpan ke hardisk (*commit*).
   Fase pengendalian metode pesimistik dapat diilustrasikan sebagai berikut:

Validate	Read	Compute	Write

2. **Konkurensi optimistik**: sinkronisasi transaksi ditunda hingga transaksi selesai dieksekusi. Proses validasi data hanya dilakukan ketika transaksi hendak menyimpan data secara permanen ke hardisk. Terdapat dua jenis mekanisme optmistik: *locking* dan *timestamp* (*row-version*). Dengan metode optimistik, sejumlah transaksi tetap bisa dieksekusi bersamaan tanpa harus saling menunggu. Kekurangannya terletak pada kemungkinan terjadinya tumpang tindih data, karena tidak ada jaminan bahwa data yang dimodifikasi adalah data asli, belum diubah oleh transaksi lainnya.
   Fase pengendalian metode optimistik dapat diilustrasikan sebagai berikut:

Read	Compute	Validate	Write

2. KENDALI KONKURENSI PADA BASISDATA ORACLE 10g

Basisdata Oracle 10g menggunakan mekanisme *locking* dan *row-versioning*. *Lock* berarti bahwa sebelum sebuah session diizinkan untuk menyimpan data hasil modifikasi (bisa berupa proses *insert*, *update* atau *delete*), session tersebut harus mengunci data terlebih dahulu. Data yang dikunci bisa berupa satu baris data, beberapa baris data, atau bahkan keseluruhan tabel. Ketika sejumlah transaksi membutuhkan data yang sama, maka transaksi yang pertama kali meminta akan mendapatkan lock, sedangkan transaksi lainnya harus menunggu hingga transaksi tersebut selesai. Mekanisme antrian terjadi secara otomatis, tanpa perlu interaksi administrator.

Semua lock dilepas pada akhir siklus eksekusi transaksi. Transaksi dianggap selesai ketika dilakukan *commit* atau *rollback*. Jika transaksi mengalami kegagalan (*failed*), semua *lock* yang diperoleh transaksi tersebut akan dibebaskan.

Secara default, Oracle melakukan *locking* pada level baris data. Misalnya ada dua transaksi konkuren, T1 dan T2, yang mengakses baris data yang sama, maka transaksi bisa memilih di antara enam modus lock, yaitu:

- **EXCLUSIVE**: Query masih diperbolehkan, sedangkan aktivitas lainnya dilarang. Jika T1 sedang mengunci baris data X secara eksklusif, T2 diwajibkan untuk mengantri sampai T1 melepaskan kunci. Lock jenis ini terutama dibutuhkan ketika hendak menghapus (*drop*) tabel.
- **ROW SHARE**: Jika T1 sedang mengunci tabel dalam modus ini, T2 juga diperbolehkan mengakses tabel tersebut asal bukan akses eksklusif.
- **ROW EXCLUSIVE**: Hampir sama dengan Row Share, tetapi T2 juga tidak boleh mengunci secara SHARE. Lock jenis ini secara otomatis diberikan kepada transaksi yang hendak melakukan *update*, *insert* dan *delete*.
- **SHARE**: T1 dan T2 bisa dieksekusi secara konkuren, tetapi keduaranya tidak boleh memodifikasi tabel yang sedang dikunci. Lock ini digunakan terutama untuk membuat indeks pada tabel.
- **SHARE ROW EXCLUSIVE**: Jika T1 menggunakannya untuk melakukan *query* data pada tabel, T2 juga diperbolehkan untuk melakukan hal yang sama, tetapi T2 dilarang memodifikasi data atau mengunci dalam modus SHARE.

Lock juga bisa diperoleh secara manual, misalnya dengan sintaks:

```
LOCK TABLE [schema.] (table/view) IN modus-lock MODE [NOWAIT]
```

Perintah tersebut diperlakukan sama seperti permintaan lock lainnya, yaitu harus masuk dalam antrian jika sebelumnya ada transaksi yang memegang atau meminta lock. Akan tetapi jika tidak ingin menunggu, argumen NOWAIT bisa digunakan.

[3] Jika T1 terdiri dari 4 operasi dan T2 terdiri dari 3 operasi, maka akan terjadi banyak kemungkinan penjadwalan, karena eksekusi operasi-operasi T1 bisa langsung diikuti oleh eksekusi operasi dari T2, tanpa harus menunggu 4 operasi T1 tersebut diselesaikan dengan berurutan.

 ISSN: 1907-5022

Misalnya ada dua transaksi, T1 dengan perintah:

```
SQL>    UPDATE employees
        SET salary = salary + 200000
        WHERE employee_id=107;
```

dan T2 dengan perintah:

```
SQL>    UPDATE employees
        SET salary = salary * 1.1
        WHERE employee_id=107;
```

Masing-masing transaksi akan memperoleh lock:

- *row exclusive* pada baris data (row) yang akan dimodifikasi, dan
- *share* pada tabel yang berisi row tersebut.

Jika nilai salary awal adalah 1.000.000, maka akan terjadi beberapa kemungkinan:

Penjadwalan 1: jika saat hendak menyimpan hasil modifikasi data, T1 meminta lock lebih dahulu daripada T2:

Awal	1.000.000
T1	1.200.000
T2	1.320.000

Penjadwalan 2: jika saat hendak menyimpan hasil modifikasi data, T2 meminta lock lebih dahulu daripada T1:

Awal	1.000.000
T2	1.100.000
T1	1.300.000

Gaji terakhir dari kedua penjadwalan tersebut berbeda 20.000 rupiah. Kedua-duanya valid. Perbedaannya terletak pada nilai salary yang dibaca oleh masing-masing transaksi. Misalnya pada penjadwalan 1, T1 membaca nilai salary orisinil, sedangkan T2 membaca nilai salary hasil modifikasi T1. Jika dikehendaki agar T2 juga membaca nilai salary yang orisinil (yaitu 1.000.000), maka bisa digunakan fitur **Flashback**. Kemampuan fitur ini adalah:

- Menjalankan query untuk melihat nilai data yang lampau.
- Menjalankan query untuk menampilkan metadata perubahan pada basisdata secara detail.
- Mengembalikan isi tabel atau baris data pada nilai sebelumnya.

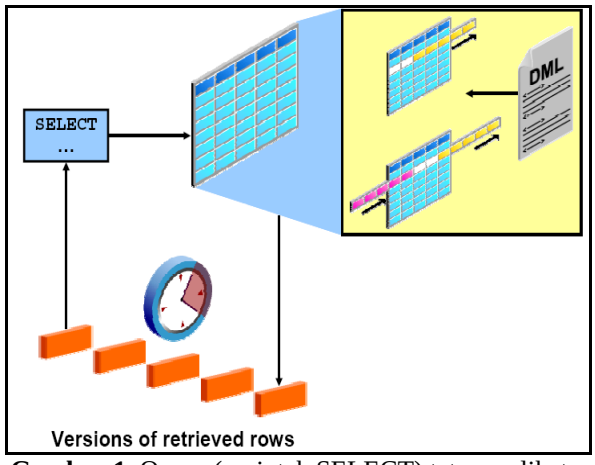
Terdapat empat macam subfitur Flashback, yaitu:

1. **Oracle Flashback Query**: mengambil data di masa lampau dengan klausa AS OF.
2. **Oracle Flashback Version Query**: mengambil metadata dan data historis untuk interval waktu tertentu dengan menggunakan klausa VERSIONS yang ditambahkan pada perintah SELECT untuk melihat semua versi data yang tersimpan di antara dua *timestamp* atau SCN[4], termasuk baris yang dihapus (*deleted*) dan diisikan kembali (*reinserted*). Sintaks klausa VERSIONS adalah:

```
VERSIONS BETWEEN {SCN | TIMESTAMP}
    MINVALUE AND MAXVALUE
```

3. **Oracle Flashback Transaction Query**: mengambil metadata dan data historis transaksi tertentu atau seluruh transaksi pada interval waktu tertentu serta kode SQL untuk keperluan *undo*. Data diambil dari view FLASHBACK_TRANSACTION_QUERY.
4. **DBMS_FLASHBACK package**: mengeset jam ke waktu lampau, untuk memeriksa data pada saat itu.

Mulai versi 9i, data-data masa lampau yang digunakan untuk proses rollback (dan *undo*) tersimpan pada segmen *undo* di tablespace UNDO, bukan segmen *rollback* di tablespace SYSTEM. Undo bertanggung jawab atas konsistensi data bagi pengguna yang sedang membaca baris data yang sedang dimodifikasi oleh pengguna lainnya. Dalam contoh ini, transaksi-transaksi lain yang sedang membaca baris data pegawai nomor 107 tidak akan melihat perubahan yang dilakukan oleh T1hingga T1 melakukan *commit*. Segmen undo digunakan untuk merekonstruksi blok-blok data yang berisi versi konsisten, sehingga nilai pra-modifikasi pada baris tersebut bisa tetap tersedia bagi para pengguna yang mengakses dengan perintah SELECT sebelum T1 melakukan *commit*, seperti yang diilustrasikan pada gambar 1.



Gambar 1. Query (perintah SELECT) tetap melihat nilai orisinil (pra-modifikasi) ketika data sedang dimodifikasi tapi belum *commit*.

[4] SCN (System Change Number): Server Oracle menggunakan SCN untuk mengidentifikasikan baris-baris data untuk setiap hasil modifikasi transaksi yang tersimpan permanen (*committed transaction*). SCN digunakan secara internal oleh Oracle dan menyimpan data waktu secara tepat, karena SCN dipetakan ke timestamp dengan granularitas setiap 3 detik.

Oleh karena itu, jika dikehendaki T2 mengubah nilai gaji yang orisinil, yakni 1.000.000, maka tahapan perintahnya adalah:

1. Memeriksa SCN atau timestamp pada salary pegawai tersebut untuk melihat perubahan nilai salary.

```
SELECT ora_rowscn, salary FROM employees
VERSIONS BETWEEN SCN MINVALUE AND
MAXVALUE WHERE employee_id = 107;
ORA_ROWSCN         SALARY
-------------------------
205553             1200000
205552             1000000
```

2. Jika nilai sudah berubah, dikembalikan ke nilai *salary* yang orisinil.

```
UPDATE employees SET salary =
(SELECT salary FROM employees AS OF SCN
20553 WHERE employee_id=107)
WHERE employee_id = 107;
```

3. Transaksi T2 bisa dijalankan.

```
UPDATE employees  SET salary = salary *
1.1 WHERE employee_id=107;
```

Hasilnya adalah:

Awal	1.000.000
T1	1.200.000
T2	1.100.000

3. KESIMPULAN

Kendali konkurensi pada basisdata Oracle 10g menggunakan metode optimistik dengan locking dan row-versioning. Delay tidak perlu terjadi jika ada satu transaksi yang memodifikasi data dan ada transaksi lain yang berusaha untuk melihat data (proses *query*). Akan tetapi, proses validasi yang terjadi sebelum data disimpan secara permanen bisa menimbulkan tumpang tindih data. Hal ini bisa diatasi dengan fitur flashback.

PUSTAKA

Van Dyke, Ric, Lowenthal, Russ. *Oracle Database 10g: Administration Workshop I*. Oracle. 2004

Adams, Drew, Paapanen, Eric. *Oracle Database Application Developer's Guide - Fundamentals, 10*g *Release 1 (10.1)*. Oracle Corporation. 2003

Loney, Kevin. Bryla, Bob. *Oracle Database 10g DBA Handbook*. Oracle Press, McGraw-Hill/Osborne, 2005.