SUBQUERY SKALAR DAN TEKNOLOGI CACHING PADA ORACLE DATABASE

Iwan Tanto

Sekolah Tinggi Manajemen Informatika dan Komputer LIKMI
Jl. Ir. Juanda 96 Bandung 40132

e-mail: iwantanto@gmail.com

ABSTRAK

Subquery skalar sangat jarang dilakukan, padahal Oracle Database menerapkan teknologi *cache* yang sangat membantu untuk memroses *subquery* skalar dengan efisien. *Subquery* skalar dapat meningkatkan waktu respon pada *query* yang melibatkan proses penggabungan tabel. Selain itu *subquery* skalar dapat dimanfaatkan untuk mengurangi pemanggilan fungsi PL/SQL dari suatu pernyataan SQL. Hal ini penting mengingat proses mengeksekusi pernyataan SQL lalu pindah ke PL/SQL berkali-kali dapat sangat membebani sistem basisdata. Dari beberapa contoh kasus dapat ditarik kesimpulan bahwa performansi dari teknologi *cache subquery* skalar jauh lebih baik daripada menggunakan fungsi dengan kata kunci `DETERMINISTIC` atau `RESULT_CACHE`.

Kata-kata kunci: skalar, *subquery, cache, hash*

1. PENDAHULUAN

Apakah yang disebut sebagai *query* skalar? Di dalam basisdata, data skalar adalah tabel yang terdiri dari satu kolom dengan satu baris data. Jadi *query* skalar adalah pernyataan SQL yang memberikan hasil sebuah data skalar. Isi dari satu kolom tersebut dapat berupa data sederhana ataupun data kompleks dengan banyak atribut. Namun isi satu kolom tersebut dapat diasumsikan sebagai satu nilai skalar (jika *subquery* menghasilkan satu baris data), atau NULL (jika menghasilkan 0 baris data).

Basis data Oracle menyediakan sebuah data skalar (sering juga disebut tabel *dummy*) yaitu tabel `dual` yang terdiri dari satu kolom `D` dengan satu baris data bernilai 'X' untuk berbagai keperluan. Misalnya, lakukan *query* skalar berikut:

```
select 'Hello' from dual;
select sqrt(10) from dual;
```

Query skalar umumnya digunakan dalam pernyataan SQL sederhana seperti untuk mencari agregasi dan jarang digunakan untuk *query* yang lebih rumit. Seperti contoh *query* berikut untuk menghitung jumlah karyawan:

```
select count(*) from emp;
```

2. PENERAPAN SUBQUERY SKALAR

Subquery skalar adalah sebuah pernyataan SQL (*query*) yang di dalamnya terdapat sebuah *query* skalar. *Subquery* skalar dapat digunakan di mana nilai literal sah untuk digunakan. Sebagai contoh, cobalah lakukan query yang melibatkan nilai literal 'Hello' sebagai berikut:

```
select deptno, dname, 'Hello'
  from dept;
```

Maka nilai literal 'Hello' boleh digantikan dengan sebuah *subquery* skalar sebagai berikut:

```
select deptno, dname,
     (select count(*)
        from emp
        where emp.deptno = dept.deptno)
  from dept;
```

Bagian yang ditandai dengan huruf tebal adalah *subquery* yang akan menghasilkan sebuah nilai skalar, yaitu banyaknya karyawan yang bekerja di suatu departemen tertentu. *Quer*y ini akan menghasilkan daftar seluruh departemen dan jumlah karyawan yang bekerja di setiap departemen.

Query berikut ini ekivalen secara semantik dengan *query* di atas sehingga akan memberikan hasil yang sama:

```
select dept.deptno, dept.dname, count(emp.empno)
  from dept left outer join emp
    on (dept.deptno = emp.deptno)
```

```
group by dept.deptno, dept.dname;
```

Basisdata memroses kedua *query* di atas dengan cara yang berbeda. Dengan memahami cara kerja basisdata dalam memroses pernyataan SQL, maka dapat dituliskan formula yang paling efisien yang disesuaikan dengan kebutuhan.

Pada *query* pertama, Oracle Database akan mengambil data pertama dari tabel `dept`, menjalankan *subquery* skalar (`select count ...`) terhadap tabel `emp` dan melakukan proses agregasi (menghitung jumlah karyawan yang bekerja di departemen pertama) kemudian menampilkan hasilnya. Proses menampilkan hasil pertama ini akan berjalan dengan sangat cepat.

Pada *query* kedua, Oracle Database akan melakukan proses penggabungan dua buah tabel (`left outer join`) dan membentuk sebuah tabel *hash* yang memakan waktu, kemudian melakukan proses agregasi. Hasil *query* baru dapat ditampilkan setelah seluruh proses agregasi selesai dilakukan. Untuk data yang besar, waktu respon terhadap *query* ini akan terasa lambat. Jadi jika tujuannya adalah mengoptimalkan proses *query* untuk menghasilkan waktu respon yang baik, maka penggunaan *subquery* skalar adalah pilihan yang tepat.

3. TEKNOLOGI CACHE PADA SUBQUERY SKALAR

Contoh sederhana di atas menunjukkan salah satu contoh penerapan *subquery* skalar. Perlu dipahami bagaimana cara Oracle Database menggunakan *cache* untuk memroses *subquery* skalar. Pertama-tama bagian *subquery* skalar dipisahkan dan ditulis ulang dengan menggunakan variabel taut (*bind variable*), sehingga pernyataan yang diproses adalah:

```
(select count(*)
   from emp
   where emp.deptno = ?)
```

Karena data pada kolom `deptno` adalah unik untuk tabel `dept`, maka Oracle Database akan memroses *subquery* skalar tersebut untuk setiap baris data yang ada di tabel `dept`. Hasil pemrosesan ini disimpan di dalam memori dalam bentuk sebuah tabel *hash*, yang kemudian dipakai pada saat *query* utama diproses.

select count(*) from emp where emp.deptno=:deptno	
: deptno	**Count(*)**
…	…

Bagaimana jika ternyata tidak dilakukan *query* terhadap tabel `dept`, tetapi terhadap tabel lain yang mengandung kolom `deptno`? Misalnya terdapat sebuah tabel `projects` yang mengandung kolom `project_name` dan `deptno` sehingga *query* seperti:

```
select project_name, deptno,
     (select count(*)
        from emp
        where emp.deptno = projects.deptno)
  from projects;
```

mengakibatkan Oracle harus memroses ulang *subquery* skalar **maksimal** sebanyak nilai unik `deptno` yang terdapat di dalam tabel `projects`. Ditekankan di sini kata **maksimal**, berarti Oracle tidak perlu memroses ulang *subquery* skalar untuk setiap baris data di tabel `projects` jika hasilnya telah tersedia di dalam *cache*.

Oracle Database menggunakan tabel *hash* untuk mengingat setiap *subquery* skalar, nilai masukan, :`deptno` pada contoh di atas, dan nilai keluaran. Pada setiap awal eksekusi suatu *query*, *cache* dalam keadaan kosong. Misalkan query di atas dijalankan dan data pertama yang diperoleh dari tabel `projects` memiliki `deptno` bernilai 10. Oracle Database akan memeriksa apakah angka 10 telah ada di dalam tabel *hash*. Dalam kasus ini belum ada sehingga Oracle Database harus menjalankan *subquery* skalar dengan data `deptno=10` untuk mendapatkan jawabannya. Misalkan *subquery* skalar menghasilkan nilai 42, maka tabel *hash* akan terisi data seperti berikut:

select count(*) from emp where emp.deptno=:deptno	
: deptno	**Count(*)**
…	…
10	42
…	…

Seandainya data kedua yang diperoleh dari tabel `projects` memiliki `deptno` bernilai 20. Oracle Database akan kembali memeriksa apakah angka 20 telah ada di dalam tabel *hash*. Dalam kasus ini belum ada sehingga Oracle Database harus menjalankan *subquery* skalar dengan data `deptno=20` untuk mendapatkan jawabannya. Misalkan *subquery* skalar menghasilkan nilai 55, maka tabel *hash* akan terisi data seperti berikut:

select count(*) from emp where emp.deptno=:deptno	
: deptno	**Count(*)**
…	…
10	42
…	…
20	55
…	…

Sekarang andaikan data ketiga yang diperoleh dari tabel `projects` memiliki `deptno` bernilai 10. Oracle Database akan memeriksa tabel *hash* dan ternyata angka 10 telah ada di dalam tabel *hash*. Oracle Database tidak perlu memroses *subquery* skalar tersebut dan menggunakan hasil yang telah tercatat di tabel *hash*.

Tabel *hash* dapat menampung hingga 255 data unik, sesuai dengan kapasitas *cache* yang dimiliki oleh Oracle Database 10g dan Oracle Database 11g. Apa yang terjadi jika jumlah data unik `deptno` lebih dari 255, yakni melebihi kapasitas tabel *hash*? Jawabannya cukup sederhana, Oracle Database tidak dapat memasukkan data selebihnya ke dalam tabel *hash*. Dalam kasus ini *subquery* skalar hanya di-*cache* sebagian.

Contoh, jika Oracle Database menemukan *subquery* skalar dengan nilai `deptno=40` yang tidak tersedia di tabel *hash*, maka Oracle Database terpaksa memroses *subquery* skalar tersebut. Seandainya tabel *hash* telah penuh, maka hasil pemrosesan *subquery* skalar tersebut langsung digunakan oleh *query* utama dan tidak dapat disimpan di *cache*. Jika kemudian Oracle Database kembali menemukan *subquery* skalar dengan nilai `deptno=40`, maka Oracle Database terpaksa memroses kembali *subquery* skalar tersebut.

4. FUNGSI PL/SQL DAN SUBQUERY SKALAR

Jika dalam sebuah *query* terdapat pemanggilan fungsi PL/SQL, maka Oracle Database akan mengalihkan proses dari lingkungan SQL ke lingkungan PL/SQL. Pengalihan proses ini dapat membebani sistem basisdata. Jika pengalihan ini cukup sering terjadi, maka performansi basisdata akan menurunkan secara signifikan.

Cobalah buat sebuah simulasi sederhana untuk menunjukkan efek dari pengalihan proses dari lingkungan SQL ke lingkungan PL/SQL. Buat sebuah fungsi PL/SQL sederhana seperti di Kode 1. Fungsi ini mengembalikan sebuah nilai, yaitu jumlah karakter dari variabel masukan. Selain itu fungsi ini juga menambahkan data di tabel `V$SESSION` kolom `client_info` dengan nilai 1. Ini akan digunakan untuk mencatat berapa kali terjadi pemanggilan fungsi `f`.

Kode 1. Fungsi `f` pada PL/SQL

```
SQL> create or replace function f(x in varchar2)
     return number
  2  as
  3  begin
  4     dbms_application_info.set_client_info
        (userenv('client_info')+1);
  5     return length(x);
  6  end;
  7  /
Function created.
```

Sekarang cobalah panggil fungsi di atas berkali-kali, misalnya seperti pada Kode 2. Lakukan *query* terhadap tabel `ALL_OBJECTS` dan tampilkan kolom `owner` dan hitung panjang string dari kolom `owner` dengan menggunakan fungsi `f` yang telah dibuat di Kode 1. Tabel `ALL_OBJECTS` ini digunakan oleh Oracle Database untuk mencatat semua objek yang terdapat di dalam basisdata dan berisikan data yang cukup banyak, yaitu 72841 baris. Perhatikan bahwa hasil *query* dapat berbeda bergantung pada jumlah objek di dalam basisdata, namun ini tidak penting. Kemudian lakukan *query* untuk menampilkan waktu yang dibutuhkan oleh sistem basisdata memroses *query* tersebut dan jumlah pemanggilan fungsi. Dibutuhkan 118 milidetik untuk memroses *query* dan 72841 kali pemanggilan fungsi, sama dengan jumlah baris data karena memang demikian seharusnya.

Kode 2. Pemanggilan berulang terhadap fungsi f

```
SQL> begin
  2    :cpu := dbms_utility.get_cpu_time;
  3    dbms_application_info.set_client_info(0);
  4  end;
  5  /
PL/SQL procedure successfully completed.

SQL> select owner, f(owner) from ALL_OBJECTS;
...
72841 rows selected

SQL> select
  2    dbms_utility.get_cpu_time-:cpu cpu_hsecs,
  3    userenv('client_info')
  4    from dual;

CPU_HSECS      USERENV('CLIENT_INFO')
---------      ----------------------
      118      72841
```

Kode 3. Mengurangi pemanggilan fungsi dengan *subquery* skalar

```
SQL> begin
  2    :cpu := dbms_utility.get_cpu_time;
  3    dbms_application_info.set_client_info(0);
  4  end;
  5  /
PL/SQL procedure successfully completed.

SQL> select owner, (select f(owner) from dual) f
  2    from ALL_OBJECTS;
...
72841 rows selected

SQL> select
  2    dbms_utility.get_cpu_time-:cpu cpu_hsecs,
  3    userenv('client_info')
  4    from dual;

CPU_HSECS      USERENV('CLIENT_INFO')
---------      ----------------------
       29      66
```

Sekarang terapkan teknik *subquery* skalar. Pemanggilan fungsi `f` dilakukan di dalam sebuah *subquery* terhadap tabel `dual`, lihat Kode 3. Tabel `dual` adalah sebuah tabel skalar yang disediakan oleh Oracle Database untuk keperluan *query* skalar. Dapat dilhat bahwa hasil *query* ini sama saja dengan hasil *query* pada Kode 2. Tetapi karena Oracle Database menggunakan *cache* untuk memroses *subquer*y skalar, *query* ini hanya membutuhkan waktu proses 29 milidetik dengan pemanggilan fungsi sebanyak 66 kali saja. Pemanggilan fungsi sebanyak 66 kali karena terdapat 66 data

unik pada kolom `owner` di dalam tabel `ALL_OBJECTS`. Dari 72841 kali pemanggilan fungsi menjadi hanya 66 kali, sebuah penghematan yang luar biasa.

Kode 4. Penggunaan fungsi deterministic

```
SQL> create or replace function f(x in varchar2)
     return number
  2  DETERMINISTIC
  3  as
  4  begin
  5     dbms_application_info.set_client_info
        (userenv('client_info')+1);
  6     return length(x);
  7  end;
  8  /
Function created.

SQL> begin
  2    :cpu := dbms_utility.get_cpu_time;
  3    dbms_application_info.set_client_info(0);
  4  end;
  5  /
PL/SQL procedure successfully completed.

SQL> select owner, f(owner) from ALL_OBJECTS;
...
72841 rows selected

SQL> select
  2    dbms_utility.get_cpu_time-:cpu cpu_hsecs,
  3    userenv('client_info')
  4    from dual;

CPU_HSECS      USERENV('CLIENT_INFO')
---------      ----------------------
       69      8316
```

Apakah tidak ada teknik lain untuk mengurangi jumlah pemanggilan fungsi? Bagi yang cukup menguasai perintah PL/SQL akan menganjurkan untuk menggunakan kata kunci `DETERMINISTIC` pada saat mendefinikan sebuah fungsi PL/SQL, seperti pada Kode 4. Pada kenyataannya kolom `owner` di dalam tabel `ALL_OBJECTS` memang berisikan data yang bernilai sama, sehingga kata kunci `DETERMINISTIC` tampaknya akan sangat membantu. Dari Kode 4 dapat dilihat bahwa waktu pemrosesan *query* turun menjadi 69 milidetik dan pemanggilan fungsi juga turun menjadi 8316 kali. Ternyata penggunaan *subquery* skalar masih jauh lebih unggul. Sebagai catatan, kata kunci `DETERMINISTIC` hanya efektif untuk Oracle Database versi 10g ke atas.

Pada Oracle Database versi 11g, tersedia kata kunci RESULT_CACHE untuk fungsi PL/SQL. Penggunaan kata kunci RESULT_CACHE menyebabkan Oracle Database akan menyimpan hasil pemrosesan fungsi PL/SQL di dalam *cache*. Fitur ini cukup menarik untuk dicoba sejauh mana efektivitasnya. Dari Kode 5 dapat dilihat bahwa waktu pemrosesan *query* turun menjadi 73 milidetik dan pemanggilan fungsi juga turun menjadi 32 kali. Waktu pemrosesan ini masih kalah sedikit dari penggunaan kata kunci DETERMINISTIC.

Kode 5. Penggunaan RESULT_CACHE pada fungsi

```
SQL> create or replace function f(x in varchar2)
     return number
  2  RESULT_CACHE
  3  as
  4  begin
  5     dbms_application_info.set_client_info
        (userenv('client_info')+1);
  6     return length(x);
  7  end;
  8  /
Function created.

SQL> begin
  2    :cpu := dbms_utility.get_cpu_time;
  3    dbms_application_info.set_client_info(0);
  4  end;
  5  /
PL/SQL procedure successfully completed.

SQL> select owner, f(owner) from ALL_OBJECTS;
...
72841 rows selected

SQL> select
  2    dbms_utility.get_cpu_time-:cpu cpu_hsecs,
  3    userenv('client_info')
  4    from dual;

CPU_HSECS      USERENV('CLIENT_INFO')
---------      ----------------------
       73      32
```

Mengapa jumlah pemanggilan fungsi RESULT_CACHE yang hanya 32 kali tetapi waktu pemrosesannya masih kalah dari fungsi DETERMINISTIC yang 8316 kali pemanggilan fungsi? Kata kunci RESULT_CACHE hanya mengurangi jumlah pemanggilan fungsi, tetapi ternyata tidak mengurangi jumlah pengalihan proses dari lingkungan SQL ke lingkungan PL/SQL, yaitu tetap 72841 kali. Ini yang menyebabkan hasil *subquery* skalar jauh lebih efisien dari cara lainnya.

Salah satu hal yang menarik dari kata kunci `RESULT_CACHE` adalah jika menjalankan *query* yang sama satu kali lagi, maka jumlah pemanggilan fungsi akan turun menjadi 0 kali (lihat Kode 6). Artinya tidak ada lagi pemanggilan fungsi karena sudah tersedia di *cache*. Tetapi perhatikan bahwa waktu pemrosesan *query* hanya turun sedikit menjadi 63 milidetik dan masih kalah jauh dari teknik *subquery* skalar yang hanya membutuhkan waktu 29 milidetik.

Kode 6. Pemanggilan ulang fungsi dengan `RESULT_CACHE`

```
SQL> begin
  2    :cpu := dbms_utility.get_cpu_time;
  3    dbms_application_info.set_client_info(0);
  4  end;
  5  /
PL/SQL procedure successfully completed.

SQL> select owner, f(owner) from ALL_OBJECTS;
...
72841 rows selected

SQL> select
  2    dbms_utility.get_cpu_time-:cpu cpu_hsecs,
  3    userenv('client_info')
  4    from dual;

CPU_HSECS      USERENV('CLIENT_INFO')
---------      ----------------------
       63      0
```

5. KESIMPULAN

Dari contoh-contoh di atas dapat ditarik kesimpulan bahwa walaupun telah dibuat fungsi dengan kata kunci `DETERMINISTIC` atau `RESULT_CACHE`, lebih bijaksana jika pemanggilan fungsi tersebut selalu 'dibungkus' dengan `SELECT FROM DUAL` agar dapat dinikmati performansi dari teknologi *cache subquery* skalar.

Teknologi *cache* pada Oracle Database yang digunakan oleh untuk memroses *subquery* skalar sangat efektif untuk meningkatkan waktu respon pada pernyataan SQL murni. Selain itu juga sangat efektif untuk *query* yang melibatkan fungsi PL/SQL karena dapat menghindari terjadinya penurunan performansi akibat pengalihan proses dari lingkungan SQL ke lingkungan PL/SQL.

6. DAFTAR PUSTAKA

[1]. Kevin Loney, *Oracle Database 10g - The Complete Reference*, Oracle Press, 2004

[2]. Lance Ashdown & Thomas Kyte, *Oracle Database Concepts 11g Release 2(11.2)*, Oracle Press, 2011

[3]. Thomas Kyte, *Effective Oracle by Design*, Oracle Press, 2003

[4]. Thomas Kyte, *Expert Oracle Database Architecture: 9i and 10g rogramming Techniques and Solutions*, Apress, 2005

[5]. Thomas Kyte, *Expert Oracle Database Architecture: Oracle Database 9i, 10g, and 11g Programming Techniques and Solutions*, Second edition, Apress, 2010