MENINGKATKAN EFEKTIFITAS PENGELOLAAN DATABASE DENGAN OPTIMASI SQL

Arifin

Abstraksi : *Database merupakan kumpulan informasi dari suatu organisasi yang tersimpan secara elektronis dan dapat diolah untuk menghasilkan informasi yang dapat memenuhi kebutuhan informasi organisasi tersebut. Untuk mengelola database dibutuhkan bahasa basis data yang dikenal dengan SQL (Structure Query Language). Dengan SQL database dapat diubah, dihapus, dikelompokan dan diorganisasikan agar menghasilkan informasi yang dibutuhkan. Database yang kompleks dan kebutuhan informasi yang cepat membutuhkan penanganan SQL yang kompleks pula, sehingga biasanya menimbulkan masalah ketidakefektifan. Oleh karena dibutuhkan penanganan SQL yang efektif dengan optimasi SQL.*

Kata Kunci : *SQL, Database, Optimasi*

PENDAHULUAN

Database

Basis data (bahasa Inggris: database), adalah kumpulan informasi yang disimpan di dalam komputer secara sistematik sehingga dapat diperiksa menggunakan suatu program komputer untuk memperoleh informasi dari basis data tersebut. Atau dengan istilah lainnya database adalah kumpulan data yang diorganisasikan sedemikian rupa sehingga pengguna dapat mengakses, mengambil, dan menggunakan data.

SQL

SQL (*Structured Query Language*) merupakan sebuah bahasa atau pemrograman standar untuk RDBMS (*Relational Database Management System*). Walaupun disebut bahasa, mungkin sedikit janggal saat kita menyebut bahasa pemrograman SQL, lebih familiar jika yang terdengar adalah pemrograman C, Visual Basic, Java, Delphi, dan seterusnya.

Bahasa-bahasa yang disebut belakangan termasuk dalam pemrograman imperative, mudahnya adalah bahasa yang berbentuk instruksi-instruksi inti. Sedangkan, SQL termasuk dalam pemrograman declarative, yang lebih berbentuk kalimat atau pernyataan.

SQL paling tidak mempunyai dua macam perintah yang digunakan untuk mengelola dan mengorganisasikan basis data, yaitu :

1. *Data Definition Language* (DDL)

   Adalah perintah-perintah yang biasa digunakan oleh administrator basis data (DBA) untuk mendefinisikan skema ke dalam DBMS. Skema adalah deskripsi lengkap tentang struktur table, rekaman dan hubungan data pada basis data.

   DDL juga digunakan untuk mendefinisikan subskema. Subskema adalah pandangan (view) bagi pengguna basis data yang merupakan himpunan bagian dari skema. Bila suatu item tidak tercantum dalam skema seseorang pengguna, maka item tersebut tidak tersedia bagi pengguna bersangkutan. Subskema dapat menjadi mekanisme pengamanan system basis adata, yakni dengan mengatur hak pengaksesan item-item dalam basis data. DDL juga digunakan untuk menciptakan, mengubah dan menghapus basis data.

2. *Data Manipulation Language* (DML)

   Adalah perintah-perintah yang digunakan untuk mengubah, memanipulasi dan mengambil data pada basis data. Tindakan seperti menghapus, mengubah dan mengambil data menjadi bagian dari DML. DML pada dasarnya dibagi menjadi dua :

   a. Prosedural, yang menuntut pengguna menentukan data apa saja yang diperlukan dan bagaimana cara mendapatkannya.
   b. Nonprosedural, yang menuntut pengguna menentukan data apa saja yang diperlukan, tetapi tidak perlu menyebutkan cara mendapatkannya.

Optimasi dapat dilakukan dengan berbagai cara, dengan memahami tuning performance pada database. Beberapa teknik dan metoda mungkin memerlukan perlakuan khusus yang berbeda, tergantung pada database yang digunakan. Sebagai contoh, peningkatan kinerja bisa dilakukan dari sisi administrasi database seperti konfigurasi file dan pengupdatean service atau security pack, yang tentunya masing-masing database memiliki keunikan dan teknik tersendiri.

Terdapat seperangkat metode dan teknik yang umum diterapkan dengan RDBMS, mungkin tidak semuanya dapat diimplementasikan karena sangat tergantung pada lingkungan aplikasi masing-masing, tetapi setidaknya dapat digunakan sebagai panduan dan referensi untuk membentuk sistem yang terbaik sesuai dengan kondisi yang dihadapi.

Optimasi melalui perintah SQL juga memegang peranan yang tidak kalah penting. Inti dari SQL itu sendiri adalah perintah untuk melakukan pengambilan (*retrieval*), penambahan (*insertion*), modifikasi (*updating*), dan penghapusan (*deletion*) data, disertai dengan fungsi-fungsi pendukung administrasi dan managemen database.

SQL mempunyai struktur yang cukup sederhana, terdiri dari tiga operasi *SELECT, FROM* dan *WHERE*. Struktur tersebut dapat dituliskan :

Select field1, field2, field3, …..
From <*table*>
Where <*condition*>

Dimana :
Table adalah nama tabel yang akan digunakan dalam query tersebut.
Condition adalah kondisi yang disyaratkan (bentuknya adalah ekspresi logika)

PEMBAHASAN

Optimasi SQL

Pada pembahasan ini, untuk meningkatkan efektifitas pengelolaan database ditekankan pada beberapa optimasi sederhana yang dapat dilakukan. Cara-cara berikut ini setidaknya memperbaiki atau mencegah permasalahan, dan meningkatkan performa RDBMS.

1. **Index**

   Index dapat meningkatkan kecepatan pencarian pada record yang diinginkan. Tetapi, yang harus diperhatikan adalah bagaimana memilih field yang digunakan untuk kunci index, karena tidak semua field memerlukannya.

   Ibaratnya membaca buku, proses pencarian akan membaca dari awal hingga akhir halaman. pada field yang di-index, pencarian dilakukan secara index scan, atau membaca pada index, tidak langsung pada tabel yang bersangkutan. Sementara pencarian yang dilakukan langsung dengan membaca record demi record pada tabel disebut dengan *table scan*.

   Apakah *index scan* selalu lebih cepat dibandingkan dengan table scan? Ternyata tidak juga, *table scan* bisa jadi bekerja lebih cepat saat mengakses record dalam jumlah relatif kecil, ataupun pada saat aplikasi memang memerlukan pembacaan tabel secara keseluruhan. Sebaliknya dalam mengakses record yang besar pada field tertentu, *index scan* dapat mengurangi operasi pembacaan I/O sehingga tidak jarang menghasilkan kinerja yang lebih cepat. Sebagai patokan, dapat ditentukan index pada field yang sering digunakan, misalnya field yang sering diakses oleh klausa *Where, Join, Order By, Group By*.

2. **Menentukan Tipe Data**

   Tipe data merupakan permasalahan yang gampang-gampang susah. Dari sisi daya tampung, tipe data yang terlalu kecil atau sebaliknya terlalu besar bagi suatu field, dapat menimbulkan bom waktu yang menimbulkan masalah seiring dengan pertambahan data yang pesat setiap harinya.

   Menentukan tipe data yang tepat memerlukan ketelitian dan analisa yang baik. Sebagai contoh, kita perlu mengetahui kapan kita menggunakan tipe data char atau varchar. Keduanya menampung karakter, bedanya char menyediakan ukuran penyimpanan yang tetap (*fixed-length*), sedangkan varchar menyediakan ukuran penyimpanan sesuai dengan isi data (*variable-length*).

   Patokan umum adalah menggunakan tipe data char jika field tersebut diperuntukkan untuk data dengan panjang yang konsisten. Misalnya kode pos, bulan yang terdiri dari dua digit (01 sampai 12), dan seterusnya. Varchar digunakan jika data yang ingin disimpan memiliki panjang yang bervariasi, atau gunakan varchar(max) jika ukurannya melebihi 8000 byte.

3. **Menghindari Field bernilai *Null***

   Jika memungkinkan, kurangi penggunaan field yang memperbolehkan nilai null. Sebagai gantinya, dapat diberikan nilai default pada field tersebut. Nilai null kadang rancu dalam intepretasi programer dan dapat mengakibatkan kesalahan logika pemrograman. Selain itu, field null mengonsumsi byte tambahan sehingga menambah beban pada query yang mengaksesnya.

4. **Query yang Mudah Terbaca**

   Karena SQL merupakan bahasa declarative, maka tidak mengherankan jika Anda membuat query berbentuk kalimat yang panjang walaupun mungkin hanya untuk keperluan menampilkan satu field. Jangan biarkan query susah dibaca dan dipahami, kecuali memang berniat membuat pusing siapapun yang melihat query tersebut. Query panjang yang ditulis dalam 1 baris jelas akan menyulitkan modifikasi dan pemahaman, akan jauh lebih baik jika menuliskan query dalam format yang mudah dicerna.

Pemilihan huruf besar dan kecil juga dapat mempermudah pembacaan, misalnya dengan konsisten menuliskan keyword SQL dalam huruf kapital, dan tambahkan komentar bilamana diperlukan.

5. **Hindari *SELECT ****

   Select mungkin merupakan keyword yang paling sering digunakan, karena itu optimasi pada perintah *SELECT* sangat mungkin dapat memperbaiki kinerja aplikasi secara keseluruhan. *SELECT* * digunakan untuk melakukan query semua field yang terdapat pada sebuah tabel, tetapi jika hanya ingin memproses field tertentu, maka sebaiknya dituliskan field yang ingin diakses saja, sehingga query menjadi *SELECT field1, field2, field3* dan seterusnya (jangan pedulikan kode program yang menjadi lebih panjang). Hal ini akan mengurangi beban lalu lintas jaringan dan lock pada tabel, terutama jika tabel tersebut memiliki banyak field dan berukuran besar.

6. **Batasi Printah *ORDER BY***

   Penggunaan *ORDER BY* yang berfungsi untuk mengurutkan data, ternyata memiliki konsekuensi menambah beban query, karena akan menambah satu proses lagi, yaitu proses sort. Karena itu gunakan *ORDER BY* hanya jika benar-benar dibutuhkan oleh aplikasi tersebut. Atau jika dimungkinkan, dapat dilakukan pengurutan pada sisi client dan tidak pada sisi server. Misalnya dengan menampung data terlebih dahulu pada komponen grid dan melakukan sortir pada grid tersebut sesuai kebutuhan pengguna.

7. ***Subquery* Atau *JOIN***

   Adakalanya sebuah instruksi dapat dituliskan dalam bentuk subquery atau perintah *JOIN*, disarankan memprioritaskan penggunaan *JOIN* karena dalam kasus yang umum akan menghasilkan performa yang lebih cepat. Walaupun demikian, mengolah query merupakan suatu seni, selalu ada kemungkinan ternyata subquery bekerja lebih cepat dibandingkan *JOIN*, misalnya dalam kondisi penggunaan *JOIN* yang terlalu banyak, ataupun logika query yang belum optimal.

8. Gunakan *WHERE* dalam *SELECT*

Sebagai programer database, pasti sangat familier dengan pepatah "di mana ada *SELECT* di sana ada *WHERE*", untuk mengingatkan pentingnya klausa *WHERE* sebagai kondisi untuk menyaring record sehingga meminimalkan beban jaringan.

Saat sebuah tabel dengan jumlah data yang sangat besar diproses, juga terjadi proses *lock* terhadap tabel tersebut sehingga menyulitkan pengaksesan tabel yang bersangkutan oleh pengguna yang lain. Bahkan jika bermaksud memanggil seluruh record, tetap menggunakan *WHERE* merupakan kebiasaan yang baik. Jika telah menggunakan *WHERE* pada awal query, maka kapanpun ingin ditambahkan kondisi tertentu, tinggal menyambung query tersebut dengan klausa *AND* diikuti kondisi yang diinginkan.

Tapi bagaimana menggunakan *WHERE* jika benar-benar tidak ada kondisi apapun? Kita dapat menuliskan suatu kondisi yang pasti bernilai true, misalnya *SELECT .... WHERE 1=1*.

Bahkan tools *open source phpMyAdmin* yang berfungsi untuk mena ngani database MySQL selalu menyertakan default klausa *WHERE 1* pada perintah *SELECT*, di mana angka 1 pada MySQL berarti nilai true.

9. Kecepatan Akses Operator

WHERE 1=1 dan *WHERE 0 <> 1* sama-sama merupakan kondisi yang menghasilkan nilai true. Tetapi, dalam hal ini lebih baik digunakan *WHERE 1=1* daripada *WHERE 0 <> 1*. Hal ini dikarenakan operator = diproses lebih cepat dibandingkan dengan operator <>. Dari sisi kinerja, urutan operator yang diproses paling cepat adalah:

1. =
2. >, >=, <. <=
3. LIKE
4. <>

Tidak dalam setiap kondisi operator dapat disubtitusikan seperti contoh sederhana di atas, tetapi prioritaskanlah penggunaan operator yang tercepat.

10. Membatasi Jumlah Record

Bayangkan apabila akan ditampilkan isi sebuah tabel dengan menggunakan *SELECT*, dan ternyata tabel tersebut memiliki jutaan record yang sangat tidak diharapkan untuk tampil

seluruhnya. Skenario yang lebih buruk masih dapat terjadi, yaitu query tersebut diakses oleh ratusan pengguna lain dalam waktu bersamaan. Oleh karena itu, perlu dibatasi jumlah record yang berpotensi mengembalikan record dalam jumlah besar (kecuali memang benar-benar dibutuhkan), pada SQL Server, Anda dapat menggunakan operator *TOP* di dalam perintah *SELECT*.

Contohnya :

SELECT TOP 100 nama... (akan menampilkan 100 record teratas field nama).

Jika menggunakan MySQL, dapat digunakan perintah LIMIT untuk keperluan yang sama.

11. Batasi Penggunaan Function

Gunakan fungsi-fungsi yang disediakan SQL seperlunya saja. Sebagai contoh, jika ditemukan query sebagai berikut:

SELECT nama FROM tbl_teman WHERE ucase(nama) = 'ABC';

Tampak query tersebut ingin mencari record yang memiliki data berisi "abc", fungsi ucase digunakan untuk mengubah isi field nama menjadi huruf besar dan dibandingkan dengan konstanta "ABC" untuk meyakinkan bahwa semua data "abc" akan tampil, walaupun dituliskan dengan huruf kecil, besar, ataupun kombinasinya.

Tetapi, cobalah mengganti query tersebut menjadi :

SELECT nama FROM tbl_teman WHERE nama = 'ABC';

Query di atas tidak menggunakan function ucase. Apakah menghasilkan *result* yang sama dengan query pertama? Jika pengaturan database tidak case-sensitive (dan umumnya secara default memang tidak case-sensitive), maka hasil kedua query tersebut adalah sama. Artinya, dalam kasus ini, sebenarnya tidak perlu menggunakan function ucase.

12. Baca dari Kiri ke Kanan

Query yang Anda tulis akan diproses dari kiri ke kanan, misalkan terdapat query *WHERE kondisi1 AND kondisi2 AND kondisi3*, maka kondisi1 akan terlebih dahulu dievaluasi, lalu

kemudian kondisi2, kondisi3, dan seterusnya. Tentunya dengan asumsi tidak ada kondisi yang diprioritaskan/dikelompokkan dengan menggunakan tanda kurung.

Logika operator *AND* akan langsung menghasilkan nilai *false* saat ditemukan salah satu kondisi false, maka letakkan kondisi yang paling mungkin memiliki nilai *false* pada posisi paling kiri. Hal ini dimaksudkan agar SQL tidak perlu lagi mengevaluasi kondisi berikutnya saat menemukan salah satu kondisi telah bernilai *false*.

Jika bingung memilih kondisi mana yang layak menempati posisi terkiri karena kemungkinan *false*-nya sama atau tidak bisa diprediksi, pilih kondisi yang lebih sederhana untuk diproses.

13. Gambar dalam Database

Database memang tidak hanya diperuntukkan sebagai penyimpanan teks saja, tetapi dapat juga berupa gambar. Kalau pepatah mengatakan sebuah gambar bermakna sejuta kata, tidak berarti kita harus menyediakan tempat penyimpanan seukuran sejuta kata untuk menampung satu gambar. Akan lebih baik bagi kinerja database jika kita hanya menyimpan link atau lokasi gambar di dalam database, dibandingkan menyimpan file fisik gambar tersebut.

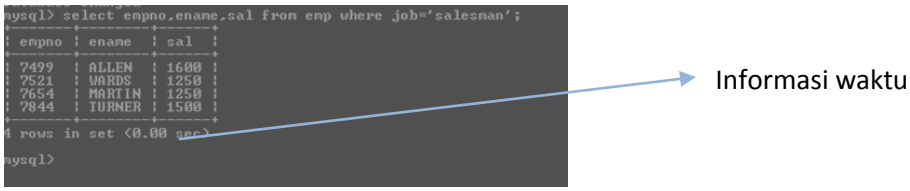
Kecuali jika tidak memiliki pilihan lain, misalnya karena alasan keamanan atau tidak tersedianya tempat penyimpanan lain untuk gambar selain di dalam database.

Tetapi, jelas jika dapat memisahkan gambar secara fisik dari database, maka ukuran dan beban database akan relatif berkurang drastis, proses seperti *back-up* dan migrasi akan lebih mudah dilakukan.

Pengukuran Kinerja

Terdapat tools optimizer yang bervariasi untuk tiap RDBMS, Kita dapat menggunakannya sebagai panduan untuk meningkatkan kinerja query. Dengan tools tersebut dapat diketahui berapa lama waktu eksekusi atau operasi apa saja yang dilakukan sebuah query.

Informasi mengenai berapa lama waktu yang dibutuhkan untuk menkases query, juga dapat dilihat dibagian bawah output query yang ditampilkan.



Gambar 1. Contoh Query

Jika Anda menemukan sebuah query tampak tidak optimal, berusahalah menulis ulang query tersebut dengan teknik dan metode yang lebih baik. Semakin banyak query yang dapat dioptimasi, akan semakin baik kinerja aplikasi Anda. Terutama saat frekuensi pemakaian query tersebut relatif tinggi.

Back-up

Buatlah back-up otomatis secara periodik, sebaiknya tes dan simulasikan prosedur restore database dan perhitungkan waktu yang diperlukan untuk membuat sistem pulih kembali jika terjadi sesuatu yang tidak diharapkan pada database. Lakukan proses back-up pada waktu di mana aktivitas relatif rendah agar tidak mengganggu kegiatan operasional.

Alternatif Lain

Jika kita melihat kemungkinan peningkatan kinerja karena penulisan query yang kurang baik, contohnya pada potongan query berikut:

Select nama from mahasiswa WHERE SUBSTRING(nama,1,1) ='b';

Query di atas akan mengambil record dengan kondisi karakter pertama kolom nama adalah "b", sehingga akan tampil isi record seperti "Budi", "Badu", "Benny" dan seterusnya. Cara lain untuk menghasilkan record yang sama adalah sebagai berikut:

Select nama from mahasiswa WHERE nama LIKE 'b%';

Hasil yang ditampilkan kedua query tersebut akan sama, tetapi performa yang dihasilkan (terutama untuk record berukuran besar) akan berbeda. Umumnya kondisi LIKE akan bekerja

dengan lebih cepat dibandingkan function SUBSTRING. Contoh lain yang lebih kompleks adalah seperti query berikut:

SELECT NIP, nama FROM tbl_pegawai WHERE dept = 'IT' OR kota = 'jakarta' OR divisi = 'programer';

Perhatikan query di atas memiliki tiga kondisi yang dipisahkan oleh klausa OR. Alternatif lain adalah dengan menuliskan query sebagai berikut:

```
SELECT NIP, nama FROM tbl_pegawai WHERE dept = 'IT'
    UNION ALL
      SELECT NIP, nama FROM tbl_pegawai WHERE kota = 'jakarta'
        UNION ALL
          SELECT NIP, nama FROM tbl_pegawai WHERE divisi = 'programer';
```

Walaupun penulisan query menjadi lebih panjang, bisa jadi alternatif ini akan lebih baik. Hal ini disebabkan field dept memiliki index, sementara field kota dan divisi tidak diindex, query pertama tidak akan menggunakan index dan melakukan *table scan*. Berbeda dengan query kedua, index akan tetap dilakukan pada sebagian query sehingga akan menghasilkan kinerja yang relatif lebih baik.

Pastinya masih banyak terdapat teknik lain yang tidak akan dapat dibahas semuanya dalam artikel ini. Di antara (atau mungkin semua) teknik optimasi yang dibahas di atas, mungkin akan ditemukan bahwa setelah diuji dengan data sampel maka kinerja sebelum dan sesudah optimasi ternyata sama sekali tidak signifikan perbedaannya, beda tipis, atau tidak ada bedanya sama sekali. Memang benar, dengan spesifikasi hardware yang semakin meningkat, data yang relatif kecil, dan alur yang sederhana, mungkin tidak akan mendapatkan perbedaan yang signifikan. Tetapi jika kita siap untuk terjun menghadapi tantangan menangani aplikasi yang lebih besar, maka perbedaan antara tanpa dan dengan optimasi akan sangat nyata, dengan pemahaman dan kebiasaan coding yang baik, kita akan dapat menghasilkan aplikasi yang juga lebih baik. Tidak ada salahnya menerapkan optimasi yang diketahui sedini mungkin dalam pengembangan sistem aplikasi tersebut.

Bahkan jika sebuah aplikasi tampaknya memiliki kinerja yang cukup baik, tidak berarti lepas dari usaha optimasi lebih lanjut. Terutama jika diharapkan aplikasi tersebut mampu berkembang lebih jauh, tidak pernah ada kata sempurna bagi suatu sistem aplikasi, tetapi setiap sistem selalu ada kesempatan menjadi lebih berguna. Salah satunya dengan selalu mencari cara yang lebih baik

KESIMPULAN

Berdasarkan pembahasan di atas, dapat diambil beberapa simpulan sebagai berikut :

1. Perlu adanya cara untuk penulisan query yang efektif, sehingga SQL dapat bekerja dengan lebih efektif dalam menghasilkan informasi.
2. Optimasi SQL dapat diukur berdasarkan jumlah waktu yang dibutuhkan dalam mengakses query tersebut, semakin sedikit waktu yang dibutuhkan berarti query lebih efektif.

DAFTAR PUSTAKA

1. Aripin, "Modul Praktikum Basis Data Dengan Database Server MySQL"
2. Bunafit Nugroho, "Administrasi Database MySQL", Graha Ilmu, Yogyakarta, 2005
3. Henry F. Korth, "*Database System Concepts",* McGraw-Hill Company, 2002
4. Paul DuBois, "MySQL", Indianapolis, Indiana, 2000
5. http://www.vibiru.co.cc/virtual-work/website/189-optimasi-sql.pdf, diakses Tanggal 15 Mei 2010.