Penggunaan Index di SQL Server 2005

Rangga Praduwiratna
ziglaret@yahoo.co.nz
http://geeks.netindonesia.net/blogs/ziglaret

Lisensi Dokumen:
Copyright © 2003-2007 IlmuKomputer.Com
Seluruh dokumen di IlmuKomputer.Com dapat digunakan, dimodifikasi dan disebarkan secara bebas untuk tujuan bukan komersial (nonprofit), dengan syarat tidak menghapus atau merubah atribut penulis dan pernyataan copyright yang disertakan dalam setiap dokumen. Tidak diperbolehkan melakukan penulisan ulang, kecuali mendapatkan ijin terlebih dahulu dari IlmuKomputer.Com.

Pendahuluan

Penggunaan index pada database merupakan salah satu teknik pembuatan database yang baik. Hal ini terutama sangat berguna pada implementasi database dengan skala VLDB (*Very Large Database*) atau OLDB (*Online Large Database*). Saat database dibuat tanpa menggunakan index, maka kinerja server database dapat menurun secara drastis. Hal ini dikarenakan *resource* komputer banyak digunakan untuk pencarian data atau pengaksesan *query* SQL dengan metode *table-scan*. Index pada kolom-kolom tabel database mempunyai fungsi seperti indeks kamus atau indeks buku. Hal ini membuat pencarian data akan lebih cepat dan tidak banyak menghabiskan *resource* komputer. Oleh karena itu, memiliki pemahaman akan konsep indexing, teknik implementasi, dan memahami kapan saat tepat untuk menerapkan metode ini merupakan hal yang berguna terutama bagi para *enterprise application developer* atau *database administrator*.

Konsep *Heap* dan *Table Scan* pada SQL Server 2005

Sebelum saya membahas lebih lanjut mengenai indexing pada SQL Server 2005, saya akan sedikit membahas mengenai metode table-scan yang saya singgung sedikit sebelumnya.

Tabel pada sebuah database yang tidak menggunakan index (disebut heaps) menggunakan metode table-scan saat dilakukan pencarian data atau pengaksesan query SQL. Apa yang dimaksud dengan *table-scan*? *Table scan* dapat diumpamakan seperti mencari sebuah arti kata dalam pada sebuah buku yang tidak memiliki indeks huruf.

Jika kita ingin mengetahui arti kata ‘mobil’ di Kamus Besar Bahasa Indonesia (KBBI), apa yang kita lakukan? Pertama-tama, kita tentu akan membuka kamus berdasarkan index hurufnya bukan? Kita akan mencari huruf ‘m’, kemudian meneruskan pencarian hingga menemukan halaman yang mengandung kata ‘mo’, dan mengakhiri proses pencarian hingga kita menemukan kata mobil. Mudah bukan?

Nah, sekarang bayangkan jika Anda mencari arti kata tertentu pada sebuah buku yang tidak mempunyai indeks huruf. Anda mungkin akan mencari dari awal halaman hingga menemukan kata yang Anda cari. Nah, perumpamaan *table-scan* adalah seperti hal ini. *Table-scan* akan mencari data pada tabel database dari awal hingga menemukan data yang dicari.

Anda tentu akan heran saat ini. Lho, bukannya data pada tabel tersebut sudah terurut berdasarkan abjad berdasarkan kolom tabel tertentu? Nah, konsep ini sedikit salah. Anda mungkin saja melihat hasil pencarian data yang telah terurut, namun perlu diingat bahwa sistem penyimpanan data pada SQL Server adalah melalui *pages*. Setiap *pages* hanya terdiri dari 8Kb data Anda, dan 8 buah *pages* (64kb) tersebut akan disimpan pada sebuah *extent*. Walaupun data yang dimunculkan terlihat terurut, namun sebenarnya cara pencarian data tersebut tidak dilakukan dengan cara yang saya umpamakan tadi (index huruf pada KBBI), bahkan data yang tersimpan tidak terurut sama sekali!

Begini ilustrasi sederhananya : Saat Anda akan mencari data, maka pencarian data akan dimulai dari *pages* pertama hingga *pages* terakhir dari sebuah *extent*, jika tidak ditemukan maka akan dilanjutkan ke *extent* berikutnya. Nah, mengapa metode ini memiliki kelemahan? Hal ini dikarenakan saat kita memasukkan data pada tabel tertentu, data tidak secara otomatis disimpan secara terurut, namun disimpan pada pages yang masih bisa menyimpan data. Jadi saat Anda memasukkan data yang harusnya berada di urutan 2 pada tabel, data ini akan disimpan pada *pages* terakhir yang masih bisa menyimpan data. Hal inilah yang seringkali membuat proses pencarian data menjadi lebih lama pada database yang tidak memiliki index, terutama pada

Copyright © 2003-2007 IlmuKomputer.Com

database dengan skala OLDB (Online Large Database) dan VLDB (Very Large Database).

Perlu diingat juga, bahwa saat Anda menerapkan index pada kolom-kolom tabel database, data Anda tidak akan terurut secara fisik. Hal ini berarti, data Anda tidak diurutkan secara terurut pada harddisk Anda. Data pada tabel database Anda akan terurut secara *logical* pada level *pages* dan *extent*. *Clustered* atau *non-clustered key* inilah yang membantu Anda saat melakukan pencarian data.

Penggunaan Index pada SQL Server 2005

Jika Anda sudah memahami kekurangan metode *table-scan*, maka kita dapat melangkah ke pembahasan mengenai index. Index pada dasarnya dapat dibagi menjadi 2, yaitu *clustered index* dan *non-clustered index*.

Clustered Index

Clustered index dapat diumpamakan seperti index huruf pada sebuah kamus. Saya telah memberikan ilustrasinya pada bab sebelumnya. Penggunaan clustered index akan membuat pencarian data lebih cepat. Hal ini dikarenakan, saat Anda memasukkan sebuah data baru, maka SQL Server akan memaksa untuk memasukkan data tersebut pada urutan yang seharusnya. Anda tentu bertanya bagaimana cara SQL tahu urutan yang sebenarnya dan kita inginkan. Anda perlu menerapkan clustered index pada kolom tabel yang paling sering diakses oleh user. Oleh karena itu, Microsoft sangat menyarankan pembuatan kolom tabel yang spesifik. Misalnya, jika kita membuat TabelKTP pada database DataKaryawan, maka daripada Anda membuat tabel seperti ini :

ID	NamaLengkap	TempatTanggal Lahir	Alamat
Int	Varchar(50)	Varchar(50)	Varchar(100)
105..	Adi Sasono	Bandung, 28 Oktober 1980	Jl. Sangkuriang 9 Bandung 40116
106..	Budi Sucipto	Sukabumi, 9 Januari 1981	Jl. Teuku Umar 48 Bandung 40117
107..			
108..			

Tabel 1. TabelKTP seperti ini kurang spesifik dan dapat menyebabkan masalah saat dilakukan pencarian data

,akan jauh lebih baik jika Anda membagi-bagi tabel menjadi lebih spesifik, seperti ini :

Copyright © 2003-2007 IlmuKomputer.Com

ID	NamaDepan	NamaBelakang	TempatLahir	TanggalLahir	Alamat	Kota	KodePos
Int	Varchar(20)	Varchar(20)	Varchar(15)	Smalldatetime	Varchar(50)	Varchar(15)	Int
105..	Adi	Sasono	Bandung	28-10-1980	Jl. Dago 4	Bandung	40116
106..	Budi	Sucipto	Sukabumi	9-1-1981	Jl. Otista 48	Bandung	40117
107..							
108..							

Tabel 2. TabelKTP yang kolom-kolomnya telah dibuat lebih spesifik

Nah, misalnya karena pencarian data pada database KTP lebih banyak menggunakan parameter nama belakang, maka Anda dapat menerapkan *clustered index* pada kolom *Nama Belakang*. Hal ini akan membuat proses pencarian data pada server database lebih cepat.

Bagaimana cara menerapkan clustered index pada tabel database? Anda dapat menggunakan query CREATE…INDEX untuk membuat clustered index. Berikut adalah sintaks T-SQL pembuatan clustered index :

```
CREATE [ UNIQUE ] [ CLUSTERED ] INDEX index_name
ON <object> ( column [ ASC | DESC ] [ ,...n ] )
[ INCLUDE ( column_name [ ,...n ] ) ]
[ WITH ( <relational_index_option> [ ,...n ] ) ]
[ ON { partition_scheme_name ( column_name)
| filegroup_name
| default
}
][ ; ]
```

Anda juga bisa membuat *clustered index* dengan menggunakan SSMS (SQL Server Management Studio). Anda dapat mengimplementasikan *clustered index* pada kolom nama belakang pada tabel KTP di DatabaseKaryawan (dengan asumsi tidak ada primary key pada tabel ini) :

```
CREATE CLUSTERED INDEX ci_namabelakang ON TabelKTP(NamaBelakang);
```

ci_namabelakang adalah nama *clustered index* yang kita buat, sedangkan TabelKTP adalah nama tabel dimana kita ingin menerapkan *clustered index* pada salah satu kolomnya, sedangkan (NamaBelakang) adalah kolom yang akan diberikan *clustered index*.

Copyright © 2003-2007 IlmuKomputer.Com

Perlu diingat bahwa clustered index hanya bisa diterapkan sebanyak 1 kali pada 1 tabel, dan secara otomatis, sebuah primary key juga akan menjadi clustered index pada tabel tersebut. Clustered index sebaiknya diterapkan pada kolom tabel yang paling sering digunakan pada saat pencarian data.

Non-clustered Index

Jika kita mengumpamakan clustered index seperti index huruf pada sebuah kamus, maka non-clustered index dapat diumpamakan seperti sebuah daftar indeks pada sebuah buku. Jika kita mencari sebuah arti kata atau pembahasan mengenai sebuah kata pada buku yang memiliki indeks pada bagian belakangnya, maka yang kita lakukan pertama kali adalah mencari kata tersebut pada indeks buku. Setelah kata ditemukan, maka apakah kita langsung mendapatkan hasil yang kita cari? Tidak. Kita masih harus mencari penjelasan mengenai kata tersebut pada halaman yang tercantum di sebelah kata tersebut bukan? Nah, ilustrasi tersebut berlaku juga pada *non-clustered index*. *Non-clustered index* berisi pointer-pointer yang menunjukkan lokasi sesungguhnya dari data yang kita cari saat dilakukan pencarian data. Cara ini sedikit lebih membutuhkan waktu pencarian dibanding dengan metode *clustered index*, namun pada database dengan skala OLDB atau VLDB, metode ini sangat membantu dibanding dengan penggunaan metode *table-scan*.

Sedikit berbeda dengan *clustered index* yang hanya bisa diimplementasikan sebanyak 1 buah pada sebuah tabel, *non-clustered index* dapat diimplementasikan sebanyak 249 buah pada sebuah tabel.

Berikut adalah T-SQL untuk membuat *non-clustered index* :

```
CREATE [ UNIQUE ] [ NONCLUSTERED ] INDEX index_name
ON <object> ( column [ ASC | DESC ] [ ,...n ] )
[ INCLUDE ( column_name [ ,...n ] ) ]
[ WITH ( <relational_index_option> [ ,...n ] ) ]
[ ON { partition_scheme_name ( column_name)
| filegroup_name
| default
}
][ ; ]
```

Misalnya, jika pada tabel KTP pada database DataKaryawan, parameter yang juga sering digunakan dalam pencarian data (selain nama belakang) adalah tanggal lahir, maka Anda dapat mengimplementasikan *non-clustered index* dengan cara sebagai berikut :

Copyright © 2003-2007 IlmuKomputer.Com

```
CREATE NONCLUSTERED INDEX nci_tanggallahir ON TabelKTP(TanggalLahir);
```

Anda dapat menerapkan non-clustered index pada kolom-kolom yang juga sering digunakan oleh pengguna pada saat pencarian data.

Penutup

Pada dasarnya ada banyak cara untuk membuat performance server komputer menjadi lebih baik saat dilakukan pengaksesan data pada database. Penggunaan index merupakan salah satu cara untuk mencapai hal ini. Perlu diingat, bahwa metode *table-scan* tidaklah selalu buruk. Saya perlu sedikit meluruskan hal ini agar tidak terjadi kesalahpahaman. Metode *table-scan* bisa saja tidak menurunkan performance server database bahkan lebih cepat menyediakan data saat Anda mencari data tertentu jika diterapkan pada database dengan ukuran kecil dan penambahan datanya cenderung lambat, misalnya pada database karyawan perusahaan. Anda perlu banyak mencoba dan berlatih menyelesaikan masalah agar mengetahui kapan saat yang tepat untuk menerapkan index pada database.

Pada artikel ini, saya memang tidak membahas secara detail dan teoritis bagaimana cara pengambilan data dari sebuah tabel yang memiliki index pada kolom-kolomnya. Hal ini dikarenakan lebih mudah memahami penggunaan index ini lewat ilustrasi yang saya contohkan sebelumnya. Saya akan membahas penjelasan konsep pengambilan data dari sebuah tabel database yang memiliki index pada artikel berikutnya. Salam.

Referensi

Jorden, Joseph. 2007. SQL Server 2005 DBA Street Smarts. Indiana. Wiley Publishing
Solid Quality Learning. 2006. SQL Server 2005 Implementation and Maintenance. USA. Microsoft Press
SQL Server 2005 BOL (Books Online)

Biografi Penulis

Rangga Praduwiratna. Rangga Praduwiratna merupakan mahasiswa jurusan Teknologi Informasi Universitas Kristen Maranatha dan Teknik Arsitektur Institut Teknologi Bandung. Ia

Copyright © 2003-2007 IlmuKomputer.Com

aktif menulis artikel mengenai teknologi Microsoft di INDC (Indonesia .NET Developer Community) dan merupakan MCTS (Microsoft Certified Technology Specialist) untuk teknologi SQL Server 2005 – Implementation and Maintenace. Ia memiliki ketertarikan pada teknologi .NET Framework, C#, SQL Server 2005, 3d Max, dan AutoCAD.

Copyright © 2003-2007 IlmuKomputer.Com