ページングを高速化する最適化テクニック

Web DBアプリケーションの最大のボトルネックである巨大結果のクエリを解決する手法

秋月 巌
AKIZUKI, Iwao
秋月巌ソリューション事務所

Technology Tools

- ☑ Visual Basic .NET
- ❑ Visual C# .NET
- ❑ SQL Server 2000
- ❑ Oracle 9i
- ❑ Access 2002
- ☑ ASP.NET
- ☑ Internet Information Services
- ❑ Other:

Level

★ ★ ★ ★ ☆

Samples

・この記事で取り上げたソースコードおよびサンプルプログラムは、付録CD-ROMの\DOTNET\F01_05ディレクトリに収録しています。

\ASPXPAGING
本稿で解説した「PagingForm」コンポーネントとページングサンプル（SAMPLE1～7）

はじめに

ページングをサポートしているほとんどのWebデータベースアプリケーションは、巨大結果のクエリに対してデータベースアクセスが最適化されていない。そこで、本稿では巨大結果クエリに最適化したページングテクニックをコンポーネントとして提供する。

何が最大のボトルネットなのか、いかに解決するのか

リソースに恵まれている今日、最適化の必要な場面は少ない

データベースにアクセスするASP.NETアプリケーションに限定して、パフォーマンスについて書いてみたい。

ASP.NETは速い。多少、無駄なコードが記述されているとしても、それがWebアプリケーションの大きなボトルネックになることはあまりない。それに最近のマシンの高速化は著しい。大体、どんなデータベースエンジンを使っても、適切にインデックスが設定された100万件程度のデータからの検索ならば、パフォーマンスが問題になることはあまりない。ましてや、自動チューニング機能にすぐれたVersion 7.0以降のSQL Serverは、初期のインストール状態で性能的に問題になることはあまりないだろう。

Webアプリケーションのようにステートレスなアプリケーションでは、データベースに接続する負荷がばかにできないが、Internet Information Services（IIS）はデータベースへの接続を最適化する。

検索結果の行数が多くなるとパフォーマンスは一気に悪化する

データベースとクエリが適切に設定されているならば、開発者があまり最適化に神経質になることはない。しかし、開発者がWebデータベースを作るうえで、どうしても配慮しなければいけないことがある。それは「クエリの結果セットの行数」である。

結論からいえば、クエリの結果を受け取る行の数が最少になるようにクエリを設計しなければいけないというこ

とである。100万件あるテーブルから1件のデータを検索するのは（インデックスが設定されていれば）瞬時だが、100万件のデータを取得するのはかなりのコストが発生する。このような問題に対処するには、当然、まず始めにデータベースとクエリの設計を検討すべきである。しかし、今回、ここで解説するのは、その方法についてではない。仕様上、どうしても仕方なく巨大な結果セットを返すクエリを実行しなければならない場合に、どのように対処するかの有効な提案である。仕方なく巨大な結果セットのクエリを実行しなければならないケースは案外多く存在する。何かしらの条件でデータを強力に絞り込むことが不可能なケースである。

どのような場合に検索結果の行数が多くなるか

たとえば、膨大なニュース情報を蓄積したデータベースがあったとしよう。ユーザーはその中から目的の記事を探したいが検索のためのキーワードを思いつかないとする。そのため、ユーザーは最新のニュースからしらみつぶしに閲覧して必要な情報を探したいという要求をもつ。このようなケースでは、日付で逆ソートして今日の日付のニュースだけを表示すれば問題は解決する。それで見つからないならば、前日の分を表示するためのリンクなりボタンを用意すればいい。これが設計による問題の回避である。しかし、たとえば1日のデータ自体が莫大である可能性もある。その場合、1時間分だけ表示されるようにクエリを実行すべきだろうか。難しい判断である。また、パッケージソフトのように多くのユーザーが使用するソフトウェアでは時間あたりのデータ数を特定できないので、適切なインターバルというもの自体が想定できない。

また、ユーザーがキーワードを思いついたとしても、その条件にヒットする結果が膨大かもしれない。その結果をみればユーザーは検索条件を考え直すかもしれないが、すでに負荷の大きいクエリが実行されたという事実は変わらない。

ほとんどのWebアプリのページングはDBアクセスに最適化されていない

このようなケースでは、やはり、ページングでページ単位に分割するのが妥当というものだ。それにページングは表示行数を一定にできるので、レイアウト的にも美しい。ページングを利用するならば、結果セットを小さくするなんて簡単じゃないかと思われるかもしれない。確かにページング機能がついたWebアプリケーションはどれも1ページ分のデータしかWebブラウザに返さないという意味で、ネットワークトラフィックが最適化されている。しかし、ほとんどの場合、データベースにアクセスして取得した結果セットは大きいままである。それは大きな結果セットの内容を分割してWebブラウザに送信しているだけで、クエリを分割実行していないためだ。

ASP.NETのWebフォーム用のコントロールであるDataGridコントロールは、ページングを最適化するためのカスタムページングに対応している。以下はカスタムページングに関するドキュメントの抜粋である。

通常、 DataGridコントロールのすべての行を格納しているデータソースは、そのDataGridコントロールが別のページに移動するたびに読み込まれます。データソースが大きい場合は、この動作によって多量のリソースが利用されます。カスタムページングでは、1ページの表示に必要なデータのセグメントのみが読み込まれます。
カスタムページングを有効にするには、AllowPagingプロパティとAllowCustomPagingプロパティをtrueに設定します。

ここでは最適化の内容について詳しくは記述されていない。ただ、クエリの分割が行なわれていないことは間違いないだろう。クエリの分割が行なわれない限り、結果セット後半のデータを取得するのに時間がかかることは回避できない。おそらく目的は高速化ではなく、サーバー側のメモリ消費をおさえるための方策だろう。ただ、Microsoftが提供する製品は、ときどき一般に知られていない手法を用いて、圧倒的な高性能化を図ることができる場合もあるので、あながち無視はできない。いずれ、そのパフォーマンスを実測する必要はあるだろう。

なお、クエリの分割の詳細については、後述のSAMPLE6の解説を参考にしていただきたい。