.NET Framework アイデアノート

なにを使うか、どう使えるのか

第8回

秋月巌ソリューション事務所
秋月 巌 *AKIZUKI, Iwao*
http://www.akizuki.co.jp

ADO.NET自体をデータベースエンジンとして使用する ―その2―

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¥NOTEディレクトリに収録しています。

¥SAMPLE1
連結テーブルの追加、変更、削除サンプル

¥SAMPLE2
3つのテーブルを連結するサンプル

¥FORMS
帳票画面サンプル

Delphi 8を使いたい！

本誌先月号の特集2「Delphi 8オーバービュー」を読んで、本当にすばらしい開発ツールが登場したものだと思った。単に技術的な点だけでいうならば、Visual Studio .NETを使う理由など、あまりないのではないだろうか。もっとも、それを言うならば、Delphiの最初のバージョンが出たとき、すでにVisual Basicを使用する理由などなくなっていたのである。

プロフェッショナルの開発者ならば、コンピュータ技術に関するトレンドが技術的な、あるいはコンセプトやデザイン的な優劣ではなく、政治的な要因で決まることを何度も経験してきているはずである。私がコンピュータプログラミングに関する最初の本を書く直前、私の手元にはDelphiの最初のバージョンがあったがVisual Basicの最新バージョンであるVisual Basic 4.0はなかった。私はVisual BasicもDelphiもどちらもほとんど経験がなかったので、本を書くために新たに学習しなければいけないという意味では同じ条件だった。また、私は貧しかったので、新しくVisual Basicを買うことを躊躇したが、結局、Visual Basic 4.0 Learningエディションを新たに購入し、Visual Basicに関する本を書いた。

当時、Delphiと比較して、Visual Basicが技術やコンセプト面で優れている点というのは、おそらく、何ひとつなかったはずである。しかし、当時、海外から帰国したばかりでほとんど無職だった私は、生活の糧を得るために、迷う余地なくVisual Basicを選択した。一応、いっておくと当時、MicrosoftとBorlandの政治的な力は、今日の差よりもはるかに少なかった。にも関わらず、選択肢は限られていたのである。

現在、.NETの置かれている立場は微妙である。Microsoftはランタイム環境としての.NET Frameworkを普及させることに、それほどの情熱を持っていないように見える。そのような状況の中で新規案件を、Win32アプリでこなすか、あるいは.NETを採用するかの判断は難しい。そのような今だからこそ、Delphi 8でクロスプラットフォームとして開発することには大きな意味があるだろう。とはいえ、そのような過度期のためだけに、新しい開発環境をマスターするというのも憂鬱である。結局のところ、Delphi 8は従来のDelphiユーザーのためのものということになってしまうのではないだろうか。このようなよいツールを積極的に導入できない自分を残念に思う。

ADO.NETをDBエンジンレスで使うサブプロシージャの改良

前回は、ADO.NETをデータベースエンジンレスで使用する方法について説明し、その他に連結テーブルの取得を実現するサブプロシージャ（JoinTable）を紹介した。また、もうひとつのサブプロシージャ（JoinTable Write）は、連結テーブルの更新を可能にした。

今回はサブプロシージャに追加と修正を加え、データの新規追加と削除も可能にしている。ただし、同じ名前にも関わらず、前回のサブプロシージャとの互換性は失ってしまった。もっとも、互換性を失ったのは、機能を追加したからではない。より広い範囲での使用を考慮したからである。

まず、連結テーブルを実現する「JoinTable」サブプロシージャには、2つの引数を追加した。ひとつは6番目の引数として追加した「右テーブル用の連結キー」である。前回の仕様では、2つのテーブルにある連結キーは、常に同じ名前でなければならなかった。多くの場合、テーブル連結を行なうキー列は同じ名前だろうが、そうでない場合もありえるのでそれに対応した。ちなみにこの引数にヌルストリング（長さ0の文字列）を指定すると、5番目の引数と同一と判断するので、連結キーの名前が両テーブルで同じ場合はヌルストリングで代用できる。

もうひとつは、引数の最後にDataSetオブジェクトのTablesプロパティで参照できるコレクションの識別文字列を指定するようにした。先月号のバージョンでは、内部で自動生成していた。DataTableオブジェクト名がわかるから問題ないと思っていたのだが、事前にこの文字列がわからないと、データグリッドで各列の表示設定を事前に設定しておくことができないことがわかったので、この引数を設定した。また、後述するように3つのテーブルを連結するために、「JoinTable」サブプロシージャを連続して使用するような場合にも、事前にこの登録名を知る必要がある。

結果として、「JoinTable」サブプロシージャのプロトタイプは次のようになった。

```
JoinTable(1-列リスト(,) As String, _
          2-左テーブル As DataTable, _
          3-連結の種類 As String, _
          4-右テーブル As DataTable, _
          5-左テーブル連結キー As String, _
          6-右テーブル連結キー As String, _
          7-左テーブル用検索評価式 As String, _
          8-左テーブル用ソート順, As String, _
          9-右テーブル用検索評価式 As String, _
          10-右テーブル用ソート順, As String, _
          11-作業用DataSetオブジェクト As DataSet, _
          12-DataSetオブジェクトへの登録名 As String _
          ) As DataTable
```

引数の数が多すぎて覚えるのは難しいと思うので、サンプルを見ながら理解してほしい。

今回の「JoinTable」サブプロシージャは、その他にもいくつかの重大な障害を解決しているので、今回のものだけを使うようにしてほしい。

また、もうひとつのサブプロシージャである「JoinTableWrite」の最後の引数は、ユニークキー列を示す文字列型だったが、これを「文字列配列」に変更した。複数列でユニークにしているケースに対応するためである。

データの削除を元テーブルに反映させるサブプロシージャも開発

また、連結テーブルのデータの削除に対応するため、新しく「JoinTableRowDeleting」サブプロシージャを追加した。このサブプロシージャの使い方は、少し特殊な