Viewing .NET Data

Visual Studio và Data Access – Phần 1

Với phiên bản mới của Visual studio đưa ra vài cách mới để truy cập dữ liệu trong các ứng dụng của bạn. Phần này sẽ bàn luận về một số cách mà Visual Studio.NET cho phép dữ liệu được hợp nhất trong GUI, để bạn có thể tương tác với dữ liệu.

Các công cụ cho phép bạn tạo một sự kết nối cơ sở dữ liệu là sử dụng các lớp *OleDbConnection* hay *SqlConnection*. Lớp mà bạn sẽ dùng phụ thuộc vào cơ sở dữ liệu nào bạn muốn kết nối. Khi định nghĩa một sự kết nối, bạn có thể tạo một *DataSet* và định vị nó từ bên trong Visual studio.NET. Vấn đề này sẽ tạo ra một tập tin XSD cho *DataSet* như là chúng ta đã làm bằng

tay trong chương trước và tự động phát ra các mã .cs cho bạn. Kết quả này nằm trong sự tạo thành của một type-safe *DataSet*.

Trong phần này ta sẽ học cách tạo một sự kết nối, chọn một số dữ liệu và tạo ra một *DataSet*, và sử dụng tất cả đối tượng được tạo ra để làm một ứng dụng đơn giản.

Tạo một sự kết nối

Để bắt đầu phần này, ta phải tạo một ứng dụng Windows. Khi tạo bạn sẽ thấy một form trống. Công việc đầu tiên là tạo một sự kết nối cơ sở dữ liệu mới. Mở *Server Explorer* bằng cách gõ *Ctrl+Alt+S* hay chọn mục *Server Explorer* từ menu. Cửa sổ sẽ hiển thị như sau:

Trong cửa sổ này bạn có thể quản lý nhiều khía cạnh khác nhau của việc truy cập dữ liệu. Theo ví dụ này, bạn cần tạo một sự kết nối đến cơ sở dữ liệu *Northwind*. Chọn *Add Connection*...từ menu trên mục *Data Connections* sẽ tự động hiện lên một trình thông minh để bạn có thể chọn OLEBD

provider nào được dùng- ở đây ta chọn *Microsoft OLEDB Provider* cho SQL server, khi bạn sẽ được kết nối với cơ sở dữ liệu *Northwind* được cài đặt như một phần của mẫu Framework SDK. Trang thứ hai của hộp thoại *Data Link* như sau:

Phụ thuộc vào cách bạn cài đặt các cơ sở dữ liệu mẫu Framework thì bạn sẽ có một thể hiện của cơ sở dữ liệu *Northwind* trong SQL Server, và một thể hiện trong một cơ sở dữ liệu local MSDE (Microsoft Data Engine), hay cả hai.

Để kết nối với cơ sở dữ liệu MSDE thì gõ (*local)\NETSDK* và tên của server. Để kết nối một thể hiện của SQL server bạn gõ *(local)* như hiện ở trên cho bộ máy hiện tại hay tên của server muốn kết nối trên mạng.

Tiếp theo, bạn cần chọn thông tin đăng nhập. Bạn phải chọn lại một lần nữa phụ thuộc vào cách cơ sở dữ liệu của bạn được cài đặt. Đối với cơ sở dữ liệu local MSDE, bạn có thể dùng một username và Password đặc biệt tương ứng với *QSUser* và *QSPassword*.

Chọn cơ sở dữ liệu *Northwind* từ danh sách cơ sở dữ liệu, và để chắc rằng bạn đã cài đặt mọi thứ chính xác thì click vào nút *Test Connection*. Hành động này sẽ kết nối cơ sở dữ liệu và hiện một hộp tin khi hoàn tất. Dĩ nhiên, bạn phải cài server trên cấu hình của máy bạn. vì thế Username, password và tên server sẽ khác nhau.

Để tạo một đối tượng kết nối, click và kéo server mới đến cửa sổ ứng dụng chính. Nó sẽ tạo một biến thành viên của kiểu *System.Data.SqlClient.SqlConnection*, hay *System.Data.OleDb.OleDbConnection* nếu bạn chọn một provider khác và thêm đoạn mã sau vào phương thức *InitializeComponent* của form chính:

```
this.sqlConnection1 = new System.Data.SqlClient.SqlConnection();
```

```
//

// sqlConnection1

//

this.sqlConnection1.ConnectionString = "data source=skinnerm\\NETSDK;"
+

                                "initial catalog=Northwind;" +

                                "user id=QSUser;password=QSPassword;" +

                                "persist security info=True;" +

                                "workstation id=SKINNERM;" +

                                "packet size=4096";
```

Như bạn thấy, sự kết nối thông tin chuỗi được gắn trực tiếp trong đoạn mã.

Khi bạn thêm đối tượng này và dự án bạn sẽ chú ý đối tượng *sqlConnection1* xuất hiện trong vùng bên dưới của cửa sổ visual studio.

Chọn dữ liệu

Khi bạn định nghĩa một sự kết nối dữ liệu, bạn có thể chọn một bản từ danh sách và kéo bảng đó đến một form trên dự án của bạn.

Ví dụ, ta chọn bảng *Customer*. khi bạn kéo đối tượng này vào dự án của bạn nó sẽ thêm một đối tượng vào form của bạn được thừa hưởng từ *SqlDataAdapter*, hay *OleDbDataAdapter* nếu bạn không dùng SQL Server.

Data adapter đã tạo ra chứa đựng các lệnh SELECT, INSERT, UPDATE, và DELETE. Đoạn mã tạo trình thông minh sẽ thực hiện ngay lúc này nhưng visual studio.NET thêm đoạn mã sau vào tập tin .cs của bạn.

```
private System.Data.SqlClient.SqlCommand sqlSelectCommand1;

private System.Data.SqlClient.SqlCommand sqlInsertCommand1;

private System.Data.SqlClient.SqlCommand sqlUpdateCommand1;

private System.Data.SqlClient.SqlCommand sqlDeleteCommand1;

private System.Data.SqlClient.SqlDataAdapter sqlDataAdapter1;
```

Có một đối tượng đã định nghĩa cho mọi lệnh SQL và một *sqlDataAdapter*. Trong phương thức *InitializeComponent()*, trình thông minh tạo ra đoạn mã để tạo mọi lệnh này và data adapter. Đoạn mã thì dông dài, vì thế tôi chỉ đưa ra một đoạn ở đây.

Có hai khía cạnh của đoạn mã được tạo bởi Visual studio.NET là các giá trị được nhìn thấy từ các thuộc tính *UpdateCommand* và *InsertCommand*. Đây là một phiên bản tóm tắt hiện thông tin thích đáng:

```
// sqlInsertCommand1
```

```
//
this.sqlInsertCommand1.CommandText = @"INSERT INTO dbo.Customers
                    (CustomerID, CompanyName, ContactName,
                    ContactTitle, Address, City, Region,
                    PostalCode, Country, Phone, Fax)
    VALUES(@CustomerID, @CompanyName, @ContactName, @ContactTitle,
        @Address, @City, @Region, @PostalCode, @Country, @Phone, @Fax);
    SELECT CustomerID, CompanyName, ContactName, ContactTitle, Address,
        City, Region, PostalCode, Country, Phone, Fax
    FROM dbo.Customers WHERE (CustomerID = @Select2_CustomerID)";
this.sqlInsertCommand1.Connection = this.sqlConnection1;
```

```
//
// sqlUpdateCommand1
//
this.sqlUpdateCommand1.CommandText = @"UPDATE dbo.Customers
        SET CustomerID = @CustomerID, CompanyName = @CompanyName,
            ContactName = @ContactName, ContactTitle = @ContactTitle,
            Address = @Address, City = @City, Region = @Region,
            PostalCode = @PostalCode, Country = @Country,
            Phone = @Phone, Fax = @Fax
        WHERE (CustomerID = @Original_CustomerID)
        AND (Address = @Original_Address) AND (City = @Original_City)
        AND (CompanyName = @Original_CompanyName)
        AND (ContactName = @Original_ContactName)
```

```
        AND (ContactTitle = @Original_ContactTitle)

        AND (Country = @Original_Country)

        AND (Fax = @Original_Fax)

        AND (Phone = @Original_Phone)

        AND (PostalCode = @Original_PostalCode)

        AND (Region = @Original_Region);

        SELECT CustomerID, CompanyName, ContactName,
ContactTitle,

            Address, City, Region, PostalCode, Country, Phone, Fax

        FROM dbo.Customers

        WHERE (CustomerID = @Select2_CustomerID)";

this.sqlUpdateCommand1.Connection = this.sqlConnection1;
```

Điểm chú ý chính trong những lệnh này là SQL đã được tạo. Cả hai lệnh INSERT và UPDATE là hai SQL statement thực sự: một để thực hiện INSERT hay UPDATE, và cái còn lại để chọn lại hàng từ cơ sở dữ liệu:

Các mệnh đề dư thừa này được dùng như một cách để đồng bộ hoá lại dữ liệu trên các máy client trên server. Có những mặc định được áp dụng vào các cột khi chèn vào, hay các trigger dữ liệu kích thích để cập nhật một số cột trong mẫu tin. Vì thế việc đồng bộ hoá lại dữ liệu có vài thuận lợi. Thông số *@Select2_CustomerID* dùng để chọn lại dữ liệu thì cùng giá trị truyền vào cho statement INSERT/UPDATE của khoá chính; tên thì được tự tạo ra bởi trình thông minh.

Các bảng gồm một cột IDENTITY, SQL đựơc tạo sử dụng giá trị @@IDENTITY sau statement INSERT. Như mô tả ở chương trứơc, dựa vào @@IDENTITY để tạo khoá chính có thể dẫn đến vài vấn đề, vì thế có một vùng của SQL bạn sẽ muốn thay đổi. Nếu bạn không đếm số cột, nó xem như một sự phí phạm để chọn lại tất cả các cột từ bảng ban đầu trong trường hợp có vài thứ đã thay đổi.

Tạo ra một DataSet

Bây giờ bạn định nghĩa adapter dữ liệu, bạn có thể dùng nó để tạo một *DataSet*. Để tạo một *DataSet,* click vào adapter dữ liệu và hiển thị các thuộc tính của đối tượng. Dưới đáy của bảng thuộc tính bạn chú ý ba tuỳ chọn sau:

Click trên *Generate DataSet*… sẽ cho phép bạn chọn một tên cho đối tượng *DataSet* mới. Nếu bạn kéo vài bảng từ Server Explorer lên form thì bạn có thể liên kết chúng với nhau từ trong hộp dialog vào một *DataSet* đơn.

Những gì được tạo là một lược đồ XSD, định nghĩa *DataSet* và mọi bảng mà bạn đã chứa bên trong *DataSet*. Nó giống như ví dụ hand-crafted trong chương trước, nhưng ở đây tập tin XSD đã được tạo nên cho bạn.

Tập tin XSD có một tập tin .cs mà định nghĩa một số lượng các lớp type-safe. Để xem tập tin này, click trên nút thanh công cụ *Show All Files* để hiện

và sau đó mở rộng tập tin XSD. Bạn chú ý là tập tin .cs cùng tên với tập tin XSD. Các lớp được định nghĩa như dưới đây:

- Một lớp thừa hưởng từ DataSet
- Một lớp thừa hưởng từ DataTable cho adapter bạn chọn
- Một lớp thừa hưởng từ DataRow, định nghĩa các cột có thể truy cập bên trong DataTable
- Một lớp thừa hưởng từ EventArgs, được sử dụng khi một hàng thay đổi.

Bạn sẽ đoán được những công cụ nào được dùng để tạo tập tin này và các lớp này. Nó là XSD.EXE .

Bạn có thể chọn để cập nhật tập tin XSD một lần khi trình thông minh thực hiện việc của nó. Nhưng không nên sửa đổi tập tin .cs để vặn nó vào một số cách, nó sẽ được tạo lại khi bạn biên dịch lại dự án và tất cả sự thay đổi đó sẽ bị mất.

Viewing .NET Data

Visual Studio và Data Access – Phần 2

Cập nhật nguồn dữ liệu

Bây giờ chúng ta đã tạo một ứng dụng mà có thể chọn dữ liệu từ cơ sở dữ liệu, chúng ta sẽ học cách để khôi phục cơ sở dữ liệu. Nếu bạn làm theo vài bước sau cùng bạn sẽ có một ứng dụng chứa sự kết nối, adapter dữ liệu và đối tượng *DataSet*. Tất cả bị bỏ qua việc móc *DataSet* vào một *DataGrid*, thêm vài tính logic để khôi phục dữ liệu từ cơ sở dữ liệu và hiện nó, sau đó tạo sự thay đổi trở lại cơ sở dữ liệu.

Chúng ta cài đặt một form như bên dưới và sau đó tìm hiểu đoạn mã của ứng dụng , nó nằm trong thư mục 10_UpdatingData:

Form bao gồm một control *DataGrid* và hai nút. Khi người dùng click vào nut *Retrive* thì đoạn mã sau sẽ chạy:

```
private void retrieveButton_Click(object sender, System.EventArgs e)
{
  sqlDataAdapter1.Fill (customerDataSet , "Customer") ;
  dataGrid1.SetDataBinding (customerDataSet , "Customer") ;
}
```

Đoạn mã này dùng adapter dữ liệu được tạo ra dễ dàng hơn để điền một *DataSet*. Chúng ta điền vào bảng dữ liệu *Customer* với tất cả mẫu tin từ cơ sở dữ liệu. Việc gọi phương thức *SetDataBinding()* sẽ hiển thị những mẫu tin này trên màn hình.

Sau khi điều khiển các dữ liệu và tạo một số thay đổi bạn có thể click vào nút *Update*. Đoạn mã sau được hiện tiếp theo:

```
private void updateButton_Click(object sender, System.EventArgs e)

{

  sqlDataAdapter1.Update(customerDataSet , "Customer" ) ;

}
```

Đoạn mã này cũng rất đơn giản, như adapter dữ liệu đang làm mọi công việc. Phương thức *Update()* lập qua dữ liệu trong bảng chọn của *DataSet*, và cho một sự thay đổi sẽ thực thi các statement SQL chống lại cơ sở dữ liệu. Chú ý rằng phương thức này trả về một kiểu int là số lượng hàng được chỉnh sửa.

Công dụng của adapter dữ liệu được bàn luận chi tiết trong chương trước, nhưng nhắc lại một tý, nó tượng trưng cho các SQL statement như các tác vụ SELECT, INSERT, UPDATE, và DELETE. Khi phương thức *Update()* được gọi, nó thực thi các statement thích hợp cho mọi hàng chỉnh sửa. Nó là nguyên nhân của tất cả hàng chỉnh sửa thực thi một statement UPDATE, tất cả hàng bị xoá phát ra một statement DELETE, và vân vân.

Nếu bạn muốn có tất cả lợi ích của việc dùng các thủ tục lưu trữ, nhưng không có thời gian hay kiến thức để viết. Có một cách dễ dàng hơn trong Visual studio.NET. Hiển thị một menu ngữ cảnh cho adapter dữ liệu và chọn menu *Configure Data Adapter*. Nó sẽ hiện một trình thông minh để chọn nguồn của dữ liệu cho adapter.

Sau khi chọn tạo một thủ tục lưu trữ mới. Click *Next* để tiến trình tự động tạo mới các thủ tục cho các statement SELECT, INSERT, UPDATE, và DELETE. Và sửa đổi mã bên trong dự án để gọi các thủ tục lưu trữ này thay cho việc gọi các SQL statements.

Ngoài việc tạo các thủ tục lưu trữ mới, bạn có thể chọn các thủ tục lưu trữ đang tồn tại để phổ biến bốn lệnh SQL trên adapter. Nó sẽ có lợi khi hand -

crafted các thủ tục lưu trữ hay khi một vài chức năng khác được biểu diễn bởi một thủ tục như là thay đổi kiểm toán hay cập nhật liên kết các mẫu tin.

Xây dựng một lược đồ

Chúng ta mất vài trang để xây dựng một lược đồ XSD bằng tay nhưng đó không phải là cách duy nhất để làm. Visual studio bao gồm một editor để tạo lược đồ XSD - từ menu *Project* chọn *Add New Item* sau đó chọn mục *XML Schema* từ category *Data* và gọi *TestSchema.xsd*.

Nó thêm hai tập tin mới vào dự án của bạn - tập tin .xsd và một tập tin .xsx. Để tạo một tập hợp tương ứng của mã cho lược đồ ta chọn *Generate Dataset* từ menu *Schema* như bên dưới:

Chọn tuỳ chọn này sẽ thêm một tập tin C# vào dự án, dựa án sẽ hiện lên tập tin XSD trong Solution Explorer. Tập tin này được tự động tạo ra bất cứ khi nào có sự thay đổi trong lược đồ XSD và không nên chỉnh sửa bằng tay; nó được tạo như ở chương trước với công cụ XSD.EXE.

Nếu click từ cửa sổ xem *Schema* đến cửa sổ xem XML, bạn sẽ thấy các mẫu lược đồ:

```
<?xml version="1.0" encoding="utf-8" ?>

<xs:schema id="TestSchema"

        targetNamespace="http://tempuri.org/TestSchema.xsd"
```

```
        elementFormDefault="qualified"

        xmlns="http://tempuri.org/TestSchema.xsd"

        xmlns:mstns="http://tempuri.org/TestSchema.xsd"

        xmlns:xs="http://www.w3.org/2001/XMLSchema">

</xs:schema>
```

XSD này tạo ra đoạn mã bên dưới trong tập tin *TestSchema.cs*. Trong đoạn mã bên dưới, tôi đã bỏ qua phần thân của phương thức và định dạng để đọc dễ dàng hơn.

```
using System;

using System.Data;

using System.Xml;

using System.Runtime.Serialization;

[Serializable()]

[System.ComponentModel.DesignerCategoryAttribute("code")]

[System.Diagnostics.DebuggerStepThrough()]
```

```
[System.ComponentModel.ToolboxItem(true)]

public class TestSchema : DataSet

{

  public TestSchema() { ... }

  protected TestSchema(SerializationInfo info, StreamingContext context)

  { ... }

  public override DataSet Clone() { ... }

  protected override bool ShouldSerializeTables() { ... }

  protected override bool ShouldSerializeRelations() { ... }

  protected override void ReadXmlSerializable(XmlReader reader) { ... }

  protected override System.Xml.Schema.XmlSchema
GetSchemaSerializable()

  { ... }

  internal void InitVars() { ... }
```

```
  private void InitClass() { ... }

  private void SchemaChanged(object sender,

                System.ComponentModel.CollectionChangeEventArgs e)

  { ... }

}
```

Thêm một yếu tố

Đều đầu tiên để làm là thêm một phần tử cấp cao mới. Click phải trên workspace và chọn *add/New Element*:

Nó sẽ tạo ra một phần tử mới không có tên trên màn hình. Bạn nên gõ tên cho phần tử này. Trong ví dụ này chúng ta sẽ dùng *Product*. Và ta cũng thêm vài attribute cho phần tử:

Khi bạn lưu tập tin XSD, tập tin C# sẽ được sửa đổi và một số lượng lớp mới được tạo. Chúng ta sẽ bàn luận những khía cạnh thích hợp nhất của đoạn mã được tạo trong tập tin này, *TestSchema.cs*:

```
public class TestSchema : DataSet
{
  private ProductDataTable tableProduct;
  [System.ComponentModel.DesignerSerializationVisibilityAttribute
        (System.ComponentModel.DesignerSerializationVisibility.Content)]
  public ProductDataTable Product
```

```
  {
    get
    {
        return this.tableProduct;
    }
  }
}
```

Một biến thành viên mới của lớp *ProductDataTable* được tạo. Đối tượng này được trả về bởi thuộc tính *Product*, và được xây dựng trong phương thức cập nhật *InitClass()*. Từ phần nhỏ này của đoạn mã chứng minh rằng người dùng của những lớp này có thể xây dựng một *DataSet* từ lớp trong tập tin này, và sử dụng *DataSet.Procducts* để trả về products *DataTable*.

Tạo DataTable

Đoạn mã bên dưới được tạo cho *DataTable* mà được thêm vào mẫu lược đồ:

```
public delegate void ProductRowChangeEventHandler
```

```
                    (object sender, ProductRowChangeEvent e);

public class ProductDataTable : DataTable, System.Collections.IEnumerable

{

  internal ProductDataTable() : base("Product")

  {

    this.InitClass();

  }

  [System.ComponentModel.Browsable(false)]

  public int Count

  {

    get { return this.Rows.Count;}

  }

  public ProductRow this[int index]

  {
```

```
    get { return ((ProductRow)(this.Rows[index]));}

  }

  public event ProductRowChangeEventHandler ProductRowChanged;

  public event ProductRowChangeEventHandler ProductRowChanging;

  public event ProductRowChangeEventHandler ProductRowDeleted;

  public event ProductRowChangeEventHandler ProductRowDeleting;
```

Lớp *ProductDataTable* được thừa hưởng từ *DataTable*, và bao gồm một sự thực thi của giao diện *IEnumerable*. Bốn sự kiện được định nghĩa là sử dụng delegate được định nghĩa trên lớp khi được gọi lên. Delegate này được truyền qua một thể hiện của lớp *ProductRowChangeEvent*, được định nghĩa lại bởi Visual studio.NET.

Đoạn mã được tạo bao gồm một lớp thừa hưởng từ *DataRow*, cho phép truy cập đến các cột bên trong bảng. Bạn có thể tạo một hàng mới theo một trong hai cách sau:

- Gọi phương thức *NewRow()* để trả về một thể hiện mới của lớp Row. truyền hàng mới này đến phương thức *Rows.Add()*

- Gọi phương thức *Rows.Add()* và truyền một mảng đối tượng, một cho mọi hàng trong bảng.

Phương thức *AddProductRow()* được trình bày bên dưới:

```
public void AddProductRow(ProductRow row)
{
   this.Rows.Add(row);
}
public ProductRow AddProductRow ( ... )
{
   ProductRow rowProductRow = ((ProductRow)(this.NewRow()));
   rowProductRow.ItemArray = new Object[0];
   this.Rows.Add(rowProductRow);
   return rowProductRow;
}
```

Từ đoạn mã, phương thức thứ hai không chỉ tạo một hàng mới, nó sau đó chèn hàng đó vào tập hợp *Rows* của *DataTable*, và sau đó trả về đối tượng này cho người gọi.

Viewing .NET Data

Visual Studio và Data Access – Phần 3

Tạo DataRow

Lớp *ProductRow* được tạo như trình bày bên dưới:

```
public class ProductRow : DataRow
{
  private ProductDataTable tableProduct;
  internal ProductRow(DataRowBuilder rb) : base(rb)
  {
   this.tableProduct = ((ProductDataTable)(this.Table));
  }
  public string Name { ... }
```

```
  public bool IsNameNull { ... }

  public void SetNameNull { ... }

 // Other accessors/mutators omitted for clarity

}
```

Khi các attribute được thêm vào một phần tử, một thuộc tính được thêm vào lớp *DataRow* như ở trên. Thuộc tính có cùng tên như attribute, vì thế trong ví dụ trên cho hàng *Product*, có các thuộc tính như *Name, SKU, Description,* và *Price.*

Để mọi attribute thêm vào, sự thay đổi được tạo trong tập tin.cs. Trong ví dụ bên dưới sẽ chỉ ta thêm một attribute gọi là ProductID

Lớp *ProductDataTable* đầu tiên có một thành viên riêng được thêm là *DataColumn*:

```
private DataColumn columnProductId;
```

Nó được tham gia bởi một thuộc tính có tên *ProductIDColumn* :

```
internal DataColumn ProductIdColumn

{
```

```
  get { return this.columnProductId; }

}
```

Phương thức *AddProductRow()* trình bày ở trên được sửa đổi, nó mang một *ProductId* số nguyên và lưu trữ giá trị trong một cột được tạo mới:

```
public ProductRow AddProductRow ( ... , int ProductId)

{

  ProductRow rowProductRow = ((ProductRow)(this.NewRow()));

  rowProductRow.ItemArray = new Object[] { ... , ProductId};

  this.Rows.Add(rowProductRow);

  return rowProductRow;

}
```

Cuối cùng, trong *ProductDataTable*, có một sự sửa đổi đến phương thức *InitClass()*:

```
private void InitClass()

{
```

```
  ...

  this.columnProductID = new DataColumn("ProductID", typeof(int), null,

                                System.Data.MappingType.Attribute);

  this.Columns.Add(this.columnProductID);

  this.columnProductID.Namespace = "";

}
```

Nó tạo *DataColumn* mới và thêm nó vào *Columns Collection* của *DataTable*. Tham số cuối cùng cho hàm dựng *DataColumn* định nghĩa cách cột này được vẽ lên XML. Điều này có lợi khi DataSet được lưu vào một tập tin XML

Lớp *ProductRow* được cập nhật để thêm một bộ truy cập cho cột này:

```
public int ProductId

{

   get { return ((int)(this[this.tableProduct.ProductIdColumn])); }

   set { this[this.tableProduct.ProductIdColumn] = value; }
```

```
}
```

Tạo EventArgs

Lớp cuối cùng được thêm vào mã nguồn là một sự thừa hưởng của *EventArgs*, lớp này cung cấp các phương thức truy cập trực tiếp vào hàng đã được thay đổi, và hành động được áp dụng vào hàng đó. Đoạn mã này đã bị xoá cho ngắn gọn hơn.

Những yêu cầu khác

Một yêu cầu chung khi hiển thị dữ liệu là cung cấp một menu Pop-up cho một hàng. Có nhiều cách để thực hiện nhưng ta tập trung vào một cách có thể đơn giản các đoạn mã được yêu cầu, Nếu phạm vi hiển thị là một *DataGrid*, nơi có một *DataSet* với vài mối quan hệ được hiển thị. Vấn đề ở đây là menu ngữ cảnh phụ thuộc vào hàng đang được chọn, và hàng đó có thể đến từ bất kỳ *DataTable* nguồn nào trong *DataSet*.

Chức năng của menu ngữ cảnh thì thích hợp để đạt mục đích chung, sự thực thi ở đây sử dụng một lớp cơ sở để hổ trợ một menu pop-up thừa hưởng từ lớp cơ sở này.

Khi người dùng click phải trên bất kỳ phần nào của một hàng trong *DataGrid*, chúng ta sẽ tìm kiếm hàng và kiểm tra nếu nó thừa hưởng từ *ContextDataRow* và phương thức *PopupMenu()* có thể được gọi. Bạn nên thực thi nó bằng cách sử dụng một giao diện nhưng trong thể hiện này một lớp cơ sở thì đơn giản hơn.

Ví dụ này sẽ chỉ cách để tạo các lớp *DataRow* và *Datatable*, các lớp này có thể sử dụng để cung cấp truy cập type-safe đến dữ liệu.

Minh hoạ bên dưới trình bày thừa kế lớp cho ví dụ này:

Đoạn mã đầy đủ nằm trong thư mục 11_Miscellaneous:

```
using System;

using System.Windows.Forms;

using System.Data;

using System.Data.SqlClient;
```

```
using System.Reflection;

public class ContextDataRow : DataRow

{

  public ContextDataRow(DataRowBuilder builder) : base(builder)

  {

  }

  public void PopupMenu(System.Windows.Forms.Control parent, int x, int
y)

  {

    // Use reflection to get the list of popup menu commands

    MemberInfo[]  members  =  this.GetType().FindMembers
(MemberTypes.Method,

                    BindingFlags.Public | BindingFlags.Instance ,

                    new System.Reflection.MemberFilter(Filter),

                    null);
```

```
    if (members.Length > 0)
    {
        // Create a context menu
        ContextMenu menu = new ContextMenu();
        // Now loop through those members and generate the popup menu
        // Note the cast to MethodInfo in the foreach
        foreach (MethodInfo meth in members)
        {
            // Get the caption for the operation from the
            // ContextMenuAttribute
            ContextMenuAttribute[] ctx = (ContextMenuAttribute[])
              meth.GetCustomAttributes(typeof(ContextMenuAttribute), true);
            MenuCommand callback = new MenuCommand(this, meth);
            MenuItem item = new MenuItem(ctx[0].Caption, new
```

```
                                EventHandler(callback.Execute));

            item.DefaultItem = ctx[0].Default;

            menu.MenuItems.Add(item);

        }

        System.Drawing.Point pt = new System.Drawing.Point(x,y);

        menu.Show(parent, pt);

    }

  }

  private bool Filter(MemberInfo member, object criteria)

  {

    bool bInclude = false;

    // Cast MemberInfo to MethodInfo

    MethodInfo meth = member as MethodInfo;

    if (meth != null)
```

```
        if (meth.ReturnType == typeof(void))
        {
            ParameterInfo[] parms = meth.GetParameters();
            if (parms.Length == 0)
            {
                // Lastly check if there is a ContextMenuAttribute on the
                // method...
                object[] atts = meth.GetCustomAttributes
                        (typeof(ContextMenuAttribute), true);
                bInclude = (atts.Length == 1);
        }
    }
    return bInclude;
  }
```

```
}
```

Lớp hàng dữ liệu ngữ cảnh được thừa hưởng từ *DataRow*, và chứa hai chức năng thành viên. Đầu tiên là *PopupMenu* dùng sự phản ánh để tìm các phương thức phù hợp với một dạng cụ thể, và nó hiện một menu pop-up của những tuỳ chọn này đến người dùng. phương thức *Filter()* được dùng như một đại diện bởi *PopupMenu* khi liệt kê các phương thức. Nó trả về kết quả *true* nếu chức năng thành viên thực hiện tương ứng với quy ước gọi:

```
MemberInfo[] members =
this.GetType().FindMembers(MemberTypes.Method,
        BindingFlags.Public | BindingFlags.Instance,
        new System.Reflection.MemberFilter(Filter),
        null);
```

Statement này được dùng để lọc tất cả phương thức trên đối tượng hiện hành, và chỉ trả về theo các tiêu chuẩn sau:

- Thành viên phải là một phương thức
- Thành viên phải là một phương thức thể hiện public

- Thành viên không có kiểu trả về
- Thành viên phải chấp nhận không tham số
- Thành viên phải bao gồm ContextMenuAttribute

Cuối cùng là một custom attribute, chúng ta sẽ bàn luần về nó sau khi tìm hiểu rõ phương thức *PopupMenu*.

```
ContextMenu menu = new ContextMenu();

foreach (MethodInfo meth in members)

{

   // ... Add the menu item

}

System.Drawing.Point pt = new System.Drawing.Point(x,y);

menu.Show(parent, pt);
```

Một thể hiện menu ngữ cảnh được tạo, và chúng ta lặp qua mọi phương thức theo tiêu chuẩn trên và thêm mục vào menu. Menu được hiển thị như trình bày trong màn hình sau:

Rắc rối chính của ví dụ này là phần mã sau, lập lại một lần cho mọi chức năng thành viên để được hiển thị trên pop-up menu.

```
System.Type ctxtype = typeof(ContextMenuAttribute);

ContextMenuAttribute[] ctx = (ContextMenuAttribute[])

                        meth.GetCustomAttributes(ctxtype);

MenuCommand callback = new MenuCommand(this, meth);

MenuItem item = new MenuItem(ctx[0].Caption,

            new EventHandler(callback.Execute));

item.DefaultItem = ctx[0].Default;

menu.MenuItems.Add(item);
```

Mọi phương thức nên trình bày trên menu ngữ cảnh được tượng trưng với *ContextMenuAttribute*. Định nghĩa này là một tên người dùng quen thuộc cho các tuỳ chọn menu, như một tên phương thức C# không thể bao gồm các khoảng trắng. Attribute được khôi phục từ phương thức và một mục menu mới được tạo và thêm vào tập hợp mục memu của pop-up menu.

Ví dụ này cũng trình bày cách dùng của một lớp Command đơn giản. Lớp *MenuCommand* dùng trong thể hiện này được trigger từ người dùng chọn một mục trên menu ngữ cảnh, và nó định dạng việc gọi đến bộ nhận của phương thức.

Visual Studio và Data Access – Phần 4

Tạo Tables và Rows

Ví dụ XSD dễ dàng hơn trong chương chỉ đoạn mã được viết ra khi visual studio editor được dùng để tạo một tập hợp lớp truy cập dữ liệu, và bạn sẽ được vui với đoạn mã cho các lớp này như sau:

```
public class CustomerTable : DataTable

{

  public CustomerTable() : base("Customers")

  {

    this.Columns.Add("CustomerID", typeof(string));

    this.Columns.Add("CompanyName", typeof(string));
```

```
    this.Columns.Add("ContactName", typeof(string));
  }
  protected override System.Type GetRowType()
  {
    return typeof(CustomerRow);
  }
  protected override DataRow NewRowFromBuilder(DataRowBuilder
builder)
  {
    return(DataRow) new CustomerRow(builder);
  }
}
```

Điều cần thiết đầu tiên của một *DataTable* là bạn override phương thức *GetRowtype()*. Nó được dùng bởi các đặc tính .NET khi tạo các dòng mới cho bảng. Bạn nên trả về kiểu của lớp dùng để mô tả mọi hàng.

Điều cần thiết tiếp theo là bạn thực thi phương thức *NewRowFromBuilder()*, được gọi lại khi tạo ra các hàng mới cho bảng. Bấy nhiêu đủ cho một sự thực thi nhỏ. Sự thực thi của chúng ta bao gồm thêm các cột vào *DataTable*. từ khi chúng ta biết các cột trong ví dụ này là gì, chúng ta có thể thêm chúng cho phù hợp. Lớp *CustomerRow* thì khá đơn giản. Nó thực thi các thuộc tính cho mọi cột trong hàng, và sau đó thực thi các phương thức để hiển thị trên menu ngữ cảnh:

```
public class CustomerRow : ContextDataRow
{
  public CustomerRow(DataRowBuilder builder) : base(builder)
  {
  }
  public string CustomerID
  {
    get { return (string)this["CustomerID"];}
    set { this["CustomerID"] = value;}
```

```
  }

  // Other properties omitted for clarity

  [ContextMenu("Blacklist Customer")]

  public void Blacklist()

  {

    // Do something

  }

  [ContextMenu("Get Contact",Default=true)]

  public void GetContact()

  {

    // Do something else

  }

}
```

Lớp thừa hưởng từ *ContextDataRow* bao gồm các phương thức getter/setter trên các thuộc tính được đặt tên cùng với mọi trường và sau đó một tập phương thức sẽ được thêm để được dùng khi phản hồi trên lớp:

```
[ContextMenu("Blacklist Customer")]

public void Blacklist()

{

    // Do something

}
```

Mọi phương thức mà bạn muốn hiển thị trên menu ngữ cảnh có cùng kiểu và bao gồm attribute *ContextMenu* tuỳ biến.

Sử dụng một Attribute

Ý tưởng sau viết attribute *ContextMenu* là để cung cấp một tên văn bản tự do cho một tuỳ chọn menu. Ta thực thi một cờ *Default* để cho biết sự chọn lựa menu mặc định. Lớp attribute được mô tả như sau:

```
[AttributeUsage(AttributeTargets.Method,AllowMultiple=false,Inherited=tr
ue)]
```

```
public class ContextMenuAttribute : System.Attribute
{
  public ContextMenuAttribute(string caption)
  {
    Caption = caption;
    Default = false;
  }
  public readonly string Caption;
  public bool Default;
}
```

Ở đây, attribute *AttributeUsage* trên lớp đánh dấu *ContextMenuAttrinbute* như chỉ có thể dùng trong một phương thức, và nó cũng định nghĩa là chỉ có một thể hiện của đối tượng này trên bất kỳ phương thức nào.

Bạn có thể nghĩ về một số lượng các thành viên khác để thêm vào attribute này như là:

- một hotkey cho tuỳ chọn menu
- Một hình ảnh để hiển thị
- Vài văn bản đểhiển thị trong thanh công cụ
- Một help context ID

Dispatching Methods

Khi một menu được hiển thị trong .NET, mọi tuỳ chọn menu được liên kết với mã xử lý. Trong quá trình thực thi cơ chế móc menu chọn đến mã, bạn có hai chọn lựa như sau:

- Thực thi một phương thức với cùng dạng như *System.EventHandler*.
- public delegate void EventHandler(object sender, EventArgs e);
- Định nghĩa một lớp đại diện thực thi và gọi đến lớp nhận. Nó được biết như mẫu Command.

Mẫu Command tách người gửi và người nhận của sự gọi bằng các phương tiện của một lớp đơn giản. Nó tạo nên các phương thức trên mọi *DataRow* đơn giản hơn, và nó có thể mở rộng hơn:

```
public class MenuCommand
```

```
{
  public MenuCommand(object receiver, MethodInfo method)
  {
    Receiver = receiver;
    Method = method;
  }
  public void Execute(object sender, EventArgs e)
  {
    Method.Invoke(Receiver, new object[] {} );
  }
  public readonly object Receiver;
  public readonly MethodInfo Method;
}
```

Lớp cung cấp một delegate *EventHandler* để gọi phương thức trên đối tượng nhận .Ví dụ của chúng ta sử dụng hai kiểu hàng khác nhau: Hàng từ bảng *Customer* và hàng từ bảng *Orders*. Các tuỳ chọn xử lý cho mọi kiểu dữ liệu này từ giống đến khác. Hình ảnh trên trình bày các tác vụ cho một hàng *Customer*. Hình bên dưới trình bày các tuỳ chọn cho một hàng *Order*:

Getting the Selected Row

Vấn đề cuối cùng trong ví dụ này là cách để làm việc ngoài các hàng trong *DataSet*. Bạn nghĩ rằng nó là một thuộc tính trên *DataGrid*, nhưng bạn sẽ không tìm thấy nó ở đó. Bạn sẽ nhìn vào thông tin kiểm mà bạn có thể đạt được từ bên trong sự kiện *MouseUp()*, nhưng nó chỉ giúp nếu bạn đang hiển thị dữ liệu từ một *DataTable* đơn.

Quay lại cách khung lưới được điền ngay lập tức, dòng mã để thực hiện là:

```
dataGrid.SetDataBinding(ds,"Customers");
```

Phương thức này thêm một *CurrencyManager* mới vào *BindingContext*, để mô tả cho *DataTable* và *DataSet*. Bây giờ, *DataGrid* có hai thuộc tính *DataSource* và *DataMember*. Các thuộc tính này được cài đặt khi bạn gọi phương thức *SetDataBinding()*. *DataSource* trong thể hiện này sẽ là một *DataSet* và *ĐataMeber* sẽ là *Customers*.

Chúng ta có một nguồn dữ liệu, một thành viên dữ liệu và biết thông tin này được lưu trữ trong *BindingContext* của form.

```
protected void dataGrid_MouseUp(object sender, MouseEventArgs e)
{
    // Perform a hit test
    if(e.Button == MouseButtons.Right)
    {
        // Find which row the user clicked on, if any
        DataGrid.HitTestInfo hti = dataGrid.HitTest(e.X, e.Y);
        // Check if the user hit a cell
        if(hti.Type == DataGrid.HitTestType.Cell)
```

```
{

    // Find the DataRow that corresponds to the cell

    //the user has clicked upon
```

Sau khi gọi *dataGrid.HitTest()* để tính nơi người dùng click chuột, sau đó chúng ta khôi phục thể hiện *BindingManagerBase* cho khung dữ liệu:

```
    BindingManagerBase bmb = this.BindingContext[
dataGrid.DataSource,

                                dataGrid.DataMember];
```

Đoạn mã trên sử dụng *DataSource* và *DataMember* của *DataGrid* để đặt tên cho đối tượng chúng ta muốn được trả về.Tất cả chúng ta muốn làm bây giờ là tìm hàng mà người dùng click và hiện menu ngữ cảnh. Khi nhắp phải chuột trên một hàng, thì hàng được chọn hiện hành không di chuyển một cách bình thường. Chúng ta muốn di chuyển hàng được chọn và sau đó pop up menu.Từ đối tượng *HitTestInfo* chúng ta có số hàng, vì tất cả chúng ta cần là di chuyển vị trí hiện tại của đối tượng *BindingManagerBase* :

```
    bmb.Position = hti.Row;
```

sự thay đổi này chỉ đến ô và tại cùng một lúc các phương tiện ta gọi vào lớp để lấy Row , ta kết thúc với hàng hiện hành và không chọn một cái cuối cùng:

```
        DataRowView drv = bmb.Current as DataRowView;

        if(drv != null)

        {

            ContextDataRow ctx = drv.Row as ContextDataRow;

            if(ctx != null) ctx.PopupMenu(dataGrid,e.X,e.Y);

        }

      }

    }

  }
```

Khi *DataGrid* đang hiển thị các item từ một *DataSet*, đốitượng *Current* bên trong bộ *BindingManagerBase* là một *DataRowView*, nó được kiểm tra bởi một bố cục rõ ràng trong đoạn mã trên. Nêu thành công ta có thể khôi phục

hàng mà *DataRowView wraps* bằng cách hiện một bố cục khác để kiểm tra nếu nó là một *ContextDataRow*, và cuối cùng pop up một menu.

Trong ví dụ, bạn chú ý ta tạo ra hai bảng dữ liệu, *Customers* và *Orders*, và định nghĩa một mối quan hệ giữa các bảng này, để khi bạn click trên *CustomerOrders* bạn thấy một danh sách orders. Khi bạn làm như vậy, *DataGrid* thay đổi *DataMember* từ *Customers* đến *Customers.CustomerOrders*, nó xảy ra để xác định rằng bộ chỉ mục *BindingContext* dùng để khôi phục dữ liệu đang trình bày.

Viewing .NET Data

The Control DataGrid – Phần 1

DataGrid là một control mới hoàn toàn, được viết cho các ngôn ngữ .NET, và nó cho phép có những cái nhìn khác nhau về dữ liệu được hiển thị. Bạn có thể hiển thị dữ liệu bằng cách gọi phương thức *SetDataBinding()*.

Hiển thị dữ liệu xếp theo cột:

Phần cuối của chương sẽ trình bày nhiều cách chọn dữ liệu và lấy nó trong một bảng dữ liệu, mặc dù dữ liệu được hiển thị trong một kiểu rất cơ bản; chúng ta chỉ đơn giản dùng *Console.WriteLine()*

Ví dụ đầu tiên ở đây sẽ chỉ cách khôi phục dữ liệu và hiển thị trong một control *DataGrid*. Hình bên dưới là màn hình từ ứng dụng đã được xây dựng, mã nguồn của ứng dụng này nằm ở thư mục 01_DisplayTabularData:

Ứng dụng này chọn mỗi phần tử từ bản *Customer* trong cơ sở dữ liệu *NorthWind* và hiển thị những phần tử này cho người dùng trong một *DataGrid*. Đoạn mã này khá ngắn, ta sẽ từng bước xem xét chúng như sau:

```
using System;

using System.Windows.Forms;

using System.Data;

using System.Data.SqlClient;

public class DisplayTabularData : System.Windows.Forms.Form

{

  private System.Windows.Forms.Button retrieveButton;

  private System.Windows.Forms.DataGrid dataGrid;

  public DisplayTabularData()

  {

    this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);

    this.ClientSize = new System.Drawing.Size(464, 253);
```

```
    this.Text = "01_DisplayTabularData";
```

Tiếp đến, ta sẽ tạo control khung lưới(grid), và cài đặt những thuộc tính của nó. Dòng thứ hai: dataGrid.BeginInit(); chỉ được dùng khi tạo nhiều sự thay đổi trên control. Nếu các sự kiện không giới hạn, mọi thay đổi trên khung lưới có thể tạo ra một *Redraw* trên màn hình. Sau đó ta xác định vị trí và kích thước của Control, định nghĩa chỉ mục tab, và neo control vào cả hai góc trên bên trái và góc dưới bên phải của cửa sổ để nó cân xứng trong cửa sổ ứng dụng chính.

```
    this.dataGrid = new System.Windows.Forms.DataGrid();

    dataGrid.BeginInit();

    dataGrid.Location = new System.Drawing.Point(8, 8);

    dataGrid.Size = new System.Drawing.Size(448, 208);

    dataGrid.TabIndex = 0;

    dataGrid.Anchor = AnchorStyles.Bottom | AnchorStyles.Top |

                    AnchorStyles.Left | AnchorStyles.Right;

    this.Controls.Add(this.dataGrid);
```

```
    dataGrid.EndInit();
```

Bây giờ ta tạo nút. Cùng với những bước cơ bản theo sau trong việc khởi tạo nút:

```
    this.retrieveButton = new System.Windows.Forms.Button();

    retrieveButton.Location = new System.Drawing.Point(384, 224);

    retrieveButton.Size = new System.Drawing.Size(75, 23);

    retrieveButton.TabIndex = 1;

    retrieveButton.Anchor = AnchorStyles.Bottom | AnchorStyles.Right;

    retrieveButton.Text = "Retrieve";

    retrieveButton.Click += new System.EventHandler

                            (this.retrieveButton_Click);

    this.Controls.Add(this.retrieveButton);
```

Chúng ta có một sự kiện *click* gọi bộ điều khiển sự kiện *retrieveButton_click*

```
protected void retrieveButton_Click(object sender, System.EventArgs e)
```

```
{
    retrieveButton.Enabled = false;
    string source = "server=(local)\\NetSDK;" +
                "uid=QSUser;pwd=QSPassword;" +
                "database=Northwind";
```

Sau khi chọn dữ liệu từ bảng *Customer* và điền dữ liệu vào tập dữ liệu. Ta gọi phương thức *SetĐataBlinding* để gắn kết dữ liệu giữa tập dữ liệu và khung lưới. Phương thức này sẽ được truyền vào tập dữ liệu và tên của bảng trong *DataSet*. Một khung lưới có thể chỉ hiện dữ liệu từ một *DataTable* tại một thời điểm mặc dù *DataSet* chứa nhiều bảng.

```
    string select = "SELECT * FROM Customers" ;
    SqlConnection conn = new SqlConnection(source);
    SqlDataAdapter da = new SqlDataAdapter( select , conn);
    DataSet ds = new DataSet();
    da.Fill(ds , "Customers");
```

```
    dataGrid.SetDataBinding(ds , "Customers");
  }
  static void Main()
  {
    Application.Run(new DisplayTabularData());
  }
}
```

Để biên dịch đoạn mã trên bạn gõ dòng lệnh sau:

csc /t:winexe /debug+ /r:System.dll /r:System.Data.dll /r:system.windows.forms.dll

/recurse:*.cs

Tham số */recurse:*.cs* sẽ biên dịch tất cả tập tin .cs trong thư mục hiện hành và các thư mục con. Đó là một cách viết tắt, nên bạn không cần phải nhớ tất cả các tập tin nhưng bạn phải chắc chắn là chỉ có những tập tin bạn cần nằm trong thư mục đó.

Nguồn dữ liệu:

DataGrid là một cách rất linh động để hiển thị dữ liệu; thêm vào đó là để gọi phương thức *SetDataBlinding()* với một *DataSet* và tên của bảng để hiển thị thì phương thức này sẽ được gọi với bất kỳ nguồn dữ liệu sau:

- Một mảng
- Datatable
- DataView
- DataSet hay DataViewManager
- Những thành phần thực thi giao diện IListSource
- Những thành phần thực thi giao diện IList

Hiển thị dữ liệu từ một mảng:

Nhìn thoáng qua có vẽ rất dễ dàng. Tạo một mảng, điền dữ liệu vào mảng và gọi phương thức *SetDataBlinding(array,null)* trên *DataGrid*. Như ví dụ sau:

```
string[] stuff = new string[] {"One", "Two", "Three"};

dataGrid.SetDataBinding(stuff, null);
```

Chú ý rằng phương thức *SetDataBlinding()* chỉ có hai tham số, tham số đầu là nguồn dữ liệu trong trường hợp này là mảng, tham số còn lại: nếu nguồn dữ liệu là *DataSet* hay *DataViewMannager* thì gán bằng tên của bảng muốn hiển thị còn ngược lại được gán giá trị *null*.

Bạn có thể thay thế đoạn mã trong bộ điều khiển sự kiện *retriveButton_click()* của ví dụ trước với đoạn mã ở trên. Kết quả hiển thị sẽ có vấn đề sau:

Bạn sẽ thấy kết quả hiển thị có nhiều hơn số chuỗi mà bạn định nghĩa trong mảng, Khung lưới sẽ hiển thị thêm chiều dài của những chuỗi đó. Nguyên nhân là khi sử dụng mảng là nguồn dữ liệu của *Datagrid* thì khung lưới sẽ tìm thuộc tính chung đầu tiên của đối tượng bên trong mảng để hiển thị và trường hợp này đó chính là chiều dài của chuỗi đó.

Một cách giải quyết đó là tạo một lớp bao bọc cho các chuỗi này như bên dưới:

```
protected class Item

{

  public Item(string text)
```

```
  {
    m_text = text;
  }
  public string Text
  {
    get{return m_text;}
  }
  private string m_text;
}
```

Khi thêm một mảng lớp *Item* như trên, bạn sẽ nhận được kết quả mong muốn, mã nguồn của ứng dụng này nằm trên thư mục 02_DataSourceArray

DataTable

Có hai cách chính để hiển thị một *DataTable* trong một *DataGrid*:

- Nếu *DataTable* của bạn đứng một mình thì gọi phương thức *SetDataBlingding(DataTable,null)*

- Nếu *DataTable* của bạn chứa một *DataSet* thì gọi *SetDataBlinding(DataSet,"<Table Name>")*

Ví dụ bên dưới lấy từ đoạn mã trong thư mục 03_DatasourceDataTable để hiện một số cột:

Chú ý hiển thị của cột cuối cùng; nó hiện một checkbox để thay cho các control thông thường. *DataGrid* sẽ đọc lượt đồ từ nguồn dữ liệu và suy ra các kiểu cột mà control được hiển thị.

Dữ liệu trong cơ sở dữ liệu không thay đổi khi bạn thay đổi các trường trong khung lưới dữ liệu.

Hiển thị dữ liệu từ một DataView

Một *DataView* cung cấp phương tiện để lọc và sắp xếp dữ liệu bên trong một *DataTable*. Khi bạn chọn một dữ liệu từ một cơ sở dữ liệu, thông thường nó cho phép người dùng sắp xếp dữ liệu đó. Thêm vào đó, bạn muốn lọc dữ liệu để chỉ hiện những hàng nào đó. Một *DataView* cho phép bạn giới hạn số hàng hiển thị cho người dùng nhưng nó không giới hạn số cột trong *DataTable*.

Một DataView không cho phép bạn thay đổi các cột để hiển thị mà chỉ thay đổi các hàng.

Bên dưới đây là một dòng mã để tạo một *DataView* dựa trên một *DataTable* đang tồn tại. Đoạn mã ví dụ này nằm trên thư mục 04_DataSourceDataView :

```
DataView dv = new DataView(dataTable);
```

Khi tạo bạn có thể thay đổi các cài đặt trên *DataView*, và nó sẽ ảnh hưởng đến dữ liệu và các tác vụ khi chúng được hiển thị bên trong một *DataGrid*. Một vài ví dụ là:

- Cài đặt AllowEdit = false khoá chức năng chỉnh sửa các dòng
- Cài đặt AllowNew = false khoá chức năng tạo dòng mới
- Cài đặt AllowDelete = false khoá khả năng xoá dòng
- Cài đặt RowStateFilter chỉ hiển thị những dòng của một trạng thái được cho
- Cài đặt RowFilter để lọc hàng
- Xắp xếp các dòng bằng các cột nào đó

Lọc các hàng bằng dữ liệu:

Khi bạn tạo một *DataView*, bạn có thể thay đổi dữ liệu hiển thị bằng cách cài đặt thuộc tính *RowFilter*. Thuộc tính này như một chuỗi được dùng như một phương tiện để lọc trên những tiêu chuẩn nào đó- giá trị của chuỗi được dùng như tiêu chuẩn lọc.

Vài ví dụ về các mệnh đề lọc được chỉ trong bảng dưới đây

Mệnh đề	**Mục đích**
UnitsInStock > 50	Chỉ những hàng có UnitslnStock lớn hơn 50
Client = 'Smith'	Chỉ trả về những mẫu cho một client
County LIKE 'C*'	Trả về tất cả mẫu mà trường Country bắt đầu ký tự C

Lọc các hàng trên trạng thái

Mọi hàng trên *DataView* có một tập định nghĩa hàng, nó sẽ là một trong những giá trị sau. Tập này có thể được dùng để lọc những hàng được xem bởi người dùng:

DataViewRowState	**Mục đích**
Added	Tất cả hàng được tạo mới
CurrentRows	Tất cả hàng ngoại trừ những hàng được chọn sẽ bị xoá
Deleted	tất cả hàng được chọn bị xoá
ModifiedCurrent	Dãy tất cả hàng bị sửa đổi và hiện giá trị hiện hành cho mọi cột
ModifiedOriginal	Dãy tất cả hàng bị sửa đổi nhưng hiện giá trị ban đầu cho các cột chứ không phải giá trị hiện hành
OriginalRows	Tất cả hàng được chọn ban đầu từ nguồn dữ liệu .Không có những hàng mới. Chỉ hiện những giá trị ban đầu của các cột.

DataViewRowState	Mục đích
Unchanged	Tất cả các hàng không thể bị thay đổi

Để xem ảnh hưởng của các trạng thái này trên một khung lưới, ta viết một ví dụ hiển thị hai khung lưới: một chứa dữ liệu được chọn từ cơ sở dữ liệu mà bạn có thể tương tác, cái còn lại hiện các hàng trong một trong những trạng thái trên.

Bộ lọc không chỉ áp dụng đối với các hàng mà còn với trạng thái của các cột bên trong những hàng này.

Sắp xếp các hàng:

Khi lọc dữ liệu, đôi lúc bạn cần sắp xếp dữ liệu trong một *Dataview*. Bạn có thể click trên tiêu đề của cột trong control *DataGrid*, và nó sẽ sắp xếp một cột theo thứ tự giảm dần hoặc tăng dần. Tuy nhiên bạn chỉ có thể sắp xếp một cột, những nơi nào có *Dataview* gạch dưới thì có thể sắp xếp theo nhiều cột.

Khi một cột được sắp xếp, *DataGrid* sẽ hiển thị một mũi tên để cho biết cột đã được sắp xếp.

Để thực hiện sắp xếp trên một cột bạn sử dụng thuộc tính *Sort* của *DataView*:

```
dataView.Sort = "ProductName";
```

```
dataView.Sort = "ProductName ASC, ProductID DESC";
```

Dòng đầu tiên sẽ được sắp xếp theo cột *ProductName*. Dòng thứ hai sẽ được sắp xếp thứ tự tăng dần theo cột *ProductName* và sau đó theo thứ tự giảm dần của *ProductID*.

Dataview hỗ trợ sắp xếp tăng dần và giảm dần trên các cột. Nếu bạn chọn sắp xếp trên nhiều cột thì *DataGrid* sẽ ngừng hiện các mũi tên sắp xếp.

Viewing .NET Data

The Control DataGrid – Phần 2

Hiển thị dữ liệu từ một DataSet

Ở ví dụ trước, *DataGrid* chỉ có thể hiển thị một *DataTable* đơn tại một thời điểm. Nhưng ở ví dụ này, nó có thể điều khiển nhiều mối quan hệ trong *DataSet* trên màn hình. Đoạn mã sau được dùng để tạo ra một *DatasSet* dựa trên các bảng *Customers* và *Orders* trong cơ sở dữ liệu *Northwind*. Đoạn mã này nằm trong thư mục 05_DataSourceDataSet. Ví dụ này thêm hai *DataTable* và tạo một mối quan hệ giữa chúng gọi là *CustomerOrders*:

```
string source = "server=(local)\\NetSDK;" +
            "uid=QSUser;pwd=QSPassword;" +
            "database=northwind";
string orders = "SELECT * FROM Orders";
```

```
string customers = "SELECT * FROM Customers";

SqlConnection conn = new SqlConnection(source);

SqlDataAdapter da = new SqlDataAdapter(orders, conn);

DataSet ds = new DataSet();

da.Fill(ds, "Orders");

da = new SqlDataAdapter(customers , conn);

da.Fill(ds, "Customers");

ds.Relations.Add("CustomerOrders",

                ds.Tables["Customers"].Columns["CustomerID"],

                ds.Tables["Orders"].Columns["CustomerID"]);
```

Khi tạo, bạn có thể liên kết *DataSet* với *DataGrid* bằng cách gọi phương thức *SetDataBinding*:

```
dataGrid1.SetDataBinding(ds, "Customers");
```

Nó sẽ tạo một hiển thị như sau:

Bạn chú ý có một dấu + bên trái mỗi mẫu tin. Để biết rằng chúng ta đã tạo một *Dataset* với một mối quan hệ điều khiển giữa *customers* và *orders*. Bạn có thể định nghĩa nhiều mối quan hệ trong đoạn mã.

Khi bạn click trên các dấu +, một danh sách các mối quan hệ hiện ra, click trên tên của mối quan hệ sẽ điều khiển khung lưới liên kết với các mẫu tin.

Control *DataGrid* bao gồm một cặp biểu tượng mới ở góc trên bên phải. Mũi tên cho phép bạn quay lại hàng cha mẹ, và sẽ thay đổi hiển thị đến trang trước đó. Tiêu đề của hàng hiện chi tiết các mẫu tin cha mẹ có thể hiện hay ẩn bằng cách click trên những nút khác.

Hiển thị dữ liệu trong một DataViewManager

Hiển thị dữ liệu trong một *DataViewManager* thì giống như *DataSet*. Nhưng khi một *DataViewManager* được tạo cho một *DataSet* thì một *DataView* đặc biệt được tạo ra cho mỗi *DataTable*, cho phép bạn có thể thay đổi hiển thị hàng, dựa vào một bộ lọc hay trạng thái hàng. Nếu bạn không muốn lọc dữ liệu, bạn sẽ đề nghị luôn luôn bao một *DataSet* trong một *DataViewManager* để hiển thị. Nó cho bạn nhiều tuỳ chọn khi sửa đổi mã của bạn.

Đoạn mã dưới tạo một *DataViewManager* dựa trên *DataSet* từ ví dụ trước, và sau đó thay đổi *DataView* cho bảng *Customers* để chỉ hiện customers từ UK:

```
DataViewManager dvm = new DataViewManager(ds);

dvm.DataViewSettings["Customers"].RowFilter = "Country='UK'";

dataGrid.SetDataBinding(dvm, "Customers");
```

Kết quả hiển thị sẽ như sau, ví dụ này có thể tìm thấy trong thư mục 06_DataSourceDataViewManager:

Giao diện IListSource và IList

DataGrid cũng hổ trợ bất kỳ đối tượng mà đưa vào một trong những giao diện *IListSource* hay *IList*. I*ListSource* chỉ có một phương thức *GetList()* trả về một giao diện *IList*. *IList* được thực thi bởi rất nhiều lớp trong thời gian chạy. Vài lớp thực thi giao diện này là *Array, ArrayList, StringCollection.*

Khi sử dụng *IList*, cùng điều kiện cho đối tượng bên trong tập hợp là true thì sự thực thi *Array* sẽ hiện dễ dàng hơn- Nếu sử dụng một *StringCollection* như nguồn dữ liệu cho *DataGrid* thì chiều dài của chuỗi được hiện bên trong khung lưới.

Thừa kế lớp DataGrid

Thừa kế lớp cho những phần chính của DataGrid được hiện bên dưới:

DataGrid bao gồm 0 hay nhiều *DataGridTableStyles*. Những kiểu này bao gồm 0 hay nhiều *DataGridColumnStyles*. Một ô trong khung lưới có thể được truy cập bởi nhiều phương tiện của cấu trúc *DatagridCell*.

DataGridTableStyle và DataGridColumnStyle

Một *DataGridTableStyle* chứa sự miêu tả trực quan của *DataTable*. *DataGrid* chứa một tập hợp những kiểu này có thể truy cập được bằng thuộc tính *TableStype*. Khi một *DataTable* được hiển thị thì một sự kiểm tra được tạo xuyên qua tất cả đối tượng *DataGridTableStype* để tìm thuộc tính *MappingName* của nó bằng với thuộc tính *TableName* của *DataTable*. Trong khi tìm kiếm, kiểu đó sẽ được dùng trong việc hiển thị của bảng.

DataGridTableStyle cho phép bạn định nghĩa những biến hiện hình cho *DataGrid*, như là màu nền và màu cận cảnh, font dùng trong tiêu đề cột và các thuộc tính khác. *DataGridColumnStyle* cho phép bạn lọc những tuỳ chọn

hiển thị trên một cột, như là cài đặt trật tự của dữ liệu trong cột, văn bản được hiển thị một giá trị *null* và chiều rộng của cột trên màn hình.

Khi *DataGrid* hiển thị một định nghĩa *DataGridTableStype*, bạn có thể định nghĩa các cột của dữ liệu được hiển thị bằng cách thêm một *DataGridColumnStype*. Chỉ những cột có một kiểu định nghĩa sẽ được hiển thị và có thể là lợi ích cho những cột ẩn như là giá trị của khoá chính không được hiển thị. Bạn cũng có thể định nghĩa một kiểu cột *ReadOnly*.

Đoạn mã bên dưới là ví dụ của việc tạo một *DataGridTableStyle*. Đoạn mã tạo ra một đối tượng *DataGridTableStyle*, thêm vào hai đối tượng *DataGridColumnStyle*, và hiển thị tất cả dữ liệu bên trong bảng *Customer*.

```
using System;

using System.Windows.Forms;

using System.Data;

using System.Data.SqlClient;

public class CustomDataGridTableStyle : System.Windows.Forms.Form

{
```

```
private System.Windows.Forms.Button retrieveButton;

private System.Windows.Forms.DataGrid dataGrid;

public CustomDataGridTableStyle()

{

  this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);

  this.ClientSize = new System.Drawing.Size(464, 253);

  this.Text = "07_CustomDataGridTableStyle";

  this.dataGrid = new System.Windows.Forms.DataGrid();

  dataGrid.BeginInit();

  dataGrid.Location = new System.Drawing.Point(8, 8);

  dataGrid.Size = new System.Drawing.Size(448, 208);

  dataGrid.TabIndex = 0;

  dataGrid.Anchor = AnchorStyles.Bottom | AnchorStyles.Top |

                AnchorStyles.Left | AnchorStyles.Right;
```

```
    this.Controls.Add(this.dataGrid);

    dataGrid.EndInit();

    this.retrieveButton = new System.Windows.Forms.Button();

    retrieveButton.Location = new System.Drawing.Point(384, 224);

    retrieveButton.Size = new System.Drawing.Size(75, 23);

    retrieveButton.TabIndex = 1;

    retrieveButton.Anchor = AnchorStyles.Bottom | AnchorStyles.Right;

    retrieveButton.Text = "Retrieve";

    retrieveButton.Click += new

                        System.EventHandler(this.retrieveButton_Click);

    this.Controls.Add(this.retrieveButton);

  }

  protected void retrieveButton_Click(object sender, System.EventArgs e)

  {
```

```
    retrieveButton.Enabled = false;
```

DataSet sẽ được dùng tạo ra *DataGridTableStyles* để dùng trong ví dụ này và cuối cùng liên kết *DataGrid* với *DataSet*. Phương thức *CreateDataSet* không có gì mới như chúng ta thấy sau, nó chỉ đơn giản nhận tất cả hàng từ bảng *Customers*:

```
    DataSet ds = CreateDataSet();

    CreateStyles(dataGrid);

    dataGrid.SetDataBinding(ds, "Customers");

  }
```

Phương thức *CreateStyles()* có nhiều đặc biệt. Dòng đầu tiên tạo đối tượng *DataGridTableStype* và cài thuộc tính *MappingName* của nó. Thuộc tính này được sử dụng khi *DataGrid* hiển thị một *DataTable*. *DataGrid* có thể hiển thị hàng trong những màu thay đổi. Đoạn mã ở đây định nghĩa màu theo từng cặp hàng.

```
  private void CreateStyles(DataGrid dg)

  {
```

```
DataGridTableStyle style = new DataGridTableStyle();

style.MappingName = "Customers";

style.AlternatingBackColor = System.Drawing.Color.Bisque;

DataGridTextBoxColumn customerID = new DataGridTextBoxColumn();

customerID.HeaderText = "Customer ID";

customerID.MappingName = "CustomerID";

customerID.Width = 200;

DataGridTextBoxColumn name = new DataGridTextBoxColumn();

name.HeaderText = "Name";

name.MappingName = "CompanyName";

name.Width = 300;
```

Khi các cột được định nghĩa, chúng được thêm vào *GridColumnStypes* bộ các đối tượng *DataGridTableStype*, các đối tượng này có thể tự thêm thuộc tính *TableStype* của *DataGrid*:

```
    style.GridColumnStyles.AddRange
                (new DataGridColumnStyle[]{customerID , name});
    dg.TableStyles.Add(style);
  }
  private DataSet CreateDataSet()
  {
    string source = "server=(local)\\NetSDK;" +
                    "uid=QSUser;pwd=QSPassword;" +
                    "database=northwind";
    string customers = "SELECT * FROM Customers";
    SqlConnection con = new SqlConnection(source);
    SqlDataAdapter da = new SqlDataAdapter(customers , con);
    DataSet ds = new DataSet();
    da.Fill(ds, "Customers");
```

```
    return ds;
  }
  static void Main()
  {
    Application.Run(new CustomDataGridTableStyle());
  }
}
```

Sau khi tạo đối tượng *DataGridTableStyle*, chúng ta tạo hai đối tượng thừa hưởng từ *DataGridColumnStyle*. Mọi cột có một số lượng thuộc tính được định nghĩa. Sau đây là một danh sách các thuộc tính khoá:

Property	**Description**
Alignment	một trong những giá trị liệt kê HorizontalAlignment - Left, Center, or Right. Nó cho biết cách mà dữ liệu trong cột được định nghĩa hợp lý.

Property	**Description**
FontHeight	Kích cỡ của font theo pixels. Nó sẽ mặc định nếu không có giá trị được cài. Thuộc tính này được bảo vệ, vì thế có thể chỉ sửa đổi nếu bạn tạo lớp con.
HeaderText	Văn bản hiển thị trong cột heading.
MappingName	Cột trong DataTable mô tả bởi cột hiển thị
NullText	Văn bản hiển thị bên trong cột nếu giá trị dữ liệu nằm dưới là DBNull.
PropertyDescriptor	Nó sẽ được bàn luận phần của chương
ReadOnly	Một cờ cho biết cột là read-write or read-only.
Width	Chiều rộng của cột theo pixels.

Kết quả của đoạn mã hiện như sau, ví dụ này nằm trên thư mục 07_CustomDataGridTableStyle:

Viewing .NET Data

Gắn kết dữ liệu

Ở ví dụ trước đã xem xét tất cả control *DataGrid*, đó chỉ là một phần trong thời gian chạy.NET có thể dùng để hiển thị dữ liệu. Một tiến trình gắn kết một control và một nguồn dữ liệu được gọi là **data binding**.

Nếu bạn có những kinh nghiệm với các ứngdụng lập trình Windows trong MFC. Có lúc nào đó bạn đã sử dụng chức năng **Dialog Data Exchange** (**DDX**)để móc các biến thành viên của một lớp với bộ điều khiển Win32. Bạn sẽ vui sướng khi biết rằng bạn có thể giấu cửa trên DDX, như nó dễ dàng hơn để móc dữ liệu vào bộ điều khiển trong .NET. Bạn có thể gắn kết dữ liệu không chỉ đến các bộ điều khiển Window mà còn với các trang Web ASP.NET.

Gắn kết đơn giản

Một control hỗ trợ việc gắn kết đơn hiển thị chỉ những giá trị đơn tại một lúc, như là một hộp văn bản hay một nút chọn. Ví dụ sau chỉ cách gắn kết một cột từ một *DataTable* đến một hộp văn bản.

```
DataSet ds = CreateDataSet();

textBox1.DataBindings.Add("Text", ds , "Products.ProductName");
```

Sau khi lấy lại vài dữ liệu từ bảng Products và lưu trữ trong một *DataSet* được trả về từ phương thức *CreateDataSet()* như trên, dòng thứ hai gắn kết thuộc tính *Text* của control đến cột *Products.ProductName*. Nếu bạn viết đoạn mã này từ cơ sở dữ liệu *Northwind*, bạn sẽ thấy màn hình như bên dưới đây:

Hộp văn bản hiển thị vài thứ trong cơ sở dữ liệu. Để kiểm tra rằng nó là cột hay giá trị, bạn sẽ sử dụng công cụ SQL Server Query Analyzer để kiểm tra nội dung của bảng *Procucttool*.

Đối tượng gắn kết dữ liệu

Sơ đồ sau chỉ một thừa kế lớp cho các đối tượng được sử dụng trong gắn kết dữ liệu. Trong phần này ta bàn luận về *BindingContext*, *CurrencyManager*, và *PropertyManager* các lớp của *System.Windows.Forms*, và trình cách chúng tương tác khi dữ liệu giới hạn trong một hay nhiều control trên một form. Các đối tượng chuyển màu được dùng trong gắn kết.

Trong ví dụ trước, chúng ta sử dụng thuộc tính *DataBinding* của control *TextBox* để gắn kết một cột từ một *DataSet* đến thuộc tính *Text* của bộ điều khiển. Thuộc tính *DataBindings* là một thể hiện của *ControlBindingsCollection* :

```
textBox1.DataBindings.Add("Text", ds, "Products.ProductName");
```

Dòng này thêm một đối tượng gắn kết từ một đối tượng *Binding* đến *ControlBindingsCollection*

Binding Context

Mọi Windows form có một thuộc tính *BindingContext*. Form được thừa hưởng từ Control . Một đối tượng *BindingContext* có một tập thể hiện *BindingManagerBase*. Những thể hiện này được tạo và thêm vào đối tượng quản lý gắn kết khi một control bị giới hạn:

BindingContext sẽ chứa vài nguồn dữ liệu, được gói trong một *CurrencyManager* hay một *PropertyManager*. Sự quyết định lớp nào được dùng dựa vào chính nguồn dữ liệu.

Nếu nguồn dữ liệu chứa một dãy item như là *DataTable*, *DataView*, hay bất kỳ đối tượng khác thực thi giao diện *IList* thì một *CurrencyManager* sẽ được dùng, như nó có thể duy trì vị trí hiện tại bên trong nguồn dữ liệu. Nếu nguồn dữ liệu chỉ trả về một giá trị đơn thì một *PropertyManager* sẽ được lưu trữ trong *BindingContext*.

Một *CurrencyManager* hay *PropertyManager* chỉ được tạo một lần cho một nguồn dữ liệu. Nếu bạn gắn kết hai hộp văn bản với một hàng từ một *DataTable* thì chỉ một *currencyManage*r sẽ được tạo bên trong binding context.

Mọi control thêm vào một form được gắn kết với bộ quản lý gắn kết của form, vì thế tất cả control chia sẽ cùng một thể hiện. Khi một control được tạo thuộc tính *BindingContext* của nó là *null*. Khi control được thêm bộ Control của form thì nó sẽ cài *BindingContext* đến bộ đó của form.

Để gắn kết một control với một form, bạn cần thêm một thực thể vào thuộc tính DataBinding của nó. Đoạn mã bên dưới tạo một sự gắn kết mới:

```
textBox1.DataBindings.Add("Text", ds, "Products.ProductName");
```

Phương thức *Add()* của *ControlBindingsCollection* tạo một thể hiện mới của đối tượng Binding từ những thông số của phương thưc này và thêm chúng vào bộ những việc gắn kết

Hình trên trình bày những gì đang hoạt động khi bạn thêm một Binding đến một Control. Binding gắn kết control với một nguồn dữ liệu được duy trì bên trong *BindingContext* của Form. Sự thay đổi bên trong nguồn dữ liệu được phản ánh vào control như là những thay đổi trong control đó.

Binding

Lớp này gắn kết một thuộc tính của control với một thành viên của nguồn dữ liệu. Khi những thành viên này thay đổi thì những thuộc tính của control được cập nhật để phản ánh sự thay đổi này và ngược lại

Bindings có thể cài đặt từ bất kỳ cột nào đến bất kỳ thuộc tính nào của control, vì thế bạn sẽ gắn kết một cột với một hộp văn bản và có thể gắn kết cột khác với màu hộp văn bản..

Bạn có thể gắn kết các thuôc tính của một control đến các nguồn dữ liệu khác nhau .

CurrencyManager và PropertyManager

Khi một đối tượng Binding được tạo, một đối tượng *CurrencyManager* hay *PropertyManager* sẽ được tạo nếu đó là lần đầu tiên dữ liệu đó từ nguồn bị giới hạn. Mục đích của lớp này là định nghĩa vị trí của mẫu tin hiện hành trong nguồn dữ liệu và kết hợp tất cả dãy bindings khi mẫu tin hiện hành này bị thay đổi.

Ví dụ sau sẽ hiển thị hai trường từ bảng *Product* và bao gồm một cách để di chuyển giữa các mẫu tin bằng các phương tiện của một control *TrackBar*.

Đoạn mã cho ứng dụng này nằm hoàn toàn trong thư mục 09_ScrollingDataBinding

```
using System;

using System.Windows.Forms;

using System.Data;

using System.Data.SqlClient;

public class ScrollingDataBinding : System.Windows.Forms.Form

{

   private Button retrieveButton;

   private TextBox textName;

   private TextBox textQuan;

   private TrackBar trackBar;
```

```
private DataSet ds;
```

Ứng dụng trên tạo cửa sổ và tất cả control cho cửa sổ đó bên trong một hàm khởi tạo *ScrollingDataBinding*:

```
public ScrollingDataBinding()
{
    this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
    this.ClientSize = new System.Drawing.Size(464, 253);
    this.Text = "09_ScrollingDataBinding";
    this.retrieveButton = new Button();
    retrieveButton.Location = new System.Drawing.Point(4, 4);
    retrieveButton.Size = new System.Drawing.Size(75, 23);
    retrieveButton.TabIndex = 1;
    retrieveButton.Anchor = AnchorStyles.Top | AnchorStyles.Left;
    retrieveButton.Text = "Retrieve";
```

```
    retrieveButton.Click += new System.EventHandler
                                (this.retrieveButton_Click);
    this.Controls.Add(this.retrieveButton);
    this.textName = new TextBox();
    textName.Location = new System.Drawing.Point(4, 31);
    textName.Text = "Please click retrieve...";
    textName.TabIndex = 2;
    textName.Anchor = AnchorStyles.Top | AnchorStyles.Left |
                    AnchorStyles.Right ;
    textName.Size = new System.Drawing.Size(456, 20);
    textName.Enabled = false;
    this.Controls.Add(this.textName);
    this.textQuan = new TextBox();
    textQuan.Location = new System.Drawing.Point(4, 55);
```

```
    textQuan.Text = "";

    textQuan.TabIndex = 3;

    textQuan.Anchor = AnchorStyles.Top | AnchorStyles.Left |

                      AnchorStyles.Top;

    textQuan.Size = new System.Drawing.Size(456, 20);

    textQuan.Enabled = false;

    this.Controls.Add(this.textQuan);

    this.trackBar = new TrackBar();

    trackBar.BeginInit();

    trackBar.Dock = DockStyle.Bottom ;

    trackBar.Location = new System.Drawing.Point(0, 275);

    trackBar.TabIndex = 4;

    trackBar.Size = new System.Drawing.Size(504, 42);

    trackBar.Scroll += new System.EventHandler(this.trackBar_Scroll);
```

```
    trackBar.Enabled = false;

    this.Controls.Add(this.trackBar);

  }
```

Khi nút *Retrieve* được click, sự kiện handler chọn tất cả mẫu tin từ bảng Product và lưu trữ trong *Dataset* riêng ds:

```
  protected void retrieveButton_Click(object sender, System.EventArgs e)

  {

    retrieveButton.Enabled = false ;

    ds = CreateDataSet();
```

Tiếp theo là hai control văn bản được giới hạn

```
    textName.DataBindings.Add("Text" , ds ,

                              "Products.ProductName");

    textQuan.DataBindings.Add("Text" , ds ,

                              "Products.QuantityPerUnit");
```

```
    trackBar.Minimum = 0 ;

    trackBar.Maximum = this.BindingContext[ds,"Products"].Count – 1;

    textName.Enabled = true;

    textQuan.Enabled = true;

    trackBar.Enabled = true;

  }
```

Ở đây chúng ta có một mẫu tin cuộn để phản ứng lại với sự di chuyển của TrackBar:

```
  protected void trackBar_Scroll(object sender , System.EventArgs e)

  {

    this.BindingContext[ds,"Products"].Position = trackBar.Value;

  }

  private DataSet CreateDataSet()

  {
```

```
    string source = "server=(local)\\NetSDK;" +
                    "uid=QSUser;pwd=QSPassword;" +
                    "database=northwind";
    string customers = "SELECT * FROM Products";
    SqlConnection con = new SqlConnection(source);
    SqlDataAdapter da = new SqlDataAdapter(customers , con);
    DataSet ds = new DataSet();
    da.Fill(ds , "Products");
    return ds;
  }
  static void Main()
  {
    Application.Run(new ScrollingDataBinding());
  }
```

```
}
```

Khi dữ liệu được khôi phục, vị trí lớn nhất trên track bar được cài là số lượng của mẫu tin. Sau đó, trong phương thức scroll ở trên, chúng ta cài vị trí của *BindingContext* cho *DataTable* products đến vị trí của scroll bar thumb. Nó thay đổi mẫu tin hiện hành từ *DataTable*, vì thế tất cả control giới hạn đến hàng hiện hành.