Building a dynamically ASP.NET 2.0 GridView control

Cătălin NĂCHILĂ
Petroleum - Gas University Of Ploiesti Ploiești, România

Microsoft Visual Studio 2005 (based on ASP.NET 2.0), the successor to Visual Studio .NET 2003 has a lot of new features and goodies designed for Web developers. This article show how a ASP.NET 2.0 control can be dynamically connected to Microsoft Access database. The delete and update operation will be implemented using a GridView control and SQL queries. The connection between the database and the .NET application will be made with OleDb Data provider, the new Access Data Source control. The SQL queries will be implemented with OleDbCommand.

Keywords: dynamic GridView, AccessDataSource, OleDbConnection, OleDbCommand.

ASP.NET 2.0 defines a few key composite data-bound controls, such as GridView, FormView, and DetailsView. The GridView is a major upgrade of the ASP.NET 1.x DataGrid control, with the added ability to take advantage of data source controls. The GridView control supports several new features, including the built-in ability to sort, edit, page, select, and update data. The GridView control has several features that weren’t available in the DataGrid control, in particular: binding to the new data source controls, including AccessDataSource and SqlDataSource.

A GridView can be connected to a database in many ways:

– static (automatically) – using Server Explorer and the data source control; in this case, the GridView is connected to the database through the data source control when the Web page is designed;

– dynamic (programmed) – using object like: OleDbConnection, OleDbCommand, OleDbDataAdapter, OleDbDataReader and OleDbParameter or using the same type of object but with another data providers; the data providers which can be used with those object are: ODBC Data Provider, OleDb Data Provider and SQL Data Provider.

– dynamic (programmed) – using DataAdapter and Fill() method.

Another innovation of ASP.NET 2.0 is the new type of data source controls: AccessDataSource. This control represents a connection to an Access database. It inherits from the SqlDataSource control but throws an exception if you attempt to set the ConnectionString and ProviderName properties. The control uses the Jet 4.0 OLEDB provider to connect to the database.

The application

The following controls are brought to the Web form: two Button controls, one DropDownList, three Label controls and one GridView. The first button (Button1) is used to get the tables from the database. The names of the tables are presented in the DropDownList control (DropdownList1).

The second button (Button2) is used to assign GridView’s (GridView1) data source to the table selected in the DropDownList. The first label (Label1) is used to show eventually errors. The other two labels (Label2 and Label3) are used to keep the name of the selected table from the DropDownList1 and the primary key of the record which is in the editable mod of the GridView1. The property visible for the Label2 and Label3 will be set to false. The user doesn’t need to see those values.

The GridView1 must be set up before, for using the edit and delete operation. From the GridView1 Show Smart Tags we choose Add New Column. We select CommandField for the Field Type. We check Delete and Edit/Update and choose Button for the Button Type. In this way, we will have both buttons, Edit and Delete, in the first column.
of GridView1. It must be mentioned that the first column and the first row (except for the paging row if it has position set to top) of a GridView have the index 0. For example, the HeaderRow has the index 0. If the user wants, he can set up the properties for the two buttons (Edit and Delete) if he converts the first column of the GridView1 into a Template field (from the GridView Show Smart Tags the user choose Edit Column – figure 2). The Web form should look like in the figure 1.

**Figure 1. The Web form**

**Figure 2. Convert into a Template Field**

For the application I used a small Microsoft Access database named books.mdb which has the diagram presented in figure 3.

**Figure 3. The tables and relationships of books.mdb.**

### Connection with the database

The connection with a Microsoft Access database can be made in different ways. I used two of them:

a) if we need to make more operation on the database, we use the OleDbConnection; in the connection string the provider must be specified and the path of the database.; the general syntax is:

```csharp
OleDbConnection user_name_of_connection = new OleDbConnection
    (string connection string)
............ // user’s code
user_name_of_connection.Open();
............ // user’s code
user_name_of_connection.Close();
```
In the application the following syntax is used for Button1_Click event:

```csharp
DropDownList1.Items.Clear();
OleDbConnection connection_table = new OleDbConnection(@"Provider=Microsoft.Jet.OLEDB.4.0;
    Data Source=D:\doc\Articol\App_Data\books.mdb");
connection_table.Open();
string[] restrictions = new string[4];
restrictions[3] = "Table";
DataTable userTables = connection_table.GetSchema("Tables",
    restrictions);
for (int i = 0; i < userTables.Rows.Count; i++)
{
    DropDownList1.Items.Add(userTables.Rows[i][2].ToString());
}
connection_table.Close();
```

b) if the database records need only to be displayed, a more simple method which use an AccessDataSource object can be chosen; the general syntax is:

```csharp
AccessDataSource user_name_of_data_source = new AccessDataSource
    (string data file, string select command)
```

In the application the following syntax is used for Button2_Click event:

```csharp
AccessDataSource Access_Data_Source1 = new AccessDataSource
    ("~/App_Data/books.mdb",
     "SELECT * FROM " + DropDownList1.SelectedValue.ToString());
GridView1.DataSource = Access_Data_Source1;
GridView1.DataBind();
```

**Programming the GridView**

Before programming the events of the GridView control we write a function named GridLoad() for the GridView1 data binding. The code is similar to the Button2_Click without the Label2.Text line.

The events fired by the GridView control which are used in the application are:

- **RowEditing** — occurs when a row’s **Edit** button is clicked, but before the control enters edit mode;
- **RowUpdating** — occur when a row’s **Update** button is clicked, but before the control updates the row;
- **RowCancelingEdit** — occurs when the **Cancel** button of a row in edit mode is clicked, but before the row exits edit mode.
- **RowDeleting** — occur when a row’s **Delete** button is clicked, but before the grid control deletes the record from the data source.

The code for RowEditing and RowCancelingEdit is:

```csharp
protected void GridView1_RowEditing(object sender,
    GridViewEditEventArgs e)
{
    GridView1.EditIndex = e.NewEditIndex;
    GridLoad();
    Label3.Text = "" + 
    (((TextBox)((GridView1.Rows[GridView1.EditIndex].Cells[1].Controls[0])
    )).Text);
}
```

```csharp
protected void GridView1_RowCancelingEdit(object sender,
    GridViewCancelEditEventArgs e)
{...
```
GridViewCancelEditEventArgs e) {
    GridView1.EditIndex = -1;
    GridLoad();
}

In Label13 is kept the primary key of the record which is in edit mode. The value is needed because I want to update even the primary key.
The events GridView1_RowUpdating and GridView1_RowDeleting are similar. In the first part of those two, are used OleDbConnection, OleDbCommand and OleDbDataAdapter objects to get the selected table attributes. The general syntax for those objects is:

```csharp
OleDbConnection user_name_of_connection = new OleDbConnection
    (string connection string);
user_name_of_connection.Open();
OleDbCommand user_name_of_command = new OleDbCommand();
user_name_of_command.Connection = user_name_of_connection;
user_name_of_command.CommandType = CommandType.CommandText;
user_name_of_command.CommandText = string_SQL;
………… // user’s code
OleDbDataAdapter user_adaptor = new OleDbDataAdapter(string Select
    command text, string connection);
DataSet user_DataSet = new DataSet();
………… // user’s code
user_adaptor.Fill(user_DataSet, source table);
………… // user’s code
user_name_of_command.ExecuteNonQuery();
………… // user’s code
user_name_of_connection.Close();
```

The common code for the two events is:

```csharp
// Variables
int nr_of_table_attributes, i = 1;
string[] table_attributes = new

// Connection
OleDbConnection connection_update_record = new OleDbConnection
    (@"Provider=Microsoft.Jet.OLEDB.4.0;
    Data Source=D:\doc\Articol\App_Data\books.mdb");
connection_update_record.Open();

// Command
OleDbCommand cmd_update_record = new OleDbCommand();
cmd_update_record.Connection = connection_update_record;
cmd_update_record.CommandType = CommandType.Text;

// Get the selected table attributes
string query_update_record = "SELECT * FROM " + Label2.Text.ToString();
OleDbDataAdapter Adaptor = new OleDbDataAdapter(query_update_record, connection_update_record);
DataSet Set_de_date = new DataSet();
Adaptor.Fill(Set_de_date, Label2.Text.ToString());
DataTable Tabel_de_date = Set_de_date.Tables[0];
foreach (DataColumn Coloana_date in Tabel_de_date.Columns)
{
    table_attributes[i] = Coloana_date.ColumnName;
i++;
}

Updating the records
After the Edit button is pressed, the cells of the selected record becomes TextBox controls. For each one of this controls we make an SQL UPDATE query. If we don’t want to update the primary key or the database doesn’t admit this operation, we don’t take the primary key anymore and the for cycle will start from 2 instead of 1. The code is:

int primary_key = Convert.ToInt32(Label3.Text);
for (i = 1; i < nr_of_table_attributes; i++)
{
    cmd_update_record.CommandText = "UPDATE " + Label2.Text.ToString()
    primary_key = Convert.ToInt32(((TextBox)((GridView1.Rows[GridView1.EditIndex].Cells[1].Controls[0]))).Text);
}
connection_update_record.ExecuteNonQuery();
GridView1.EditIndex = -1;
GridLoad();

For a better understanding of the error and the problem which generated it, a try – catch block can be used, especially in the sequences in which we communicate with the database: when a connection is opened, when an Adaptor is filled or when an ExecuteNonQuery() is used.
For example, Label1 can be used to show the errors and the user can understand which was the mistake:

try
{
    cmd_update_record.ExecuteNonQuery();
    Label1.Visible = false;
}
catch
{
    Label1.Visible = true;
    Label1.Text = "Data type mismatch. Enter a correct value."
}

Delete the records
Before using the command, we must get the index of the row were we pushed the Delete button. The SQL DELETE command query is:

GridView1.EditIndex = e.RowIndex;
GridLoad();
cmd_delete_record.CommandText = "DELETE FROM " +
Conclusion

The static method is limited and is indicated when small databases are used with a few tables and in the case of simple Web applications which don’t need dynamically programming because of their field of application.

The dynamic method is more complex and is used for bigger and more complicated Web application which are connected to big databases with a lot of tables or if the field of application ask for it.

The static method perform data-manipulation tasks (such as retrieving and updating the records in a table). The Command class allows to execute any type of SQL statement. The .NET developer can use a Command class to perform data-definition tasks (such as creating and altering databases, tables, and indexes).

An advantage of the dynamic method over the static method is that it doesn’t use on the web form a large number of controls. In the dynamic method the same control can be programmed for many operations instead of putting a control on the form for every operation. As was seen in these application, the UPDATE and DELETE command can be used on every database that user select using a single GridView control.

Bibliography

2. Ion Smeureanu, Marian Dârdală, Adriana Reveiu, *Visual C#, Cison, 2004*