

Windows Programming

Chapter 3: Connecting Windows Applications With Databases

Compiled By Amsalu Dinote

Introduction

- Connection and communication between windows applications and databases is accomplished through ADO.NET(Active Data Object)
- ADO.NET is an object-oriented set of libraries that allows you to interact with data sources. Commonly, the data source is a database, but it could also be a text file, an Excel spreadsheet, or an XML file.
- As you are probably aware, there are many different types of databases available. For example, there are Microsoft SQL Server, Microsoft Access, Oracle, Borland Interbase, and IBM DB2 databases, just to name a few. For our course we'll use MS SQL Server

Introduction(cont..)

- Data sources have their own protocols to communicate with different applications. The protocols used by data sources might be ODBC protocol or OleDb protocol
- ADO.NET provides a relatively common way to interact with data sources, but comes in different sets of libraries for each way you can talk to a data source. These libraries are called **Data Providers** and are usually named for the protocol or data source type they allow you to interact with. The following table lists some data providers

Introduction(cont..)

Provider Name	API Prefix	Data Source Description
ODBC Data Provider	Odbc	Data Sources with an ODBC interface. Normally older data bases.
OleDb Data Provider	OleDb	Data Sources that expose an OleDb interface, i.e. Access or Excel.
Oracle Data Provider	Oracle	For Oracle Databases.
SQL Data Provider	Sql	For interacting with Microsof SQL Server

ADO.NET Objects

- ADO.NET includes many objects you can use to work with data.
 - Connection object
 - Command object
 - Data reader object
 - Dataset object
 - Data adapter object
- These objects have different names under respective data providers, for example, connection object under sql data provider is named as SqlConnection
- Throughout our discussion we'll consider and discuss how we could use these objects under sql data provider

SqlConnection Object

- To interact with a database, you must have a connection to it. The connection helps identify the ***database server***, the ***database name***, ***user name***, ***password***, and other parameters that are required for connecting to the data base.
- A connection object is used by command objects so they will know which database to execute the command on.
- It is an object, just like other C# objects

SqlConnection Object(cont..)

- Most of the time, you just declare and instantiate the SqlConnection object all at the same time, as shown below:

```
SqlConnection conn = new SqlConnection("Data Source=(local);Initial Catalog=Northwind; Integrated Security=SSPI");
```

Or

```
SqlConnection conn = new SqlConnection("Data Source=(local);Initial Catalog=Northwind; User ID=YourUserID;Password=YourPassword");
```

- The SqlConnection object instantiated above uses a constructor with a single argument of type string
- This argument is called a ***connection string***.

SqlConnection Object(cont..)

- **Data Source:** Identifies the server and could be local machine, machine domain name, or IP Address.
- **Initial Catalog:** Database name.
- **Integrated Security:** Set to SSPI to make connection with user's Windows login
- **User ID:** Name of user configured in SQL Server.
- **Password:** Password matching SQL Server User ID.

SqlCommand Object

- A SqlCommand object allows you to specify what type of interaction you want to perform with a database.
- For example, you can do select, insert, modify, and delete commands on rows of data in a database table.
- Like other C# objects SqlCommand object is instantiated with new instance declaration like:

***SqlCommand cmd = new SqlCommand("select
CategoryName from Categories", conn);***

SqlCommand Object(cont..)

- In the instantiation the constructor takes a string parameter that holds the command you want to execute and a reference to a SqlConnection object
- In applying any data operations using SqlCommand object we should follow the following steps:
 1. Open the connection through connection object
 2. Instantiate the command object with query string and connection object
 3. Call appropriate command object method
 4. Close the resources and connection

SqlCommand Object(cont..)

- Generally, a SqlCommand object allows you to query and send commands to a database. It has methods that are specialized for different commands.
- The ***ExecuteReader*** method returns a SqlDataReader object for viewing the results of a select query.
- For insert, update, and delete SQL commands, we use the ***ExecuteNonQuery*** method.
- If we only need a single aggregate value from a query, the ***ExecuteScalar*** is the best choice.

Using SqlConnection and SqlCommand Objects

Assume that we want to register course data through the following form, then

1. `//Use SqlClient data provider for SQL server`
`using System.Data.SqlClient;`
2. `//Create SqlConnection object;`
`SqlConnection con =new SqlConnection(("Data Source=(local);Initial Catalog=Demonst; User ID=sa;Password=mysqlserver");`
3. `//Open connection,`
`//Create SqlCommand object and call ExecuteNonQuery`
`//method inside Save button event`
`private void btnOk_Click(object sender, EventArgs e)`
`{`
`SqlCommand cmd=newSqlCommand("insert into Course Values("INSERT`
`INTO Course VALUES('" + txtCode.Text + "', '" + txtName.Text + "', '" + txtHr.Text`
`+"')",con);`
`cmd.ExecuteNonQuery();`
`}`

SqlCommand Object(cont..)

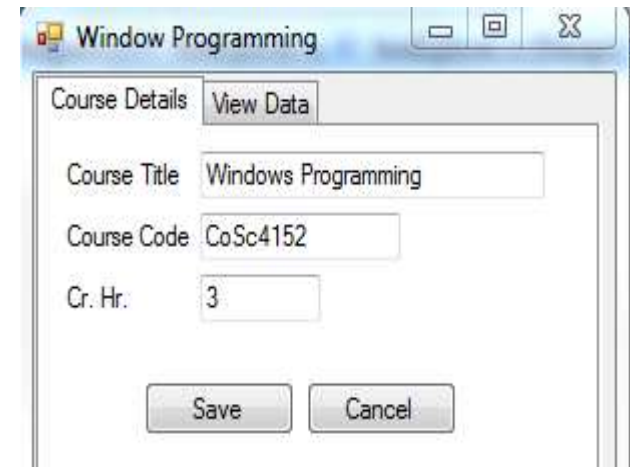
//a code to add course data to course table in Demonst database

```
using System.Data.SqlClient;
using System.Windows.Forms;
public partial class Form1 : Form
{
    static SqlConnection con = new SqlConnection("Data Source=(local);Initial Catalog=Demonst; Use
                                                ID=sa;Password=mysqlserver");

    public Form1()
    {
    }

    private void btnOk_Click(object sender, EventArgs e)
    {
        SqlCommand cmd=newSqlCommand("insert into Course Values('"+INSERT INTO Course VALUES('" +
txtCode.Text + "','"+ txtName.Text + "','"+ txtHr.Text + "')",con);
        cmd.ExecuteNonQuery();
    }

    private void btnCancel_Click(object sender, EventArgs e)
    {
        txtCode.Text = "";
        txtHr.Text = "";
        txtName.Text = "";
    }
}
```



The screenshot shows a Windows application window titled "Window Programming". Inside the window, there is a tabbed interface with two tabs: "Course Details" (which is active) and "View Data". The "Course Details" tab contains three text input fields: "Course Title" with the text "Windows Programming", "Course Code" with the text "CoSc4152", and "Cr. Hr." with the text "3". Below these fields are two buttons: "Save" and "Cancel".

SqlDataReader Object

- When using a SQL select command, you retrieve a data set for viewing. To accomplish this with a SqlCommand object, you would use the ***ExecuteReader*** method, which returns a SqlDataReader object.
- A SqlDataReader is a type that is good for reading data in the most efficient manner possible. You cannot use it for writing data. SqlDataReader's are often described as fast-forward firehose-like streams of data.
- You can read from SqlDataReader objects in a forward-only sequential manner. Once you've read some data, you must save it because you will not be able to go back and read it again. That is what makes it to be fast.

SqlDataReader Object(cont..)

- SqlDataReader is the best choice if the requirement is to only read a group of data from database or if the amount of data to read is larger than what we would prefer to hold in memory beyond a single call
- Creating SqlDataReader object is little different than creating other ADO.NET objects. We must call ExecuteReader on a command object like:
SqlDataReader rdr = cmd.ExecuteReader();

SqlDataReader Object(cont..)

Assume that we want to read course data using SqlDataReader, then

1. //Use SqlClient data provider for SQL server

using System.Data.SqlClient;

2. //Create SqlConnection object;

**SqlConnection con =new SqlConnection(("Data Source=(local);Initial Catalog=Demonst;
User ID=sa;Password=mysqlserver"));**

3. //Open connection, Create SqlCommand and SqlDataReader objects, and call
ExecuteNonQuery

//method inside form load or button event or any required event

private void btnOk_Click(object sender, EventArgs e)

{ con.Open();

SqlCommand cmd=newSqlCommand("Select*from Course",con);

SqlDataReader rdr=cmd. ExecuteReader();

While(rdr.Read())

{

String course_name=(String)rdr["Course_name"];

String Course_code=(String)rdr["Course_code"];

int Chr=(int)rdr["Chr"];

//apply any operation the read data

}}

5/29/2018

Working with Disconnected Data

- ADO.NET ***DataSet*** object and ***DataAdapter*** object are used to work on disconnected data
- DataSet objects are in-memory representations of data. They contain multiple DataTable objects, which contain columns and rows, just like normal database tables. You can even define relations between tables to create parent-child relationships.
- A ***DataSet*** is an in-memory data store that can hold numerous tables. It only holds data and does not interact with a data source.
- It is the ***SqlDataAdapter*** that manages connections with the data source and gives us disconnected behavior. It opens a connection only when required and closes it as soon as it has performed its task.

Working with Disconnected Data(cont..)

- For example, the SqlDataAdapter performs the following tasks when filling a DataSet with data:
 1. Open connection
 2. Retrieve data into DataSet
 3. Close connection
- and performs the following actions when updating data source with DataSet changes:
 1. Open connection
 2. Write changes from DataSet to data source
 3. Close connection

Working with Disconnected Data(cont..)

- DataSet object is instantiated as other C# objects like:

DataSet dsCustomers = new DataSet();

- DataSet constructor doesn't require parameters
- The **SqlDataAdapter** holds the SQL commands and connection object for reading and writing data. It can be created like:

**SqlDataAdapter daCustomers = new
SqlDataAdapter("select CustomerID,
CompanyName from Customers", conn);**

- The above code initializes data adapter with a SQL select statement and connection object

Working with Disconnected Data(cont..)

- Once data adapter object is created we can read data and fill to dataset object using Fill method of data adapter object like:

daCustomers.Fill(dsCustomers, "Customers");

- The fill method takes two parameters dataset and table name. table name is any identifier given to the table created in the dataset. It is good to give the same name as database table name
- We can bind dataset to datagrid for display onto windows forms or ASP.NET web forms like:

dgCustomers.DataSource = dsCustomers;

dgCustomers.DataMember = "Customers";

Working with Disconnected Data(cont..)

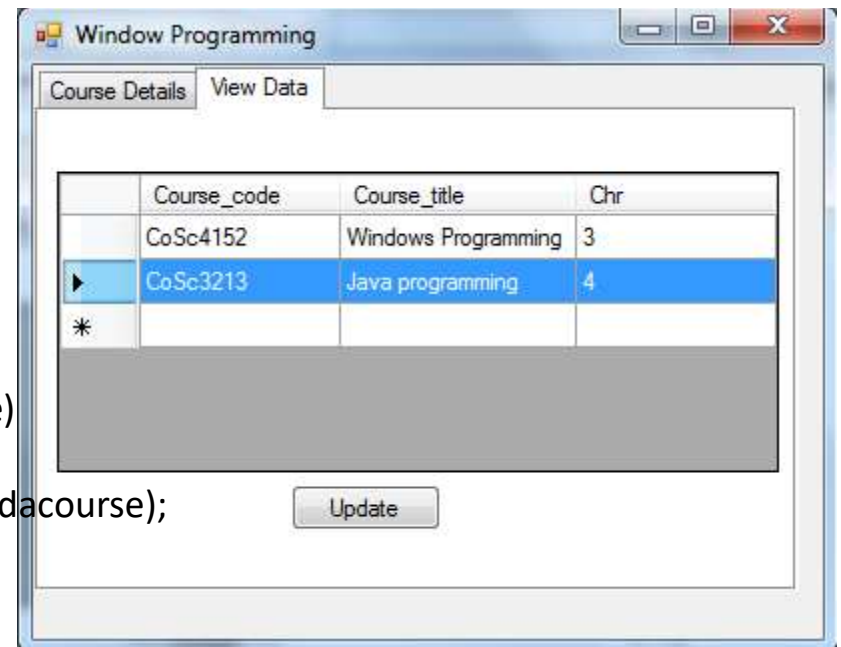
- After data is bound to grid view and modified in the grid view we can write the updates of the data from grid view back to the database through data adapter's **Update** method. Like:

**daCustomers.Update(dsCustomers,
"Customers");**

- Update method contains all SQL statements to make insert, update, and delete database modifications

Putting All Together

```
using System.Data.SqlClient
using System.Windows.Forms;
public partial class Form1 : Form
{
    static SqlConnection con = new SqlConnection("Data Source=(local);Initial Catalog=Demonst; User
                                                ID=sa;Password=mysqlserver");
    SqlCommand cmd = new SqlCommand("select*from Course", con);
    SqlDataAdapter dacourse = new SqlDataAdapter("select*from Course", con);
    DataSet dscourse = new DataSet();
    public Form1()
    {
        InitializeComponent();
        dacourse.Fill(dscourse, "Course");
        dataGridView1.DataSource = dscourse;
        dataGridView1.DataMember = "Course";
    }
    private void btnUpdate_Click(object sender, EventArgs e)
    {
        SqlCommandBuilder cb = new SqlCommandBuilder(dacourse);
        dacourse.Update(dscourse, "Course");
    }
}
```



Adding Parameters to SqlCommands

- SqlCommand query is simply string so that we can concatenate user input values to modify the result of particular query.

**SqlCommand cmd = new SqlCommand("select *
from Customers where city = " + inputCity + "");**

- However, this way is bad as it gives room for hackers to inject malicious code for. Instead of dynamically building a string, as shown in the bad example above, use parameters. Anything placed into a parameter will be treated as field data, not part of the SQL statement, which makes your application much more secure.

Adding Parameters to SqlCommands

- The way of using parameters involves three steps

1. Construct the SqlCommand command string with parameters.

```
SqlCommand cmd = new SqlCommand(  
    "select * from Customers where city = @City", conn);
```

2. Declare a SqlParameter object, assigning values as appropriate.

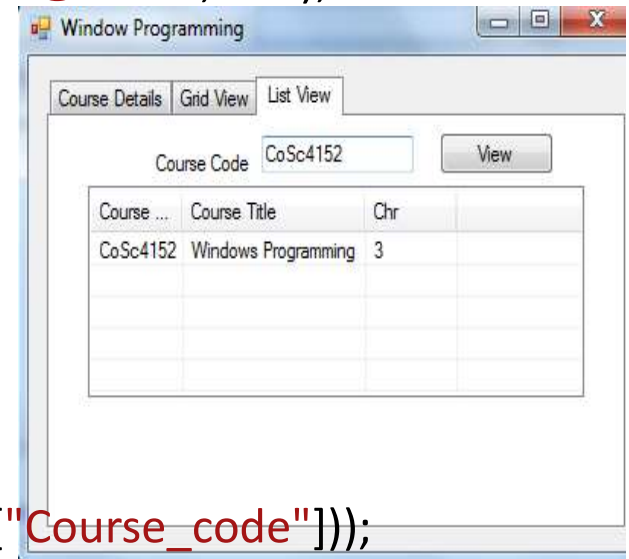
```
SqlParameter param = new SqlParameter();  
param.ParameterName = "@City";  
param.Value = inputCity;
```

3. Assign the SqlParameter object to the SqlCommand object's Parameters property
cmd.Parameters.Add(param);

Putting All Together

```
private void btnView_Click(object sender, EventArgs e)
{
    con.Open();
    SqlCommand cmd2 = new SqlCommand("select * from Course where
                                     Course_code=@code", con);

    SqlParameter para = new SqlParameter();
    para.ParameterName = "@code";
    para.Value = txtListView.Text;
    cmd2.Parameters.Add(para);
    SqlDataReader rr = cmd2.ExecuteReader();
    int i = 0;
    while (rr.Read())
    {
        listView1.Items.Add(new ListViewItem((string)rr["Course_code"]));
        listView1.Items[i].SubItems.Add((string)rr["Course_title"]);
        listView1.Items[i].SubItems.Add(rr["Chr"].ToString());
        i++;
    }
}
```



Thank you!!