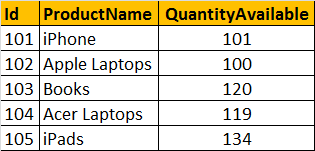
**In**[**Part 4**](http://csharp-video-tutorials.blogspot.com/2012/10/sqlcommand-in-adonet-part-4.html)**of ADO.NET video series**, we discussed about the **SqlCommand**object. In this session we will continue with another example of using **SqlCommand** object. We will be using tblProductInventory table for our example.   
   
  
**If you want to following along, use the following sql script to create the table.**  
Create table tblProductInventory  
(  
 Id int primary key,  
 ProductName nvarchar(50),  
 QuantityAvailable int  
)  
  
**Insert script to populate the table with sample data.**  
Insert into tblProductInventory values(101,'iPhone',101)  
Insert into tblProductInventory values(102,'Apple Laptops',100)  
Insert into tblProductInventory values(103,'Books',120)  
Insert into tblProductInventory values(104,'Acer Laptops',119)  
Insert into tblProductInventory values(105,'iPads',134)   
  
   
  
**Drag and drop a TextBox, Button and a GridView control onto the webform**. Change the ID of the **TextBox** to **ProductNameTextBox** and **GridView** to **ProductsGridView**. Change the ID of the **Button** to **GetProductsButton** and the **Text**to **"Get Products"**. At this point the HTML of the webform should be as shown below.  
**<asp:TextBox ID="ProductNameTextBox" runat="server"></asp:TextBox>**  
**<asp:Button ID="GetProductsButton" runat="server" Text="Get Products" />**  
**<br /><br />**  
**<asp:GridView ID="ProductsGridView" runat="server">**  
**</asp:GridView>**  
  
**Now double click the Button control to generate the Click event handler**in the code behind file, and then copy and paste the following code. In this example, we are building the query dynamically by concatenating the strings that the user has typed into the textbox. This is extremely dangerous, as it is vulnerable to SQL injection attacks.  
protected void GetProductsButton\_Click(object sender, EventArgs e)  
{  
    string ConnectionString = ConfigurationManager.ConnectionStrings["DatabaseConnectionString"].ConnectionString;  
    using (SqlConnection connection = new SqlConnection("DatabaseConnectionString"))  
    {  
        //Build the query dynamically, by concatenating the text, that the user has   
        //typed into the ProductNameTextBox. This is a bad way of constructing  
        //queries. This line of code will open doors for sql injection attack  
        SqlCommand cmd = new SqlCommand("Select \* from tblProductInventory where ProductName like '" + ProductNameTextBox.Text + "%'", connection);  
        connection.Open();  
        ProductsGridView.DataSource = cmd.ExecuteReader();  
        ProductsGridView.DataBind();  
    }  
}   
  
   
  
**Now, run the project. Enter letter "i" into the textbox and click Get Products button**. The **iPhone** and **ipad** products will be listed in the **gridview** as expected. But remember, user can type some dangerous sql queries into the textbox, which in turn will be executed by the application on the database. To give you a flavour of that, just imagine what could happen if the user types the following into the TextBox, and clicks Get Products button.  
**i'; Delete from tblProductInventory --**  
  
**Now execute the following select query on the database**  
Select \* from tblProductInventory  
  
**The entire data from tblProductInventory table is deleted**. This is called **SQL injection** attack. I have seen a lot of new developers building queries dynamically by concatenating the strings, that end users enter into user interface controls like textboxes. Just imagine the extent of damage that can happen as a result of sql injection.  
  
**However, sql injection can be easily avoided, by using parameterized queries or stored procedures. We will talk about these in our next video session.**