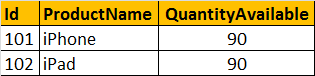
**In this video we will learn about**  
**1.** The Purpose of the SqlCommand object  
**2.** Creating an instance of the SqlCommand class  
**3.** When and how to use ExecuteReader(), ExecuteScalar() and ExecuteNonQuery() methods of the SqlCommand object.  
  
**In Part 2**, of this video series, we have discussed about the **SqlConnection**class. In this part, we will learn about **SqlCommand** class. SqlCommand class is used to prepare an SQL statement or StoredProcedure that we want to execute on a SQL Server database. In this session, we will discuss about executing Transact-SQL statements on a SQL Server. In a later session we will learn about executing stored procedures using the SqlCommand class.  
  
**The following are the most commonly used methods of the SqlCommand class.**  
**ExecuteReader** - Use when the T-SQL statement returns more than a single value. For example, if the query returns rows of data.  
**ExecuteNonQuery** - Use when you want to perform an Insert, Update or Delete operation  
**ExecuteScalar** - Use when the query returns a single(scalar) value. For example, queries that return the total number of rows in a table.  
  
We will be using **tblProductInventory** table for our examples. The table is shown below for your reference.   
  
**tblProductInventory**  
   
  
   
  
**The sample code below**, executes a T-SQL statement, that returns multiple rows of data using **ExecuteReader**() method. In this example, we are creating an instance of SqlCommand class, in just one line, by passing in the command text, and the connection object. For this purpose, we are using an overloaded constructor of the SqlCommand class that takes 2 parameters(cmdText, connection).  
protected void Page\_Load(object sender, EventArgs e)  
{  
    string ConnectionString = ConfigurationManager.ConnectionStrings["DatabaseConnectionString"].ConnectionString;  
    using (SqlConnection connection = new SqlConnection("data source=.; database=Sample\_Test\_DB; integrated security=SSPI"))  
    {  
        //Create an instance of SqlCommand class, specifying the T-SQL command that   
        //we want to execute, and the connection object.  
        SqlCommand cmd = new SqlCommand("Select Id,ProductName,QuantityAvailable from tblProductInventory", connection);  
        connection.Open();  
        //As the T-SQL statement that we want to execute return multiple rows of data,   
        //use ExecuteReader() method of the command object.  
        GridView1.DataSource = cmd.ExecuteReader();  
        GridView1.DataBind();  
    }  
}  
  
**It is also possible, to create an instance of SqlCommand class using the parameter less constructor**, and then later specify the command text and connection, using the **CommandText** and **Connection** properties of the **SqlCommand** object as shown below.  
protected void Page\_Load(object sender, EventArgs e)  
{  
    string ConnectionString = ConfigurationManager.ConnectionStrings["DatabaseConnectionString"].ConnectionString;  
    using (SqlConnection connection = new SqlConnection("data source=.; database=Sample\_Test\_DB; integrated security=SSPI"))  
    {  
        //Create an instance of SqlCommand class using the parameter less constructor  
        SqlCommand cmd = new SqlCommand();  
        //Specify the command, we want to execute using the CommandText property  
        cmd.CommandText = "Select Id,ProductName,QuantityAvailable from tblProductInventory";  
        //Specify the connection, on which we want to execute the command   
        //using the Connection property  
        cmd.Connection = connection;  
        connection.Open();  
        //As the T-SQL statement that we want to execute return multiple rows of data,   
        //use ExecuteReader() method of the command object.  
        GridView1.DataSource = cmd.ExecuteReader();  
        GridView1.DataBind();  
    }  
}   
  
   
  
**In the example below, we are using ExecuteScalar() method**, as the T-SQL statement returns a single value.  
protected void Page\_Load(object sender, EventArgs e)  
{  
    string ConnectionString = ConfigurationManager.ConnectionStrings["DatabaseConnectionString"].ConnectionString;  
    using (SqlConnection connection = new SqlConnection("data source=.; database=Sample; integrated security=SSPI"))  
    {  
        //Create an instance of SqlCommand class, specifying the T-SQL command   
        //that we want to execute, and the connection object.  
        SqlCommand cmd = new SqlCommand("Select Count(Id) from tblProductInventory", connection);  
        connection.Open();  
        //As the T-SQL statement that we want to execute return a single value,   
        //use ExecuteScalar() method of the command object.  
        //Since the return type of ExecuteScalar() is object, we are type casting to int datatype  
        int TotalRows = (int)cmd.ExecuteScalar();  
        Response.Write("Total Rows = " + TotalRows.ToString());  
    }  
}  
  
**The following example performs an Insert, Update and Delete operations** on a SQL server database using the ExecuteNonQuery() method of the SqlCommand object.  
protected void Page\_Load(object sender, EventArgs e)  
{  
    string ConnectionString = ConfigurationManager.ConnectionStrings["DatabaseConnectionString"].ConnectionString;  
    using (SqlConnection connection = new SqlConnection("data source=.; database=Sample\_Test\_DB; integrated security=SSPI"))  
    {  
        //Create an instance of SqlCommand class, specifying the T-SQL command   
        //that we want to execute, and the connection object.  
        SqlCommand cmd = new SqlCommand("insert into tblProductInventory values (103, 'Apple Laptops', 100)", connection);  
        connection.Open();  
        //Since we are performing an insert operation, use ExecuteNonQuery()   
        //method of the command object. ExecuteNonQuery() method returns an   
        //integer, which specifies the number of rows inserted  
        int rowsAffected = cmd.ExecuteNonQuery();  
        Response.Write("Inserted Rows = " + rowsAffected.ToString() + "<br/>");  
  
        //Set to CommandText to the update query. We are reusing the command object,   
        //instead of creating a new command object  
        cmd.CommandText = "update tblProductInventory set QuantityAvailable = 101 where Id = 101";  
        //use ExecuteNonQuery() method to execute the update statement on the database  
        rowsAffected = cmd.ExecuteNonQuery();  
        Response.Write("Updated Rows = " + rowsAffected.ToString() + "<br/>");  
  
        //Set to CommandText to the delete query. We are reusing the command object,   
        //instead of creating a new command object  
        cmd.CommandText = "Delete from tblProductInventory where Id = 102";  
        //use ExecuteNonQuery() method to delete the row from the database  
        rowsAffected = cmd.ExecuteNonQuery();  
        Response.Write("Deleted Rows = " + rowsAffected.ToString() + "<br/>");  
    }  
}