**In this video we will learn about**  
**1.** Instantiating a SqlConnection object  
**2.** Using the SqlConnection object  
**3.** Why is it important to close a database connection  
**4.** How to properly close a connection  
  
In [**Part 1 of ADO.NET video series**](http://csharp-video-tutorials.blogspot.com/2012/10/what-is-adonet-part-1.html), we have had an introduction to ADO.NET. We also understood about the different .NET data providers. The key to understanding ADO.NET, is to understand about the following objects.   
**1.** Connection  
**2.** Command  
**3.** DataReader  
**4.** DataAdapter  
**5.** DataSet  
  
From [Part 1](http://csharp-video-tutorials.blogspot.com/2012/10/what-is-adonet-part-1.html), it should be clear that **Connection**, **Command**, **DataReader**and **DataAdapter**objects are providers specific and **DataSet**is provider independent. So, if we are working with SQL Server, we will be using **SQLConnection**, **SqlCommand**, **SqlDataReader** and **SqlDataAdapter** objects from **System.Data.SqlClient**namespace. On the other hand if we are working with Oracle database, then we will be using **OracleConnection**, **OracleCommand**, **OracleDataReader** and **OracleDataAdapter** objects from **System.Data.OracleClient** namespace. The same logic applies when working with OleDb and Odbc data sources.  
  
If we understand how to work with one database, then we should be able to easily work with any other database. All we have to do is, change the provider specific string (Sql, Oracle, Oledb, Odbc) on the Connection, Command, DataReader and DataAdapter objects depending on the data source you are working with.  
  
So, in the next few sessions of this video series we will be discussing about each of the ADO.NET objects(Connection, Command, DataReader, DataSet and DataAdapter). In this session we will be discussing about SqlConnection object. The concepts that we discuss here are applicable to all the .NET data providers.   
  
   
  
**The first thing that we will have to do**, when working with databases is to create a connection object. There are 2 ways to create an instance of **SqlConnection** class as shown below.  
  
**Create an instance of SqlConnection class uwing the constructor that takes ConnectionString parameter**  
SqlConnection connection = new SqlConnection("data source=.; database=SampleDB; integrated security=SSPI");  
  
//First create an instance of SqlConnection class using the parameter-less constructor   
SqlConnection connection = new SqlConnection();  
//Then set the ConnectionString property of the connection object  
connection.ConnectionString = "data source=.; database=SampleDB; integrated security=SSPI";  
  
**The ConnectionString parameter** is a string made up of Key/Value pairs that has the information required to create a connection object.   
  
**To create a connection object with windows authentication**  
string ConnectionString = "data source=.; database=SampleDB; integrated security=SSPI";  
  
**To create a connection object with SQL Server authentication**  
string ConnectionString = "data source=.; database=SampleDB; user id=MyUserName; password=MyPassword";  
  
The **"data source"** is the **name or IP Address** of the **SQL Server** that we want to connect to. If you are working with a local instance of sql server, you can just specify DOT(**.**). If the server is on a network, then use Name or IP address.  
  
**Sample ADO.NET code that**  
**1.** Creates a connection  
**2.** The created connection object is then passed to the command object, so that the command object knows on which sql server connection to execute this command.  
**3.** Execute the command, and set the command results, as the data source for the gridview control.  
**4.** Call the DataBind() method  
**5.** Close the connection in the finally block. Connections are limited and are very valuable. Connections must be closed properly, for better performance and scalability.  
  
**Note:** Connections should be opened as late as possible, and should be closed as early as possible.   
  
   
  
protected void Page\_Load(object sender, EventArgs e)  
{  
    //Create the connection object  
    SqlConnection connection = new SqlConnection("data source=.; database=Sample\_Test\_DB; integrated security=SSPI");;  
    try  
    {  
        // Pass the connection to the command object, so the command object knows on which  
        // connection to execute the command  
        SqlCommand cmd = new SqlCommand("Select \* from tblProductInventory", connection);  
        // Open the connection. Otherwise you get a runtime error. An open connection is  
        // required to execute the command  
        connection.Open();  
        GridView1.DataSource = cmd.ExecuteReader();  
        GridView1.DataBind();  
    }  
    catch (Exception ex)  
    {  
        // Handle Exceptions, if any  
    }  
    finally  
    {  
        // The finally block is guarenteed to execute even if there is an exception.   
        //  This ensures connections are always properly closed.  
        connection.Close();  
    }  
}  
  
**We can also use "using" statement to properly close the connection as shown below.** We don't have to explicitly call **Close()** method, when **using**is used. The connection will be automatically closed for us.  
protected void Page\_Load(object sender, EventArgs e)  
{  
    using (SqlConnection connection = new SqlConnection("data source=.; database=Sample\_Test\_DB; integrated security=SSPI"))  
    {  
        SqlCommand cmd = new SqlCommand("Select \* from tblProductInventory", connection);  
        connection.Open();  
        GridView1.DataSource = cmd.ExecuteReader();  
        GridView1.DataBind();  
    }  
}  
  
**Common Interview Question: What are the 2 uses of an using statement in C#?**  
**1.** To import a namespace. Example: using System;  
**2.** To close connections properly as shown in the example above