**In this video we will learn about**  
**1.** The purpose of SqlDataReader  
**2.** Creating an instance of SqlDataReader  
**3.** Reading data using SqlDataReader's Read() method  
  
**SqlDataReader** reads data in the most efficient manner possible. SqlDataReader is read-only and forward only, meaning once you read a record and go to the next record, there is no way to go back to the previous record. It is also not possible to change the data using SqlDataReader. SqlDataReader is connection oriented, meaning it requires an active connection to the data source, while reading data. The forward-only nature of SqlDataReader is what makes it an efficient choice to read data.  
  
**You cannot create an instance of SqlDataReader using the new operator** as shown below. If you try to new up SqlDataReader, you will get a compilation error stating - The type 'System.Data.SqlClient.SqlDataReader' has no constructors defined.  
SqlDataReader rd = new SqlDataReader();  
  
**The SqlCommand object's ExecuteReader() method creates and returns an instance of SqlDataReader.**  
SqlCommand command = new SqlCommand("Select \* from tblProductInventory", connection);  
SqlDataReader reader = command.ExecuteReader();   
  
   
  
**Another important point to keep in mind is that, SqlDataReader is connection oriented** and the connection needs to be opened explicitly, by calling the Open() method on the connection object, before calling the ExecuteReader() method of the command object.  
  
**The simplest way to bind a SqlDataReader object to a GridView**(Data-bound control), is to assign it to the DataSource property of the GridView control and then call the DataBind() method as shown below. Notice that, just like the SqlConnection object, SqlDataReader is wrapped in an using block. This will ensure that the SqlDataReader is closed in a timely fashion, and that we don't run out of available connections to the database.  
string ConnectionString = ConfigurationManager.ConnectionStrings["DBConnectionString"].ConnectionString;  
using (SqlConnection connection = new SqlConnection(ConnectionString))  
{  
    connection.Open();  
    SqlCommand command = new SqlCommand("Select \* from tblProductInventory", connection);  
    using (SqlDataReader reader = command.ExecuteReader())  
    {  
        ProductsGridView.DataSource = reader;  
        ProductsGridView.DataBind();  
    }  
}  
  
**Please note that, finally block, can also be used to close the SqlConnection and SqlDataReader objects.**   
  
   
**If for some reason, you want to loop thru each row in the SqlDataReader object**, then use the **Read**() method, which returns **true** as long as there are rows to read. If there are no more rows to read, then this method will return false. In the following example, we loop thru each row in the **SqlDataReader** and then compute the **10% discounted price.**  
string ConnectionString = ConfigurationManager.ConnectionStrings["DBConnectionString"].ConnectionString;  
using (SqlConnection connection = new SqlConnection(ConnectionString))  
{  
    connection.Open();  
    SqlCommand command = new SqlCommand("Select \* from tblProductInventory",connection);  
    using (SqlDataReader reader = command.ExecuteReader())  
    {  
        // Create the DataTable and columns. This will   
        // be used as the datasource for the GridView  
        DataTable sourceTable = new DataTable();  
        sourceTable.Columns.Add("ID");  
        sourceTable.Columns.Add("Name");  
        sourceTable.Columns.Add("Price");  
        sourceTable.Columns.Add("DiscountedPrice");  
  
        while (reader.Read())  
        {  
            //Calculate the 10% discounted price  
            int OriginalPrice = Convert.ToInt32(reader["UnitPrice"]);  
            double DiscountedPrice = OriginalPrice \* 0.9;  
  
            // Populate datatable column values from the SqlDataReader  
            DataRow datarow = sourceTable.NewRow();  
            datarow["ID"] = reader["ProductId"];  
            datarow["Name"] = reader["ProductName"];  
            datarow["Price"] = OriginalPrice;  
            datarow["DiscountedPrice"] = DiscountedPrice;  
  
            //Add the DataRow to the DataTable  
            sourceTable.Rows.Add(datarow);  
        }  
  
        // Set sourceTable as the DataSource for the GridView  
        ProductsGridView.DataSource = sourceTable;  
        ProductsGridView.DataBind();  
    }  
}  
  
**SQL script to create the table we used in the Demo.**  
Create table tblProductInventory   
(  
 ProductId int identity primary key,  
 ProductName nvarchar(50),  
 UnitPrice int  
)  
  
**Script to populate data**  
Insert into tblProductInventory values('iPhone',350)  
Insert into tblProductInventory values('Apple Laptops',1250)  
Insert into tblProductInventory values('Books',110)  
Insert into tblProductInventory values('Acer Laptops',1150)  
Insert into tblProductInventory values('iPads',450)  
  
**Note: Please make sure you have the following namespace declarations in the code behind file.**  
using System.Data;  
using System.Data.SqlClient;  
using System.Configuration;