

Adding a Parameterized Query

XML

Extensible Markup Language

- A language for describing data and the structure of data
- Text-based
- Uses tags similar to HTML
- Starts with an XML declaration
- Always includes a root element

Accepted as a standard

XML

Extensible Markup Language

- A language for describing data and the structure of data
- Text-based
- Uses tags similar to HTML
- Starts with an XML declaration
- Always includes a root element

Accepted as a standard

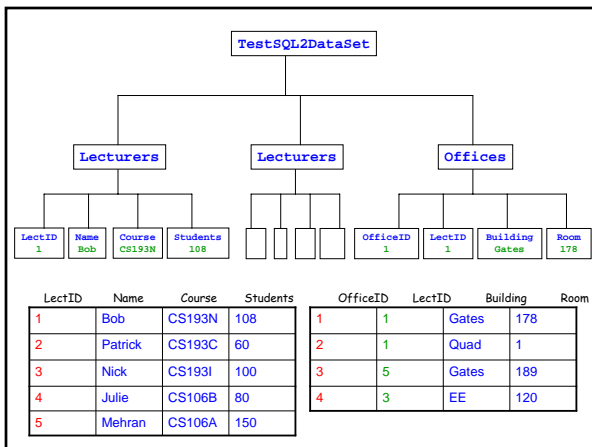
```

<?xml version="1.0"?>
<TestSQL2DataSet>
.
.
.
.
</TestSQL2DataSet>
    
```

```

<?xml version="1.0"?>
<TestSQL2DataSet>
  <Lecturers>
    <LectID>1</LectID>
    <Name>Bob </Name>
    <Course>CS193N </Course>
    <Students>80</Students>
  </Lecturers>
  <Lecturers>
    <LectID>2</LectID>
    <Name>Patrick </Name>
    <Course>CS193C </Course>
    <Students>60</Students>
  </Lecturers>
  <Offices>
    <OfficeID>1</OfficeID>
    <LectID>1</LectID>
    <Building>Gates </Building>
    <Room>178 </Room>
  </Offices>
</TestSQL2DataSet>
    
```

1,Bob,CS193N,80
 2,Patrick,CS193C,60
 1,1,Gates,178



XML Schema

- An XML Schema Definition is a document used to describe and validate the structure and content of XML data
- XML Schemas use XML syntax
- Allows us to represent both our data and a description of it as text

NOTE: the XML on the following slides has been simplified for easier presentation.

```
<?xml version="1.0"?>
<xs:schema id="TestSQL2DataSet">
  <xs:element name="TestSQL2DataSet" msdata:IsDataSet="true">
    <xs:complexType>
      <xs:choice maxOccurs="unbounded">
        <xs:element name="Lecturers">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="LectID" type="xs:int" />
              <xs:element name="Name" type="xs:string" />
              <xs:element name="Course" type="xs:string" />
              <xs:element name="Students" type="xs:int" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="Offices">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="OfficeID" type="xs:int" />
              <xs:element name="LectID" type="xs:int" />
              <xs:element name="Building" type="xs:string" />
              <xs:element name="Room" type="xs:string" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

```
<xs:unique name="Constraint1" msdata:PrimaryKey="true">
  <xs:selector xpath="//mstns:Lecturers" />
  <xs:field xpath="mstns:LectID" />
</xs:unique>
<xs:unique name="Offices_Constraint1"
  msdata:ConstraintName="Constraint1" msdata:PrimaryKey="true">
  <xs:selector xpath="//mstns:Offices" />
  <xs:field xpath="mstns:OfficeID" />
</xs:unique>
</xs:element>
<xs:annotation>
  <xs:appinfo>
    <msdata:Relationship name="FK_Offices_Lecturers"
      msdata:parent="Lecturers" msdata:child="Offices"
      msdata:parentkey="LectID" msdata:childkey="LectID" />
  </xs:appinfo>
</xs:annotation>
</xs:schema>
```

```
<Lecturers>
  <LectID>1</LectID>
  <Name>Bob </Name>
  <Course>CS193N </Course>
  <Students>80</Students>
</Lecturers>
<Lecturers>
  <LectID>2</LectID>
  <Name>Patrick </Name>
  <Course>CS193C </Course>
  <Students>60</Students>
</Lecturers>
<Offices>
  <OfficeID>1</OfficeID>
  <LectID>1</LectID>
  <Building>Gates </Building>
  <Room>178 </Room>
</Offices>
</TestSQL2DataSet>
```

XML Support in .NET

- Visual Studio can build an XML schema for a database
- The schema can be used to generate a typed DataSet

XML Support in .NET

We can easily write the contents of a DataSet to an XML file:

```
TestSQL2DataSet1.WriteXml("dsLect1WithSchema.xml",
  XmlWriteMode.WriteSchema);
```

or

```
TestSQL2DataSet1.WriteXml("dsLect1NoSchema.xml",
  XmlWriteMode.IgnoreSchema);
```

XML Support in .NET

Since XML is just text, an XML representation of data is perfect for transmitting over the Internet.

The receiver can reverse the process and reconstitute the DataSet.

XML Support in .NET

There are similar methods to read an XML file into a DataSet :

```
TestSQL2DataSet1.ReadXml("dsLect1WithSchema.xml",
    XmlReadMode.ReadSchema);

or

TestSQL2DataSet1.ReadXml("dsLect1NoSchema.xml",
    XmlReadMode.IgnoreSchema);

or

TestSQL2DataSet1.ReadXml("dsLect1NoSchema.xml");
```

If no XmlReadMode is specified, the default is "Auto", which uses the schema if there is one.

Writing a DataSet to XML



Writing a DataSet to XML

```
private void Form1_Load(object sender, EventArgs e)
{
    lecturersTableAdapter1.Fill(testSQL2DataSet1.Lecturers);
    officesTableAdapter1.Fill(testSQL2DataSet1.Offices);
    string pathToLecturersXML =
        Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments)
        + @"\\" + "Lecturers.XML";
    testSQL2DataSet1.WriteXml(pathToLecturersXML,
        XmlWriteMode.WriteSchema);
    webBrowser1.Navigate(pathToLecturersXML);
}
```

Writing a DataSet to XML

```
private void btnBack_Click(object sender, EventArgs e)
{
    webBrowser1.GoBack();
}

private void btnForward_Click(object sender, EventArgs e)
{
    webBrowser1.GoForward();
}

private void btnGo_Click(object sender, EventArgs e)
{
    webBrowser1.Navigate(textBox1.Text);
}

private void webBrowser1_Navigated(object sender,
    WebBrowserNavigatedEventArgs e)
{
    btnBack.Enabled = webBrowser1.CanGoBack;
    btnForward.Enabled = webBrowser1.CanGoForward;
}
```

Reading a DataSet from XML

```
private void Form1_Load(object sender, EventArgs e)
{
    TestSQL2DataSet ds = new TestSQL2DataSet();
    string pathToLecturersXML =
        Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments)
        + @"\\" + "Lecturers.XML";
    ds.ReadXml(pathToLecturersXML, XmlReadMode.ReadSchema);
    dataGridView1.DataSource = ds;
    dataGridView1.DataMember = ds.Offices.TableName;
}
```

.NET I/O: Streams

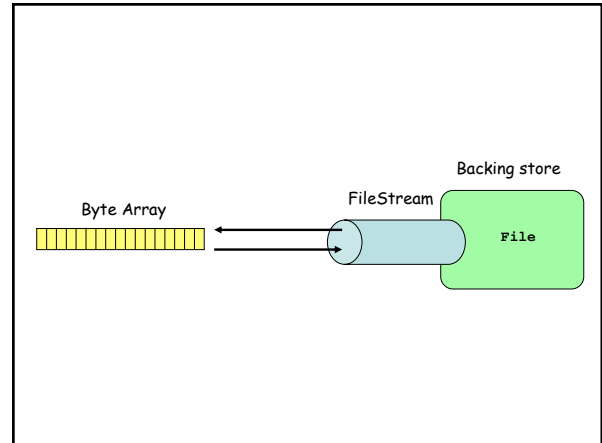
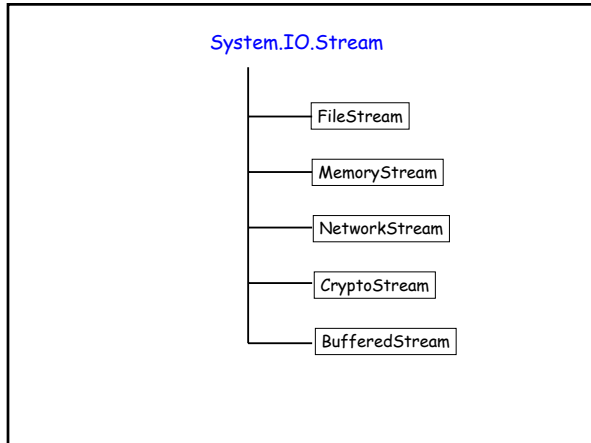
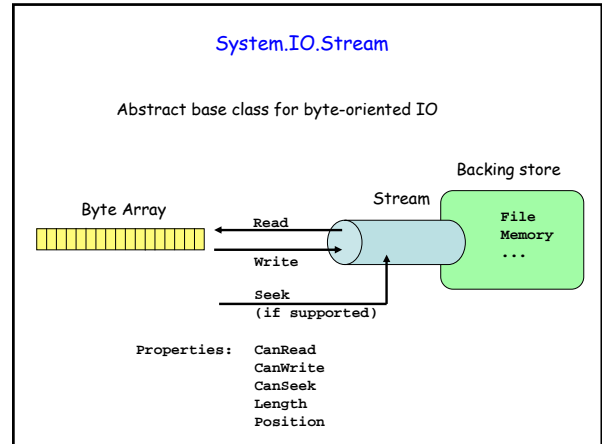
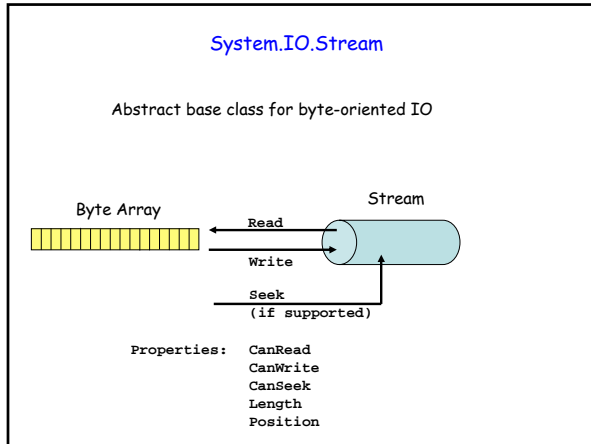
First we will remind ourselves about some data types:

byte : an unsigned integral type with range 0..255

char : a Unicode character (16-bit representation)

string : a sequential collection of Unicode characters

"binary" data types : int, long, double, ...



```

static void Main(string[] args)
{
    //Create a file stream from an existing file.
    FileStream fs = new FileStream("MyFile",
        FileMode.Open, FileAccess.Read);

    //Read 100 bytes into an array.
    int nBytes = 100;
    byte[] byteArray = new byte[nBytes];
    int nBytesRead = fs.Read(byteArray, 0, nBytes);
    Console.WriteLine("{0} bytes have been read.",
        nBytesRead.ToString());
    Console.WriteLine("The length of the file is {0} " +
        "and the position is {1}.",
        fs.Length, fs.Position);

    fs.Close();

    //Write 50 bytes to a new file.
    FileStream fs2 = new FileStream("MyFile2",
        FileMode.Create, FileAccess.Write);
    fs2.Write(byteArray, 10, 50);
    fs2.Close();
}
    
```

```

static void Main(string[] args)
{
    //Create a file stream from an existing file.
    FileStream fs = new FileStream("MyFile",
        FileMode.Open, FileAccess.Read);

    //Read 100 bytes into an array.
    int nBytes = 100;
    byte[] byteArray = new byte[nBytes];
    int nBytesRead = fs.Read(byteArray, 0, nBytes);
    Console.WriteLine("{0} bytes have been read.",
        nBytesRead.ToString());
    Console.WriteLine("The length of the file is {0} " +
        "and the position is {1}.",
        fs.Length, fs.Position);

    fs.Close();

    //Write 50 bytes to a new file.
    FileStream fs2 = new FileStream("MyFile2",
        FileMode.Create, FileAccess.Write);
    fs2.Write(byteArray, 10, 50);
    fs2.Close();
}
    
```

Offset into the byte array

Alternate way to open a FileStream

```
FileStream fs = File.Open(path, FileMode.Open, FileAccess.Read,
                          FileShare.None);
```

Static method of class File.

Any request to open the file will fail until it is closed.

Alternate way to open a FileStream

```
FileInfo fi = new FileInfo(path);
FileStream fs = fi.Open( FileMode.OpenOrCreate,
                        FileAccess.ReadWrite, FileShare.None );
```

Open for reading or writing.

Open existing file or create a new one.

Alternate way to open a FileStream

```
FileInfo fi = new FileInfo(path);
FileStream fs = fi.Open( FileMode.OpenOrCreate,
                        FileAccess.ReadWrite, FileShare.None );
```

Since file operations can often throw exceptions, the normal practice is to place them in try..catch blocks.

Reading Character Data

System.IO.TextReader

```

├── StreamReader
└── StringReader
    
```

The StreamReader is associated with an underlying byte stream coming from a file.

The data is read according to an encoding scheme. UTF-8 is the default, or we can specify the encoding or infer it from the file.

```

fs = new FileStream("Eliot.txt", FileMode.Open,
                  FileAccess.Read);
StreamReader sr = new StreamReader(fs);
string line;

while (true)
{
    line = sr.ReadLine();
    if (line == null) break;
    Console.WriteLine(line);
}
sr.Close();
fs.Close();
    
```

Between the idea
And the reality
Between the motion
And the act
Falls the Shadow

We can also pass the path directly to the StreamReader

```

StreamReader sr = null;
try
{
    sr = new StreamReader(path);
    string line;

    while (true)
    {
        line = sr.ReadLine();
        if (line == null) break;
        Console.WriteLine(line);
    }
}
catch (Exception e)
{
    Console.WriteLine("The file could not be read:");
    Console.WriteLine(e.Message);
}
finally
{
    if (sr != null)
        sr.Close();
}
    
```

A using statement makes it simpler

```

try
{
    using (StreamReader sr2 = new StreamReader("Eliot.txt"))
    {
        string line;
        while ((line = sr2.ReadLine()) != null)
        {
            Console.WriteLine(line);
        }
    }
}
catch (Exception e)
{
    Console.WriteLine(e.Message);
}
    
```

← When the using statement exits, Dispose is called on sr2

