Unit-V

What is XML?

XML stands for **E**xtensible **M**arkup **L**anguage. It is a text-based markup language derived from **S**tandard **G**eneralized **M**arkup **L**anguage (**SGML**).

XML tags identify the data and are used to store and organize the data, rather than specifying how to display it like HTML tags, XML is used to describe the data. XML is not going to replace HTML in the near future.

There are three important characteristics of XML that make it useful in a variety of systems and solutions −

- **XML is extensible** − XML allows you to create your own self-descriptive tags, or language, that suits your application.
- **XML carries the data, does not present it** − XML allows you to store the data irrespective of how it will be presented.
- **XML is a public standard** − XML was developed by an organization called the World Wide Web Consortium (W3C) and is available as an open standard.
- **XML simplifies data sharing—**XML data is stored in plain text format. This provides a software and hardware independent way of storing data.
- **Platform Independent and Language Independent:** The main benefit of xml is that you can use it to take data from a program like Microsoft SQL, convert it into XML then share that XML with other programs and platforms. You can communicate between two platforms which are generally very difficult.
- **XML simplifies data transport—**One of the most time-consuming challenges for developers is to exchange data between incompatible systems over the Internet. Exchanging data as XML greatly reduces this complexity, since the data can be read by different incompatible applications.
- **XML can be used to create new internet languages**

A lot of new Internet languages are created with XML. Here are some examples:

  - **WSDL (W**eb **S**ervices **D**escription **L**anguage) for describing available web services
  - **WAP** (**W**ireless **A**pplication **P**rotocol) and **WML** (**W**ireless **M**arkup **L**anguage) as markup languages for handheld devices
  - **RDF (R**esource **D**escription **F**ramework**)** and **OWL** (**W**eb **O**ntology **L**anguage) for describing resources and ontology
  - **SMIL** (**S**ynchronized **M**ultimedia **I**ntegration Language)for describing multimedia for the web

An XML document consists of the following parts

1. Prolog
2. Body

1. Prolog

This part of the XML document may contains the following parts

1.1 XML declaration
1.2 Optional processing instruction
1.3 Comments
1.4 Document Type Definition.

1.1 XML declaration

Every XML document should start with a one-line XML Declaration. It is not mandatory, but W3C recommends. The declaration describes the document itself. Here is a sample XML declaration

<?xml version="1.0" encoding="UTF-8" Standalone="no"?>

The XML declaration is a processing instruction that tells the processing agent that the document is an XML document. The mandatory version attribute indicates the version used in this document. The current version is 1.0. The declaration may use two optional attributes **encoding** and **standalone.**

1.2 Processing Instruction

Processing instruction start with **<? and end with ?>** they allow XML documents to contain special instruction that are used to pass parameters to the application. These parameters instruct the application about how to interpret the XML document.

Example

<?xml-stylesheet href=”simple.xsl” type=”text/xsl”?>

This processing instruction states that the XML document should be transformed using the stylesheet simple.xsl.

1.3 Comments

Like HTML comments may appear anywhere in the XML documents. An XML Comments starts with <!-- and ends with →. Everything within these characters

sequence will be ignored by the parsers and will not be parsed.

<!-- comment text -->

1.4 Document Type Declaration

The document type declaration(or DOCTYPE declaration) is used to specify the logical structure of the XML document. The structure is specified by imposing constraint on what tags can be used and where. A parser reads this section and checks whether the XML document has been written according to the rules specified.

A Document Type declaration may contain the following

1. Name of the root element.
2. Reference to an external DTD.
3. Element declaration
4. Entity declaration.

2. Body

This portion of the XML document contains textual data marked up by tags. It must have one element called the root element, which defines the content in the XML document. In an HTML document , the root element is <html>.

The root element contains other elements which in turn contains other elements and so on.

```
<?xml version = "1.0"?>
<contact-info>
   <name>SVIT</name>
   <company>Engineering College</company>
   <phone>(011) 123-4567</phone>
</contact-info>
```

Well-formed XML (V.Imp)

An XML document is said to be well-formed if it contains text and tags that conforms with basic XML well-formed constraints.

The following rules must be followed by the XML documents to be well formed.

1. **An XML document must have one and exactly one root element.**

   Eg:

valid XML Document

```
<contact>
        <person>
        <name>ABC</name>
        <number>12345</number>
        </person>
<contact>
```

invalid XML Document

```
<phonebook>
        <contact>
                <person>
                        <name>ABC</name>
                        <number>12345</number>
                </person>
        </contact>
<phonebook>
                        <name>ABC</name>
                        <number>12345</number>
</phonebook>
```

2. **All tags must be closed**

   Example

   <name>SVIT →inCorrect(No closing tag)

   <name>SVIT</name>→Correct

3. **All tags must be properly nested**

   Example

   This is incorrect nesting----→incorrect

   This is incorrect nesting----→correct

4. **XML tags are case sensitive**

   Example

   <Message>This is Incorrect</message>--→inCorrect (becoz opening tag not matching with closing tag)

   <Message>This is Incorrect</Message>--→Correct

5. **Attributes must be quoted(either single or double quotes)**

   Example
   <speed unit=rpm></speed>-→ inCorrect
   <speed unit="rpm"></speed>-→ Correct

6. **Certain characters are reserved for processing**
   Example
   <condition>if salary < 1000</condition> ---→incorrect
   <condition>if salary < 1000</condition> ---→correct

Predefined entities

in W3C XML specification, defines certain entities which cannot be used directly in XML document. All XML processors recognize those entities.

Result	Description	Entity Name	Entity Number
	non-breaking space		
<	less than	<	<
>	greater than	>	>
&	ampersand	&	&
"	quotation mark	"	"
'	apostrophe	' (does not work in IE)	'

Valid XML

Well-formed XML documents obeys only well-formed constraint. So valid XML Documents are those that

1. Are well-formed.
2. Comply with rules specified in the DTD or Schema.

The basic difference between a well-formed and a valid document is that the former is defined without any DTD or schema and the latter requires a DTD or schema. A DTD or schema specifies a set of rules , which the valid XML document must follow. The rules usually specify the name and content of the element that can occur in the valid document. XML parser check whether the XML documents are developed using these rules.

Purpose of DTD

XML lets applications share data easily. XML lets it happen by allowing you to make up your own set of tags. The purpose of a DTD is to define the legal building blocks of an XML document. It specifies the document structure with a list of legal elements. It is also used to specify a content model for each element and attributes used in an XML document. The content description is a part of the element declaration in the DTD and specifies the order and quantity of elements that can be contained within the element being declared.

An XML document is said to be valid with respect to a DTD, if it conforms with the rules specified by a given XML DTD.

Using a DTD in an XML Document

We should be aware of how DTDs and conforming XML document are linked. To validate an XML document against a DTD, we must tell the validator where to find the DTD so that it knows the rules to be verified during validation. A Document type Definition is used to make such a link and the keyword DOCTYPE is used for this purpose.

There are three ways to make this link

1. The DTD can be embedded directly in the XML document as a part of it-→called internal DTD
2. From an XML document, a reference to an file containing the DTD can be made→ called external DTD.

Internal DTD

When we embed a DTD in an XML, the DTD information is included within the XML document itself. Specifically, the DTD information is placed between the square brackets in a DOCTYPE declaration.

The general syntax for an internal Document Type Declaration is

<!DOCTYPE root-element [

<!-- doctype-declartion --

]>

The keyword DOCTYPE specifies that a DTD is to be used by the document. It states that, we are going to define the structure of the element **root-element.** Since every XML document must have one and exactly one root element, this is also a structure definition of the entire XML document.

The following rules must be followed

1. The keyword DOCTYPE must be in uppercase(well-formedness constraint)
2. The Document Type Declaration must appear before the first element in the document(well-formedness constraint).
3. The name following the word DOCTYPE(root-element in this case) must match with the name of the root element(top-level element) in the XML document(validity constraint)

Internal DTD Example

Employee.xml

```
<!DOCTYPE employee [
<!ELEMENT employee (empNo,name,salary)>
<!ELEMENT empNo (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT salary (#PCDATA)>
] >

<employee>
<empNo>101</empno>
<name>sai</name>
<salary>12000</salary>
</employee>
```

Disadvantages

1. This internal DTD is specific to this XML document only. Hence we cannot reuse the DTD.
2. In this approach the readability of the xml document will be reduced as document potentially becomes packed with markups.

External DTD

Another way of connecting a DTD to the XML document is to reference it within the XML document i.e create a separate document, put DTD information there, and point to it from the XML document. The general syntax for an external document Type Definition is

<!DOCTYPE root-element (SYSTEM or PUBLIC) "+ or - //vendorname//version//EN" "dtdFileName.dtd" >

+→ meansRegistered DTD from ISO
-→ means not Registered

Example

Emp.dtd

```
<!ELEMENT book (book-name,title)>
<!ELEMENT book-name (#PCDATA)>
<!ELEMENT title (#PCDATA)>
```

emp.xml

```
<!DOCTYPE employee SYSTEM "C:\Users\SVITIT\Desktop\EMP.DTD">
    <employee>
    <empNo>101</empNo>
    <name>sai</name>
    <salary>12000</salary>
    </employee>
```

Case 1: ***(Valid)***

```
<book>
<book-name> java</book-name>
<title>Core Java</title>
</book>
```

Case 2: ***(invalid becoz <title> is missing)***

```
<book>
<book-name> java</book-name>
</book>
```

Case 3: ***(invalid becoz <book-name> is missing)***

```
<book>
<title>Core Java</title>
</book>
```

Case 4: ***(invalid becoz order is not folowed)***

```
<book>
<title>Core Java</title>
<book-name> java</book-name>
</book>
```

Important points

1. In DTD if we use comma it means both child elements are mandatory and order is also mandatory.

<!ELEMENT book (book-name , title)>

2. In DTD if we use | it means either of child elements are allowed but both child elements are not allowed.

<!ELEMENT book (book-name | title)>

1. Element Type Declaration

Elements are primary building blocks in an XML document. Element type declaration set the rules for the type and number of elements that may appear in an XML document, how these elements may be nested, and what older they must appear in. the general syntax for an element declaration is

<!ELEMENT element-name type>

or

<!ELEMENT element-name (content)>

Here element-name is the name of the element we are defining. Here , the value of type can be either EMPTY or ANY. The content could indicate a specific rule, data, or another element.

The following rules must be followed

2. The keyboard ELEMENT must be in uppercase.
3. Element name are case-sensitive.
4. All elements types used in an XML document must be declared in the Document Type Definition(DTD) using an element type declaration.
5. The same name cannot be used in multiple element type declaration.
6. The name in the element type's end tag must match the name in the element type's start.

In DTD, elements are classified depending upon their content.

a. **Standalone elements:** These elements are also called singleton elements. They cannot contain anything and as a consequence they are also empty elements
b. **Simple elements:** These are elements that contains text or "**Parsed Character Data"** (represented as **#PCDATA** in your Data)

c. **Compound elements:** These elements contains text, other(children) elements, or both text and other(children) elements

Example

<!ELEMENT empname (#PCDATA) >
<!ELEMENT empsalary (#PCDATA) >

1.1 Standalone/Empty Elements

A standalone or empty element, as the name implies, cannot have any content. However, it may have attributes.

Empty elements are declared using the type keyword EMPTY(in Uppercase) as follows

<!ELEMENT element-name **EMPTY**>

Example:

The following example shows how to declare this element in our DTD

<!ELEMENT br EMPTY>

This element can then used in an XML declaration as

 or
</br>

1.2 Unrestricted Elements

It does not matter what your element contains, you can create such an element using the content model of ANY(in uppercase). These elements are declared as follows

<!ELEMENT element-name ANY>

The keyword **ANY** indicates that the content of the element element-name can be anything including text and other elements in any order and any number of times, as long as the XML well-formedness rule is followed.

1.3 Simple Elements

Simple elements cannot contains other elements. They can contains only text, which may have reference to entities. These elements, in general, are declared as follows

<!ELEMENT element-name (#PCDATA)>

Following is a declaration of the simple element greeting

<!ELEMENT greeting (#PCDATA)>

1.4 **Compound elements**:-These are elements can contain other elements. In the simplest case, an element contains one child ellemnt. It is speciied

<!ELEMENT element-name (child_element_name)>

Example

<!ELEMENT employee(name)>

This indicates that the element employee can contain another child element name exactly once.

Attributes

Attributes are used to describe elements or provide more information about elements. They appear in the starting tag of the element.

The syntax for specifying an attribute in an element is:

<element-name attribute-name="attribute-value">…</element-name>

Eg:

<employee gender="male">...</employee>

Emp.dtd

```
<!DOCTYPE employees [
<!ELEMENT employees (employee*)>
<!ELEMENT employee EMPTY>
]>
<employees>
<employee></employee>
<employee/>
</employees>
```

Cardinality operators in DTD

Cardinality specifies how many number of times an element can occur in a XML document. In DTD we have the following cardinality operators. They are

Symbol	Min	Max
*	0	n times
+	1	n times

?	0	1 time
No Symbol	1 time	1 time
|	Either/ or	

Examples

1. **<!ELEMENT books (book)>**

here no cardinality operator is used hence <book> tag is used only once in <books> tag.

2. **<!ELEMENT books(book*)>**
   here '*' is cardinality operator is used hence <book> tag is can be used for 0 to n times in <books> tag.

3. **<!ELEMENT books (book*)>**

<!ELEMENT book (book-name,price,author+,publications?)>

<!ELEMENT book ((book-name|title),price,author+,publications?)>

Example :1

Book.dtd

```
<!ELEMENT books (book*)>
<!ELEMENT book ((book-name|title),price,author+,publications?)>
<!ELEMENT book-name (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT price(#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT publication (#PCDATA)>
```

Book.xml

```
<books>
        <book>
                <book-name>servlets</book-name>
                <price>500</price>
                <author>SVIT/author>
                <author>IT</author>
                <publication>ABC</publication>
        </book>
</books>
```

Example :2

Emp1.dtd

```
<!DOCTYPE  employees  [
<!ELEMENT  employees   (employee+)>
<!ELEMENT   employee   ANY>
<!ELEMENT empname (#PCDATA)>
<!ELEMENT empsalary (#PCDATA)>
]>
<employees>

<employee>
The employee name is  <empname>SVIT</empname>
</employee>

<employee></employee>
<employee>
<empsalary>1200</empsalary>
</employee>
<employee>this employee information</employee>
</employees>
```

Limitation of DTD

The basic purpose of DTDs was to define languages based on SGML, which is considered to be parent of XML. Although DTD are well accepted and used very frequently, they have several limitations. Some of which are as follows

1. There are no built-in data type in DTDs.
2. No new Data type can be created in DTDs.
3. We cannot put any restrictions on text elements.
4. DTDs are written in a strange format and are difficult to validate.
5. Namespaces are not supported.
6. The use of cardinality(the number of occurrence of an element within its enclosing element) in DTDs is limited.
7. DTDs provide very limited support for modularity and reuse.

Introduction to XML Schema (XSD)

XML Schema Definition(XSD) is a specific XML schema document written using an XML Schema and typically has the filename extension ".xsd". so XSD and XML Schema are tow completely different things. XML Schema refers to a Schema language whereas XSD represents a Schema document written using the rules specified by that XML Schema Language. However sometimes XSD is referred to informally as XML Schema.

Strengths of schema

Schemas are XML based alternatives to DTDs in that they are used to create classes of XML documents that conform to the schema. XML schema is a much more powerful language than DTD. Here are some reasons.

1. XML Schemas provide must greater specificity than DTDs.
2. They support large number of built-in data type.
3. They are extensible to further additions.
4. They support the uniqueness and referential integrity constraints in a much better way.
5. It is easier to define data facets(restrictions on data).

XSD(XML Scheme Definition)

1. **Simple Type Elements**:-An element(tg) contains only data which has no child elements (or) attributes.
2. **Complex Type Elements**:-An Element(tag) contains either child or attributes, some times both is called Complex Type Elements.

Example:

```
<book>
<bid>10<bid>                    ----->Simple element
<bname sno="A">ABC</bname>----->Complex element
</book>
```

Simple type Element Syntax

Syntax:1

<xs:element name= " " type=" ">
</xs:element>
Here XSD datatype should be specified in type=" "

Java	**XSD**
int	xs:int
double	xs:double
String	xs:string
boolean	xs:boolean

Example:1

Define one XSD simple type element "bid" holds int type data

Ans) <xs:element name="eid" type="xs:int">
</xs:element>

Example:2

Define XSD for the following elements

element	**datatype**
eid	int
ename	string
sal	double

Ans) <xs:element name="eid" type="xs:int"/>
<xs:element name="ename" type="xs:string"/>
<xs:element name="esal" type="xs:double"/>

interchanging of elements is allowed.

<xs:element type="xs:double" name="esal"/>

XSD simple type element with restrictions

```
<xs:element name=" ">
<cs:simpleType>
<xs:restriction base=" ">
    -------
    -----
</xs:restriction>
</xsd:simpleTypes>
</xs:element>
```

xs:int type restrictions

Type	symbol	example
minInclusive	(>=)	empid>=10
maxInclusive	(<=)	empid<=100
minExclusive	(>)	empid>10
maxExclusive	(<)	empid<100

Example :3

Define an XSD element empid of type int that has limits min=1, max=500,(include current numbers)

XSD code

```
<xs:element name="empid">
    <xs:simpleType>
```

```
            <xs:restriction base="xs:int">
            <xs:minInclusive value="1"/>
            <xs:manInclusive value="500"/>
            </xs:restriction>
        </xs:simpleType>
</xs:element>
```

xs:string type restrictions

To specify the tange of characters in string type of data

1. Range:

    minLength (default=0)

    maxLength(default=no imit)

2. Exact length

    length

Example 4:

Define empname of type string, which accepts only 4 to 6 characters (min=4, max=6)

```
            <xs:element name="empname">
            <xs:simpleType>
            <xs:restriction base="xs:string">
            <xs:minLength value="4"/>
            <xs:manLength value="6"/>
            </xs:restriction>
            </xs:simpleType>
            </xs:element>
```

Complex Type element

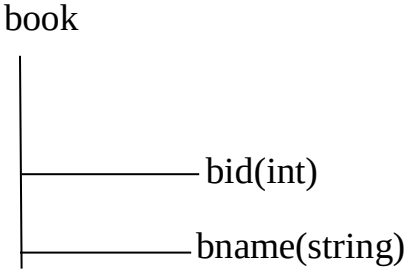
An element that contains child or attributes.

Syntax

```
<xs:element name=" ">
    <xs:complexType>
    <xs:sequence>
    [Child Elements ]
    </xs:sequence>
    </xs:compleType>
</xs:element>
```

Example

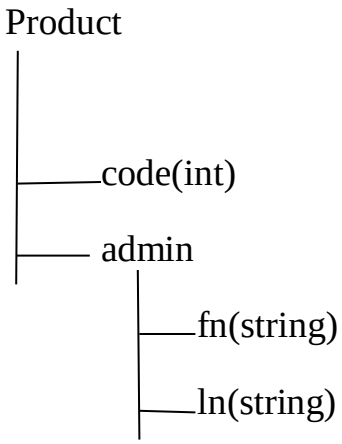
consider below tree model and write XSD code



XSD Code

```
<xs:element name="book">
<xs:complexType>
<xs:sequence>
<xs:element name="bid" type="xs:int/>
<xs:element name="bname" type="xs:string/>
</xs:sequence>
</xs:complexType>
</xs:element>
```

Example:2



XSD Code

```
<xs:element  name="product">
<xs:complexType>
<xs:sequence>
<xs:elemnt name="code"  Type="xs:int">
<xs:element  name="admin">
<xs:complexType>
<xs:sequence>
<xs:element  name="fn"  Type="xs:string">
<xs:element  name="ln"  Type="xs:string">
</xs:sequence>
</xs:seqeunce>
</xs:compleType>
<xs:sequence>
<xs:complexType>
</xs:element>
```

Displaying XML Document with CSS

Style sheet information can be provided to the browser for an XML document in two ways. First, a Cascading Style Sheet(CSS) file that has style information for the elements in the XML document can be developed. Second the XSLT style sheet technology, which was developed W#C can be used. Although using CSS is effective, XSLT provides far more power over the appearance of the document's display. On the other hand , XSLT is not yet available on all of the most commonly used browsers.

What is XSL

In HTML documents, tags are predefined but in XML documents, tags are not predefined. World Wide Web Consortium (W3C) developed XSL to understand and style an XML document, which can act as XML based Stylesheet Language.

An XSL document specifies how a browser should render an XML document.

Main parts of XSL Document

- **XSLT:** It is a language for transforming XML documents into various other types of documents.
- **XPath:** It is a language for navigating in XML documents.
- **XQuery:** It is a language for querying XML documents.
- **XSL-FO:** It is a language for formatting XML documents.

What is XSLT

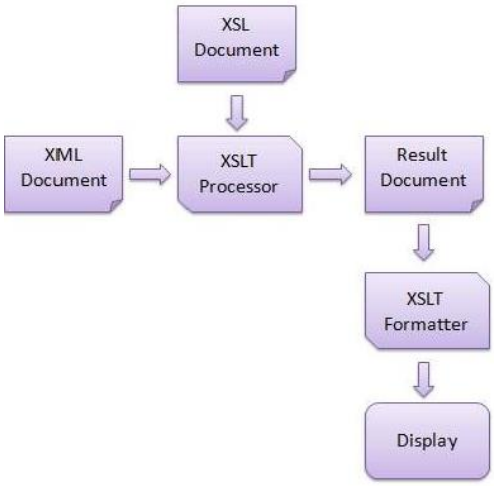
Before XSLT, first we should learn about XSL. XSL stands for EXtensible Stylesheet Language. It is a styling language for XML just like CSS is a styling language for HTML.

XSLT stands for XSL Transformation. It is used to transform XML documents into other formats (like transforming XML into HTML).

How XSLT Works

The XSLT stylesheet is written in XML format. It is used to define the transformation rules to be applied on the target XML document. The XSLT processor takes the XSLT stylesheet and applies the transformation rules on the target XML document and then it generates a formatted document in the form of XML, HTML, or text format. At the end it is used by XSLT formatter to generate the actual output and displayed on the end-user.

Image representation:



Advantage of XSLT

A list of advantages of using XSLT:

- XSLT provides an easy way to merge XML data into presentation because it applies user defined transformations to an XML document and the output can be HTML, XML, or any other structured document.
- XSLT provides Xpath to locate elements/attribute within an XML document. So it is more convenient way to traverse an XML document rather than a traditional way, by using scripting language.
- XSLT is template based. So it is more resilient to changes in documents than low level DOM and SAX.
- By using XML and XSLT, the application UI script will look clean and will be easier to maintain.
- XSLT templates are based on XPath pattern which is very powerful in terms of performance to process the XML document.
- XSLT can be used as a validation language as it uses tree-pattern-matching approach.
- You can change the output simply modifying the transformations in XSL files.

XSLT Syntax

Let's take an example to take a sample XML file and transform it into a well formatted HTML document.

See this example:

Create an XML file named employee.xml, having the following code:

Employee.xml

```
<?xml version = "1.0"?>
<class>
  <employee id = "001">
    <firstname>Aryan</firstname>
    <lastname>Gupta</lastname>
    <nickname>Raju</nickname>
    <salary>30000</salary>
  </employee>
  <employee id = "024">
    <firstname>Sara</firstname>
    <lastname>Khan</lastname>
    <nickname>Zoya</nickname>
    <salary>25000</salary>
  </employee>
  <employee id = "056">
    <firstname>Peter</firstname>
    <lastname>Symon</lastname>
    <nickname>John</nickname>
    <salary>10000</salary>
  </employee>
</class>
```

Define an XSLT stylesheet document for the above XML document. You should follow the criteria give below:

- Page should have a title employee.
- Page should have a table of employee's details.
- Columns should have following headers: id, First Name, Last Name, Nick Name, Salary
- Table must contain details of the employees accordingly.

Step1: Create XSLT document

Create the XSLT document which satisfies the above requirements. Name it as employee.xsl and save it in the same location of employee.xml.

Employee.xsl

```
<?xml version = "1.0" encoding = "UTF-8"?>
<xsl:stylesheet version = "1.0"
xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">

  <xsl:template match = "/">
    <!-- HTML tags
      Used for formatting purpose. Processor will skip them and browser
        will simply render them.
    -->
    <html>
      <body>
        <h2>Employee</h2>
        <table border = "1">
          <tr bgcolor = "#9acd32">
            <th>ID</th>
            <th>First Name</th>
            <th>Last Name</th>
            <th>Nick Name</th>
            <th>Salary</th>
          </tr>
          <!-- for-each processing instruction
          Looks for each element matching the XPath expression
          -->
          <xsl:for-each select="class/employee">
            <tr>
              <td>
                <!-- value-of processing instruction
          process the value of the element matching the XPath expression
                -->
                <xsl:value-of select = "@id"/>
              </td>
              <td><xsl:value-of select = "firstname"/></td>
              <td><xsl:value-of select = "lastname"/></td>
              <td><xsl:value-of select = "nickname"/></td>
              <td><xsl:value-of select = "salary"/></td>
            </tr>
```

```
        </xsl:for-each>
      </table>
    </body>
  </html>
 </xsl:template>
</xsl:stylesheet>
```

Step2: List the XSLT document to the XML document

Update employee.xml document with the following xml-stylesheet tag. Set href value to employee.xsl

```
<?xml version = "1.0"?>
<?xml-stylesheet type = "text/xsl" href = "employee.xsl"?>
<class>
...
</class>
```

Updated "employee.xml"

```
<?xml version = "1.0"?>
<?xml-stylesheet type = "text/xsl" href = "employee.xsl"?>
<class>
  <employee id = "001">
    <firstname>Aryan</firstname>
    <lastname>Gupta</lastname>
    <nickname>Raju</nickname>
    <salary>30000</salary>
  </employee>
  <employee id = "024">
    <firstname>Sara</firstname>
    <lastname>Khan</lastname>
    <nickname>Zoya</nickname>
    <salary>25000</salary>
  </employee>
  <employee id = "056">
    <firstname>Peter</firstname>
    <lastname>Symon</lastname>
    <nickname>John</nickname>
    <salary>10000</salary>
```

```
  </employee>
</class>
```

Step3: View the XML document in Internet Explorer

The output will look like this:

Output: