“ UNIT 2 DATABASE AND XML

Structure **Page Nos.**

2.0 INTRODUCTION

XML stands for Extensible Markup Language. It is used to describe documents and data in a standardised, text-based format, easily transportable *via* standard Internet protocols. XML, is based on the *mother* of all markup languages–Standard Generalised Markup Language (SGML).

SGML is remarkable inspiration and basis for all modern markup languages. The first popular adaptation of SGML was HTML, primarily designed as a common language for sharing technical documents. The advent of the Internet facilitated document exchange, but not document display. Hypertext Markup Language (HTML) standardises the description of document layout and display, and is an integral part of every Web site today.

Although SGML was a good format for document sharing, and HTML was a good language for describing the document layout in a standardised way, there was no standardised way of describing and sharing data that was stored in the document. For example, an HTML page might have a body that contains a listing of today’s share prices. HTML can structure the data using tables, colours etc., once they are rendered as HTML; they no longer are individual pieces of data to extract the top ten shares. You may have to do a lot of processing.

Thus, there was a need for a tag-based markup language standard that could describe data more effectively than HTML, while still using the very popular and standardised HTTP over the Internet. Therefore, in 1998 the World Wide Web Consortium (W3C) came up with the first Extensible Markup Language (XML) Recommendations.

Now, the XML (eXtended Markup Language) has emerged as the standard for structuring and exchanging data over the Web.

XML can be used to provide more details on the structure and meaning of the data pages rather than just specifying the format of the Web pages. The formatting aspects can be specified separately, by using a formatting language such as XSL (eXtended Stylesheet Language). XML can describe data as records of data store or as a single document.

As a language, XML defines both syntax and grammar rules. The rules are called Document Type Definition (DTD), and are one of the major differences between HTML and XML. XML uses metadata for describing data. The metadata of XML is not complex and adds to the readability of the document. XML, like HTML, also uses tags to describe data however, tags, unlike HTML, describes data and not how to present it. To display XML data, you often transform it using XSLT into an HTML page.

HTML is comprised of a defined set of tags, XML on the other hand has very few defined tags. However, it does not mean that XML is powerless, the greatest power of XML is that it is extensible. You can create your own tags with your own semantic meaning. For example, you can create a tag to use for your customer information data such as:

<Customer_First_Name> Manoj </Customer_ First_Name>

This tag has meaning for you and, thus, to your application. This tag has been created by you to designate customer's first name but its tells nothing about its presentation. But how is this tag useful to us? Consider now that data stream contains multiple customers information. If you want to find all customers with first name "Manoj" you can easily search for the <Customer_First_Name> tags. You cannot perform such types of operation in HTML with the same ease and consistency, as HTML was not designed for such purposes.

Please note: XML is case sensitive whereas HTML is not. So, you may see that XML and databases have something in common. So, let us discuss more about XML and databases in this unit.

2.1 OBJECTIVES

After going through this unit, you should be able to:

- identify XML & XML Document Basics;
- define XML Data Type Definition (DTD);
- identify XML Schema;
- discuss XML Transformation (XSLT) ;
- give overview of XPath, XLink & XQuery;
- give overview of XML Databases & Storage of XML data, and
- discuss a few real life examples of the usage of XML.

2.2 STRUCTURED, SEMI STRUCTURED AND UNSTRUCTURED DATA

The data can be categorised in three categories on the basis of its schema: structured, Semi-structured & Unstructured.

Information stored in databases is known as *structured data* because it is represented in a predefined format. The DBMS ensures that all data follows the defined structures and constraints specified in the schema.

In some applications, data is collected in an *ad-hoc* manner, way before you decide on how to store and manage it. This data may have a certain structure, but not all the information collected will have identical structure. This type of data is termed as semi-structured data. In semi-structured data, the schema or format information is mixed with the data values, since each data object can have different attributes that are not known earlier. Thus, this type of data is sometimes referred to as self-describing data.

A third category is known as unstructured data, as there is very limited indication of the type of data. For example, a text document that contains information embedded within it such as web pages in HTML.

2.3 XML HIERARCHICAL (TREE) DATA MODEL

The basic object in XML is the XML document. There are two main structuring concepts that construct an XML document:

Elements and attributes

Attributes in XML describe elements. Elements are identified in a document by their start tag and end tag. The tag names are enclosed between angular brackets <…>, and end tags are further identified by a backslash </…>. Complex elements are constructed from other elements hierarchically, whereas simple elements contain data values. Thus, there is a correspondence between the XML textual representation and the tree structure. In the tree representation of XML, internal nodes represent complex elements, whereas leaf nodes represent simple elements. That is why the XML model is called a tree model or a hierarchical model.

There are three main types of XML documents:

1) **Data-centric XML documents:** These documents have small data items that follow a specific structure, and hence may be extracted from a structured database. They are formatted as XML documents in order to exchange or display them over the Web.

2) **Document-centric XML documents:** These are documents with large amounts of text, such as articles. There is little or no structured data elements in such documents.

3) **Hybrid XML documents:** These documents may have parts of both that is structured data and textual or unstructured.

2.4 XML TAG SETS

The following section presents a closer look at some of the syntactical rules of XML and also looks at why tags are used at all.

Most tags, and all user-defined tags in the XML document instance (i.e. data section), of an XML document follow the convention of a start tag:

```
< Some_Tag >
Followed by an end tag:
</ Some_Tag >
```

Some elements in an XML document contain no data – more specifically the data is contained only in one-or-more attributes. In this case, you can reduce the notation to the “empty” form of tag:

```
< Some_Tag />
```

Note: The white space after the “<” and before the “>” or “/>” is not required but only used here for asthetic purposes. Also, we will use “…….” in some examples to show additional options/information may or may not exist but is omitted for brevity.

XML document declaration: Every XML document must start with an XML declaration: <?xml ?>. The W3C strongly recommends that at a minimum, you should include the version information to ensure parser compatibility:

```
<?xml version=”1.0” ?>
```

XML Comments: Comments, in XML are same as they are used in programming languages. They are delimited by a special tag: <!-- -->. **Please note:** Two dashes are required for both the start and end tag. For example:

```
<!-- A comment -->
```

XML promotes logical structuring of data and document organisation through the hierarchically nested nature of tag sets and it can create tags that have meaning for your application.

```
<a> ABC Corporation
  <b> K Kumar</b>
</a>
```

Is certainly less meaningful than:

```
<CompanyName> ABC Corporation
  <ContactName> K Kumar</ContactName>
</CompanyName>
```

2.5 COMPONENTS OF XML DOCUMENT

An XML document have three parts:

- The XML processing Instruction(s), also called the XML declaration;
- The Document Type Declaration;
- The document instance.

2.5.1 Document Type Declaration (DTD)

A DTD is used to define the syntax and grammar of a document, that is, it defines the meaning of the document elements. XML defines a set of key words, rules, data types, etc to define the permissible structure of XML documents. In other words, we can say that you use the DTD grammar to define the grammar of your XML documents. The form of DTD is:

```
<! DOCTYPE name >
```

Or
<! DOCTYPE name [a_dtd_definition_or_declaration]>

The name, while not necessarily the document name, must be the same name as that of the document root node.

The second point of interest with DOCTYPE is that after the name you can declare your Document Type Definition (DTD), the assembling instructions for the document.

You can define them "in-line" or reference external definitions - which is something like an "include" or "import" statement in a language like C. The advantage of creating one-or-more external DTDs is that external DTDs are reusable – more than one XML document may reference the same DTD. DTDs can also reference each other. But how do we define the structure of a XML document?

The structure of the XML document is created by defining its elements, attributes, their relation to one another and the types of data that each may or may not have. So, how do you define these elements and attributes and how are they related to one another in a DTD?

Elements are defined using the <!ELEMENT> keyword. Its attributes are related to the <!ATTLIST> keyword. The following are a few rules for XML DTD notation:

- A * following the element name implies that the element can be repeated zero or more times in the document.
- A + following the element name means that the element can be repeated one or more times. Such elements are required at least once.
- A ? following the element name means that the element can be repeated zero or one times.
- An element appearing without any of the symbols as above must appear exactly once in the document.
- The type of the element is specified using parentheses following the element. If the parentheses include names of other elements, the element that is being defined would be the children of the element in the tree structure. If the parentheses include the keyword #PCDATA or one of the other data types available in XML DTD, the element is at the leaf node of the tree. PCDATA stands for Parsed Character Data, which is roughly similar to a string data type.
- Parentheses can be nested when specifying elements.
- A bar symbol (e1 | e2) specifies that either e1 or e2 can appear in the document.

For example, if your XML document models the components of a house you might define an element, foundation, that contains another element, floor, and has two attributes, material and size. You would write this as follows:

```
<!ELEMENT foundation (floor) >
<!ELEMENT floor (#PCDATA) >
<!ATTLIST foundationmaterial (#PCDATA) >
<!ATTLIST foundationsize (#PCDATA) >
```

Another short example that will appeal to anyone dealing with customers is as follows:

```
<?xml version = "1.0" encoding = "UTF-8" ?>
<!DOCTYPE CustomerOrder >

<!ELEMENT CustomerOrder (Customer, Orders*) >

<!ELEMENT Customer (Person, Address+) >
<!ELEMENT Person (FName, LName) >
<!ELEMENT FName (#PCDATA) >
<!ELEMENT LName (#PCDATA) >
<!ELEMENT Address (#PCDATA) >
<!ATTLIST Address
        AddrType ( billing |  home ) "home" >

<!ELEMENT Orders (OrderNo, ProductNo+) >
<!ELEMENT OrderNo (#PCDATA) >
<!ELEMENT ProductNo (#PCDATA) >
]>
```

In the CustomerOrder example, **please note** the following points:

- A CustomerOrder element contains one-and-only-one Customer element and zero-or-more Orders elements (specified by the * in Orders*).

- The Customer element, contains one-and-only-one Person (defined by name) element and one-or-more Address elements (designated by the + in Address+). Thus, showing an emerging hierarchy that defines the structure. **Please note**: In the defined structure, some elements must exist, some may exist once or more and some may or may not.

- The elements FName and LName do not include elements themselves but have something called #PCDATA; the parsed character data.

- Now look at the attribute declaration:

```
<!ATTLIST Address
 AddrType (billing | home) "home">
```

Here an *attribute* AddrType is declared and is associated with the element Address. Furthermore, the attribute is declared to have one of two values (billing or home) and if none were specified then the default would be "home".

Let us take an example.

Program 1: A Sample XML Document

```
<?xml version = "1.0" encoding = "UTF-8" ?>
<! DOCTYPE CustomerOrder
SYSTEM "http://www.mycompany.com/dtd/order.dtd" >
<CustomerOrder>
        <Customer>
                <Person>
                        <FName> Jayesh </FName>
                        <LName> Kumar </LName>
                </Person>
                <Address AddrType = "home">
                        D-204, Saket, New Delhi 110012 </Address>
                <Address AddrType = "billing">
                        C-123, Janpath, NewDelhi 110015</Address>
        </Customer>
        <Orders>
                <OrderNo> 10 </OrderNo>
                <ProductNo> 100 </ProductNo>
                <ProductNo> 200 </ProductNo>
        </Orders>
        <!-- More Customers can be put here ... -->
</CustomerOrder>
```

This is of an example of an XML data stream containing Customer Orders. As you can see, a <CustomerOrder> contains a <Customer> node that, in turn, contains information about a customer. Notice in this example that, a customer can have only one name, but two addresses – "Home" and "Billing".

2.5.2 XML Declaration

The XML processing instruction declares, the document to be an XML document. For an application or parser this declaration is important. It may also include: the version of XML, encoding type; whether the document is stand-alone; what namespace, if any, is used etc. and much more. The encoding attribute is used to inform the XML processor of the type of character encoding that is used in the document. UTF-8 and UTF-16 and ISO-10646-UCS-2 are the more common encoding types. The standalone attribute is optional and if, present, has a value of yes or no. The following is an example of an XML declaration:

```
<?xml version = ”1.0” encoding = “UTF-16”  standalone=”yes” ?>
```

2.5.3 Document Instance

The other components of an XML document provide information on how to interpret actual XML data, whereas the document instance is the actual XML data. There are basically three types of elemental markup that are used in the making of an XML document: i) The document's root element; ii) child elements to the root; and iii) attributes.

Document:

i) Root Element: There is no difference between the document root element and other elements in an XML document except that the root is the root. The document root element is required if and only if a document type declaration (DOCTYPE) is present. Document root element must have the same name as the name given in the DOCTYPE declaration. If it does not, the XML document will not be valid.

ii) Child Elements to the Root: Elements are nodes in the XML hierarchy that may contain other nodes and may or may not have attributes assigned to them. Elements may or may not contain a value.

iii) Attributes: Attributes are properties that are assigned to elements. They provide additional information about the element to which they are assigned.

☞ Check Your Progress 1

1) What is semi-structured data?

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

2) What is XML? How does XML compare to SGML and HTML?

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

3) Why is XML case sensitive, whereas SGML and HTML are not?

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

4) Is it easier to process XML than HTML?

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

5) Why is it possible to define your own tags in XML but not in HTML?

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

6) Discuss the advantages of XML?

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

7) What is an XML element? What is an XML attribute?

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

8) Which three attributes can appear in an XML declaration?

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

2.6 XML SCHEMA

The W3C defines XML Schema as a structured framework of XML documents. Schema is a definition language with its own syntax and grammar, It provides a means to structure XML data and does so with semantics. Unlike a DTD, XML Schema are written in XML. Thus, you do not need to learn a second markup language for the purpose of providing document definitions. Schemata are actually composed of two parts: Structure and datatypes.

2.6.1 XML Schema Datatypes

There are two kinds of datatypes in XML Schema: Built-in and User-defined.

The built in datatypes include the primitive datatypes and the derived datatypes. The primitive types include, but are not limited to:

- string
- double
- recurringDuration
- decimal
- Boolean

The derived types are derived from primitive datatypes and include:

- integer: Derived from decimal
- nonPositiveInteger: Derived from integer
- CDATA: Derived from string
- time: Derived from recurringDuration

The user-defined datatypes are those types that are derived from either a built in datatype or other user-defined type.

The Simplest Type Declaration

The simple types are defined for basic elements, such as a person’s first name, in the following manner:

```
<simpleType name = “FirstName” type = “string” />
```

The value of the name attribute is the type name you are defining and expect in your XML documents. The type specifies the datatype upon which the type is based defined by you.
The value of the type attribute must be either a primitive type, such as string, or a derived built-in type, such as integer. You cannot define a simpleType based on a user-defined type. When you need to define types, based on a user-defined type you should use the complexType.

Furthermore, simpleType definitions cannot declare sub-elements or attributes; for such cases you need to use the complexType. However, the simpleType can define various constraining properties, known in XML Schema as facets, such as minLength or Length. This is accomplished by applying a restriction as shown in the following example:

```
<simpleType name=“FirstName”>
        <restriction base =  “string”>
                <minLength value = “0” />
                <maxLength value = “25” />
        </restriction>
</simpleType>
```

Lastly, simpleTypes may be used as the basis of complexTypes.

The Complex Type: <complexType>

The complexType is used to define types that are not possible with the simpleType declaration. complexTypes may declare sub-elements or element references.

```
<complexType name = “colonialStyleWindow”
        type = “window” >
  <sequence>
    <element name = “frame” type = “frame Type” />
  </sequence>
</complexType>
```

The <sequence> element is used to define a sequence of one or more elements. In the above example colonialStyleWindow has only one sub-element but it could have more, as you will see in Defining a complexType By Example. There are additional control tags, such as <choice>, which you may use. ComplexTypes may also declare attributes or reference attribute groups.

```
<complexType name = “colonialStyleWindow”
          type = “window” >
  <sequence>
    <element name = “frame” type = “frame Type” />
  </sequence>
  <attribute name = “width” type = “integer” />
  <attribute name = “height” type = “integer” />
</complexType>
```

Or

```
<complexType name = “colonialStyleWindow”
          type = “window” >
  <sequence>
    <element name = “frame” type = “frame Type” />
  </sequence>
  <attributeGroup ref = “windowDimensions” />
</complexType>
```

However, the real power and flexibility of complex types lies in their extensibility – that you can define two complexTypes and derive one from the other. A detailed discussion on them is beyond the scope of this unit. However, let us explain them with the help of an example:

```
<!-- Define a complexType -->
<complexType name = “window” type = “someType”/>

<!-- Now lets define a type based on it! -->
<complexType name = “colonialStyleWindow”
          type = “window” … />
```

Defining a complexType by Example: Let us look at a more interesting and complete example of defining a complexType. Here, we define a type Customer that declares and must have one Name element and one Address element. The Person element is of the Name type, defined elsewhere, and the Address type is the Address type, the definition of which follows Customer. Examining the definition of the Address type, you see that, it in turn, declares a sequence of elements: Street, City, PostalCode, and Country. A partial schema for the complexType AddrType may be:

```
<complexType name = “Address”
          type = “AddrType” >
          <sequence>
                    <element name = “Street” type = “string” />
                              …..
                      </sequence>
</complexType>
```

Given this Schema, the following XML data fragment would be valid:

```
          <Customer>
                    <Person>
                              <FName> Jayesh </FName>
                              <LName> Kumar </LName>
                    </Person>
```

```
                <Address AddrType = "Home">
                        <Street> A-204, Professor's Colony </Street>
                        <City> New Delhi </City>
                        <State> DL </State>
                        <PostalCode> 110001</PostalCode>
                        <Country> INDIA </Country>
                </Address>
                <Address AddrType = "billing">
                        <Street> B-104, Saket</Street>
                        <City> New Delhi </City>
                        <State> DL </State>
                        <PostalCode> D-102345</PostalCode>
                        <Country> INDIA </Country>
                </Address>
        </Customer>
```

2.6.2 Schema vs. DTDs

Both DTDs and Schema are document definition languages. Schemata are written in XML, while DTDs use EBNF (Extended Backus Naur Format) notation. Thus, schemata are extensible as they are written in XML. They are also easy to read, write and define.
DTDs provide the capability for validation the following:

- Element nesting.
- Element occurrence constraints.
- Permitted attributes.
- Attribute types and default values.

However, DTDs do not provide control over the format and data types of element and attribute values. For example, once an element or attribute has been declared to contain character data, no limits may be placed on the length, type, or format of that content. For narrative documents such as, web pages, book chapters, newsletters, etc., this level of control may be all right. But as XML is making inroads into more record-like applications, such as remote procedure calls and object serialisation, it requires more precise control over the text content of elements and attributes. The W3C XML Schema standard includes the following features:

- Simple and complex data types
- Type derivation and inheritance
- Element occurrence constraints
- Namespace-aware element and attribute declarations.

Thus, schema can use simple data types for parsed character data and attribute values, and can also enforce specific rules on the contents of elements and attributes than DTDs can. In addition to built-in simple types (such as string, integer, decimal, and dateTime), the schema language also provides a framework for declaring new data types, deriving new types from old types, and reusing types from other schemas.

2.7 XML PARSER

Figure 1 shows the interaction between application, XML parser, XML documents and DTD or XML Schema.. An XML source document is fed to the parser, that loads a definition document (DTD or XML Schema) and validates the source document from these definition document.

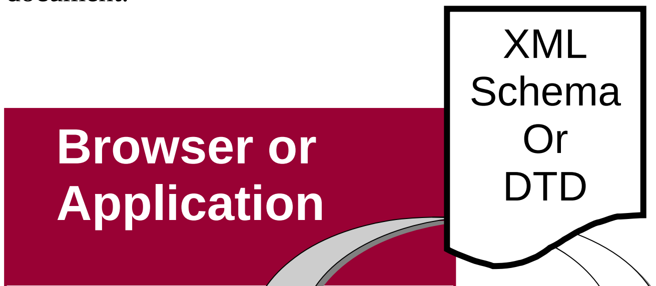

Figure 1: An XML Parser

XML parsers know the rules of XML, which obviously includes DTDs or XML Schemata, and how to act on them. The XML parser reads the XML document instance, which may be a file or stream, and parse it according to the specifications of XML. The parser creates an in-memory map of the document creating traversal tree of nodes and node values.

The parser determines whether the document is well-formed. It may also determine if the document instance is valid. But what is a well-formed XML document?

Well-formed XML document contains the required components of an XML document that has a properly nested hierarchy. That is, all tag sets are indeed sets with a begin and end tag, and that intersecting tags do not exist. For example, the following tags are not properly nested because <a> includes <b> but the end tag of <b> is outside the end tag of <a>:

```
<a><b> </a></b>
```

The correct nesting is:

```
<a> <b> </b> </a>
```

A validating parser interprets DTDs or schemata and applies it to the given XML instance. Given below are two popular models for reading an XML document programmatically:

DOM (Document Object Model)

This model defines an API for accessing and manipulating XML documents as tree structures. It is defined by a set of W3C recommendations. The most recently completed standard DOM Level 3, provides models for manipulating XML documents, HTML documents, and CSS style sheets. The DOM enables us to:

- Create documents and parts of documents.
- Navigate the documents.
- Move, copy, and remove parts of the document.
- Add or modify attributes.

The Document Object Model is intended to be an operating system- and language-independent, therefore, the interfaces in this model are specified using the Interface Description Language (IDL) notation defined by the Object Management Group.

Simple API for XML (SAX)

The Simple API for XML (SAX) is an event-based API for reading XML documents. Many different XML parsers implement the SAX API, including Xerces, Crimson, the Oracle XML Parser for Java, etc. SAX was initially defined as a Java API and is primarily intended for parsers written in Java. However, SAX has been ported to most other object-oriented languages, including C++, Python, Perl, and Eiffel. The SAX API is unusual among XML APIs because it is an event-based push model rather than a tree-based pull model, as the XML parser reads an XML document in real time. Each time the parser sees a start-tag, an end-tag, character data, or a processing instruction, it tells the program. You do not have to wait for the entire document to be read before acting on the data. Thus, the entire document does not have to reside in the memory. This feature makes SAX the API of choice for very large documents that do not fit into available memory.

2.8 XML NAMESPACES

Namespaces have two purposes in XML:

1) To distinguish between elements and attributes from different vocabularies with different meanings that happen to share the same name.

2) To group all the related elements and attributes from a single XML application together so that software can easily recognise them.

The first purpose is easier to explain and grasp, but the second purpose is more important in practice.

Namespaces are implemented by attaching a prefix to each element and attribute. Each prefix is mapped to a URI by an xmlns:prefix attribute. Default URIs can also be provided for elements that do not have a prefix. Default namespaces are declared by xmlns attributes. Elements and attributes that are attached to the same URI are in the same namespace. Elements from many XML applications are identified by standard URIs. An example namespace declaration that associates the namespace prefix ‘lib’ with the namespace name http://www.library.com/schema is shown below:

```
<book xmlns:lib = 'http://www.library.com/schema'>
        <!-- the "lib" prefix is now bound to http://ecommerce.org/schema
                for the element "book" and its contents -->
</book>
```

In an XML 1.1 document, an Internationalised Resource Identifier (IRI) can be used instead of a URI. An IRI is just like a URI except it can contain non-ASCII characters such as é: etc. In practice, parsers do not check that namespace names are legal URIs in XML 1.0, so the distinction is mostly academic.

2.9 XSL TRANSFORMATIONS (XSLT)

XSLT stands for XML Stylesheet Language Transformations and is yet another widely used and open standard defined by the W3C. Although the W3C defines XSLT as “a language for transforming documents” it is more than that. Unlike XML, XSLT is an active language permitting you to perform Boolean logic on nodes and selected XML sub-trees. Thus, it is closer to a programming language.

It is precisely because of its programmable nature that XSLT enables you to write XSLT transformation documents (a sort of programs). You use these "programs", known as XSL stylesheets (denoted by convention with the file type .XSL) in conjunction with an XSLT processor to transform documents. Although designed for transforming source XML documents into a new target XML documents, XSLT can transform an XML document into another type of text stream, such as an HTML file. A common use of XSL stylesheets is to translate between Schema formats.

The XSLT Process – Overview

In the case of XSLT, the processor loads a source document and using the already loaded stylesheet, transforms the source into a target document.

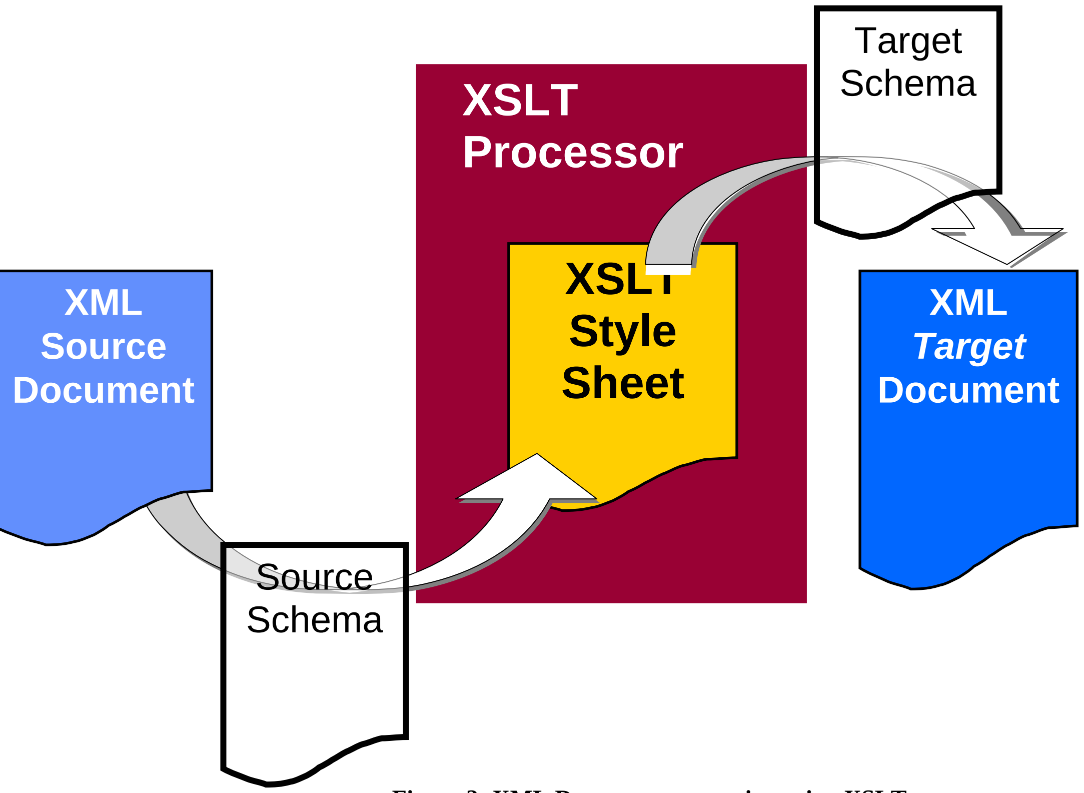

Figure 2: XML Document conversion using XSLT

The XSLT process first loads the specified stylesheet. The process then parses the stylesheet and loads the stylesheet templates into memory. It then traverses the source document, node by node, comparing the node values to the directives (or "search conditions") of the stylesheet templates. If there is a match between the current source document node and one of the templates, the process applies the template to the current node. This process continues until the processor has finished traversing the source document node tree and applied all matching templates. The result is a new, transformed document that the XSLT processor then emits as a stream or to file.

In order to perform any of the transformations the right tools; namely, an XSLT processor and a proper XSLT Stylesheet is required. The stylesheet is prefaced with the familiar <?xml?> declaration. But you also need to include the "stylesheet node" which declares the stylesheet namespace. You accomplish this by following your XML processing declaration with:

```
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <!-- The rest of your stylesheet goes here! -->
</ xsl:stylesheet>
```

For example, perhaps your XML source document has the following data:

```
<OrderNo>
  <ProductNo> 100 </ProductNo>
  <ProductNo> 200 </ProductNo>
  <ProductNo> 300 </ProductNo>
</OrderNo>
```

However, you want to display only the ProductNo data and do so as an HTML list:
Products in this order:

```
<UL>
  <LI> 100 </LI>
  <LI> 200 </LI>
  <LI> 300 </LI>
</UL>
```

The template will need to contain the <UL> and <LI> HTML codes, but what is needed is the means to select the nodes value you desire to insert between each <LI> and </LI>.

The Elements - Templates

The most basic tools at your disposal in XSLT are the <xsl:template> and <xsl:apply-templates/>.   The former is used to define a rule and the later to apply it.

<xsl:template/>

The <template> tag is used to select a node or node sub-trees. By using the tag's match attribute you can set a pattern to identify the node upon which you wish to apply a rule – or transformation. The pattern value can be a node path, such as Customer/Order/OrderNo, or it can be a predefined pattern value, such as * to select all nodes.

In the previous example, you wanted to select all ProductNo values from a particular Customer node. For the sake of simplicity, we will omit how to iterate over all Customers or Orders for a particular Customer. Let us assume, you have managed to select a particular Customer/Orders/OrderNo and now you wish to display the ProductNo values in that OrderNo.

```
<template match = "Customer/Orders/OrderNo" >
  <! -- apply some rules -->
</template>
```

But this only shows how to select the node or node set on which you wish to work. Now you need to apply some rules – or changes.

<xsl:apply-templates />

To apply rules you use the <apply-templates> rule. You can apply a template rule to all child nodes of the currently selected node (selected using <template match = "…" />) by using <apply-template /> without a select attribute value pair. Alternatively, you can apply the rule to all child nodes in the set that meet a specific selection criteria by using <apply-template select = "…" /> where the ellipses are some pattern such as ProductNo. The following fragment should help things come together:

```
<xsl:stylesheet ...>
  <html>
    <body>
```

```
      <ul>
      <xsl:template
           match="Customer/Orders/OrderNo">
       <li>
         <xsl:apply-templates select="ProductNo" />
              </li>
             </xsl:template>
             </ul>
             </body>
  </html>
</xsl:stylesheet>
```

Here we have a full example that uses a few XSLT elements not yet covered, such as the <for-each> element that is used to iterate over a selection.

```
<?xml version='1.0'?>
<xsl:stylesheet
       xmlns:xsl="http://www.w3.org/TR/2000/CR-xsl-20001121/">
<xsl:template match="/">
  <HTML>
    <BODY>
      <TABLE border = “3”>
        <xsl:for-each select=“Customer/Orders/OrderNo">
          <xsl:for-each select=“Customer/Orders/ProductNo">
          <TR>
              <TD> <xsl:value-of select=“OrderNo"/></TD>
              <TD> <xsl:value-of select=“ProductNo"/></TD>
          </TR>
          </xsl:for-each>
          <TR></TR>
        </xsl:for-each>
      </TABLE>
    </BODY>
  </HTML>
</xsl:template>
<xsl:stylesheet>
```

Thus, the goal is to create an HTML document that has a table with two columns, one for OrderNo and one for ProductNo, and print out all the OrderNo elements for a Customer. Between each different OrderNo, there will be a blank row.

The first <template> element selects the document, meaning the transformation will apply to the entire source document. Alternatively, you could select a subset of your source document by using different match criteria. Skipping over the HTML code, the first interesting XSLT element is a <for-each> element that iterates over Customer/Orders/OrderNo elements. This means that all the OrderNos for this Customer will be processed.

Recall that each OrderNo can have multiple products, or ProductNo elements. For this reason, the stylesheet has another <for-each> element that iterates over the ProductNo elements. Now, notice that inside the iteration of the ProductNo elements, there are two <value-of> elements. These XSLT elements select the values of the OrderNo and ProductNo respectively and insert them into separate columns.

As you saw in this example, there is more to XSLT than template and apply-templates.

<xsl: value-of select = “...” />

One of the more common XSLT elements to use is the <value-of …>, which creates a new node in the output stream based on the value of the node you select. It also contains many more elements, however, a discussion on those are beyond the scope of this unit.

☞ Check Your Progress 2

1) What are some of the requirements for an XML document to be well-formed?

……………………………………………………………………………………..

. ……………………………………………………………………………………..

……………………………………………………………………………………..

2) What are two XML constructs that let you specify an XML document's syntax so that it can be checked for validity?

………………………..…………………………………………………………..

………………………..…………………………………………………………..

……………………….……………………………………………………………..

3) What is the difference between a well formed XML document and a valid XML Document?

………………………………………………………………………………………

…………………….……………………………………………………………..

………………………………………………………………………………………

4) Why is the following XML document not well-formed?

```
<?xml version = "1.0" standalone="yes"?>
<employee>
   <name>Jayesh</name>
   <position>Professor</position>
</employee>
<employee>
   <name>Ramesh</name>
   <position>Clerk</position>
</employee>
```

………………………………………………………………………………………

………………………..…………………………………………………………..

………………………………………………………………………………………

………………………………………………………………………………………

5) What's wrong with this XML document?

```
<?xml version = "1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE document [
<!ELEMENT document (employee)*>
<!ELEMENT employee (hiredate, name)>
]>
<document>
   <employee>
      <hiredate>October 15, 2005</hiredate>
      <name>
         Somya Mittal
      </name>
   </employee>
```

```
</document>
```

…………………………………………………………………………………………

……………………….……………………………………………………………..

…………………………………………………………………………………………

…………………………………………………………………………………………

6) Where do you see a problem with this XML document?

```
<?xml version = "1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE document [
<!ELEMENT document (employee)*>
<!ELEMENT employee (name, hiredate)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT hiredate (#PCDATA)>
]>
<document>
   <employee>
      <hiredate>October 15, 2005</hiredate>
      <name>
         Somya Mittal
      </name>
   </employee>
</document>
```

…………………………………………………………………………………………

..……………………………………………………………………………………..

…………………………………………………………………………………………

…………………………………………………………………………………………

7) Describe the differences between DTD and the XML Schema?

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

8) Which namespace uses XML schemas?

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

9) How can you use an XML schema to declare an element called <name> that holds text content?

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

10) What is the difference between DOM & SAX APIs?

…………………………………………………………………………………………

…………………………………………………………………………………………

…………………………………………………………………………………………

11) What is XSL Transformation (XSLT)?

…………………………………………………………………………………

…………………………………………………………………………………

…………………………………………………………………………………

2.10 XPATH

XPath is a language that permits you to go through the XML document hierarchy. The language also permits the selection of nodes in an XML document based on path and/or selection criteria. Viewing an XML document as a forest of nodes, XPath enables you to navigate and select a specified tree or leaf.

The path selection notation is similar to URL's in that you can use both absolute and relative paths. For example, using the CustomerOrder as a base example, you can navigate to an Order node using the following absolute path:

/CustomerOrder/Orders

The general XPath syntax is axis::node_test[predicate]

Axis

It defines the area from where you wish to start navigating. For instance, the absolute path example you saw previously has an axis of "/" which denotes the document. Some other possible axis values are: parent, child, ancestor, attribute, etc.

Node test

The node test is used to select one or more nodes. It can be done by tag name, using a node selector or by using the wildcard (*). The following are several node selelectors: node( ), text( ), comment( ), etc.

Predicates

Optional function or expression enclosed in "[...]". A few of the functions available are: position( ), count( ), etc.

Examples of using predicates:

/Address:: * [@AddrType="billing"]

Or

OrderNo[position=1]/ProductNo[position=3]

Fortunately, you do not always have to write Order No.
[position =1]/ProductNo[position=3] when you wish to select the third ProductNo of the first OrderNo. XPath does provide some means of abbreviating. The following is an equivalent expression.

OrderNo[1]/ProductNo[3]

There are other abbreviations also. Please refer to them in the further readings.

2.11 XLINKS

XLinks are an attribute-based syntax for attaching links to XML documents. XLinks can be a simple Point A-to-Point B links. XLinks can also be bidirectional, linking two documents in both directions so you can go from A to B or B to A. XLinks can even be multidirectional, presenting many different paths between any number of XML documents. The documents do not have to be XML documents—XLinks can be placed in an XML document that lists connections between other documents that may or may not be XML documents themselves. At its core, XLink is an XML syntax for describing directed graphs, in which the vertices are documents at particular URIs and the edges are the links between the documents.

Current web browsers at most support simple Xlinks. Many browsers, do not support XLinks at all. A simple link defines a one-way connection between two resources. The source or starting resource of the connection is the link element itself. The target or ending resource of the connection is identified by a Uniform Resource Identifier (URI). The link goes from the starting resource to the ending resource. The starting resource is always an XML element. The ending resource may be an XML document, a particular element in an XML document, a group of elements in an XML document, an MPEG movie or a PDF file which are not the part of XML document. The URI may be something other than a URL, may be a book ISBN number like urn:isbn:1283222229.

A simple XLink is encoded in an XML document as an element of an arbitrary type that has an xlink:type attribute with the simple value and an xlink:href attribute whose value is the URI of the link target. The xlink prefix must be mapped to the http://www.w3.org/1999/xlink namespace URI. As usual, the prefix can change as long as the URI is the same. For example, suppose this novel element appears in a list of children's literature and we want to link it to the actual text of the novel available from the URL ftp://HindiLibrary.org/Premchand/Karmabhumi.txt:

```
<novel>
  <title>Karmabhumi</title>
  <author>Munshi Premchand</author>
  <year>1925</year>
</novel>
```

We give the novel element an xlink:type attribute with the value simple, an xlink:href attribute that contains the URL to which we're linking, and an xmlns:xlink attribute that associates the prefix xlink with the namespace URI http://www.w3.org/1999/xlink like so:

```
<novel xmlns:xlink= "http://www.w3.org/1999/xlink"
      xlink:type = "simple"
      xlink:href = "ftp://HindiLibrary.org/Premchand/Karmabhumi.txt">
  <title>Karmabhumi</title>
  <author>Munshi Premchand</author>
  <year>1925</year>
</novel>
```

This establishes a simple link from this novel element to the plain text file found at ftp://HindiLibrary.org/Premchand/karmabhumi.txt. Browsers are free to interpret this link as they like. However, the most natural interpretation, and the one implemented by a few browsers that do support simple XLinks, is to make this a blue underlined phrase. The user can click on to replace the current page with the file being linked to.

2.12 XQUERY

The Xquery as explained by W3C is: "XML is a versatile markup language, capable of labelling the information content of diverse data sources including structured and semi-structured documents, relational databases, and object repositories. A query language that uses the structure of XML intelligently can express queries across all these kinds of data, whether physically stored in XML or viewed as XML via middleware. This specification describes a query language called XQuery, which is designed to be broadly applicable across many types of XML data sources."

XQuery is a query language something like SQL that we can use with XML documents. XQuery is designed to access data much as if we were working with a database, even though we are working with XML. The creation of such a query language is to provide a way to work with data in XML. XQuery not only gives a data model to interpret XML documents, but also a set of operators and functions to extract data from those documents.

The W3C XQuery specification is divided into several working drafts; the main XQuery 1.0 working drafts you download at http://www.w3.org/TR/xquery.
One major difference between XQuery and other query languages, such as SQL and XPath, is that XQuery will have multiple syntaxes. At the minimum it will have a syntax in XML and another more human-readable syntax. The human-readable syntax is defined in the working draft itself, while the XML syntax, called XQueryX, is defined in a separate working draft, at http://www.w3.org/TR/xqueryx.

A glance at the XQueryX working draft shows that the XML syntax for XQuery will be much more than the human-readable syntax. For example, suppose we have a document that conforms to the following DTD (from the XML Query Use Cases document):

```
<!ELEMENT bib  (book* )>
<!ELEMENT book  (title,  (author+ | editor+ ), publisher, price )>
<!ATTLIST book  year CDATA  #REQUIRED >
<!ELEMENT author  (last, first )>
<!ELEMENT editor  (last, first, affiliation )>
<!ELEMENT title  (#PCDATA )>
<!ELEMENT last  (#PCDATA )>
<!ELEMENT first  (#PCDATA )>
<!ELEMENT affiliation  (#PCDATA )>
<!ELEMENT publisher  (#PCDATA )>
<!ELEMENT price  (#PCDATA )>
```

The XQuery Working draft gives an example query to list each publisher in the XML document, and the average price of its books, which is

```
FOR $p IN distinct(document("bib.xml")//publisher)
LET $a := avg(document("bib.xml")//book[publisher = $p]/price)
RETURN
  <publisher>
    <name> {$p/text()} </name>
    <avgprice> {$a} </avgprice>
  </publisher>
```

Since XQuery is only in Working Draft stage, the final syntax could change, meaning that the query syntax above may not be proper XQuery syntax.

The XQuery as above creates a variable, named **p,** which will contain a list of all of the distinct <publisher> elements from the document bib.xml. That is, if there are multiple <publisher> elements which contain the text "IGNOU Press", the **p** variable will only contain one of them, and ignore the rest. For each of the <publisher> elements in the **p** variable, another variable, called **a** will be created, which will contain the average price of all of the books associated with this publisher. It does this by:

Getting a node-set of all of the <price> elements, which are a child of a <book> element whose <publisher> element has the same value as the current value of p. Passing this node-set to the avg() function, which will return the average value. Once the publisher’s name, and the average price of its books, have been discovered, an XML fragment similar to

```
<publisher>
  <name>Publisher's name</name>
  <avgprice>Average price</avgprice>
</publisher>
```

will be returned. This is very similar to the types of queries we can create using SQL.

Thus, the final Xquery generated for the problem above would be:

```
<q:query xmlns:q="http://www.w3.org/2001/06/xqueryx">
  <q:flwr>
    <q:forAssignment variable="$p">
      <q:function name="distinct">
        <q:step axis="SLASHSLASH">
          <q:function name="document">
            <q:constant datatype="CHARSTRING">bib.xml</q:constant>
          </q:function>
          <q:identifier>publisher</q:identifier>
        </q:step>
      </q:function>
    </q:forAssignment>
    <q:letAssignment variable="$a">
      <q:function name="avg">
        <q:step axis="CHILD">
          <q:function name="document">
            <q:constant datatype="CHARSTRING">bib.xml</q:constant>
          </q:function>
          <q:step axis="CHILD">
            <q:predicatedExpr>
              <q:identifier>book</q:identifier>
              <q:predicate>
                <q:function name="EQUALS">
                  <q:identifier>publisher</q:identifier>
                  <q:variable>$p</q:variable>
                </q:function>
              </q:predicate>
            </q:predicatedExpr>
            <q:identifier>price</q:identifier>
          </q:step>
        </q:step>
      </q:function>
    </q:letAssignment>
    <q:return>
      <q:elementConstructor>
        <q:tagName>
          <q:identifier>publisher</q:identifier>
        </q:tagName>
        <q:elementConstructor>
          <q:tagName>
            <q:identifier>name</q:identifier>
          </q:tagName>
          <q:step axis="CHILD">
            <q:variable>$p</q:variable>
            <q:nodeKindTest kind="TEXT" />
```

```
     </q:step>
    </q:elementConstructor>
    <q:elementConstructor>
     <q:tagName>
      <q:identifier>avgprice</q:identifier>
     </q:tagName>
     <q:variable>$a</q:variable>
    </q:elementConstructor>
   </q:elementConstructor>
  </q:return>
 </q:flwr>
</q:query>
```

That is a long text for a query. But remembers the XML syntax is primarily not meant to be read by humans. XQueryX is meant to be generated and processed by, software, thereby, making the query explicit. In the XML syntax will make it easier to be processed by these tools.

2.13 XML AND DATABASES

With both XML and databases being data-centric technologies, are they in competition with each other? XML is best used to communicate data, whereas database is best used to store and retrieve data. Thus, both of these are complementary, rather than competitive. XML will never replace the database, however, the two will become more closely integrated with time.

For this reason, database vendors have realised the power and flexibility of XML, and are building support for XML right into their products. This potentially makes the programmer's job easier while writing Data Objects, as there is one less step to perform. Instead of retrieving data from the database, and transforming it to XML, programmers now can retrieve data from the database in XML format.

Let us look at two database vendors who were quick to market XML integration technologies: Oracle, and Microsoft. Both companies offer a suite of tools that can be used for XML development, when communicating with a database and otherwise.

2.13.1 Microsoft's XML Technologies

Microsoft has provided an extensive documentation on XML at MSDN (Microsoft Developer Network), the online site devoted to developers working with Microsoft technologies is http://msdn.microsoft.com/xml/.

MSXML

The first, and most obvious, form of XML support from Microsoft is that Internet Explorer comes bundled with the MSXML – a COM-based parser. MSXML 3 (that is being shipped with IE 6) provides validating and non-validating modes, as well as support for XML namespaces, SAX 2, and XSLT. MSXML 4 also includes support for the XML Schemas Recommendation.

.NET

.NET is Microsoft's development model in which software becomes a platform and is device-independent, and data becomes available over the Internet. The .NET Framework is the infrastructure of .NET and XML is the backbone to it. XML is a common communication language for .NET components and servers.

SQL Server

The XML support has also been built into SQL Server. SQL Server provides the ability to perform a SQL query through an HTTP request, via an ISAPI filter for

Internet Information Server. So programmers might perform a query like the following (replace servername with the name of your web server, and databasename with the name of the database you are querying):

```
http://servername/databasename?sql=SELECT+last_name+FROM+Customer+FOR+
XML+RAW
```

If we do not want to be passing our complex SQL statements in a URL, we can also create an XML template file to store the query. It would look something like this:

```
<root>
  <sql:query xmlns:sql="urn:schemas-microsoft-com:xml-sql">
    SELECT last_name FROM Customer FOR XML RAW
  </sql:query>
</root>
```

Notice the words FOR XML RAW added to the end of that SQL statement. This is a language enhancement Microsoft has added to allow SQL queries to natively return XML.

If this template is named lastname.xml, we need to execute the SQL by using the following URL:

```
http://servername/databasename/lastname.xml
```

And this query would return XML similar to the following:

```
<?xml version="1.0" encoding="UTF-8" ?>
<root>
  <row last_name="Gupta" />
  <row last_name="Bhushan" />
  <row last_name="Kumar" />
</root>
```

2.13.2 Oracle's XML Technologies

Like Microsoft, Oracle provides a number of tools to work with XML, in its XML Developer's Kit (XDK). Even though Microsoft had some XML support built right into the database, it took them much longer to get their database-integration tools to market, whereas Oracle had these tools in the market very quickly indeed. Oracle's Technology Network has a good web site devoted to the XML Developer's Kit, which is located at http://technet.oracle.com/tech/xml/.

XML Parsers

The first tool available from Oracle is the XML parser. Oracle provides parsers written in Java, C, C++, and PL/SQL. These parsers provide:

- A DOM interface
- A SAX interface
- Both validating and non-validating support
- Support for namespaces
- Fully compliant support for XSLT.

Like MSXML, the Oracle XML parsers provide extension functions to the DOM, selectSingleNode() and selectNodes(), which function just like the Microsoft methods.

Code Generators

Oracle offers Java and C++ class generating applications, just like the Visual Basic class building application. However, these generators work from DTDs, not schemas, meaning that they are already fully conformant to the W3C specifications. Also, since these tools are part of the XDK, they are fully supported by Oracle, instead of just being sample codes.

XML SQL Utility for Java

Along with these basic XML tools, the XDK also provides the XML SQL Utility for Java. This tool can generate an XML document from a SQL query, either in text form or in a DOM. The XML results may differ slightly from the Microsoft SQL Server FOR XML clause, but they are just as easy to use. For example, the following SQL query:

```
SELECT last_name FROM customer
```

might return XML like the following:

```
<?xml version="1.0"?>
<ROWSET>
  <ROW id="1">
    <last_name>Gupta</last_name>
  </ROW>
  <ROW id="2">
    <last_name>Bhushan</last_name>
  </ROW>
  <ROW id="3">
    <last_name>Kumar</last_name>
  </ROW>
</ROWSET>
```

So, instead of including the information in attributes of the various <row> elements, Oracle decided to include the information in separate child elements. And, just like Microsoft’s enhancements to SQL Server 2000, Oracle’s XML SQL Utility for Java can take in XML documents, and use the information to update the database.

XSQL Servlet

Microsoft decided to make SQL queries available over HTTP by providing an ISAPI filter for IIS. Oracle, on the other hand, decided to create the Java XSQL Servlet instead. Since it is a Java Servlet, it will run on any web server that supports servlets– which is most of them. This servlet takes in an XML document that contains SQL Queries, like the XML templates used by SQL Server. The servlet can optionally perform an XSLT transformation on the results (if a stylesheet is specified), so the results can potentially be any type of file that can be returned from an XSLT transformation, including XML and HTML. (To accomplish this, the XSQL Servlet makes use of the XML parser mentioned above). Because it is a servlet, it will run on any web server that has a Java Virtual Machine and can host servlets.

2.13.3 XML Databases

So far, we have focused on integrating XML with traditional databases. The general idea has been to store the data in the database, and transform it to and from XML when needed.

However, there is a fundamental difference between relational databases and XML documents – relational databases are based on relations, tuples and attributes whereas XML documents are hierarchical in nature, using the notions of parents, children, and descendants. This means that there may be cases where the structure of an XML document will not fit into the relational model.

For such cases, you might want to consider a specialised XML database, instead of transforming back- and-forth from XML, like SQL Server and Oracle. An XML database stores data natively in XML. XML format can offer the following benefits:

- The speed of your application have to be increased as all of the components in your application need to deal with the data in XML format. The XML database will eliminate the need to transform the data back-and-forth – it starts in XML, and stays in XML.

- Many of the XML databases coming out provide functionality to query the database using XPath queries, just as relational databases today use SQL. Since XML is hierarchical in nature, XPath can provide a more natural query language than SQL, which is a more focused relational database.

There are already a number of such XML databases available. The features and focus of each product can vary greatly, meaning that not all of these products are in direct competition with each other. Some of the XML databases that you may encounter include:

- Extensible Information Server, from eXcelon (http://www.exceloncorp.com)
- GoXML, from XML Global (http://www.xmlglobal.com/prod/db/)
- dbXML, an open source product from dbXML Group (http://www.dbxml.com)
- Tamino XML Database, from Software AG (http://www.softwareag.com/tamino/)

2.14 STORAGE OF XML DATA

When it comes to storing XML there are a number of options available to us. We can store it in:

- plain flat files (such as plain text files held on the operating system’s native file system),
- a relational database (such as Oracle, SQL Server or DB2),
- an object database (which stores DOM representations of the document), and
- a directory service (such as Novell NDS or Microsoft Active Directory).

The first two options are by far the most common. If we use flat files they tend to be accessed from the file system of the operating system we are running. It is just like holding a collection of text documents or word processing documents on our file server. The important thing to do if, we take this approach is to make sure that we have a good naming convention for our files so that they are easily accessed.

Relational databases, however, are still by far the most common data storage format. There are many ways to store XML data in relational databases. One, XML data can be stored as strings in a relational database. Second, relations can represent XML data as tree. Third, XML data can be mapped to relations in the same way that E-R schemes are mapped to relational schemas. Increasingly, relational database vendors are adding more sophisticated XML support to their RDBMS so that users can store data in relational tables, but simply send the database XML for it to insert, update or delete the information it holds; they can also request that contents of the database be retrieved and output in XML format. It is no longer necessary with many databases to convert the XML into another form before the database can accept it – if we are able to just send the database an XML file this will save us doing the conversion ourself.

2.15 XML DATABASE APPLICATIONS

XML is platform and language independent, which means it does not matter that one computer may be using, for example, Visual Basic on a Microsoft operating system, and the other is a UNIX/LINUX machine running Java code. Any time one computer program needs to communicate with another program, XML is a potential fit for the exchange format. The following are just a few examples.

Reducing Server Load

Web-based applications can use XML to reduce the load on the web servers. This can be done by keeping all information on the client for as long as possible, and then sending the information to those servers in one big XML document.

Web Site Content

The W3C uses XML to write its specifications. These XML documents can then be transformed to HTML for display (by XSLT), or transformed to a number of other presentation formats. Some web sites also use XML entirely for their content, where traditionally HTML would have been used. This XML can then be transformed to HTML via XSLT, or displayed directly in browsers via CSS. In fact, the web servers can even determine dynamically the kind of browser that is retrieving the information, and then decide what to do. For example, transform the XML to HTML for older browsers, and just send the XML straight to the client for newer browsers, reducing the load on the server.

In fact, this could be generalised to any content, not just web site content. If your data is in XML, you can use it for any purpose, with presentation on the Web being just one possibility.

Remote Procedure Calls

XML is also used as a protocol for Remote Procedure Calls (RPC). RPC is a protocol that allows objects on one computer to call objects on another computer to do work, allowing distributed computing. Using XML and HTTP for these RPC calls, using a technology called the Simple Object Access Protocol (SOAP), allows this to occur even through a firewall, which would normally block such calls, providing greater opportunities for distributed computing.

Web-Enabling Business Using XML

XML can be a very effective way to represent business data for transfer between different systems and organisations. With the increasing importance of the Internet and related technologies, XML can be a key factor in helping organisations move their businesses to the Web.

Using XML in Business to Consumer (B2C) Solutions: When most people think of using the Internet for business, they think of online retailers selling goods and services through a Web site. Retail e-commerce solutions, such as these usually rely on HTML pages accessed by means of a Web browser. The pages show product data from a database and allow customers to place orders, the details of which are also stored in a database. XML can be an extremely effective way to pass data from the database to the Web application. XSL can then be used to easily transform the XML data into HTML for display in the browser. This approach is usually more efficient than retrieving data as a rowset and writing presentation logic in a Web page script or component to render the data. In addition, as more devices are used to connect to the Internet, the same XML data can be transformed using different style sheets to suit different client devices. For example, an XML product catalogue could be transformed into HTML for display in a browser or into Wireless Markup Language (WML) for display on a Wireless Application Protocol (WAP) – enabled cell phone.

This flexibility makes XML a great choice for developing Web-based applications that will be accessed by multiple client types.

Using XML in Business to Enterprise (B2E) Solutions: Of course, Internet technologies such as HTTP are often used to build internal applications. This is particularly helpful in environments where multiple platforms and development languages are used because an Intranet-based solution allows any application that can communicate over TCP/IP to be integrated. For applications that allow users to access and manipulate data, XML-aware browsers such as Microsoft Internet Explorer can be used to download and render the data. Users can then manipulate the data in the browser using XML data islands before sending the updated data back to the server for storage in the database. Existing applications running on platforms such as mainframes or UNIX can use XML as a neutral way to describe data. For example, a mail-order company with a large existing Information Management System (IMS) application running on a mainframe could decide to build a Web-based e-commerce program in this case, the existing telephone sales orders can continue to be entered into the IMS application as before, and new orders placed through the Web site can be represented as XML and passed on to be stored in the IMS application.

Using XML in Business to Business (B2B) Solutions: One of the most important aspects of Web development is the integration of business processes across trading partners. Most interbusiness processes involve an exchange of business documents, such as orders, invoices, delivery notes, and so on. XML provides an ideal way to describe these business documents for exchange across the Internet. XML schemas can be used to define the XML representation of the business documents, allowing trading partners to agree on a format for the data being exchanged. Of course, each organisation can represent data differently internally, and, use XSL to transform data for exchange. Because XML is a text-based language, business documents can be exchanged by using any protocol, such as HTTP, SMTP, or FTP, or by using a message queuing solution. This flexibility makes it possible for any business to integrate with its trading partners over the Internet.

☞ Check Your Progress 3

1) What is the difference between XPath & XLink?

…………………………………………………………………………………

…………………………………………………………………………………

…………………………………………………………………………………

2) What is XQuery?

…………………………………………………………………………………

…………………………………………………………………………………

…………………………………………………………………………………

3) How can you store an XML document in any relational database?

…………………………………………………………………………………

…………………………………………………………………………………

…………………………………………………………………………………

2.16 SUMMARY

XML (eXtensible Markup Language) is a meta language (a language for describing other languages) that enables designers to create their own customised tags to provide functionality not available within HTML. XML is a restricted form of SGML, designed as a less complex markup language than SGML i.e., which at the same time is network aware.

AN XML document consists of elements, attributes, entity, references, comments and processing instructions. An XML document can optionally have a Document Type Definition (DTD) which defines the valid syntax of an XML document.

An XML schema is the definition of a specific XML structure. XML schema uses the W3C XML Schema language to specify how each type of element in the schema is defined and what data type that element has associated with it. The schema is itself an XML document, so it can be read by the same tools that read the XML it describes.

XML APIs generally fall into two categories: tree based and event based. DOM (Document Object Model) is a tree based API for XML that provides an object oriented view of the data. The API was created by W3C and describes a set of platform and language neutral interfaces that can represent any well formed XML or HTML document. SAX (Simple API for XML) is an event based, serial access API for XML that uses callbacks to report parsing events to the applications. The application handles these events through customised event handlers.

The World Wide Web Consortium (W3C) has recently formed an XML Query Working group to produce a data model for XML documents, a set of query operators on this model, and a query language based on these query operators.

Today, XML has become a *de facto* communication language between cross vendor products, components written in different technologies and running on different operating systems; and cross databases. The key of today's distributed architecture is XML based web service, which is supported by all majors' players of the software industry such as like Microsoft, IBM, Oracle, Sun (i.e., Java).

2.17 SOLUTIONS/ANSWERS

Check Your Progress 1

1) Semi-structured data is data that has some structure, but the structure may not be rigid, regular or complete and generally the data does not conform to a fixed schema. Sometimes the term schema-less or self-describing is used to describe such data.

2) Most documents on Web are currently stored and transmitted in HTML. One strength of HTML is its simplicity. However, it may be one of its weaknesses with the growing needs of users who want HTML documents to be more attractive and dynamic. XML is a restricted version of SGML, designed especially for Web documents. SGML defines the structure of the document (DTD), and text separately. By giving documents a separately defined structure, and by giving web page designers ability to define custom structures, SGML has and provides extremely powerful document management system but has not been widely accepted as it is very complex. XML attempts to provide a similar function to SGML, but is less complex. XML retains the key SGML advantages of extensibility, structure, and validation.
XML cannot replace HTML.

3) XML is designed to work with applications that might not be case sensitive and in which the case folding (the conversion to just one case) cannot be predicted. Thus, to avoid making unsafe assumptions, XML takes the safest route and opts for case sensitivity.

4) Yes, for two reasons. The first is that you normally model your data better with XML than with HTML. You can capture the hierarchical structure giving meaning to your information. The second reason is that XML files are well formed. You have a much better idea what comes in the data stream. HTML files, on the other hand, can take many forms and appearances.

5) In HTML, the tags tell the browser what to do to the text between them. When a tag is encountered, the browser interprets it and displays the text in the proper form. If the browser does not understand a tag, it ignores it. In XML, the interpretation of tags is not the responsibility of the browser. It is the programmer who defines the tags through DTD or Schema.

6)
1. Simplicity
2. Open standard and platform/vendor-independent
3. Extensibility
4. Reuse
5. Separation of content and presentation
6. Improved load balancing
7. Support for integration of data from multiple sources
8. Ability to describe data from a wide variety of applications.

7) An XML element is the basic data-holding construct in an XML document. It starts with an opening tag, can contain text or other elements, and ends with a closing tag, like this: <greeting>hello</greeting>. An attribute gives you more information, and is always assigned a value in XML. Here's how you might add an attribute named language to this element:

<greeting language = "en">hello</greeting>.

8) The attributes you can use in an XML document are: version (required; the XML version), encoding (optional; the character encoding), and standalone (optional; "yes" if the document does not refer to any external documents or entities, "no" otherwise).

Check Your Progress 2

1)
1. An XML document must contain one or more elements.
2. One element, the root element, must contain all the other elements.
3. Each element must nest inside any enclosing elements correctly.

2) Document Type Definitions (DTDs) and XML schemas.

3) XML document that conforms to structural and notational rules of XML is considered well-formed. XML Validating processor checks that an XML document is well-formed and conforms to a DTD, in which case the XML document is considered valid. A well formed XML document may not be a valid document.

4) In a well-formed XML document, there must be one root element that contains all the others.

5) The <hiredate> and <name> elements are not declared in the DTD.

6) The <hiredate> and <name> elements appear in the wrong order.

7)
1. It is written in a different (non-XML) syntax;
2. It has no support for namespaces;

3.It only offers extremely limited data typing.

8) The namespace that is used by XML schemas is www.w3.org/2001/XMLSchema.

9) You can declare the element like this:

   <xsd:element name="name" type="xsd:string"/>

10) Document Object Model (DOM) & Simple API for XML (SAX) are two popular models to interpret an XML document programmatically. DOM (Document Object Model) is a tree-based API that provides object-oriented view of data. It describes a set of platform and language-neutral interfaces that can represent any well-formed XML/HTML document. While SAX is an event-based, serial-access API for XML that uses callbacks to report parsing events to the application. Unlike tree-based APIs, event-based APIs do not build an in-memory tree representation of the XML document.

11) XML Stylesheet Langauge (XSL) is created specifically to define how an XML document's data is rendered and to define how one XML document can be transformed into another document. XSLT, a subset of XSL, is a language in both the markup and programming sense, providing a mechanism to transform XML structure into either another XML structure, HTML, or any number of other text-based formats (such as SQL). XSLT's main ability is to change the underlying structures rather than simply the media representations of those structures, as with Cascading Style Sheet (CSS).

Check Your Progress 3

1) XPath is a declarative query language for XML that provides a simple syntax for addressing parts of an XML document. It is designed for use with XSLT (for pattern matching). With XPath, collections of elements can be retrieved by specifying a directory-like path, with zero or more conditions placed on the path. While XLink allows elements to be inserted into XML documents to create and describe links between resources. It uses XML syntax to create structures that can describe links similar to simple unidirectional hyperlinks of HTML as well as more sophisticated links.

2) Data extraction, transformation, and integration are well-understood database issues that rely on a query language. SQL does not apply directly to XML because of the irregularity of XML data. W3C recently formed an XML Query Working Group to produce a data model for XML documents, set of query operators on this model, and query language based on query operators. Queries operate on single documents or fixed collections of documents, and can select entire documents or subtrees of documents that match conditions based on document content/structure. Queries can also construct new documents based on what has been selected.

3) 1. XML data can be stored as strings in a relational database.
   2. Relations can represent XML data as tree.
   3. XML data can be mapped to relations in the same way that E-R schemes are mapped to relational schemas.