UNIT IV XML DATABASES

Structured, Semi structured, and Unstructured Data – XML Hierarchical Data Model – XML Documents – Document Type Definition – XML Schema – XML Documents and Databases – XML Querying – XPath – XQuery

Difference between Structured, Semi-structured and Unstructured data

Big Data includes huge volume, high velocity, and extensible variety of data. These are 3 types: Structured data, Semi-structured data, and Unstructured data.

Structured data –

Structured data is data whose elements are addressable for effective analysis. It has been organized into a formatted repository that is typically a database. It concerns all data which can be stored in database SQL in a table with rows and columns. They have relational keys and can easily be mapped into pre-designed fields. Today, those data are most processed in the development and simplest way to manage information. *Example:* Relational data.

Semi-Structured data –

Semi-structured data is information that does not reside in a relational database but that has some organizational properties that make it easier to analyze. With some processes, you can store them in the relation database (it could be very hard for some kind of semi-structured data), but Semi-structured exist to ease space. *Example*: XML data.

Unstructured data –

Unstructured data is a data which is not organized in a predefined manner or does not have a predefined data model, thus it is not a good fit for a mainstream relational database. So for Unstructured data, there are alternative platforms for storing and managing, it is increasingly prevalent in IT systems and is used by organizations in a variety of business intelligence and analytics applications. *Example*: Word, PDF, Text, Media logs.

Differences between Structured, Semi-structured and Unstructured data:

Properties	Structured data	Semi-structured data	Unstructured data
Technology	It is based on Relational database table	It is based on XML/RDF(Resource Description Framework).	It is based on character and binary data
Transaction management	Matured transaction and various concurrency techniques	Transaction is adapted from DBMS not matured	No transaction management and no concurrency
Version management	Versioning over tuples,row,tables	Versioning over tuples or graph is possible	Versioned as a whole
Flexibility	It is schema dependent and less flexible	It is more flexible than structured data but less flexible than unstructured data	It is more flexible and there is absence of schema
Scalability	It is very difficult to scale DB schema	It's scaling is simpler than structured data	It is more scalable.
Robustness	Very robust	New technology, not very spread	—
Query performance	Structured query allow complex joining	Queries over anonymous nodes are possible	Only textual queries are possible

XML Hierarchical Data Model

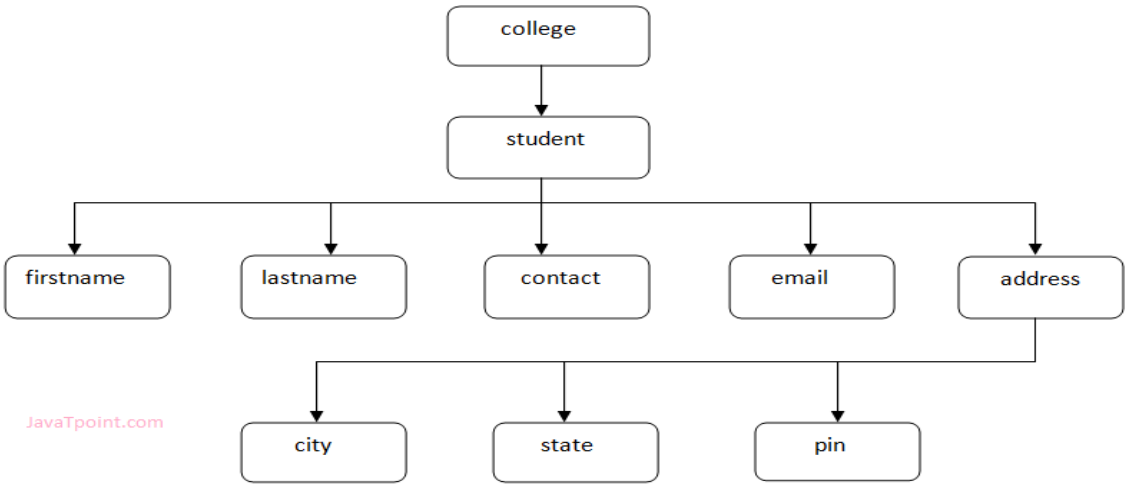
XML Tree Structure

An XML document has a self descriptive structure. It forms a tree structure which is referred as an XML tree. The tree structure makes easy to describe an XML document.

A tree structure contains root element (as parent), child element and so on. It is very easy to traverse all succeeding branches and sub-branches and leaf nodes starting from the root.

```
<?xml version="1.0"?>
<college>
  <student>
     <firstname>Tamanna</firstname>
     <lastname>Bhatia</lastname>
     <contact>09990449935</contact>
     <email>tammanabhatia@abc.com</email>
     <address>
        <city>Ghaziabad</city>
        <state>Uttar Pradesh</state>
        <pin>201007</pin>
     </address>
  </student>
</college>
```

lets see the tree-structure representation of the above example.



In the above example, first line is the XML declaration. It defines the XML version 1.0. Next line shows the root element (college) of the document. Inside that there is one more element (student). Student element contains five branches named <firstname>, <lastname>, <contact>, <Email> and <address>.<address> branch contains 3 sub-branches named <city>, <state> and <pin>.

XML Tree Rules

These rules are used to figure out the relationship of the elements. It shows if an element is a child or a parent of the other element.

Descendants: If element A is contained by element B, then A is known as descendant of B. In the above example "College" is the root element and all the other elements are the descendants of "College".

Ancestors: The containing element which contains other elements is called "Ancestor" of other element. In the above example Root element (College) is ancestor of all other elements.

What is xml?

Xml (eXtensible Markup Language) is a mark up language.

XML is designed to store and transport data.

Xml was released in late 90's. it was created to provide an easy to use and store self describing data.

XML became a W3C Recommendation on February 10, 1998.

XML is not a replacement for HTML.

XML is designed to be self-descriptive.

XML is designed to carry data, not to display data.

XML tags are not predefined. You must define your own tags.

XML is platform independent and language independent.

XML Example

```
<bookstore>
  <book category="COOKING">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
```

```
    <price>30.00</price>
  </book>
  <book category="CHILDREN">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="WEB">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

The root element in the example is <bookstore>. All elements in the document are contained within <bookstore>.The <book> element has 4 children: <title>,< author>, <year> and <price>.

```
<?xml version="1.0" encoding="UTF-8"?>
<emails>
<email>
  <to>Vimal</to>
  <from>Sonoo</from>
  <heading>Hello</heading>
  <body>Hello brother, how are you!</body>
</email>
<email>
  <to>Peter</to>
  <from>Jack</from>
  <heading>Birth day wish</heading>
  <body>Happy birth day Tom!</body>
</email>
<email>
  <to>James</to>
```

<from>Jaclin</from>
<heading>Morning walk</heading>
<body>Please start morning walk to stay fit!</body>
</email>
<email>
<to>Kartik</to>
<from>Kumar</from>
<heading>Health Tips</heading>
<body>Smoking is injurious to health!</body>
</email>
</emails>

XML Attributes

XML elements can have attributes. By the use of attributes we can add the information about the element.

<book publisher='Tata McGraw Hill'>**</book>**

Metadata should be stored as attribute and data should be stored as elements

<book>
<book category="computer">
<author> A & B **</author>**
</book>

XML Comments

XML comments are just like HTML comments. We know that the comments are used to make codes more understandable other developers.

An XML comment should be written as:

<!-- Write your comment-->

XML Validation

A well formed XML document can be validated against DTD or Schema.

A well-formed XML document is an XML document with correct syntax. It is very necessary to know about valid XML document before knowing XML validation.

Valid XML document

It must be well formed (satisfy all the basic syntax condition)

It should be behave according to predefined DTD or XML schema

Rules for well formed XML

It must begin with the XML declaration.

It must have one unique root element.

All start tags of XML documents must match end tags.

XML tags are case sensitive.

All elements must be closed.

All elements must be properly nested.

All attributes values must be quoted.

XML entities must be used for special characters.

XML Validation

XML DTD:

DTD stands for Document Type Definition. It defines the legal building blocks of an XML document. It is used to define document structure with a list of legal elements and attributes.

Purpose of DTD:

elements and define the strucIts main purpose is to define the structure of an XML document. It contains a list of legal ture with the help of them.

Example:

```
<?xml version="1.0"?>

<!DOCTYPE employee SYSTEM "employee.dtd">

<employee>

  <firstname>vimal</firstname>

  <lastname>jaiswal</lastname>
```

<email>vimal@javatpoint.com</email>

</employee>

Description of DTD:

<!DOCTYPE employee : It defines that the root element of the document is employee.

<!ELEMENT employee: It defines that the employee element contains 3 elements "firstname, lastname and email".

<!ELEMENT firstname: It defines that the firstname element is #PCDATA typed. (parse-able data type).

<!ELEMENT lastname: It defines that the lastname element is #PCDATA typed. (parse-able data type).

<!ELEMENT email: It defines that the email element is #PCDATA typed. (parse-able data type).

XML DTD

A DTD defines the legal elements of an XML document

In simple words we can say that a DTD defines the document structure with a list of legal elements and attributes.

XML schema is a XML based alternative to DTD.

Actually DTD and XML schema both are used to form a well formed XML document.

We should avoid errors in XML documents because they will stop the XML programs.

XML schema

It is defined as an XML language

Uses namespaces to allow for reuses of existing definitions

It supports a large number of built in data types and definition of derived data types

Valid and well-formed XML document with External DTD

Let's take an example of well-formed and valid XML document. It follows all the rules of DTD.

employee.xml

<?xml version="1.0"**?>**

```
<!DOCTYPE employee SYSTEM "employee.dtd">
<employee>
  <firstname>vimal</firstname>
  <lastname>jaiswal</lastname>
  <email>vimal@javatpoint.com</email>
</employee>
```

In the above example, the DOCTYPE declaration refers to an external DTD file. The content of the file is shown in below paragraph.

employee.dtd

```
<!ELEMENT employee (firstname,lastname,email)>

<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT email (#PCDATA)>
```

Valid and well-formed XML document with Internal DTD

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "yes" ?>

<!DOCTYPE address [

    <!ELEMENT address (name,company,phone)>

    <!ELEMENT name (#PCDATA)>

    <!ELEMENT company (#PCDATA)>

    <!ELEMENT phone (#PCDATA)>

]>

<address>

    <name>Tanmay Patil</name>

    <company>TutorialsPoint</company>

    <phone>(011) 123-4567</phone>

</address>
```

Description of DTD

<!DOCTYPE employee : It defines that the root element of the document is employee.

<!ELEMENT employee: It defines that the employee element contains 3 elements "firstname, lastname and email".

<!ELEMENT firstname: It defines that the firstname element is #PCDATA typed. (parse-able data type).

<!ELEMENT lastname: It defines that the lastname element is #PCDATA typed. (parse-able data type).

<!ELEMENT email: It defines that the email element is #PCDATA typed. (parse-able data type).

XML CSS

Purpose of CSS in XML

CSS (Cascading Style Sheets) can be used to add style and display information to an XML document. It can format the whole XML document.

How to link XML file with CSS

To link XML files with CSS, you should use the following syntax:

<?xml-stylesheet type="text/css" href="cssemployee.css"**?>**

XML CSS Example

cssemployee.css

```
employee
{
background-color: pink;
}
firstname,lastname,email
{
font-size:25px;
display:block;
color: blue;
margin-left: 50px;
}
```

employee.dtd

```
<!ELEMENT employee (firstname,lastname,email)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT email (#PCDATA)>
```

employee.xml

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="cssemployee.css"?>
```

<!DOCTYPE employee SYSTEM "employee.dtd">

<employee>

<firstname>vimal**</firstname>**

<lastname>jaiswal**</lastname>**

<email>vimal@javatpoint.com**</email>**

</employee>

CDATA vs PCDATA

CDATA

CDATA: (Unparsed Character data): CDATA contains the text which is not parsed further in an XML document. Tags inside the CDATA text are not treated as markup and entities will not be expanded.

Let's take an example for CDATA:

```
<?xml version="1.0"?>

<!DOCTYPE employee SYSTEM "employee.dtd">

<employee>

<![CDATA[

   <firstname>vimal</firstname>

   <lastname>jaiswal</lastname>

   <email>vimal@javatpoint.com</email>

]]>

</employee>
```

In the above CDATA example, CDATA is used just after the element employee to make the data/text unparsed, so it will give the value of employee:

```
<firstname>vimal</firstname><lastname>jaiswal</lastname><email>vimal@javatpoint.com</e
mail>
```

PCDATA

PCDATA: (Parsed Character Data): XML parsers are used to parse all the text in an XML document. PCDATA stands for Parsed Character data. PCDATA is the text that will be parsed by a parser. Tags inside the PCDATA will be treated as markup and entities will be expanded.

In other words you can say that a parsed character data means the XML parser examine the data and ensure that it doesn't content entity if it contains that will be replaced.

Let's take an example:

```
<?xml version="1.0"?>

<!DOCTYPE employee SYSTEM "employee.dtd">

<employee>

  <firstname>vimal</firstname>

  <lastname>jaiswal</lastname>

  <email>vimal@javatpoint.com</email>

</employee>
```

In the above example, the employee element contains 3 more elements 'firstname', 'lastname', and 'email', so it parses further to get the data/text of firstname, lastname and email to give the value of employee as:

vimaljaiswal vimal@javatpoint.com

XML Schema:

XML schema is a language which is used for expressing constraint about XML documents. There are so many schema languages which are used now a days for example Relax- NG and XSD (XML schema definition).

An XML schema is used to define the structure of an XML document. It is like DTD but provides more control on XML structure**.**

Example:

```
<?xml version="1.0"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"

targetNamespace="http://www.javatpoint.com"

xmlns="http://www.javatpoint.com"

elementFormDefault="qualified">

  <xs:element name="employee">
```

```
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
      <xs:element name="email" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
  </xs:schema>
```

Description of XML Schema:

<xs:element name="employee"> : It defines the element name employee.

<xs:complexType> : It defines that the element 'employee' is complex type.

<xs:sequence> : It defines that the complex type is a sequence of elements.

<xs:element name="firstname" type="xs:string"/> : It defines that the element 'firstname' is of string/text type.

<xs:element name="lastname" type="xs:string"/> : It defines that the element 'lastname' is of string/text type.

<xs:element name="email" type="xs:string"/> : It defines that the element 'email' is of string/text type.

XML Schema Data types:

There are two types of data types in XML schema.

1.SimpleType 2.ComplexType

SimpleType

The simpleType allows you to have text-based elements. It contains less attributes, child elements, and cannot be left empty**.**

ComplexType

The complexType allows you to hold multiple attributes and elements. It can contain additional

sub elements and can be left empty.

XML Database:

XML database is a data persistence software system used for storing the huge amount of information in XML format. It provides a secure place to store XML documents.

You can query your stored data by using XQuery, export and serialize into desired format. XML databases are usually associated with document-oriented databases.

DTD vs XSD

There are many differences between DTD (Document Type Definition) and XSD (XML Schema Definition). In short, DTD provides less control on XML structure whereas XSD (XML schema) provides more control.

No.	DTD	XSD
1)	DTD stands for **Document Type Definition**.	XSD stands for XML Schema Definition.
2)	DTDs are derived from **SGML** syntax.	XSDs are written in XML.
3)	DTD **doesn't support datatypes**.	XSD **supports datatypes** for elements and attributes.
4)	DTD **doesn't support namespace**.	XSD **supports namespace**.
5)	DTD **doesn't define order** for child elements.	XSD **defines order** for child elements.
6)	DTD is **not extensible**.	XSD is **extensible**.
7)	DTD is **not simple to learn**.	XSD is **simple to learn** because you don't need to learn new language.

8)	DTD provides **less control** on XML structure.	XSD provides **more control** on XML structure.

XML Database:

XML database is a data persistence software system used for storing the huge amount of information in XML format.

It provides a secure place to store XML documents.

Types of XML databases:

There are two types of XML databases.

1. XML-enabled database

2. Native XML database (NXD)

XML-enable Database:

XML-enable database works just like a relational database. It is like an extension provided for the conversion of XML documents. In this database, data is stored in table, in the form of rows and columns.

Native XML Database:

Native XML database is used to store large amount of data. Instead of table format, Native XML database is based on container format. You can query data by XPath expressions. Native XML database is preferred over XML-enable database because it is highly capable to store, maintain and query XML documents.

Example:

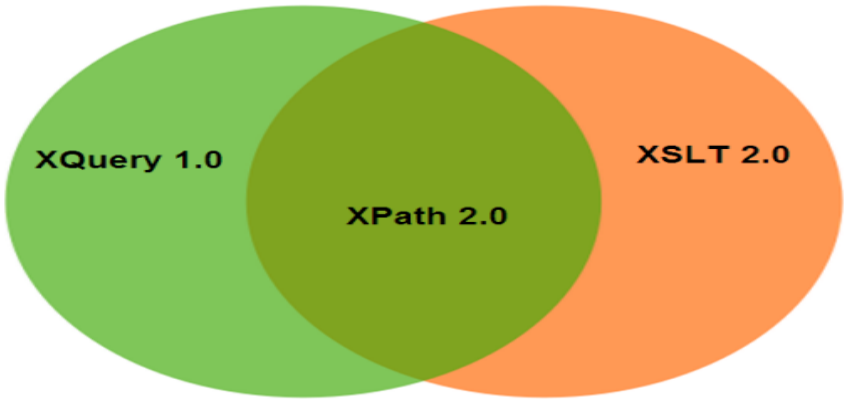
```
<?xml version="1.0"?>

<contact-info>

    <contact1>

        <name>Vimal Jaiswal</name>

        <company>SSSIT.org</company>

        <phone>(0120) 4256464</phone>

    </contact1>
```

```
    <contact2>
        <name>Mahesh Sharma </name>
        <company>SSSIT.org</company>
        <phone>09990449935</phone>
    </contact2>
</contact-info>
```

XPath:

XPath is an important and core component of XSLT standard. It is used to traverse the elements and attributes in an XML document.

XPath is a W3C recommendation. XPath provides different types of expressions to retrieve relevant information from the XML document. It is syntax for defining parts of an XML document.



Important features of XPath:

XPath defines structure: XPath is used to define the parts of an XML document i.e. element, attributes, text, namespace, processing-instruction, comment, and document nodes.

XPath provides path expression: XPath provides powerful path expressions, select nodes, or list of nodes in XML documents.

XPath is a core component of XSLT: XPath is a major element in XSLT standard and must be followed to work with XSLT documents.

XPath is a standard function: XPath provides a rich library of standard functions to manipulate string values, numeric values, date and time comparison, node and QName manipulation,

sequence manipulation, Boolean values etc.

Path is W3C recommendation.

XPath Expression

XPath defines a pattern or path expression to select nodes or node sets in an XML document. These patterns are used by XSLT to perform transformations. The path expressions look like very similar to the general expressions we used in traditional file system.

XPath specifies seven types of nodes that can be output of the execution of the XPath expression.

- Root
- Element
- Text
- Attribute
- Comment
- Processing Instruction
- Namespace

We know that XPath uses a path expression to select node or a list of nodes from an XML document.

A list of useful paths and expression to select any node/ list of nodes from an XML document:

XPath Expression Example

Let's take an example to see the usage of XPath expression. Here, we use an xml file "employee.xml" and a stylesheet for that xml file named "employee.xsl". The XSL file uses the XPath expressions under **select** attribute of various XSL tags to fetchvalues of id, firstname, lastname, nickname andsalary of each employee node.

Employee.xml

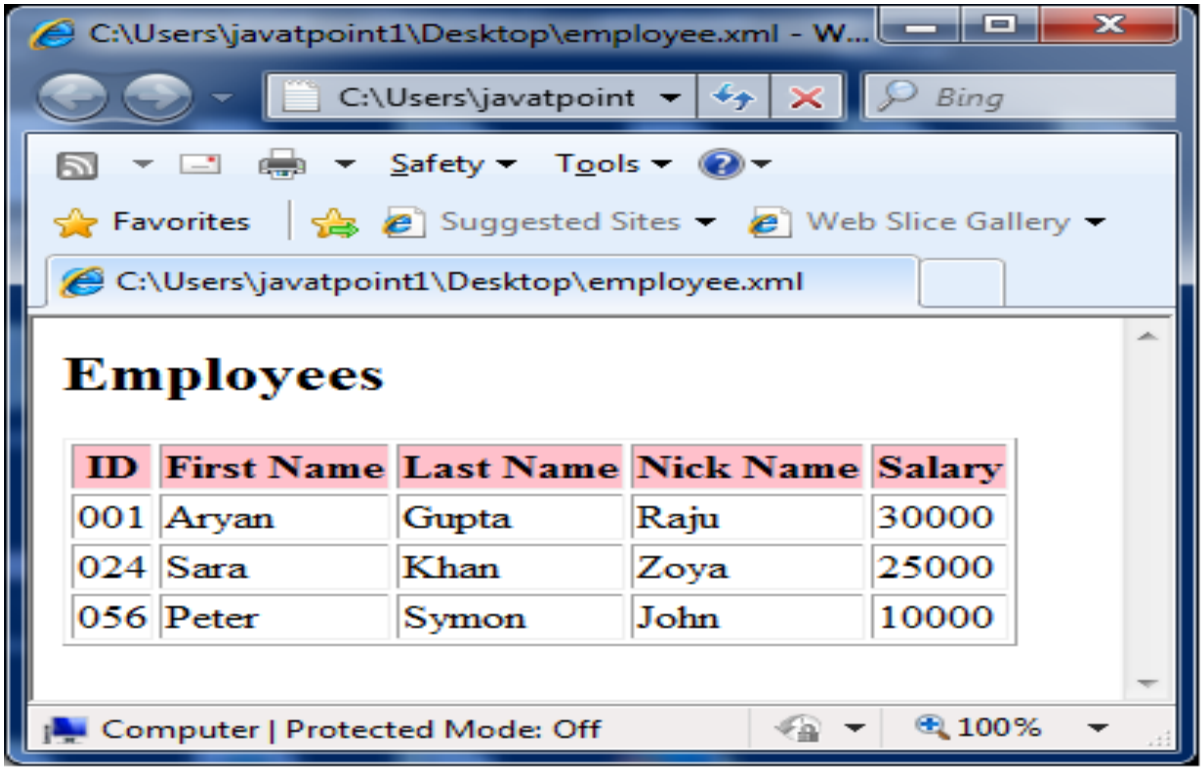
```
<?xml version = "1.0"?>
<?xml-stylesheet type = "text/xsl" href = "employee.xsl"?>
<class>
  <employee id = "001">
    <firstname>Aryan</firstname>
    <lastname>Gupta</lastname>
```

```
    <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>
```

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>
      <body>
        <h2> Employees</h2>
        <table border = "1>
          <tr bgcolor = "pink">
            <th> ID</th>
            <th> First Name</th>
            <th> Last Name</th>
            <th> Nick Name</th>
            <th> Salary</th>
          </tr>
          <xsl:for-each select = "class/employee">
```

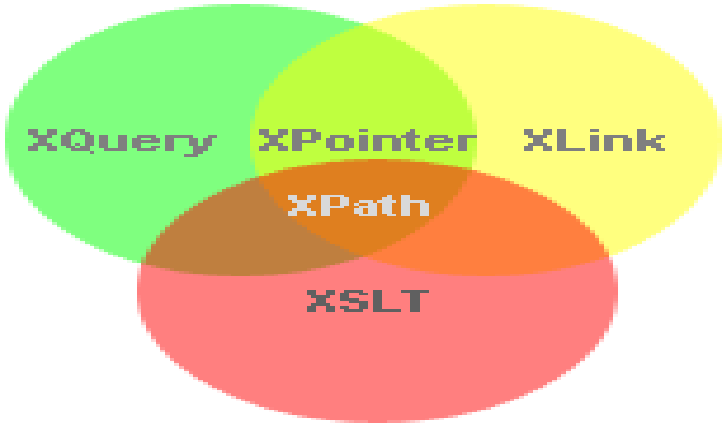
```
            <tr>
              <td> <xsl:value-of select = "@id"/> </td>
              <td> <xsl:value-of select = "firstname"/> </td>
              <td> <xsl:value-of select = "lastname"/> </td>
              <td> <xsl:value-of select = "nickname"/> </td>
              <td> <xsl:value-of select = "salary"/> </td>
            </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```



XQuery:

XQuery is a functional query language used to retrieve information stored in XML format. It is same as for XML what SQL is for databases. It was designed to query XML data.

XQuery is built on XPath expressions. It is a W3C recommendation which is supported by all major databases.



What does it do

XQuery is a functional language which is responsible for finding and extracting elements and attributes from XML documents.

It can be used for following things:

To extract information to use in a web service.

To generates summary reports.

To transform XML data to XHTML.

Search Web documents for relevant information.

XQuery Features:

There are many features of XQuery query language. A list of top features are given below:

XQuery is a functional language. It is used to retrieve and query XML based data.

XQuery is expression-oriented programming language with a simple type system.

XQuery is analogous to SQL. For example: As SQL is query language for databases, same as XQuery is query language for XML.

XQuery is XPath based and uses XPath expressions to navigate through XML documents.

XQuery is a W3C standard and universally supported by all major databases.

Advantages of XQuery:

XQuery can be used to retrieve both hierarchal and tabular data.

XQuery can also be used to query tree and graphical structures.

XQUery can used to build web pages.

XQuery can be used to query web pages.

XQuery is best for XML-based databases and object-based databases. Object databases are much more flexible and powerful than purely tabular databases.

XQuery can be used to transform XML documents into XHTML documents.

XQuery Environment Setup

Let's see how to create a local development environment. Here we are using the jar file of Saxon XQuery processor. The Java-based Saxon XQuery processor is used to test the ".xqy" file, a file containing XQuery expression against our sample XML document.

You need to load Saxon XQuery processor jar files to run the java application.

For eclipse project, add build-path to these jar files. Or, if you are running java using command prompt, you need to set classpath to these jar files or put these jar files inside JRE/lib/ext directory.

How to Set CLASSPATH in Windows Using Command Prompt

Type the following command in your Command Prompt and press enter.

1.set CLASSPATH=%CLASSPATH%;C:\Program Files\Java\jre1.8\rt.jar;

XQuery First Example

Here, the XML document is named as courses.xml and xqy file is named as courses.xqy

courses.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<courses>
    <course category="JAVA">
        <title lang="en">Learn Java in 3 Months.</title>
        <trainer>Sonoo Jaiswal</trainer>
        <year>2008</year>
        <fees>10000.00</fees>
    </course>
     <course category="Dot Net">
        <title lang="en">Learn Dot Net in 3 Months.</title>
        <trainer>Vicky Kaushal</trainer>
        <year>2008</year>
        <fees>10000.00</fees>
```

```
    </course>
     <course category="C">
        <title lang="en">Learn C in 2 Months.</title>
        <trainer>Ramesh Kumar</trainer>
        <year>2014</year>
        <fees>3000.00</fees>
    </course>
     <course category="XML">
        <title lang="en">Learn XML in 2 Months.</title>
        <trainer>Ajeet Kumar</trainer>
        <year>2015</year>
        <fees>4000.00</fees>
    </course>
</courses>
```

courses.xqy

```
for $x in doc("courses.xml")/courses/course
where $x/fees>5000
return $x/title
```

This example will display the title elements of the courses whose fees are greater than 5000.

Create a Java based XQuery executor program to read the courses.xqy, passes it to the XQuery expression processor, and executes the expression. After that the result will be displayed.

XQueryTester.java

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import javax.xml.xquery.XQConnection;
import javax.xml.xquery.XQDataSource;
```

```
import javax.xml.xquery.XQException;
import javax.xml.xquery.XQPreparedExpression;
import javax.xml.xquery.XQResultSequence;
  import com.saxonica.xqj.SaxonXQDataSource;
  public class XQueryTester {
   public static void main(String[] args){
       try {
           execute();
       }
          catch (FileNotFoundException e) {
           e.printStackTrace();
       }
       catch (XQException e) {
           e.printStackTrace();
       }
   }
   private static void execute() throws FileNotFoundException, XQException{
       InputStream inputStream = new FileInputStream(new File("courses.xqy"));
       XQDataSource ds = new SaxonXQDataSource();
       XQConnection conn = ds.getConnection();
       XQPreparedExpression exp = conn.prepareExpression(inputStream);
       XQResultSequence result = exp.executeQuery();
        while (result.next()) {
           System.out.println(result.getItemAsString(null));
       }
   } }
```

Execute XQuery against XML

Put the above three files to a same location. We put them on desktop in a folder name XQuery2. Compile XQueryTester.java using console. You must have JDK 1.5 or later installed on your computer and classpaths are configured.

Compile:

javac XQueryTester.java

Execute:

javaXQueryTester

XQuery FLWOR

FLWOR is an acronym which stands for "For, Let, Where, Order by, Return".

- For - It is used to select a sequence of nodes.
- Let - It is used to bind a sequence to a variable.
- Where - It is used to filter the nodes.
- Order by - It is used to sort the nodes.
- Return - It is used to specify what to return (gets evaluated once for every node).

XQuery FLWOR Example

Example

Following is a sample XML document that contains information on a collection of books. We will use a FLWOR expression to retrieve the titles of those books with a price greater than 30.

books.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<books>
        <book category="JAVA">
        <title lang="en">Learn Java in 24 Hours</title>
        <author>Robert</author>
        <year>2005</year>
        <price>30.00</price>
    </book>
        <book category="DOTNET">
        <title lang="en">Learn .Net in 24 hours</title>
        <author>Peter</author>
```

```
        <year>2011</year>
        <price>70.50</price>
    </book>
        <book category="XML">
        <title lang="en">Learn XQuery in 24 hours</title>
        <author>Robert</author>
        <author>Peter</author>
        <year>2013</year>
        <price>50.00</price>
    </book>
        <book category="XML">
        <title lang="en">Learn XPath in 24 hours</title>
        <author>Jay Ban</author>
        <year>2010</year>
        <price>16.50</price>
    </book>
    </books>
```

The following Xquery document contains the query expression to be executed on the above XML document.

books.xqy

```
let $books := (doc("books.xml")/books/book)
return <results>
{
    for $x in $books
    where $x/price>30
    order by $x/price
    return $x/title
}</results>
```

Result

```
<title lang="en">Learn XQuery in 24 hours</title>
```

<title lang="en">Learn .Net in 24 hours</title>

2. Let's take an XML document having the information on the collection of courses. We will use a FLWOR expression to retrieve the titles of those courses which fees are greater than 2000.

courses.xml

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

<courses>

    <course category="JAVA">

        <title lang="en">Learn Java in 3 Months.</title>

        <trainer>Sonoo Jaiswal</trainer>

        <year>2008</year>

        <fees>10000.00</fees>

    </course>

     <course category="Dot Net">

        <title lang="en">Learn Dot Net in 3 Months.</title>

        <trainer>Vicky Kaushal</trainer>

        <year>2008</year>

        <fees>10000.00</fees>

    </course>

     <course category="C">

        <title lang="en">Learn C in 2 Months.</title>

        <trainer>Ramesh Kumar</trainer>

        <year>2014</year>

        <fees>3000.00</fees>

    </course>

     <course category="XML">

        <title lang="en">Learn XML in 2 Months.</title>

        <trainer>Ajeet Kumar</trainer>

        <year>2015</year>
```

```
        <fees>4000.00</fees>
    </course>
</courses>
```

Let's take the Xquery document named "courses.xqy" that contains the query expression to be executed on the above XML document.

courses.xqy

```
let $courses := (doc("courses.xml")/courses/course)
return <results>
{
    for $x in $courses
    where $x/fees>2000
    order by $x/fees
    return $x/title
}
</results>
```

Create a Java based XQuery executor program to read the courses.xqy, passes it to the XQuery expression processor, and executes the expression. After that the result will be displayed.

XQueryTester.java

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import javax.xml.xquery.XQConnection;
import javax.xml.xquery.XQDataSource;
import javax.xml.xquery.XQException;
import javax.xml.xquery.XQPreparedExpression;
import javax.xml.xquery.XQResultSequence;
 import com.saxonica.xqj.SaxonXQDataSource;
```

```
public class XQueryTester {
    public static void main(String[] args){
        try {
            execute();
        }
        catch (FileNotFoundException e) {
            e.printStackTrace();
        }
        catch (XQException e) {
            e.printStackTrace();
        }
    }

    private static void execute() throws FileNotFoundException, XQException{
        InputStream inputStream = new FileInputStream(new File("courses.xqy"));
        XQDataSource ds = new SaxonXQDataSource();
        XQConnection conn = ds.getConnection();
        XQPreparedExpression exp = conn.prepareExpression(inputStream);
        XQResultSequence result = exp.executeQuery();
         while (result.next()) {
            System.out.println(result.getItemAsString(null));
        }
    }
}
```

XQuery XPath Example

Let's take an XML document having the information on the collection of courses. We will use XQuery expression to retrieve the titles of those courses.

courses.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<courses>
  <course category="JAVA">
    <title lang="en">Learn Java in 3 Months.</title>
    <trainer>Sonoo Jaiswal</trainer>
    <year>2008</year>
    <fees>10000.00</fees>
  </course>
  <course category="Dot Net">
    <title lang="en">Learn Dot Net in 3 Months.</title>
    <trainer>Vicky Kaushal</trainer>
    <year>2008</year>
    <fees>10000.00</fees>
  </course>
  <course category="C">
    <title lang="en">Learn C in 2 Months.</title>
    <trainer>Ramesh Kumar</trainer>
    <year>2014</year>
    <fees>3000.00</fees>
  </course>
  <course category="XML">
    <title lang="en">Learn XML in 2 Months.</title>
    <trainer>Ajeet Kumar</trainer>
    <year>2015</year>
    <fees>4000.00</fees>
  </course>
</courses>
```

courses.xqy

```
(: read the entire xml document :)
let $courses := doc("courses.xml")
```

```
for $x in $courses/courses/course
where $x/fees > 2000
return $x/title
```

Here, we use three different types of XQuery statement that will display the same result having fees is greater than 2000.

Execute XQuery against XML

Put the above three files to a same location. We put them on desktop in a folder name XQuery3. Compile XQueryTester.java using console. You must have JDK 1.5 or later installed on your computer and classpaths are configured.

XQuery vs XPath:

Index Xquery XPath

1)XQuery is a functional programming and query language that is used to query a group of XML data.

XPath is a xml path language that is used to select nodes from an xml document using queries.

2)XQuery is used to extract and manipulate data from either xml documents or relational databases and ms office documents that support an xml data source.

XPath is used to compute values like strings, numbers and boolean types from another xml documents.

3)Xquery is represented in the form of a tree model with seven nodes, namely processing instructions, elements, document nodes, attributes, namespaces, text nodes, and comments.

xpath is represented as tree structure, navigate it by selecting different nodes.

4)Xquery supports xpath and extended relational models.

xpath is still a component of query language.

5)Xquery language helps to create syntax for new xml documents.

xpath was created to define a common syntax and behavior model for xpointer and xslt.