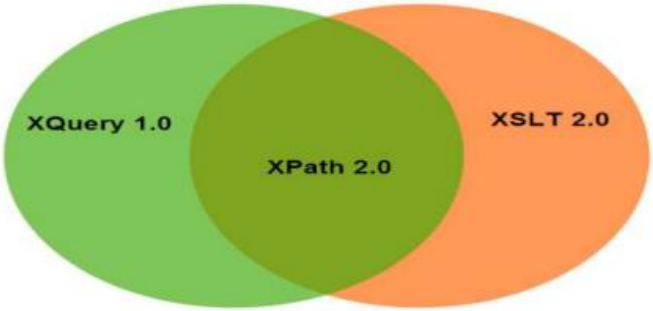
**UNIT IV XML DATABASES 9**

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

---

**XPath:**

XPath is an important and core component of the 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 very similar to the general expressions we used in the 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 nodes 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 expressions. 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 the select attribute of various XSL tags to fetch values of id, firstname, lastname, nickname and salary 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>
```

**Employees**

| ID | First Name | Last Name | Nick Name | Salary |
|---|---|---|---|---|
| 001 | Aryan | Gupta | Raju | 30000 |
| 024 | Sara | Khan | Zoya | 25000 |
| 056 | Peter | Symon | John | 10000 |

**XQuery:**

XQuery is a functional query language used to retrieve information stored in XML format. It is the 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 generate summary reports.
- To transform XML data to XHTML.

**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 an expression-oriented programming language with a simple type system.
- XQuery is analogous to SQL. For example: SQL is a query language for databases, same as XQuery is a 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 be 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 the 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 the 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 the 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, pass it to the XQuery expression processor, and execute 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**

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>
```

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 whose 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, pass it to the XQuery expression processor, and execute 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 expressions 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 statements that will display the same result having fees greater than 2000.

**Execute XQuery against XML**

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

**XQuery vs XPath:**

| **Sl.No** | **XQuery** | **XPath** |
|---|---|---|
| | 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 |
| | 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 other xml documents. |
| | 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 |

| | | |
|---|---|---|
| | XQuery supports XPath and extended relational models | XPath is still a component of query language |
| | 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 |