XQUERY - QUICK GUIDE

XQUERY - OVERVIEW

What is XQuery

XQuery is a functional language that is used to retrieve information stored in XML format. XQuery can be used on XML documents, relational databases containing data in XML formats, or XML Databases. XQuery 3.0 is a W3C recommendation from April 8, 2014.

The definition of XQuery as given by its official documentation is as follows −

> *XQuery is a standardized language for combining documents, databases, Web pages and almost anything else. It is very widely implemented. It is powerful and easy to learn. XQuery is replacing proprietary middleware languages and Web Application development languages. XQuery is replacing complex Java or C++ programs with a few lines of code. XQuery is simpler to work with and easier to maintain than many other alternatives.*

Characteristics

- **Functional Language** − XQuery is a language to retrieve/querying XML based data.
- **Analogous to SQL** − XQuery is to XML what SQL is to databases.
- **XPath based** − XQuery uses XPath expressions to navigate through XML documents.
- **Universally accepted** − XQuery is supported by all major databases.
- **W3C Standard** − XQuery is a W3C standard.

Benefits of XQuery

- Using XQuery, both hierarchical and tabular data can be retrieved.
- XQuery can be used to query tree and graphical structures.
- XQuery can be directly used to query webpages.
- XQuery can be directly used to build webpages.
- XQuery can be used to transform xml documents.
- XQuery is ideal for XML-based databases and object-based databases. Object databases are much more flexible and powerful than purely tabular databases.

XQUERY - ENVIRONMENT SETUP

This chapter elaborates how to set up XQuery library in a local development environment.

We are using an open source standalone XQuery processor Saxon Home Edition $Saxon - HE$ which is widely used. This processor supports XSLT 2.0, XQuery 3.0, and XPath 3.0 and is highly optimized for performance. Saxon XQuery processor can be used without having any XML database. We'll use a simple XML document as our database in our examples.

In order to use Saxon XQuery processor, you should have saxon9he.jar, saxon9-test.jar, saxon9-unpack, saxon9-xqj.jar in your application's classpath. These jar files are available in the download file **SaxonHE9-6-0-1J.zip** Download SaxonHE9-6-0-1J.zip.

Example

We'll use the Java-based Saxon XQuery processor to test books.xqy, a file containing XQuery expression against our sample XML document, i.e., books.xml.

In this example, we'll see how to write and process a query to get the title elements of the books whose price is 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>40.50</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>
```

books.xqy

```
for $x in doc("books.xml")/books/book
where $x/price>30
return $x/title
```

XQueryTester.java

```
package com.tutorialspoint.xquery;

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("books.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));
      }
   }
}
```

Steps to Execute XQuery against XML

- **Step 1** − Copy XQueryTester.java to any location, say, **E:** > **java**
- **Step 2** − Copy books.xml to the same location, **E:** > **java**
- **Step 3** − Copy books.xqy to the same location, **E:** > **java**
- **Step 4** − Compile XQueryTester.java using console. Make sure you have JDK 1.5 or later installed on your machine and classpaths are configured. For details on how to use JAVA, see our JAVA Tutorial

```
E:\java\javac XQueryTester.java
```

- **Step 5** − Execute XQueryTester

```
E:\java\java XQueryTester
```

Output

You'll get the following result −

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

Understanding Example

- books.xml represents the sample data.
- books.xqy represents the XQuery expression which is to be executed on books.xml. We'll understand the expression in details in next chapter.
- XQueryTester, a Java-based XQuery executor program, reads the books.xqy, passes it to the XQuery expression processor, and executes the expression. Then the result is printed.

XQUERY - FIRST APPLICATION

Example

Following is a sample XML document containing the records of a bookstore of various books.

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>
```

Following is a sample Xquery document containing the query expression to be executed on the above XML document. The purpose is to get the title elements of those XML nodes where the price is greater than 30.

books.xqy

```
for $x in doc("books.xml")/books/book
where $x/price>30
return $x/title
```

Result

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

Verify Result

To verify the result, replace the contents of *books.xqy* (given in the Environment Setup chapter) with the above XQuery expression and execute the XQueryTester java program.

XQuery Expressions

Let us understand each piece of the above XQuery expression.

Use of functions

```
doc("books.xml")
```

doc is one of the XQuery functions that is used to locate the XML source. Here we've passed "books.xml". Considering the relative path, books.xml should lie in the same path where books.xqy is present.

Use of XPath expressions

```
doc("books.xml")/books/book
```

XQuery uses XPath expressions heavily to locate the required portion of XML on which search is to be made. Here we've chosen all the book nodes available under books node.

Iterate the objects

```
for $x in doc("books.xml")/books/book
```

XQuery treats xml data as objects. In the above example, $x represents the selected node, while the for loop iterates over the collection of nodes.

Apply the condition

```
where $x/price>30
```

As $x represents the selected node, "/" is used to get the value of the required element; "where" clause is used to put a condition on the search results.

Return the result

```
return $x/title
```

As $x represents the selected node, "/" is used to get the value of the required element, price; "return" clause is used to return the elements from the search results.

XQUERY - FLWOR

FLWOR is an acronym that stands for "For, Let, Where, Order by, Return". The following list shows what they account for in a FLWOR expression −

- **F** - For - Selects a collection of all nodes.
- **L** - Let - Puts the result in an XQuery variable.
- **W** - Where - Selects the nodes specified by the condition.
- **O** - Order by - Orders the nodes specified as per criteria.
- **R** - Return - Returns the final result.

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>
```

Verify Result

To verify the result, replace the contents of *books.xqy* (given in the Environment Setup chapter) with the above XQuery expression and execute the XQueryTester java program.

XQUERY - HTML FORMAT

XQuery can also be easily used to transform an XML document into an HTML page. Take a look at the following example to understand how XQuery does it.

Example

We will use the same books.xml file. The following example uses XQuery extract data from books.xml and create an HTML table containing the titles of all the books along with their respective prices.

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>
```

Given below is the Xquery expression that is to be executed on the above XML document.

books.xqy

```
let $books := (doc("books.xml")/books/book)
return <table><tr><th>Title</th><th>Price</th></tr>
{
   for $x in $books
   order by $x/price
   return <tr><td>{data($x/title)}</td><td>{data($x/price)}</td></tr>
}
</table>
</results>
```

Result

```
<table>
   <tr>
      <th>Title</th>
      <th>Price</th>
   </tr>
   <tr>
      <td>Learn XPath in 24 hours</td>
      <td>16.50</td>
   </tr>
   <tr>
```

```
      <td>Learn Java in 24 Hours</td>
      <td>30.00</td>
   </tr>
   <tr>
      <td>Learn XQuery in 24 hours</td>
      <td>50.00</td>
   </tr>
   <tr>
      <td>Learn .Net in 24 hours</td>
      <td>70.50</td>
   </tr>
</table>
```

Verify Result

To verify the result, replace the contents of *books.xqy* (given in the Environment Setup chapter) with the above XQuery expression and execute the XQueryTester java program.

XQuery Expressions

Here we've used the following XQuery expressions −

- data function to evaluate the value of the title element, and
- {} operator to tell the XQuery processor to consider data as a function. If {} operator is not used, then data will be treated as normal text.

XQUERY - XPATH

XQuery is XPath compliant. It uses XPath expressions to restrict the search results on XML collections. For more details on how to use XPath, see our XPath Tutorial.

Recall the following XPath expression which we have used earlier to get the list of books.

```
doc("books.xml")/books/book
```

XPath Examples

We will use the books.xml file and apply XQuery to it.

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>40.50</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>
```

We have given here three versions of an XQuery statement that fulfil the same objective of displaying the book titles having a price value greater than 30.

XQuery – Version 1

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

for $x in $books/books/book
where $x/price > 30
return $x/title
```

Output

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

XQuery – Version 2

```
(: read all books :)
let $books := doc("books.xml")/books/book

for $x in $books
where $x/price > 30
return $x/title
```

Output

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

XQuery – Version 3

```
(: read books with price > 30 :)
let $books := doc("books.xml")/books/book[price > 30]

for $x in $books
return $x/title
```

Output

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

Verify the Result

To verify the result, replace the contents of *books.xqy* (given in the Environment Setup chapter) with the above XQuery expression and execute the XQueryTester java program.

XQUERY - SEQUENCES

Sequences represent an ordered collection of items where items can be of similar or of different types.

Creating a Sequence

Sequences are created using parenthesis with strings inside quotes or double quotes and numbers as such. XML elements can also be used as the items of a sequence.

XQuery Expression

```
let $items := ('orange', <apple/>, <fruit type="juicy"/>, <vehicle
type="car">sentro</vehicle>, 1,2,3,'a','b',"abc")
let $count := count($items)
return
<result>
   <count>{$count}</count>

   <items>
      {
      for $item in $items
         return <item>{$item}</item>
      }
   </items>

</result>
```

Output

```
<result>
   <count>10</count>
   <items>
      <item>orange</item>
      <item>
         <apple/>
      </item>
      <item>
         <fruit type="juicy"/>
      </item>
      <item>
         <vehicle type="car">Sentro</vehicle>
      </item>
      <item>1</item>
      <item>2</item>
      <item>3</item>
      <item>a</item>
      <item>b</item>
      <item>abc</item>
   </items>
</result>
```

Viewing the Items of a Sequence

Items of a sequence can be iterated one by one, using index or by value. The above example iterated the items of a sequence one by one. Let's see the other two ways in action.

XQuery Expression *Index*

```
let $items := (1,2,3,4,5,6)
let $count := count($items)
return
   <result>
      <count>{$count}</count>
```

```
      <items>
      {
         for $item in $items[2]
         return <item>{$item}</item>
      }
      </items>

   </result>
```

Output

```
<result>
   <count>6</count>
   <items>
      <item>2</item>
   </items>
</result>
```

XQuery Expression *Value*

```
let $items := (1,2,3,4,5,6)
let $count := count($items)
return
   <result>
      <count>{$count}</count>

      <items>
      {
         for $item in $items[. = (1,2,3)]
         return <item>{$item}</item>
      }
      </items>

   </result>
```

Output

```
<result>
   <count>6</count>
   <items>
      <item>1</item>
      <item>2</item>
      <item>3</item>
   </items>
</result>
```

XQUERY - SEQUENCE FUNCTIONS

The following table lists the commonly used sequence functions provided by XQuery.

Sr.No	Name & Description
1	count$*seqasitem*(*) Counts the items in a sequence.
2	sum$*seqasitem*(*) Returns the sum of the items in a sequence.

3

avg$*seqasitem*(*)

Returns the average of the items in a sequence.

4

min$*seqasitem*(*)

Returns the minimum valued item in a sequence.

5

max$*seqasitem*(*)

Returns the maximum valued item in a sequence.

6

distinct-values$*seqasitem*(*)

Returns select distinct items from a sequence.

7

subsequence$*seqasitem*(*, *startingLocasxs*: *double*, length as xs:double)

Returns a subset of provided sequence.

8

insert-before$*seqasitem*(*, *positionasxs*: *integer*, inserts as item*)

Inserts an item in a sequence.

9

remove$*seqasitem*(*, $position as xs:integer)

Removes an item from a sequence.

10

reverse$*seqasitem*(*)

Returns the reversed sequence.

11

index-of$*seqasanyAtomicType*(*, $target as anyAtomicType())

Returns indexes as integers to indicate availability of an item within a sequence.

12

last

Returns the last element of a sequence when used in predicate expression.

13

position

Used in FLOWR expressions to get the position of an item in a sequence.

XQUERY - STRING FUNCTIONS

The following table lists the commonly used string manipulation functions provided by XQuery.

Sr.No	Name & Description
1	string-length$*stringasxs: string* as xs:integer Returns the length of the string.
2	concat$*inputasxs: anyAtomicType*? as xs:string Returns the concatenated string as output.
3	string-join$*sequenceasxs: string* ∗ , $*delimiterasxs: string* as xs:string Returns the combination of items in a sequence separated by a delimiter.

XQUERY - DATE FUNCTIONS

The following table lists the commonly used date functions provided by XQuery.

Sr.No	Name & Description
1	current-date Returns the current date.
2	current-time Returns the current time.
3	current-dateTime Returns both the current date and the current time.

XQUERY - REGULAR EXPRESSIONS

Following is the list of commonly used regular expression functions provided by XQuery

Sr.No	Name & Description
1	matches$*input, $regex* Returns true if the input matches with the provided regular expression.
2	replace$*input, $regex, $string*

Replaces the matched input string with given string.

3

tokenize$*input*, $*regex*

Returns a sequence of items matching the regular expression.

XQUERY - IF THEN ELSE

XQuery provides a very useful if-then-else construct to check the validity of the input values passed. Given below is the syntax of the if-then-else construct.

Syntax

```
if (condition) then
 ...
else
 ...
```

Example

We will use the following books.xml file and apply to it XQuery expression containing an if-then-else construct to retrieve the titles of those books with a price value that is 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>40.50</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 expression is to be applied on the above XML document.

books.xqy

```
<result>
{
   if(not(doc("books.xml"))) then (
      <error>
         <message>books.xml does not exist</message>
      </error>
   )
   else (
      for $x in doc("books.xml")/books/book
      where $x/price>30
      return $x/title
   )
}
</result>
```

Output

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

Verify the Result

To verify the result, replace the contents of *books.xqy* (given in the Environment Setup chapter) with the above XQuery expression and execute the XQueryTester java program.

XQUERY - CUSTOM FUNCTIONS

XQuery provides the capability to write custom functions. Listed below are the guidelines to create a custom function.

- Use the keyword **declare function** to define a function.
- Use the data types defined in the current XML Schema
- Enclose the body of function inside curly braces.
- Prefix the name of the function with an XML namespace.

The following syntax is used while creating a custom function.

Syntax

```
declare function prefix:function_name($parameter as datatype?...)
as returnDatatype?
{
   function body...
};
```

Example

The following example shows how to create a user-defined function in XQuery.

XQuery Expression

```
declare function local:discount($price as xs:decimal?,$percentDiscount as xs:decimal?)
as xs:decimal? {
   let $discount := $price - ($price * $percentDiscount div 100)
   return $discount
};

let $originalPrice := 100
```

```
let $discountAvailed := 10

return ( local:discount($originalPrice, $discountAvailed))
```

Output

```
90
```

Verify the Result

To verify the result, replace the contents of *books.xqy* (given in the Environment Setup chapter) with the above XQuery expression and execute the XQueryTester java program.