XQUERY TUTORIAL

Introduction to XQuery
XQuery is to XML what SQL is to database tables.
XQuery is designed to query XML data - not just XML files, but anything that can appear as XML, including databases.

What You Should Already Know
Before you continue you should have a basic understanding of the following:
- HTML / XHTML
- XML / XML Namespaces
- XPath

What is XQuery?
- XQuery is the language for querying XML data
- XQuery for XML is like SQL for databases
- XQuery is built on XPath expressions
- XQuery is supported by all the major database engines (IBM, Oracle, Microsoft, etc.)
- XQuery is a W3C Recommendation
- XQuery is About Querying XML

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

Here is an example of a question that XQuery could solve:
"Select all CD records with a price less than $10 from the CD collection stored in the XML document called cd_catalog.xml"
XQuery and XPath
XQuery 1.0 and XPath 2.0 share the same data model and support the same functions and operators. If you have already studied XPath you will have no problems with understanding XQuery.
You can read more about XPath in our XPath Tutorial.

XQuery - Examples of Use
XQuery can be used to:
- Extract information to use in a Web Service
- Generate summary reports
- Transform XML data to XHTML
- Search Web documents for relevant information
- XQuery is a W3C Recommendation

XQuery is compatible with several W3C standards, such as XML, Namespaces, XSLT, XPath, and XML Schema.
XQuery 1.0 became a W3C Recommendation January 23, 2007.

XQuery Example

The XML Example Document
We will use the following XML document in the examples below.

"books.xml":
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<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">XQuery Kick Start</title>
  <author>James McGovern</author>
  <author>Per Bothner</author>
  <author>Kurt Cagle</author>
  <author>James Linn</author>
  <author>Vaidyanathan Nagarajan</author>
  <year>2003</year>
  <price>49.99</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>
```

View the "books.xml" file in your browser.
How to Select Nodes From "books.xml"?

Functions

- XQuery uses functions to extract data from XML documents.
- The doc() function is used to open the "books.xml" file:doc("books.xml")

Path Expressions

XQuery uses path expressions to navigate through elements in an XML document.

The following path expression is used to select all the title elements in the "books.xml" file:doc("books.xml")/bookstore/book/title

(/bookstore selects the bookstore element, /book selects all the book elements under the bookstore element, and /title selects all the title elements under each book element)

The XQuery above will extract the following:

```
<title lang="en">Everyday Italian</title>
<title lang="en">Harry Potter</title>
<title lang="en">XQuery Kick Start</title>
<title lang="en">Learning XML</title>
```

Predicates

XQuery uses predicates to limit the extracted data from XML documents.
The following predicate is used to select all the book elements under the bookstore element that have a price element with a value that is less than 30:doc("books.xml")/bookstore/book[price<30]

The XQuery above will extract the following:

```
<book category="CHILDREN">
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
```

XQuery FLWOR Expressions

The XML Example Document

We will use the "books.xml" document in the examples below (same XML file as in the previous chapter).
View the "books.xml" file in your browser.
How to Select Nodes From "books.xml" With FLWOR

Look at the following path expression:

```
doc("books.xml")/bookstore/book[price>30]/title
```

The expression above will select all the title elements under the book elements that are under the bookstore element that have a price element with a value that is higher than 30.
The following FLWOR expression will select exactly the same as the path expression above:for $x in

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

The result will be:

```
<title lang="en">XQuery Kick Start</title>
<title lang="en">Learning XML</title>
```

With FLWOR you can sort the result:for $x in doc("books.xml")/bookstore/book

```
where $x/price>30
order by $x/title
return $x/title
```

FLWOR is an acronym for "For, Let, Where, Order by, Return".
The for clause selects all book elements under the bookstore element into a variable called $x.
The where clause selects only book elements with a price element with a value greater than 30.
The order by clause defines the sort-order. Will be sort by the title element.
The return clause specifies what should be returned. Here it returns the title elements.

The result of the XQuery expression above will be:

```
<title lang="en">Learning XML</title>
<title lang="en">XQuery Kick Start</title>
```

XQuery FLWOR + HTML

The XML Example Document
We will use the "books.xml" document in the examples below (same XML file as in the previous chapters).
Present the Result In an HTML List

Look at the following XQuery FLWOR expression:

```
for $x in doc("books.xml")/bookstore/book/title
order by $x
return $x
```

The expression above will select all the title elements under the book elements that are under the bookstore element, and return the title elements in alphabetical order.

Now we want to list all the book-titles in our bookstore in an HTML list. We add <ul> and <li> tags to the FLWOR expression:

```
<ul>
{
for $x in doc("books.xml")/bookstore/book/title
order by $x
return <li>{$x}</li>
}
</ul>
```

XQuery Terms

In XQuery, there are seven kinds of nodes: element, attribute, text, namespace, processing-instruction, comment, and document (root) nodes.

XQuery Terminology

Nodes

In XQuery, there are seven kinds of nodes: element, attribute, text, namespace, processing-instruction, comment, and document (root) nodes. XML documents are treated as trees of nodes. The root of the tree is called the document node (or root node).

Look at the following XML document:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bookstore>
<book>
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
</bookstore>
```

Example of nodes in the XML document above:<bookstore> (document node)
<author>J K. Rowling</author> (element node)
lang="en" (attribute node)

Atomic values
Atomic values are nodes with no children or parent.
Example of atomic values:J K. Rowling
"en"
Items
Items are atomic values or nodes.

Relationship of Nodes
Parent
Each element and attribute has one parent.

In the following example; the book element is the parent of the title, author, year, and price:

```
<book>
  <title>Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
```

Children
Element nodes may have zero, one or more children.

In the following example; the title, author, year, and price elements are all children of the book element:

```
<book>
  <title>Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
```

Siblings
Nodes that have the same parent.

In the following example; the title, author, year, and price elements are all siblings:

```
<book>
  <title>Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
```

Ancestors

A node's parent, parent's parent, etc.

In the following example; the ancestors of the title element are the book element and the bookstore element:

```
<bookstore>
<book>
  <title>Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
</bookstore>
```

Descendants
A node's children, children's children, etc.

In the following example; descendants of the bookstore element are the book, title, author, year, and price elements:

```
<bookstore>
<book>
  <title>Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
</bookstore>
```

The result of the above will be:

```
<ul>
<li><title lang="en">Everyday Italian</title></li>
<li><title lang="en">Harry Potter</title></li>
<li><title lang="en">Learning XML</title></li>
<li><title lang="en">XQuery Kick Start</title></li>
</ul>
```

Now we want to eliminate the title element, and show only the data inside the title element:

```
<ul>
{
for $x in doc("books.xml")/bookstore/book/title
order by $x
return <li>{data($x)}</li>
}
</ul>
```

The result will be (an HTML list):<ul>

```
<li>Everyday Italian</li>
<li>Harry Potter</li>
<li>Learning XML</li>
<li>XQuery Kick Start</li>
</ul>
```

XQuery Syntax
XQuery is case-sensitive and XQuery elements, attributes, and variables must be valid XML names.

XQuery Basic Syntax Rules
Some basic syntax rules:

- XQuery is case-sensitive
- XQuery elements, attributes, and variables must be valid XML names
- An XQuery string value can be in single or double quotes
- An XQuery variable is defined with a $ followed by a name, e.g. $bookstore
- XQuery comments are delimited by (: and :), e.g. (: XQuery Comment :)
- XQuery Conditional Expressions

- "If-Then-Else" expressions are allowed in XQuery.

Look at the following example:

```
for $x in doc("books.xml")/bookstore/book
return if ($x/@category="CHILDREN")
then <child>{data($x/title)}</child>
else <adult>{data($x/title)}</adult>
```

Notes on the "if-then-else" syntax:
parentheses around the if expression are required. else is required, but it can be just else ().

The result of the example above will be:

```
<adult>Everyday Italian</adult>
<child>Harry Potter</child>
<adult>Learning XML</adult>
<adult>XQuery Kick Start</adult>
```

XQuery Comparisons
In XQuery there are two ways of comparing values.
1. General comparisons: =, !=, <, <=, >, >=
2. Value comparisons: eq, ne, lt, le, gt, ge

The difference between the two comparison methods are shown below.

The following expression returns true if any q attributes have a value greater than 10:

```
$bookstore//book/@q > 10
```

The following expression returns true if there is only one q attribute returned by the expression, and its value is greater than 10. If more than one q is returned, an error occurs:

```
$bookstore//book/@q gt 10
```

XQuery Adding Elements and Attributes

The XML Example Document
We will use the "books.xml" document in the examples below (same XML file as in the previous chapters).

Adding Elements and Attributes to the Result
As we have seen in a previous chapter, we may include elements and attributes from the input document ("books.xml) in the result:

```
for $x in doc("books.xml")/bookstore/book/title
order by $x
return $x
```

The XQuery expression above will include both the title element and the lang attribute in the result, like this:

```
<title lang="en">Everyday Italian</title>
<title lang="en">Harry Potter</title>
<title lang="en">Learning XML</title>
<title lang="en">XQuery Kick Start</title>
```

The XQuery expression above returns the title elements the exact same way as they are described in the input document.
We now want to add our own elements and attributes to the result!

Add HTML Elements and Text
Now, we want to add some HTML elements to the result. We will put the result in an HTML list - together with some text:

```
<html>
<body>
<h1>Bookstore</h1>
<ul>
```

```
{
for $x in doc("books.xml")/bookstore/book
order by $x/title
return <li>{data($x/title)}. Category: {data($x/@category)}</li>
}
</ul>
</body></html>
```

The XQuery expression above will generate the following result:

```
<html>
<body>
<h1>Bookstore</h1>
<ul>
<li>Everyday Italian. Category: COOKING</li>
<li>Harry Potter. Category: CHILDREN</li>
<li>Learning XML. Category: WEB</li>
<li>XQuery Kick Start. Category: WEB</li>
</ul>
</body></html>
```

Add Attributes to HTML Elements

Next, we want to use the category attribute as a class attribute in the HTML list:

```
<html>
<body>
<h1>Bookstore</h1>
<ul>
{
for $x in doc("books.xml")/bookstore/book
order by $x/title
return <li class="{data($x/@category)}">{data($x/title)}</li>
}
</ul>
</body></html>
```

The XQuery expression above will generate the following result:

```
<html>
<body>
<h1>Bookstore</h1>
<ul>
<li class="COOKING">Everyday Italian</li>
<li class="CHILDREN">Harry Potter</li>
<li class="WEB">Learning XML</li>
<li class="WEB">XQuery Kick Start</li>
</ul>
</body></html>
```

XQuery Selecting and Filtering

The XML Example Document

We will use the "books.xml" document in the examples below (same XML file as in the previous chapters).
View the "books.xml" file in your browser.

Selecting and Filtering Elements

As we have seen in the previous chapters, we are selecting and filtering elements with either a Path expression or with a FLWOR expression.

Look at the following FLWOR expression:

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

```
return $x/title
```

- for - (optional) binds a variable to each item returned by the in expression
- let - (optional)
- where - (optional) specifies a criteria
- order by - (optional) specifies the sort-order of the result
- return - specifies what to return in the result

The for Clause
The for clause binds a variable to each item returned by the in expression. The for clause results in iteration. There can be multiple for clauses in the same FLWOR expression.

To loop a specific number of times in a for clause, you may use the to keyword:
```
for $x in (1 to 5)
return <test>{$x}</test>
```

Result:
```
<test>1</test>
<test>2</test>
<test>3</test>
<test>4</test>
<test>5</test>
```

The at keyword can be used to count the iteration:
```
for $x at $i in doc("books.xml")/bookstore/book/title
return <book>{$i}. {data($x)}</book>
```

Result:
```
<book>1. Everyday Italian</book>
<book>2. Harry Potter</book>
<book>3. XQuery Kick Start</book>
<book>4. Learning XML</book>
```

It is also allowed with more than one in expression in the for clause. Use comma to separate each in expression:
```
for $x in (10,20), $y in (100,200)
return <test>x={$x} and y={$y}</test>
```

Result:
```
<test>x=10 and y=100</test>
<test>x=10 and y=200</test>
<test>x=20 and y=100</test>
<test>x=20 and y=200</test>
```

The let Clause
The let clause allows variable assignments and it avoids repeating the same expression many times. The let clause does not result in iteration.let $x := (1 to 5)
return <test>{$x}</test>

Result:
```
<test>1 2 3 4 5</test>
```

The where Clause
The where clause is used to specify one or more criteria for the result:where $x/price>30 and $x/price<100

The order by Clause
The order by clause is used to specify the sort order of the result. Here we want to order the result by category and title:
```
for $x in doc("books.xml")/bookstore/book
order by $x/@category, $x/title
```

return $x/title

Result:
<title lang="en">Harry Potter</title>
<title lang="en">Everyday Italian</title>
<title lang="en">Learning XML</title>
<title lang="en">XQuery Kick Start</title>

The return Clause
The return clause specifies what is to be returned.for $x in doc("books.xml")/bookstore/book
return $x/title
Result:
<title lang="en">Everyday Italian</title>
<title lang="en">Harry Potter</title>
<title lang="en">XQuery Kick Start</title>
<title lang="en">Learning XML</title>

XQuery Functions
XQuery 1.0, XPath 2.0, and XSLT 2.0 share the same functions library.

XQuery Functions
XQuery includes over 100 built-in functions. There are functions for string values, numeric values, date and time comparison, node and QName manipulation, sequence manipulation, Boolean values, and more. You can also define your own functions in XQuery.

XQuery Built-in Functions
The URI of the XQuery function namespace is:
http://www.w3.org/2005/02/xpath-functions

The default prefix for the function namespace is fn:.
Tip: Functions are often called with the fn: prefix, such as fn:string(). However, since fn: is the default prefix of the namespace, the function names do not need to be prefixed when called.
The reference of all the built-in XQuery 1.0 functions is located in our XPath tutorial.

Examples of Function Calls
A call to a function can appear where an expression may appear. Look at the examples below:
Example 1: In an element<name>{uppercase($booktitle)}</name>
Example 2: In the predicate of a path expressiondoc("books.xml")/bookstore/book[substring(title,1,5)='Harry']
Example 3: In a let clauselet $name := (substring($booktitle,1,4))

XQuery User-Defined Functions
If you cannot find the XQuery function you need, you can write your own.
User-defined functions can be defined in the query or in a separate library.
Syntaxdeclare function prefix:function_name($parameter AS datatype)
AS returnDatatype
{
 ...function code here...
}

Notes on user-defined functions:

- Use the declare function keyword
- The name of the function must be prefixed
- The data type of the parameters are mostly the same as the data types defined in XML Schema
- The body of the function must be surrounded by curly braces

Example of a User-defined Function Declared in the Querydeclare function local:minPrice($p as xs:decimal?,$d as xs:decimal?)
AS xs:decimal?
{
let $disc := ($p * $d) div 100
return ($p - $disc)
}

Below is an example of how to call the function above:
<minPrice>{local:minPrice($book/price,$book/discount)}</minPrice>

XQuery Summary
This tutorial has taught you how to query XML data.
You have learned that XQuery was designed to query anything that can appear as XML, including databases.
You have also learned how to query the XML data with FLWOR expressions, and how to construct XHTML output from the collected data.

Now You Know XQuery, What's Next?
The next step is to learn about XLink and XPointer.

- XLink and XPointer
- Linking in XML is divided into two parts: XLink and XPointer.
- XLink and XPointer define a standard way of creating hyperlinks in XML documents.

XQuery Reference
XQuery 1.0 and XPath 2.0 share the same data model and support the same functions and operators.

XQuery Functions
XQuery is built on XPath expressions. XQuery 1.0 and XPath 2.0 share the same data model and support the same functions and operators.

- **XPath Operators**

Below is a list of the operators that can be used in XPath expressions:

Operator	Description	Example	Return value
|	Computes two node-sets	//book | //cd	Returns a node-set with all book and cd elements
+	Addition	6 + 4	10
-	Subtraction	6 - 4	2
*	Multiplication		*
div	Division	8 div 4	2
=	Equal	price=9.80	true if price is 9.80 false if price is 9.90
!=	Not equal	price!=9.80	true if price is 9.90 false if price is 9.80
<	Less than	price<9.80	true if price is 9.00 false if price is 9.80
<=	Less than or equal to	price<=9.80	true if price is 9.00 false if price is 9.90
>	Greater than	price>9.80	true if price is 9.90 false if price is 9.80

>=	Greater than or equal to	price>=9.80	true if price is 9.90 false if price is 9.70
or	or	price=9.80 or price=9.70	true if price is 9.80 false if price is 9.50
and	and	price>9.00 and price<9.90	true if price is 9.80 false if price is 8.50
mod	Modulus (division remainder)	5 mod 2	1

XPath, XQuery, and XSLT Functions
The following reference library defines the functions required for XPath 2.0, XQuery 1.0 and XSLT 2.0.

Functions Reference
The default prefix for the function namespace is fn:, and the URI is:
http://www.w3.org/2005/02/xpath-functions.

Accessor Functions

Name	Description
fn:node	name(node) Returns the node
fn:nilled(node)	Returns a Boolean value indicating whether the argument node is nilled
fn:data(item.item,...)	Takes a sequence of items and returns a sequence of atomic values

fn:base-uri()
fn:base-uri(node)
	Returns the value of the base-uri property of the current or specified node

fn:document uri(node)	Returns the value of the document

Error and Trace Functions

Name	Description

fn:error()
fn:error(error)
fn:error(error,description)
fn:error(error,description,error-object)
	Example: error(fn:QName('http://example.com/test', 'err:toohigh'), 'Error: Price is too high')
	
	Result: Returns http://example.com/test#toohigh and the string "Error: Price is too high" to the external processing environment

fn:trace(value,label)	Used to debug queries

Functions on Numeric Values

Name	Description
fn:string(arg)	Returns the string value of the argument. The argument could be a number, boolean, or node-set Example: string(314) Result: "314"
fn:codepoints-to-string(int,int,...)	Returns a string from a sequence of code points Example: codepoints-to-string(84, 104, 233, 114, 232, 115, 101) Result: 'Thérèse'
fn:string-to-codepoints(string)	Returns a sequence of code points from a string

	Example: string-to-codepoints("Thérèse") Result: 84, 104, 233, 114, 232, 115, 101
fn:codepoint-equal(comp1,comp2)	Returns true if the value of comp1 is equal to the value of comp2, according to the Unicode code point collation (http://www.w3.org/2005/02/xpath-functions/collation/codepoint), otherwise it returns false

fn:compare(comp1,comp2)
fn:compare(comp1,comp2,collation)
	Returns -1 if comp1 is less than comp2, 0 if comp1 is equal to comp2, or 1 if comp1 is greater than comp2 (according to the rules of the collation that is used)
	
	Example: compare('ghi', 'ghi')
	Result: 0

fn:concat(string,string,...)	Returns the concatenation of the strings Example: concat('XPath ','is ','FUN!') Result: 'XPath is FUN!'
fn:string-join((string,string,...),sep)	Returns a string created by concatenating the string arguments and using the sep argument as the separator Example: string-join(('We', 'are', 'having', 'fun!'), ' ') Result: ' We are having fun! ' Example: string-join(('We', 'are', 'having', 'fun!')) Result: 'Wearehavingfun!' Example:string-join((), 'sep') Result: ''

fn:substring(string,start,len)
fn:substring(string,start)
	Returns the substring from the start position to the specified length. Index of the first character is 1. If length is omitted it returns the substring from the start position to the end
	
	Example: substring('Beatles',1,4)
	Result: 'Beat'
	
	Example: substring('Beatles',2)
	Result: 'eatles'

fn:string-length(string)
fn:string-length()
	Returns the length of the specified string. If there is no string argument it returns the length of the string value of the current node
	
	Example: string-length('Beatles')
	Result: 7

fn:normalize-space(string)
fn:normalize-space()
	Removes leading and trailing spaces from the specified string, and replaces all internal sequences of white space with one and returns the result. If there is no string argument it does the same on the current node
	
	Example: normalize-space(' The XML ')
	Result: 'The XML'

fn:normalize-unicode()	
fn:upper-case(string)	Converts the string argument to upper-case Example: upper-case('The XML') Result: 'THE XML'
fn:lower-case(string)	Converts the string argument to lower-case Example: lower-case('The XML')

	Result: 'the xml'
fn:translate(string1,string2,string3)	Converts string1 by replacing the characters in string2 with the characters in string3 Example: translate('12:30','30','45') Result: '12:45' Example: translate('12:30','03','54') Result: '12:45' Example: translate('12:30','0123','abcd') Result: 'bc:da'
fn:escape-uri(stringURI,esc-res)	Example: escape-uri("http://example.com/test#car", true()) Result: "http%3A%2F%2Fexample.com%2Ftest#car" Example: escape-uri("http://example.com/test#car", false()) Result: "http://example.com/test#car" Example: escape-uri ("http://example.com/~bébé", false()) Result: "http://example.com/~b%C3%A9b%C3%A9"
fn:contains(string1,string2)	Returns true if string1 contains string2, otherwise it returns false Example: contains('XML','XM') Result: true
fn:starts-with(string1,string2)	Returns true if string1 starts with string2, otherwise it returns false Example: starts-with('XML','X') Result: true
fn:ends-with(string1,string2)	Returns true if string1 ends with string2, otherwise it returns false Example: ends-with('XML','X') Result: false
fn:substring-before(string1,string2)	Returns the start of string1 before string2 occurs in it Example: substring-before('12/10','/') Result: '12'
fn:substring-after(string1,string2)	Returns the remainder of string1 after string2 occurs in it Example: substring-after('12/10','/') Result: '10'
fn:matches(string,pattern)	Returns true if the string argument matches the pattern, otherwise, it returns false Example: matches("Merano", "ran") Result: true
fn:replace(string,pattern,replace)	Returns a string that is created by replacing the given pattern with the replace argument Example: replace("Bella Italia", "l", "*") Result: 'Be**a Ita*ia' Example: replace("Bella Italia", "l", "") Result: 'Bea Itaia'
fn:tokenize(string,pattern)	Example: tokenize("XPath is fun", "\s+") Result: ("XPath", "is", "fun")

Functions on Numeric Values

Name	**Description**
fn:number(arg)	Returns the numeric value of the argument. The argument could be a boolean, string, or node-set Example: number('100') Result: 100
fn:abs(num)	Returns the absolute value of the argument Example: abs(3.14) Result: 3.14 Example: abs(-3.14) Result: 3.14
fn:ceiling(num)	Returns the smallest integer that is greater than the number argument Example: ceiling(3.14) Result: 4
fn:floor(num)	Returns the largest integer that is not greater than the number argument Example: floor(3.14) Result: 3
fn:round(num)	Rounds the number argument to the nearest integer Example: round(3.14) Result: 3
fn:round-half-to-even()	Example: round-half-to-even(0.5) Result: 0 Example: round-half-to-even(1.5) Result: 2 Example: round-half-to-even(2.5) Result: 2

Functions for anyURI

Name	**Description**
fn:resolve-uri(relative,base)	

Functions on Boolean Values

Name	**Description**
fn:boolean(arg)	Returns a boolean value for a number, string, or node-set
fn:not(arg)	The argument is first reduced to a boolean value by applying the boolean() function. Returns true if the boolean value is false, and false if the boolean value is true Example: not(true()) Result:
fn:true()	Returns the boolean value true Example: true() Result: true

fn:false()	Returns the boolean value false Example: false() Result: false

Functions on Durations, Dates and Times

Component Extraction Functions on Durations, Dates and Times

Name	**Description**
fn:dateTime(date,time)	Converts the arguments to a date and a time
fn:years-from-duration(datetimedur)	Returns an integer that represents the years component in the canonical lexical representation of the value of the argument
fn:months-from-duration(datetimedur)	Returns an integer that represents the months component in the canonical lexical representation of the value of the argument
fn:days-from-duration(datetimedur)	Returns an integer that represents the days component in the canonical lexical representation of the value of the argument
fn:hours-from-duration(datetimedur)	Returns an integer that represents the hours component in the canonical lexical representation of the value of the argument
fn:minutes-from-duration(datetimedur)	Returns an integer that represents the minutes component in the canonical lexical representation of the value of the argument
fn:seconds-from-duration(datetimedur)	Returns a decimal that represents the seconds component in the canonical lexical representation of the value of the argument
fn:year-from-dateTime(datetime)	Returns an integer that represents the year component in the localized value of the argument Example: year-from-dateTime(xs:dateTime("2005-01-10T12:30-04:10")) Result: 2005
fn:month-from-dateTime(datetime)	Returns an integer that represents the month component in the localized value of the argument Example: month-from-dateTime(xs:dateTime("2005-01-10T12:30-04:10")) Result: 01
fn:day-from-dateTime(datetime)	Returns an integer that represents the day component in the localized value of the argument Example: day-from-dateTime(xs:dateTime("2005-01-10T12:30-04:10")) Result: 10
fn:hours-from-dateTime(datetime)	Returns an integer that represents the hours component in the localized value of the argument Example: hours-from-dateTime(xs:dateTime("2005-01-10T12:30-04:10")) Result: 12
fn:minutes-from-dateTime(datetime)	Returns an integer that represents the minutes component in the localized value of the argument Example: minutes-from-dateTime(xs:dateTime("2005-01-10T12:30-04:10")) Result: 30
fn:seconds-from-dateTime(datetime)	Returns a decimal that represents the seconds component in the localized value of the argument

	Example: seconds-from-dateTime(xs:dateTime("2005-01-10T12:30:00-04:10")) Result: 0
fn:timezone-from-dateTime(datetime)	Returns the time zone component of the argument if any
fn:year-from-date(date)	Returns an integer that represents the year in the localized value of the argument Example: year-from-date(xs:date("2005-04-23")) Result: 2005
fn:month-from-date(date)	Returns an integer that represents the month in the localized value of the argument Example: month-from-date(xs:date("2005-04-23")) Result: 4
fn:day-from-date(date)	Returns an integer that represents the day in the localized value of the argument Example: day-from-date(xs:date("2005-04-23")) Result: 23
fn:timezone-from-date(date)	Returns the time zone component of the argument if any
fn:hours-from-time(time)	Returns an integer that represents the hours component in the localized value of the argument Example: hours-from-time(xs:time("10:22:00")) Result: 10
fn:minutes-from-time(time)	Returns an integer that represents the minutes component in the localized value of the argument Example: minutes-from-time(xs:time("10:22:00")) Result: 22
fn:seconds-from-time(time)	Returns an integer that represents the seconds component in the localized value of the argument Example: seconds-from-time(xs:time("10:22:00")) Result: 0
fn:timezone-from-time(time)	Returns the time zone component of the argument if any
fn:adjust-dateTime-to-timezone(datetime,timezone)	If the timezone argument is empty, it returns a dateTime without a timezone. Otherwise, it returns a dateTime with a timezone
fn:adjust-date-to-timezone(date,timezone)	If the timezone argument is empty, it returns a date without a timezone. Otherwise, it returns a date with a timezone
fn:adjust-time-to-timezone(time,timezone)	If the timezone argument is empty, it returns a time without a timezone. Otherwise, it returns a time with a timezone

Functions Related to QNames

Name	**Description**
fn:QName()	
fn:local-name-from-QName()	
fn:namespace-uri-from-QName()	
fn:namespace-uri-for-prefix()	
fn:in-scope-prefixes()	

fn:resolve-QName()	

Functions on Nodes

Name	Description

fn:name()
fn:name(nodeset)
	Returns the name of the current node or the first node in the specified node set

fn:local-name()
fn:local-name(nodeset)
	Returns the name of the current node or the first node in the specified node set - without the namespace prefix

fn:namespace-uri()
fn:namespace-uri(nodeset)
	Returns the namespace URI of the current node or the first node in the specified node set

fn:lang(lang)	Returns true if the language of the current node matches the language of the specified language Example: Lang("en") is true for <p xml:lang="en">...</p> Example: Lang("de") is false for <p xml:lang="en">...</p>

fn:root()
fn:root(node)
	Returns the root of the tree to which the current node or the specified belongs. This will usually be a document node

Functions on Sequences

General Functions on Sequences

Name	Description
fn:index-of((item,item,...),searchitem)	Returns the positions within the sequence of items that are equal to the searchitem argument Example: index-of ((15, 40, 25, 40, 10), 40) Result: (2, 4) Example: index-of (("a", "dog", "and", "a", "duck"), "a") Result (1, 4) Example: index-of ((15, 40, 25, 40, 10), 18) Result: ()
fn:remove((item,item,...),position)	Returns a new sequence constructed from the value of the item arguments - with the item specified by the position argument removed Example: remove(("ab", "cd", "ef"), 0) Result: ("ab", "cd", "ef") Example: remove(("ab", "cd", "ef"), 1) Result: ("cd", "ef") Example: remove(("ab", "cd", "ef"), 4) Result: ("ab", "cd", "ef")
n:empty(item,item,...)	Returns true if the value of the arguments IS an empty sequence, otherwise it returns false Example: empty(remove(("ab", "cd"), 1)) Result: false
fn:exists(item,item,...)	Returns true if the value of the arguments IS NOT an empty sequence, otherwise it returns false Example: exists(remove(("ab"), 1)) Result: false
fn:distinct-values((item,item,...),collation)	Returns only distinct (different) values Example: distinct-values((1, 2, 3, 1, 2))

	Result: (1, 2, 3)
fn:insert-before((item,item,...),pos,inserts)	Returns a new sequence constructed from the value of the item arguments - with the value of the inserts argument inserted in the position specified by the pos argument Example: insert-before(("ab", "cd"), 0, "gh") Result: ("gh", "ab", "cd") Example: insert-before(("ab", "cd"), 1, "gh") Result: ("gh", "ab", "cd") Example: insert-before(("ab", "cd"), 2, "gh") Result: ("ab", "gh", "cd") Example: insert-before(("ab", "cd"), 5, "gh") Result: ("ab", "cd", "gh")
fn:reverse((item,item,...))	Returns the reversed order of the items specified Example: reverse(("ab", "cd", "ef")) Result: ("ef", "cd", "ab") Example: reverse(("ab")) Result: ("ab")
fn:subsequence((item,item,...),start,len)	Returns a sequence of items from the position specified by the start argument and continuing for the number of items specified by the len argument. The first item is located at position 1 Example: subsequence(($item1, $item2, $item3,...), 3) Result: ($item3, ...) Example: subsequence(($item1, $item2, $item3, ...), 2, 2) Result: ($item2, $item3)
fn:unordered((item,item,...))	Returns the items in an implementation dependent order

Functions That Test the Cardinality of Sequences

Name	**Description**
fn:zero-or-one(item,item,...)	Returns the argument if it contains zero or one items, otherwise it raises an error
fn:one-or-more(item,item,...)	Returns the argument if it contains one or more items, otherwise it raises an error
fn:exactly-one(item,item,...)	Returns the argument if it contains exactly one item, otherwise it raises an error

Equals, Union, Intersection and Except

Name	**Description**
fn:deep equal(param1,param2,collation)	Returns true if param1 and param2 are deep

Aggregate Functions

Name	**Description**
fn:count((item,item,...))	Returns the count of nodes
fn:max((arg,arg,...))	Returns the argument that is greater than the others Example: max((1,2,3))

	Result: 3 Example: max(('a', 'k')) Result: 'k'
fn:avg((arg,arg,...))	Returns the average of the argument values Example: avg((1,2,3)) Result: 2
fn:min((arg,arg,...))	Returns the argument that is less than the others Example: min((1,2,3)) Result: 1 Example: min(('a', 'k')) Result: 'a'
fn:sum(arg,arg,...)	Returns the sum of the numeric value of each node in the specified node-set

Functions that Generate Sequences

Name	Description
fn:id((string,string,...),node)	Returns a sequence of element nodes that have an ID value equal to the value of one or more of the values specified in the string argument
fn:idref((string,string,...),node)	Returns a sequence of element or attribute nodes that have an IDREF value equal to the value of one or more of the values specified in the string argument
fn:doc(URI)	
fn:doc available(URI	) Returns true if the doc() function returns a document node, otherwise it returns false

ffn:collection()
fn:collection(string)	

Context Functions

Name	Description
fn:position()	Returns the index position of the node that is currently being processed Example: //book[position()<=3] Result: Selects the first three book elements
fn:last()	Returns the number of items in the processed node list Example: //book[last()] Result: Selects the last book element
fn:current-dateTime()	Returns the current dateTime (with timezone)
fn:current-date()	Returns the current date (with timezone)
fn:current-time()	Returns the current time (with timezone)
fn:implicit-timezone()	Returns the value of the implicit timezone
fn:default-collation()	Returns the value of the default collation
fn:static-base-uri()	Returns the value of the base-uri

By: DataIntegratedEntity
Source: http://w3schools.com/xquery/default.asp