

XQuery

by Magdalena Turska / [@magdaturska](#)

eXist Solutions 2016



Press **SPACE** to navigate through slides.



XQUERY

XQuery is a functional programming language that queries and transforms collections of structured and unstructured data, usually in the form of XML

XQuery 3.0 became a W3C Recommendation in 2014.

The mission of the XML Query project is to provide flexible query facilities to extract data from real and virtual documents on the World Wide Web, therefore finally providing the needed interaction between the Web world and the database world. Ultimately, collections of XML files will be accessed like databases".

A functional language: functions as basic building blocks

In contrast to imperative programming, the functional paradigm is based on the evaluation of functions

Avoid mutable data and state changes!

Comparable languages: Lisp, OCaml, Haskell, Scala, Erlang

XQuery is not just a query language: can be used to write entire applications

SIMPLE ARITHMETIC

```
xquery version "3.0";  
2+2
```

TITLE OF THE PLAY

```
xquery version "3.0";  
doc("/db/apps/demo/data/hamlet.xml")/PLAY/TITLE
```

ALL SPEAKERS FROM HAMLET

```
xquery version "3.0";  
distinct-values(doc("/db/apps/demo/hamlet.xml")//SPEAKER)
```

```
"Hello world!"
```

```
5 * 5
```

```
()
```

```
("Hello", "world")
```

```
<p>Hello world!</p>
```

```
sum((1, 2, 3, 4, 5))
```

```
sum(1 to 5)
```

```
//tei:head
```

XDM

The language is based on the XQuery and XPath Data Model (XDM) which uses a tree-structured model of the information content of an XML document, containing seven kinds of nodes:

document nodes, elements, attributes, text nodes, comments, processing instructions, and namespaces.

XDM

XDM also models all values as sequences (a singleton value is considered to be a sequence of length one). The items in a sequence can either be XML nodes or atomic values. Atomic values may be integers, strings, booleans, and so on: the full list of types is based on the primitive types defined in XML Schema.

SEE WHAT THE FINN SAYS

*All XQuery expressions **operate on sequences**, and **evaluate to sequences**. Sequences are ordered lists of items. Items can be either **nodes**, which represent components of XML documents, or **atomic values**, which are instances of XML Schema base types like `xs:integer` or `xs:string`. Sequences can also be empty, or consist of a single item only. No distinction is made between a single item and a singleton sequence.*

[Pekka Kilpeläinen†](#) University of Eastern Finland, Kuopio

XQUERY AND XPATH

XQuery and XPath 2.0 share a common data model, and a common set of over 100 functions and operators

Nodes of document trees in XQuery are selected using XPath location path expressions, thus XPath 2.0 is a core of XQuery language.

XQUERY AND XSLT

XQuery is capable of transforming individual documents, so its capabilities overlap with XSLT, which was designed expressly to allow input XML documents to be transformed into HTML or other formats.

The XSLT 2.0 and XQuery standards were developed by separate working groups within W3C, working together to ensure a common approach where appropriate. They share the same data model (XDM), type system, and function library, and both include XPath 2.0 as a sublanguage.

XML IN, XML OUT

XQuery also provides syntax allowing new XML documents to be constructed. Where the element and attribute names are known in advance, an XML-like syntax can be used; in other cases, expressions referred to as dynamic node constructors are available. All these constructs are defined as expressions within the language, and can be arbitrarily nested.

That said, XQuery allows to construct other output formats, such as text, JSON etc

An XQuery Script consists of

- A single main expression
- A collection of functions organized into modules
- The module containing the single main expression is called the "main module"

The result of an XQuery script is the result of the main expression ● You don't need an explicit "return"

SYNTACTICAL COMPONENTS

XPath 2.0 expressions • FLWOR expressions • Functions •
Modules and Prologs • Constructors (to construct XML
output) • Other: • Conditional expressions (if-then-else) •
try-catch • typeswitch/switch

FROM SQL TO XML: FLWOR

XQuery contains a superset of XPath expression syntax to address specific parts of an XML document. It supplements this with a SQL-like "FLWOR expression" (pronounced 'flower') for performing joins.

FLWOR supports iteration and binding of variables to intermediate results.

FLWOR

An acronym: FOR, LET, WHERE, ORDER BY, RETURN.

Loosely analogous to SQL's SELECT-FROM-WHERE can be used to provide join-like functionality to XML documents.

- for creates a sequence of nodes
- let binds a sequence to a variable
- where filters the nodes on a boolean expression
- order by sorts the nodes
- return gets evaluated once for every item in the resulting sequence

FOR EXPRESSION

```
1 xquery version "3.0";
2
3 declare namespace tei="http://www.tei-c.org/ns/1.0";
4
5 for $i in //tei:l[ft:query(., 'king')]
6 return $i
```

__new__4

XML Output ☐ Live Preview

```
1 <l xmlns="http://www.tei-c.org/ns/1.0" xml:id="sha-1h4103124" n="124"> Send us your prisoners, or you will hear of it.
  <stage>Exeunt King Henry, Blunt, and train.</stage>
</l>
2 <l xmlns="http://www.tei-c.org/ns/1.0" xml:id="sha-1h4103136" n="136"> As high in the air as this unthankful king, </l>
3 <l xmlns="http://www.tei-c.org/ns/1.0" xml:id="sha-1h4103138" n="138"> Brother, the king hath made your nephew mad. </l>
4 <l xmlns="http://www.tei-c.org/ns/1.0" xml:id="sha-1h4103148" n="148"> And then it was when the unhappy king, </l>
```

Bind the \$speech variable to each tei:sp element returned by the "in" expression

The "return" marks the end of the FLWOR expression

Result of the return expression = result of the FLWOR

LET EXPRESSION

```
1 xquery version "3.0";
2
3 declare namespace tei="http://www.tei-c.org/ns/1.0";
4
5 let $matches := //tei:l[ft:query(., 'king')]
6 let $count := count($matches)
7
8 return $count
```

__new__4

XML Output ☐ Live Preview


1213

Bind \$matches to all tei:sp elements returned by expression
Bind \$count to the result of count(\$matches)

ORDER BY

```
1 xquery version "3.0";
2
3 declare namespace tei="http://www.tei-c.org/ns/1.0";
4
5 let $matches := //tei:sp[ft:query(., "King Henry")]
6 for $match in $matches
7 order by ft:score($match)
8
9 return $match
```

__new__4

XML Output ☐ Live Preview 

```
1 <sp xmlns="http://www.tei-c.org/ns/1.0" who="HenryIV">
  <speaker>King Henry</speaker>
  <l xml:id="sha-1h4302029" n="29"> God pardon thee; yet let me wonder, Harry, </l>
  <l xml:id="sha-1h4302030" n="30"> At thy affections, which do hold a wing </l>
```

GROUP BY

```
1 xquery version "3.0";
2
3 declare namespace tei="http://www.tei-c.org/ns/1.0";
4
5 for $speech in //tei:sp[ft:query(., 'juliet')]
6 group by $play := $speech/ancestor::tei:TEI
7
8 return ($play//tei:titleStmt/tei:title, $speech)
```

__new__ 4



XML Output



Live Preview



```
1 <title xmlns="http://www.tei-c.org/ns/1.0">Romeo and Juliet</title>
2 <sp xmlns="http://www.tei-c.org/ns/1.0" who="Nurse">
  <speaker>Nurse to Juliet</speaker>
  <l xml:id="sha-roj103002" n="2"> Now, by my maidenhead, at twelve year old. </l>
  <l xml:id="sha-roj103003" n="3"> I bade her come. What, lamb! what, ladybird! </l>
  <l xml:id="sha-roj103004" n="4"> God forbid! Where's this girl? What, Juliet!
    <stage>Enter JULIET.</stage>
  </l>
</sp>
3 <sp xmlns="http://www.tei-c.org/ns/1.0" who="Juliet1">
  <speaker>Juliet</speaker>
```

Find all speeches containing "juliet" and group them by play

```
for $d in doc("depts.xml")//deptno
let $e := doc("emps.xml")//employee[deptno = $d]
where count($e) >= 10
order by avg($e/salary) descending
return
  <big-dept>
    { $d,
      <headcount>{count($e)}</headcount>,
      <avgsal>{avg($e/salary)}</avgsal>
    }
  </big-dept>
```

VARIABLE SCOPE

CONSTRUCTORS

Direct XML constructors:

```
<h1 class="title">This is a heading</h1>
```

Computed constructors:

```
element { "h1" } {  
    attribute { "class" } { "title" },  
    text { "This is a heading" }  
}
```

You only will need the constructors for working with variables

ENCLOSED EXPRESSIONS

To embed XQuery expressions into XML text or an attribute value, you need to escape with {}

```
<h1>The time is { current-dateTime() }</h1>
```

SERIALIZATION SETTINGS

```
1 xquery version "3.0";
2
3 declare namespace tei="http://www.tei-c.org/ns/1.0";
4 declare namespace output="http://www.w3.org/2010/xslt-xquery-serialization";
5
6 declare option output:method "html";
7 declare option output:media-type "text/html";
8
9 <div>
10   <h1>Plays</h1>
11 {
12   for $play in collection("/db/apps/shakespeare")//tei:TEI
13   return <li>{$play//tei:titleStmt/tei:title/string()}</li>
14 }
15 </div>
```

__new__4



XML Output



Live Preview



```
<div>
  <h1>Plays</h1>
  <li>The First Part of King Henry the Fourth</li>
  <li>The First Part of King Henry the Sixth</li>
  <li>The Second Part of King Henry the Fourth</li>
  <li>The Second Part of King Henry the Sixth</li>
  <li>The Third Part of King Henry the Sixth</li>
```

For the browser to recognize the XQuery result as HTML it is necessary to change the serialization settings

CONDITIONAL EXPRESSIONS

```
if ($condition) then $block1 else $block2
```

Note: the “else” part is required! Use the empty sequence () to do nothing in “else”

```
if ($conference/name="XML Amsterdam") then <location>Amsterdam</location>
```

TRY TO PREPARE XQUERY SCRIPTS THAT:

- present the list of files in a collection
- present the metadata for a single file
- present the list of files filtered by a criterion (eg. containing given word)

BOOLEAN EXPRESSIONS

true: 55, -3, “yes”, true(), empty(()), exists(<a/>)

false: 0, “”, false(), (), empty((<a/>,)), exists(())

Important: the empty sequence is false, so you can test:

```
if ($input/conference) then
```

but:

```
if ((1, 2, 3)) then
```

generates an error! It has to contain only one value

FUNCTION DECLARATION

General syntax:

```
declare function prefix:name($arg1, $arg2) {  
  (: body :)  
};
```

Typed:

```
declare  
function pref:name($arg1 as xs:integer?, $arg2 as xs:string+) as element  
  (: body :)  
};
```

FUNCTION EXAMPLE

```
declare
function local:multiply($n1 as xs:decimal, $n2 as xs:decimal) as xs:decimal {
    $n1 * $n2
};
```

“local” is a pre-defined namespace prefix which can be used for functions within “main” modules (see next slide to understand modules)

MODULES

XQuery code always belongs to a module:

- code outside a function as well as any “local” functions are part of the “main” module
- all other modules define a namespace in their prolog:
module namespace pfx="uri";
- A module can import other modules using import module namespace pfx="uri";

Declare the “math” module and save it to e.g.
“/db/math.xql”

```
module namespace math="http://existsolutions.com/math";  
declare  
function math:multiply($n1 as xs:decimal, $n2 as xs:decimal) as xs:decimal  
    $n1 * $n2  
};
```

and use it

```
xquery version "3.0";  
import module namespace math="http://existsolutions.com/math";  
math:multiply(1.25, 3.12)
```

```
xquery version "3.0";
declare default element namespace "http://www.tei-c.org/ns/1.0";
<ul>
  {
    for $i in collection("/db/apps/myGraves/data")//titleStmt//ti
    return
    <li>
      $i || ' ' || base-uri($i)
    </li>
  }
</ul>
```

SEQUENCE AND BANG! OPERATOR

```
xquery version "3.0";  
("Do", "you", "want", "me", "to", "shout?") ! upper-case(.)
```

```
xquery version "3.0";  
<ul>  
{  
  for $i in 1 to 5  
  return <li>{$i}</li>  
}  
</ul>
```

TYPESWITCH EXPRESSION

```
typeswitch ($node)
  case element(tei:div) return Block
  case element(tei:head) return Block
  default return Block
```

```

1 module namespace tei2="http://exist-db.org/xquery/app/tei2html";
2
3 declare namespace tei="http://www.tei-c.org/ns/1.0";
4
5 declare function tei2:tei2html($nodes as node()*) {
6     for $node in $nodes
7     return
8         typeswitch ($node)
9         case element(tei:div) return
10             <div>{ tei2:tei2html($node/node()) }</div>
11         case element(tei:head) return
12             <h1>{ tei2:tei2html($node/node()) }</h1>
13         case element() return
14             tei2:tei2html($node/node())
15         default return
16             $node
17 };

```



A recursive function to traverse a node tree
The core of many libraries we will use

THE END!