EMC²
where information lives

XML Foundations
Fall 2011

XQuery

Part II

September 29, 2011

EMC² documentum

© 2009 EMC Corporation. All rights reserved.

Expression Language

EMC²
where information lives

- XQuery is an expression language, not a statement language
  - you express what value you want, not the steps how to get it
  - any expression evaluates to a value („returns a value“)
  - or to the empty sequence
- Expressions are fully composable

EMC² documentum

© 2009 EMC Corporation. All rights reserved.

XQuery: Literals

EMC² where information lives

- Literals are the most simple expressions
- String Literals use single or double quotes
  - "Hello ' World"
  - "Hello "" World"
  - 'Hello " World'
  - No variable expansion in "Hello $foo World" – doesn't work!
- Numbers
  - 42 (xs:integer)
  - 3.5 (xs:decimal)
  - .35e1 (xs:double)
  - 1 to 5 – sequence of xs:integers (1, 2, 3, 4, 5)

EMC² documentum

© 2009 EMC Corporation. All rights reserved.

More Literals

EMC² where information lives

- Sequences of values using parenthesis and comma
  - (1, "Hello World", 4.5)
- Sequences can be of mixed types
- Other primitive values can be constructed through casts
  - xs:QName("f:bar")
  - xs:float(3.4E6)
  - Identical to casting: "f:bar" cast as xs:QName
- Time values follow ISO8601
  - xs:yearMonthDuration("PT1Y5M")
  - xs:dayTimeDuration("PT6D4H5M2.34S")
  - xs:dateTime("2006-03-24T09:30:00+01:00") → UTC+1

EMC² documentum

© 2009 EMC Corporation. All rights reserved.

Arithmetic

EMC² where information lives

- Standard operators +, -, *, div, idiv
  - * can also have an XPath meaning: foo//*
  - division is "div", not "/"
- Basic arithmetic built in functions
  - sum((1, 2, 3, 4)) = 10
  - ceiling(4.2) = 5, floor(4.2) = 4
  - round(4.5) =5
  - round-half-to-even(4.45, 1) = 4.4

EMC² documentum

© 2009 EMC Corporation. All rights reserved.

Comparison

EMC² where information lives

- Two types of comparators
- Existential (General) comparisons are set operators
  - "=", "!=", ">", ">=", ...
  - A = B means ∃ a ∈ A, b ∈ B: a = b
    - (6,7) = (7,6) → true
  - Relaxed typing (e.g. <x>5</x> = 5)
- Value comparisons
  - "eq", "neq", "gt", "ge", ...
  - Enforces exactly one element on each side and matching types (error otherwise)

EMC² documentum

© 2009 EMC Corporation. All rights reserved.

Booleans

EMC² where information lives

- Built in type xs:boolean
  - Construct using xs:boolean(“true”)
  - valid literals: “true”, “false”, “0”, “1”
  - easier: functions true() and false()
- Boolean operators
  - true() and true(), false() or true(), not(true())
- Effective boolean value “autocasts” values to boolean
  - if (<x/>) is true
  - if (“asd”) is true, if (“”) is false
  - if (5) is true, if (0) is false

EMC² documentum

© 2009 EMC Corporation. All rights reserved.

Conditional Expression

EMC² where information lives

- if/then/else

```
if (5 = 2) then "???"
else "Expected"
```

- Else is always needed (functional expression language!)
- Use empty sequence () or error() when needed

```
if ($mycond) then "foo"
else ()
```

EMC² documentum

© 2009 EMC Corporation. All rights reserved.

The Prolog

- Prolog comes in front of the query
- Declare namespaces, global variables, global options, external variables etc.
- Important declarations
  - declare namespace foo = "http://bar";
  - declare option xhive:queryplan-debug 'true';
- Predefined namespace prefixes
  - xml, xmlns, fn, xs, xsi, op, xdt, local
  - plus implementation defined prefixes (e.g. xhive)

EMC² documentum
© 2009 EMC Corporation. All rights reserved.

Specify the XML Source File to Query

- To specify an XML source file for querying use the **`doc`** or **`collection`** function
- Specify a single XML source file or an entire xDB database
  - For example, to open a single file:

```
doc("food.xml")
```

```
collection("food.xml")
```

  - For example, to open an xDB library (containing multiple XML source files):

```
doc("/63001b3f80000128")
```

```
collection("/63001b3f80000128")
```

- All XQuerys have an initial context item
  - Default starting point for unqualified XPath expressions
  - In xDB, this is the library, document, or node where you created the XQuery

EMC² documentum
© 2009 EMC Corporation. All rights reserved.

More on the doc function

- In xDB, doc and collection are synonymous
- Parameter is a string containing a URI
  - doc('foo.xml') – relative path within xDB
  - doc('/bar/foo.xml') – absolute path within xDB
  - doc('http://www.example.com/foo.xml') – HTTP request
  - doc('xhive://foo/bar/../test.xml') – complete xDB URI
  - doc('file:///my/doc.xml') – absolute local file URI
  - doc('file:doc.xml') – relative file URI
- Accessing a library (doc('lib/')) gives
  - all documents in the library
  - all documents in descendant libraries
  - only works within xDB, not for filesystem

EMC² documentum

© 2009 EMC Corporation. All rights reserved.

XPath Overview

- XQuery uses XPath expressions to access nodes
  - A *node* is any part of the XML tree: the root, an element, an attribute, a text node, etc.
- XPath
  - Expresses path patterns on XML trees, like file system paths
    - `doc('foo.xml')/a/b/c`
  - Each step
    - Results in **sequence** of nodes
    - De-duplicates nodes
    - Sorts nodes into document order

EMC² documentum

© 2009 EMC Corporation. All rights reserved.

XML documents are trees!

EMC² where information lives

- Simple XML bibliography document

```
<bib>
  <book year="1992">
    <author>W. Stevens</author>
    <title>TCP/IP Illustrated</title>
  </book>
  <book year="2000">
    <author>Serge Abiteboul</author>
    <title>Data on the Web</title>
  </book>
</bib>
```

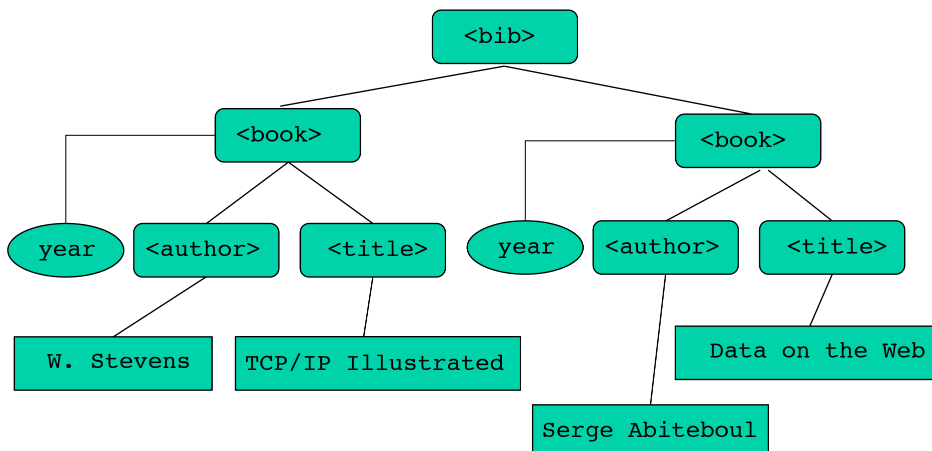
EMC² documentum

© 2009 EMC Corporation. All rights reserved.

XML documents are trees!

EMC² where information lives

- Data model behind XML is a node-labeled, ordered Tree
- Textual syntax with < and > is just a serialization form



EMC² documentum

© 2009 EMC Corporation. All rights reserved.

XPath Axes

- Abbreviated
  - `/bar = /child::bar`
  - `//bar = /descendant-or-self::node()/child:bar`
  - `/@x = /attribute::x`
  - `/../bar = /parent::node()/child::bar`
- All directions
  - `parent::, self::, child::, attribute::`
  - `descendant::, descendant-or-self::`
  - `ancestor::, ancestor-or-self::`
  - `preceding-sibling::, following-sibling::, preceding::, following::`

EMC² documentum

© 2009 EMC Corporation. All rights reserved.

Node Tests

- After each axis step, write a node test
  `axis-name::node-test`
- Pseudo-functions
  - `item(), node(), element(), attribute(), text() document-node(), processing-instruction(), comment()`
- Qualified names, wildcards
  - `/foo:bar, /*, /*:bar, /foo:*`

EMC² documentum

© 2009 EMC Corporation. All rights reserved.

XPath Predicates

- Filter node sequences from steps
  - /bib//book[@year = 1992]/author
- Filter by position: /bib/book[3]
- . (dot/full stop) selects current context item (outer item)
- Special functions
  - book[position() > 3]
  - book[last()]
- Can contain any expression, fully composable:

```
/foo[.//bar[@attr = 42]
    and count(for $x in doc('x.xml')//x
              where $x/@y = 'abc'
              return $x) > 3]
```

EMC² documentum

© 2009 EMC Corporation. All rights reserved.

XPath Examples (1 of 2)

Simple Paths	/A/B/T	Select elements T, which are nested underneath elements B and A.
Wildcards	/A/*	Selects all the elements underneath A.
Descendant Paths	/A//B/F	Selects elements F, which are children of B, which in turn can be nested at any level underneath A.
Attribute Access	/A/B/@M	Selects attribute M of element B
Subscripts	/A/B[1]/T	Selects element T, which is a child of the first B nested under A

EMC² documentum

© 2009 EMC Corporation. All rights reserved.

XPath Examples (2 of 2)

Filter	/A//B[NAME="Joe"]	Selects elements B nested at any level under A if they have a child NAME whose value equals Joe
Parent	/A/B//T..	Select the parent node of T if it is nested underneath the /A/B node tree
Group	/Video/(Movie | Documentary)/Producer	Selects producers of all movies and documentaries that are children of Video

In XPath syntax, the "|" character is a union operator, *not* an OR operator

EMC² documentum

© 2009 EMC Corporation. All rights reserved.

XPath and XQuery

- Use XPath to indicate the part(s) of the XML file to select for searching
- Example: output the **<name>** element for each dairy item

```
- <food>
  - <dairy type="cheese">
      <name>Poldark Stilton</name>
      <origin>Europe</origin>
      <price unit="pound">18.99</price>
    </dairy>
  - <dairy type="cheese">
      <name>Beau Fromage Mimolette</name>
      <origin>Europe</origin>
      <price unit="pound">14.99</price>
    </dairy>
  - <dairy type="cheese">
      <name>Marjo's Minnesota Cheddar</name>
      <origin>USA</origin>
      <price unit="pound">4.99</price>
    </dairy>
  - <dairy type="milk">
      <name>Jack's Goat Milk</name>
      <origin>USA</origin>
      <price unit="liter">1.99</price>
    </dairy>
  </food>
```

```
doc("food.xml")/food/dairy/name
```

or

```
doc("food.xml")//name
```

```
<name>Poldark Stilton</name>
<name>Beau Fromage Mimolette</name>
<name>Marjo's Minnesota Cheddar</name>
<name>Jack's Goat Milk</name>
```

EMC² documentum

© 2009 EMC Corporation. All rights reserved.

Use a Predicate to Filter Results

- Use a predicate to filter the results
  - Predicates are always in []
- Example: extract XML for all dairy items whose origin is USA

```
- <food>
  - <dairy type="cheese">
      <name>Poldark Stilton</name>
      <origin>Europe</origin>
      <price unit="pound">18.99</price>
    </dairy>
  - <dairy type="cheese">
      <name>Beau Fromage Mimolette</name>
      <origin>Europe</origin>
      <price unit="pound">14.99</price>
    </dairy>
  - <dairy type="cheese">
      <name>Marjo's Minnesota Cheddar</name>
      <origin>USA</origin>
      <price unit="pound">4.99</price>
    </dairy>
  - <dairy type="milk">
      <name>Jack's Goat Milk</name>
      <origin>USA</origin>
      <price unit="liter">1.99</price>
    </dairy>
  </food>
```

```
doc("food.xml")//dairy[origin="USA"]
```

```
<dairy type="cheese">
  <name>Marjo's Minnesota Cheddar</name>
  <origin>USA</origin>
  <price unit="pound">4.99</price>
</dairy>
<dairy type="milk">
  <name>Jack's Goat Milk</name>
  <origin>USA</origin>
  <price unit="liter">1.99</price>
</dairy>
```

EMC² documentum

© 2009 EMC Corporation. All rights reserved.

FLWOR Statements and Variables

- FLWOR statements (**for**, **let**, **where**, **order by**, **return**) provide looping, sorting, and variable assignment capabilities
  - It provides another way, in addition to predicates, to filter XQuery results

```
for $variableA [at $variableB] in expression1
[let $variableC := expression2]
[where expression3]
[order by expression4]
return expression4
```

at, **let**, **where**, and **order by** are optional, as indicated by the square brackets

- **for**: loops once for each item produced by *expression1*
- Variables: precede with **$** and declare/use as needed
- **let**: assign variables to values
- **where**: evaluate once per loop
- **order by**: sort the results in the desired order
- **return**: output results

EMC² documentum

© 2009 EMC Corporation. All rights reserved.

for expressions

EMC² where information lives

- Iterate over all elements in a sequence
- Bind current element to the variable
- Trivial example:

```
for $x in /bib/book
return $x
```

- 100% identical to simply writing /bib/book

EMC² documentum
© 2009 EMC Corporation. All rights reserved.
Using XQuery 4-23

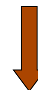
FLWOR Statements and Variables
Example: Output the <name> for European Cheeses

EMC² where information lives

Output the **<name>** element for each dairy item for all cheeses that whose origin is Europe

```
- <food>
  - <dairy type="cheese">
      <name>Poldark Stilton</name>
      <origin>Europe</origin>
      <price unit="pound">18.99</price>
    </dairy>
  - <dairy type="cheese">
      <name>Beau Fromage Mimolette</name>
      <origin>Europe</origin>
      <price unit="pound">14.99</price>
    </dairy>
  - <dairy type="cheese">
      <name>Marjo's Minnesota Cheddar</name>
      <origin>USA</origin>
      <price unit="pound">4.99</price>
    </dairy>
  - <dairy type="milk">
      <name>Jack's Goat Milk</name>
      <origin>USA</origin>
      <price unit="liter">1.99</price>
    </dairy>
  </food>
```

```
for $x in doc("food.xml")//dairy
where $x/origin ="Europe" and
$x/@type="cheese"
return $x/name
```



```
<name>Poldark Stilton</name>
<name>Beau Fromage Mimolette</name>
```

EMC² documentum
© 2009 EMC Corporation. All rights reserved.
Using XQuery 4-24

FLWOR Statements and Variables
Example: Sort the Output

EMC² where information lives

Output all **<name>** elements for European cheeses, sorted by the cheese name

- Note the **order by** clause, which does the sorting

```
- <food>
  - <dairy type="cheese">
      <name>Poldark Stilton</name>
      <origin>Europe</origin>
      <price unit="pound">18.99</price>
    </dairy>
  - <dairy type="cheese">
      <name>Beau Fromage Mimolette</name>
      <origin>Europe</origin>
      <price unit="pound">14.99</price>
    </dairy>
  - <dairy type="cheese">
      <name>Marjo's Minnesota Cheddar</name>
      <origin>USA</origin>
      <price unit="pound">4.99</price>
    </dairy>
  - <dairy type="milk">
      <name>Jack's Goat Milk</name>
      <origin>USA</origin>
      <price unit="liter">1.99</price>
    </dairy>
  </food>
```

```
for $x in doc("food.xml")//dairy
where $x/origin="Europe" and
$x/@type="cheese"
order by $x/name
return $x/name
```

```
<name>Beau Fromage Mimolette</name>
<name>Poldark Stilton</name>
```

EMC² documentum

© 2009 EMC Corporation. All rights reserved.

FLWOR Statements and Variables
Example: Remove the XML Tags

EMC² where information lives

Output all the names (in plain text) for all European cheeses, sorted by the cheese name

- Note the **data** function, which converts nodes to atomic values

```
- <food>
  - <dairy type="cheese">
      <name>Poldark Stilton</name>
      <origin>Europe</origin>
      <price unit="pound">18.99</price>
    </dairy>
  - <dairy type="cheese">
      <name>Beau Fromage Mimolette</name>
      <origin>Europe</origin>
      <price unit="pound">14.99</price>
    </dairy>
  - <dairy type="cheese">
      <name>Marjo's Minnesota Cheddar</name>
      <origin>USA</origin>
      <price unit="pound">4.99</price>
    </dairy>
  - <dairy type="milk">
      <name>Jack's Goat Milk</name>
      <origin>USA</origin>
      <price unit="liter">1.99</price>
    </dairy>
  </food>
```

```
for $x in doc("food.xml")//dairy
where $x/origin="Europe" and
$x/@type="cheese"
order by $x/name
return data($x/name)
```

```
Beau Fromage Mimolette
Poldark Stilton
```

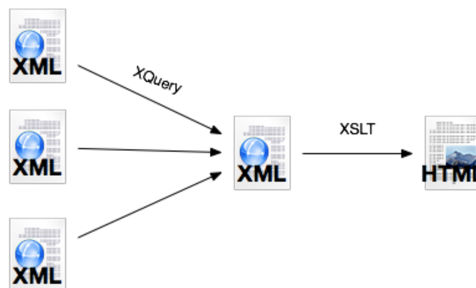
EMC² documentum

© 2009 EMC Corporation. All rights reserved.

XQuery and XML Transformation

EMC² where information lives

- XQuery includes the ability to format the query output
- Output formatting – such as HTML – is generally much easier than with XSLT (Extensible Stylesheet Language Transformation)
- For complex formatting, collect XML using XQuery, use XSLT to format to output
- For simple formatting, XQuery can be good enough



EMC² documentum

© 2009 EMC Corporation. All rights reserved.

Using XQuery 4-27