XML and Databases

Lecture 12
XQuery – XML Query Language

Sebastian Maneth
NICTA and UNSW

Thanks to Sherif Sakr
for all the following XQuery slides.

CSE@UNSW -- Semester 1, 2010

Why do we need a new query language?

<table>
<thead>
<tr>
<th>Relational Data, SQL</th>
<th>XML</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flat (rows and columns)</td>
<td>Nested and Hierarchical</td>
</tr>
<tr>
<td>Data is uniform and repetitive</td>
<td>Data is highly variable</td>
</tr>
<tr>
<td>Info schema for meta data</td>
<td>Self-describing, meta data distributed through doc</td>
</tr>
<tr>
<td>Uniform query results</td>
<td>Heterogeneous query results</td>
</tr>
<tr>
<td>Rows in table are unordered</td>
<td>Elements in document are ordered</td>
</tr>
<tr>
<td>Data is usually dense</td>
<td>Data can be sparse</td>
</tr>
</tbody>
</table>

XQuery, XSLT and XPath

XQuery

- XQuery is a declarative language in which a query is represented as an expression.
- XQuery expressions can be nested with full generality.

XML Data model life cycle

XQuery

- XQuery is based on OQL, SQL, XML-QL, XPath languages.

XQuery 1.0

XML Data Model

XPath 2.0

XQuery

XML 2.0

XSLT 2.0

XML - QL

OQL

SQL

XPointer

XSL patterns

XPath

XQuery

XQL - 99

Quit

xml

parse

XQuery Data Model

XQuery

XSLT 2.0

serialize

xml

validate

XQuery Data Model

XQuery

XSLT 2.0

serialize

xml

validate
XQuery

- The input and output of an XQuery are instances of the XML Query Data Model.

XQuery Data Model

- The XQuery language is designed to operate over ordered, finite sequences of items as its principal data type.
- The evaluation of any XQuery expression yields an ordered sequence of \( n \geq 0 \) items.
- These items can be:
  - Atomic values (integers, strings, ... etc)
  - Unranked XML tree nodes.

XQuery + ½ Programming Language + ½ Query Language

- **Programming language** features:
  - Explicit iteration and variable bindings (for, let, ...).
  - Recursive, user-defined functions.
  - Regular expressions, strong [static] typing.
  - Ordered sequences (much like lists or arrays).

- **Query language** features:
  - Filtering.
  - Grouping.
  - Joins.

XML Input

- Could be:
  - Text files that are XML documents.
  - Fragments of XML documents that are received from the web using a URI.
  - A collection of XML documents that are associated with a particular URI.
  - Data stored in native XML databases.
  - Data stored in relational databases that have an XML front-end.
  - In-memory XML documents.

Items and Ordered Sequences

- A sequence of \( n \) items \( X \), is written in parentheses and comma-separated form
  \((X_1, X_2, ..., X_n)\)
- A single item \( X \) and the singleton sequence \( (X) \) are equivalent.
- Sequences can contain other sequences (nested sequences are implicitly flattened)
  \((0, (1, 2), (3)) = (0, 1, 2, 3)\)
- Sequences can contain duplicates
  \((0, 1, 1, 2)\)
- Sequences may be heterogeneous
  \((42, "foo", 4.2, <a></a>)\)

Some Uses for XQuery

- Extracting information from a database for use in web service.
- Generating summary reports on data stored in XML database.
- Searching textual documents on the web for relevant information.
- Transforming XML data to XHTML format to be published on the web.
- Pulling data from different databases to be used for application integration.
- Splitting up an XML document into multiple XML documents.
**XQuery Syntax Rules**

- XQuery is a case-sensitive language.
- Keywords are in lower-case.
- No special end-of-line character.
- Every expression has a value and no side effects.
- Expressions are fully composable.
- Expressions can raise error.
- Comments look like this:
  `(: This is an XQuery Comment :)`

**XQuery Expressions**

- Path expressions.
- FLWOR expressions.
- Expressions involving operators and functions.
- Conditional expressions.
- Quantified expressions.
- List constructors.
- Element constructors.
- Expressions that test or modify datatypes.

---

**Path Expression**

- In a sense, the traversal or navigation of trees of XML nodes lies at the core of every XML query language.
- XQuery embeds XPath as its tree navigation sub-language.
- Every XPath expression is a correct XQuery expression.
- Since navigation expressions extract (potentially huge volumes of) nodes from input XML documents, efficient XPath implementation is a prime concern to any implementation of an XQuery processor.

**Path Expression**

- Each path consists of one or more steps, syntactically separated by `/`:
  \[ s_0 / s_1 / \ldots / s_n \]
- Each step acts like an operator that, given a sequence of nodes (the context set), evaluates to a sequence of nodes.
- XPath defines the result of each path expression to be `duplicate free` and `sorted in document order`.

---

**XQuery Expressions**

- Path expressions.
- FLWOR expressions.
- Expressions involving operators and functions.
- Conditional expressions.
- Quantified expressions.
- List constructors.
- Element constructors.
- Expressions that test or modify datatypes.
FLWOR Expression

- A FLWOR expression binds some expressions, applies a predicate, and constructs a new ordered result.

```plaintext
for $for_var in for_expression
let $let_var = let_expression
where clause
```

- The `for` construct successively binds each item of an expression (`expr`) to a variable (`var`), generating a so-called tuple stream.
- This tuple stream is then filtered by the `where` clause, retaining some tuples and discarding others.
- The `return` clause is evaluated once for every tuple still in the stream.
- The result of the expression is an ordered sequence containing the concatenated results of these evaluations.

FLWOR Expression (Variables)

- Variables are identified by a name proceeded by a `$` sign.
- Variables are defined in several places
  - FLWOR Expression.
  - Query prologs.
  - Outside the query by the processor.
  - Function signatures.

for Clauses

- Iteratively binds the variable to each item returned by the `in` expressions.
- The rest of the expression is evaluated once for each item returned.
- Multiple `for` clauses are allowed in the same FLWOR expression.

let Clauses

- Convenient way to bind variables.
- Does not result in iteration.

where Clauses

- Used to filter results.
- Can contain many sub-expressions.
- Evaluates to a Boolean value.
- If true, `return` clause is evaluated.
order by Clauses

- Only way to sort results in XQuery.
- Order by
  - Atomic values, or
  - Nodes that contain individual atomic values.
- Can specify multiple values to sort on.

```
for $item in doc("ord.xml")//item
order by $item/@dept, $item/@num
return $item
```

return Clauses

- The value that is to be returned

```
for $prod in doc("cat.xml")//product
return $prod/brand
```

- Single expression only. Multiple expressions are to be combined into single sequence.

```
return <a>{$i}</a>
<bp>{$j}</bp>
```

FLWOR Expression

**Iteration**

<table>
<thead>
<tr>
<th>Expressions</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>for $x in (3,2,1) return ($x,&quot;*&quot;)</td>
<td>(3,&quot;<em>&quot;,2,&quot;</em>&quot;,1,&quot;*&quot;)</td>
</tr>
<tr>
<td>for $x in (3,2,1) return $x,=&quot;*&quot;</td>
<td>(3,2,1,&quot;*&quot;)</td>
</tr>
<tr>
<td>for $x in (3,2,1) return for $y in (&quot;a&quot;, &quot;b&quot;) return ($x,$y)</td>
<td>(3,&quot;a&quot;,3,&quot;b&quot;, 2,&quot;a&quot;,2,&quot;b&quot;, 1,&quot;a&quot;,1,&quot;b&quot;)</td>
</tr>
</tbody>
</table>

Query:
```
for $a in document("bib.xml")//article
where $a/year < 1996
return
<early_paper>
  <fstAuth>{$a/authors/author[1]/text()}</fstAuth>
  {$a/title}
</early_paper>
```

Results:
```
<early_paper>
  <fstAuth>Maurice Bach</fstAuth>
  <title>Design of the UNIX Operating System</title>
</early_paper>
<early_paper>
  <fstAuth>Serge Arbellebou</fstAuth>
  <title>Foundations of Databases</title>
</early_paper>
```

FLWOR Expression: Test?

- What is the result of the following FLWOR expression?

```
for $x in (1, 2, 3, 4) where $x < 4 return
  for $y in (10, 20) return ($x, $y)
```

FLWOR Expression (Multiple Variables)

- Use comma to separate multiple in expressions.
- return clause evaluated for each combination of variable values.

```
for $i in (1, 2), $j in (11, 12)
return (<eval>i is {$i} and j is {$j}</eval>)
```

```xml
<eval>i is 1 and j is 11</eval>
<eval>i is 1 and j is 12</eval>
<eval>i is 2 and j is 11</eval>
<eval>i is 2 and j is 12</eval>
```
FLWOR Expression

- In a sense, FLWOR takes the role of the SELECT-FROM-WHERE block in SQL.
- The versatile FLWOR is used to express:
  - Nested Iterations.
  - Joins between sequences.
  - Groupings.
  - Orderings beyond document order.

Inner Joins

```xml
for $book in document("bib.xml")//book,
   $quote in document("quotes.xml")//listing
   where $book/isbn = $quote/isbn
return
   <book>
     { $book/title }
     { $quote/price }
   </book>
```

Outer Joins

```xml
for $book in document("bib.xml")//book
return
   <book>
     { $book/title }
     
     for $review in document("reviews.xml")//review
     where $book/isbn = $review/isbn
     return $review/rating
   
   </book>
```

Aggregation - Grouping

- for iterates on a sequence, binds a variable to each node.
- let binds a variable to a sequence as a whole.
- Together, they are used for representing aggregation and grouping expressions.

```xml
for $book in document("bib.xml")//book
let $a := $book/author
where contains($book/publisher, "Addison-Wesley")
return
   <book>
     
     { $book/title, 
       <count> Number of authors: { count($a) } </count>
     }
   </book>
```

FLWOR vs. Path

- Path expression is great if you want to copy or retrieve certain element and attributes as is.
- FLWOR Expression
  - Allow sorting.
  - Allow adding elements/attributes to results.
  - More verbose, but can be clearer.

XQuery Expressions

- Path Expressions.
- FLWOR Expressions.
- Expressions involving operators and functions.
- Conditional expressions.
- Quantified expressions.
- List constructors.
- Element Constructors.
- Expressions that test or modify datatypes
XQuery Operators and Functions

- Infix and prefix operators (+, -, *, ...).
- Parenthesized expressions.
- Arithmetic and logical operators.
- Collection operators UNION, INTERSECT and EXCEPT.
- Infix operators BEFORE and AFTER (<<, >>).
- User functions can be defined in XQuery.

XQuery Arithmetic

- Infix operators: +, -, *, div, idiv (integer division)
- Operators first atomize their operands, then perform promotion to a common numeric type.
- If at least one operand is (), the result is ().

Examples and pitfalls:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;x+y&gt; + 41</td>
<td>42.0</td>
</tr>
<tr>
<td>1 * 42</td>
<td>()</td>
</tr>
<tr>
<td>(1, 2) - (2, 3)</td>
<td>f</td>
</tr>
<tr>
<td>1 ** 2</td>
<td>1</td>
</tr>
<tr>
<td>/child:x&lt;42</td>
<td>type error</td>
</tr>
<tr>
<td>/child:x/child:y</td>
<td>use x, -42</td>
</tr>
<tr>
<td>xy</td>
<td>/child:x/child:y</td>
</tr>
</tbody>
</table>

XQuery Comparisons

- Any XQuery expression evaluates to a sequence of items. Consequently, many XQuery concepts are prepared to accept sequences (as opposed to single items).

General Comparisons:

The general comparison $e_1 \theta e_2$ with
$$\theta \in \{=, !=, <, <=, >, >=\}$$
yields $true()$ if any of the items in the sequences $e_1, e_2$ compare $true$ (existential semantics).

General and Value Comparisons

<table>
<thead>
<tr>
<th>General comparison examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, 2, 3) &gt; (2, 4, 5) =&gt; true()</td>
</tr>
<tr>
<td>(1, 2, 3) = 1 =&gt; true()</td>
</tr>
<tr>
<td>() = 0 =&gt; false()</td>
</tr>
<tr>
<td>2 &lt;= 1 =&gt; false()</td>
</tr>
<tr>
<td>(1, 2) /= 3 =&gt; true()</td>
</tr>
<tr>
<td>(1, 2) /= (1, 2) =&gt; true()</td>
</tr>
<tr>
<td>not((1, 2) = (1, 2)) =&gt; false()</td>
</tr>
</tbody>
</table>

Value comparisons:

The six value comparison operators eq, ne, lt, le, ge, gt compare single items by value (atomization):

<table>
<thead>
<tr>
<th>Value comparison examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 gt 1, 0 =&gt; true()</td>
</tr>
<tr>
<td>&lt;x&gt;42&lt;/x&gt; eq &lt;y&gt;42&lt;/y&gt; =&gt; true()</td>
</tr>
<tr>
<td>(0, 1) eq 0 =&gt; f</td>
</tr>
</tbody>
</table>

More Comparisons

- Note: The existential semantics of the general comparison operators may lead to unexpected behavior:

<table>
<thead>
<tr>
<th>Surprises</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, 2, 3) = (1, 3) =&gt; true()[2]</td>
</tr>
<tr>
<td>(&quot;a&quot;, &quot;b&quot;) = 1 =&gt; true() or f (impl. dependent)</td>
</tr>
</tbody>
</table>

Node Comparisons

- Node comparisons based on identity and document order:
  - $e_1 = e_2$ nodes $e_1, e_2$ identical?
  - $e_1 < e_2$ node $e_1$ before $e_2$?
  - $e_1 > e_2$ node $e_1$ after $e_2$?

Node comparison examples:

<table>
<thead>
<tr>
<th>Node comparison examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;x&gt;42&lt;/x&gt; eq &lt;y&gt;42&lt;/y&gt; =&gt; true()</td>
</tr>
<tr>
<td>&lt;x&gt;42&lt;/x&gt; le &lt;y&gt;42&lt;/y&gt; =&gt; false()</td>
</tr>
<tr>
<td>root(e1) = root(e2) =&gt; nodes $e_1, e_2$ in same tree?</td>
</tr>
<tr>
<td>let $S_a := &lt;x&gt;y&gt;/&lt;z&gt;/s_a$ return $S_a &lt; S_a$ =&gt; true()</td>
</tr>
</tbody>
</table>
### XQuery Comparisons

<table>
<thead>
<tr>
<th>Value</th>
<th>comparing single values Untyped data ~ string</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>Existential quantification Untyped data ~ coerced to other operand's type</td>
</tr>
<tr>
<td>Node</td>
<td>for testing identity of single nodes is, isnot</td>
</tr>
<tr>
<td>Order</td>
<td>testing relative position of one node vs. another (in document order) &lt;=, &gt;=</td>
</tr>
</tbody>
</table>

### Logical Expression

- Logical Operators "and" and "or".

- The concept of Effective Boolean Value (EBV) is key to evaluating logical expressions.
  - EBV of an empty sequence is false.
  - EBV of a non-empty sequence containing only nodes is true.
  - EBV is the value of the expression if the expression evaluates to a value of type `xs:boolean`.
  - EBV is an error in every other case.

- Example: The expression 
  
  \`'(a) and true()'\`

  evaluates to false (since `a` is false).

### XQuery: Built-in Functions

- Over 100 functions built into XQuery.
  - String-related: substring, contains, concat, ...
  - Date-related: current-date, month-from-date, ...
  - Number-related: round, avg, sum, ...
  - Sequence-related: index-of, distinct-values, ...
  - Node-related: data, empty, ...
  - Document-related: doc, collection, ...
  - Error Handling: error, exactly-one, ...
  - Others:

### XQuery: User-Defined Functions

- XQuery expressions can contain user-defined functions which encapsulate query details.

- User-defined functions may be collected into modules and then imported by a query.

#### Declaration of n-ary function f with body c

```xml
declare function f($p_1 as $t_1, ..., $p_n as $t_n) as $t_0 { c }
```

- If `t` is omitted, it defaults to `items()`.
- The pair `(f, n)` is required to be unique (overloading).
- If `t` is atomic.

### User-Defined Functions Example

**Reverse a sequence**

Reversing a sequence does not inspect the sequence's items in any way:

```xml
declare function reverse($seq)
  { for $i at $p in $seq
      order by $p descending
      return $i
  }
reverse((42,"a","b/>,doc("foo.xml")))
```

### XQuery Expressions

- Path expressions.
- FLWOR expressions.
- Expressions involving operators and functions.
- Conditional expressions.
- Quantified expressions.
- List constructors.
- Element constructors.
- Expressions that test or modify datatypes
Conditional Expression

- Syntax:
  \[
  \text{if } (\text{expr1}) \text{ then } \text{expr2} \text{ else } \text{expr3}
  \]
- If EBV of expr1 is true, the conditional expression evaluates to the value of expr2, else it evaluates to the value of expr3.
- Parentheses around if expression (expr1) are required.
- else is always required but it can be just else ().
- Useful when structure of information returned depends on a condition.
- Can be nested and used anywhere a value is expected.

\[
\text{if } ($book/@year < 1980 ) \text{ then } <old>{$x/title}</old> \text{ else } <new>{$x/title}</new>
\]

Conditional Expression

- Used as an alternative way of writing the FLWOR expressions.

<table>
<thead>
<tr>
<th>FLWOR:</th>
<th>for $a in document(&quot;bib.xml&quot;)//article where $a/year &lt; 1996 return $a</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conditional:</td>
<td>for $a in document(&quot;bib.xml&quot;)//article return if ($a/year &lt; 1996) then $a else ()</td>
</tr>
</tbody>
</table>

XQuery Expressions

- Path expressions.
- FLWOR expressions.
- Expressions involving operators and functions.
- Conditional expressions.
- Quantified expressions.
- List constructors.
- Element constructors.
- Expressions that test or modify datatypes

Quantified Expressions

- Syntax:
  \[
  \text{[some | every]} \text{ } \$var \text{ in } \text{expr} \text{ satisfies } \text{test_expr}
  \]
- Quantified expressions evaluate to a boolean value.
- Evaluation:
  - \$var is bound to each of the items in the sequence resulting from expr.
  - For each binding, the test_expr is evaluated.
  - In case of
    - Existential quantification (“some”), if at least one evaluation of test_expr evaluates "true", the entire expression evaluates "true".
    - Universal quantification (“every”), all evaluations of test_expr must result in an EBV of "true" for the expression to return "true".

Quantified Expressions

- Existential Quantification
  - Give me all books where “Sailing” appear at least once in the same paragraph.

```xml
for $b in document("bib.xml")//book
    where some $p in $b//para satisfies(contains($p,"Sailing"))
return $b/title
```

Quantified Expressions

- Universal Quantification
  - Give me all books where “Sailing” appears in every paragraph.

```xml
for $b in document("bib.xml")//book
    where every $p in $b//para satisfies(contains($p,"Sailing"))
return $b/title
```
XQuery Expressions

- Path expressions.
- FLWOR expressions.
- Expressions involving operators and functions.
- Conditional expressions.
- Quantified expressions.
- List constructors.
- Element constructors.
- Expressions that test or modify datatypes

XQuery List Constructors

- A list may be constructed by enclosing zero or more expressions in square brackets, separated by commas.
- For example, \([x, y, z]\) denotes a list containing three members represented by variables.
- \([\ ]\) denotes an empty list.

XQuery Expressions

- Path expressions.
- FLWOR expressions.
- Expressions involving operators and functions.
- Conditional expressions.
- Quantified expressions.
- List constructors.
- Element constructors.
- Expressions that test or modify datatypes

XQuery Operators on Data Types

- INSTANCEOF returns True if its first operand is an instance of the type named in its second operand.
  \(x \text{ INSTANCEOF integer}\)
- CAST is used to convert a value from one datatype to another.
  \(\text{CAST AS integer}(x \text{ DIV } y)\)
- TREAT causes the query processor to treat an expression as though its datatype were a subtype of its static type.
  \(\text{TREAT AS Cat($mypet)}\)

Library Modules

- Separate XQuery documents that contain function definitions.
- Why?
  - Reusing functions among many queries.
  - Defining standard libraries that can be distributed to a variety of query users.
  - Organizing and reducing the size of query modules.
Library Modules

Global Variables

- Declared and bound in the query prolog and used through the query.
- Can be
  - Referenced in a function that is declared in that module.
  - Referenced in other modules that import the module.

```
declare variable $maxNumItems := 3;
declare variable $ordDoc := doc("ord.xml");
for $item in $ordDoc//item[position() <= $maxNumItems]
return $item
```

Resources

- W3C XQuery
  [http://www.w3.org/TR/xquery.html](http://www.w3.org/TR/xquery.html)
- W3C XML Query Use Cases
  [http://www.w3.org/TR/xmlquery-use-cases.html](http://www.w3.org/TR/xmlquery-use-cases.html)
- W3C XML Query Requirements
  [http://www.w3.org/TR/xmlquery-req.html](http://www.w3.org/TR/xmlquery-req.html)
- W3C XML Query Data Model
  [http://www.w3.org/TR/query-datamodel.html](http://www.w3.org/TR/query-datamodel.html)
- W3C XML Query Algebra
  [http://www.w3.org/TR/query-algebra.html](http://www.w3.org/TR/query-algebra.html)

Resources (Books)


Resources (Implementations)

- Saxon
- Galax
- X-Hive
- IPSI-XQ
  [http://www.ipsi.fraunhofer.de/oasys/projects/ipsi-xq/index_e.html](http://www.ipsi.fraunhofer.de/oasys/projects/ipsi-xq/index_e.html)
- MonetDB/XQuery
  [http://monetdb.cwi.nl/XQuery/](http://monetdb.cwi.nl/XQuery/)

To play around a bit with XQuery, you can use Exist Demo
[http://demo.exist-db.org/sandbox/sandbox.xql#](http://demo.exist-db.org/sandbox/sandbox.xql#)
Other Resources

- Mailing Lists
  - talk@xquery.com
  - www-ql@w3.org

- Examples
  - http://www.xqueryfunctions.com/