Processing of structured documents

Spring 2003, Part 8
Helena Ahonen-Myka

XML Query language

- W3C 15.11.2002: working drafts
  - A data model (XQuery 1.0 and XPath 2.0)
  - XQuery 1.0: An XML Query Language
- influenced by the work of many research groups and query languages
- goal: a query language that is broadly applicable across all types of XML data sources

2

Usage scenarios

- Human-readable documents
  - perform queries on structured documents and collections of documents, such as technical manuals,
    - to retrieve individual documents,
    - to generate tables of contents,
    - to search for information in structures found within a document, or
    - to generate new documents as the result of a query

3

Usage scenarios

- Data-oriented documents
  - perform queries on the XML representation of database data, object data, or other traditional data sources
    - to extract data from these sources
    - to transform data into new XML representations
    - to integrate data from multiple heterogeneous data sources
  - the XML representation of data sources may be either physical or virtual:
    - data may be physically encoded in XML, or an XML representation of the data may be produced

4

Usage scenarios

- Mixed-model documents
  - perform both document-oriented and data-oriented queries on documents with embedded data, such as catalogs, patient health records, employment records
- Administrative data
  - perform queries on configuration files, user profiles, or administrative logs represented in XML
- Native XML repositories (databases)

5

Usage scenarios

- Filtering streams
  - perform queries on streams of XML data to process the data (logs of email messages, network packets, stock market data, newswire feeds, EDI)
    - to filter and route messages represented in XML
    - to extract data from XML streams
    - to transform data in XML streams
- DOM
  - perform queries on DOM structures to return sets of nodes that meet the specified criteria

6

Usage scenarios

- Multiple syntactic environments
  - queries may be used in many environments
  - a query might be embedded in a URL, an XML page, or a JSP or ASP page
  - represented by a string in a program written in a general-purpose programming language
  - provided as an argument on the command-line or standard input

7

Requirements

- Query language syntax
  - the XML Query Language may have more than one syntax binding
  - one query language syntax must be convenient for humans to read and write
  - one query language syntax must be expressed in XML in a way that reflects the underlying structure of the query
- Declarativity
  - the language must be declarative
  - it must not enforce a particular evaluation strategy

8

Requirements

- Reliance on XML Information Set
  - the XML Query data model relies on information provided by XML Processors and Schema Processors
  - it must ensure that it does not require information that is not made available by such processors
- Datatypes
  - the data model must represent both XML 1.0 character data and the simple and complex types of the XML Schema specification
- Schema availability
  - queries must be possible whether or not a schema is available

9

Requirements: functionality

- **Support operations (selection, projection, aggregation, sorting, etc.) on all data types:**
  - Choose a part of the data based on content or structure
  - Also operations on hierarchy and sequence of document structures
- **Structural preservation and transformation:**
  - Preserve the relative hierarchy and sequence of input document structures in the query results
  - Transform XML structures and create new XML structures
- **Combination and joining:**
  - Combine related information from different parts of a given document or from multiple documents

10

Requirements: functionality

- **References:**
  - Queries must be able to traverse intra- and inter-document references
- **Closure property:**
  - The result of an XML document query is also an XML document (usually not valid but well-formed)
  - The results of a query can be used as input to another query
- **Extensibility:**
  - The query language should support the use of externally defined functions on all datatypes of the data model

11

XQuery

- Design goals:
  - a small, easily implementable language
  - queries are concise and easily understood
  - flexible enough to query a broad spectrum of XML information sources (incl. both databases and documents)
  - a human-readable query syntax
- features borrowed from many languages
  - Quilt, XPath, XQL, XML-QL, SQL, OQL, Lorel, ...

12

XQuery vs. another XML activities

- XQuery 1.0 and XPath 2.0 Data Model
- type system is based on the type system of XML Schema
- path expressions (for navigating in hierarchic documents) = path expressions of XPath 2.0

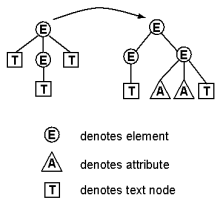
13

XQuery

- A query is represented as an expression
- several kinds of expressions -> several forms
- expressions can be nested with full generality
- the input and output of a query are instances of a data model (XQuery 1.0 and XPath 2.0 Data Model)
  - a fragment of a document or a collection of documents may lack a common root and may be modeled as an ordered forest of nodes

14

An instance of the Data Model - an ordered forest



15

XQuery expressions

- path expressions
- element constructors
- FLWOR ("flower"; for-let-where-orderby-return) expressions
- expressions involving operators and functions
- conditional expressions
- quantified expressions

16

Path expressions

- the result of a path expression is an ordered list of nodes (document order)
  - each node includes its descendant nodes -> the result is an ordered forest
- the top-level nodes in the result are ordered according to their position in the original hierarchy (in top-down, left-right order)
- no duplicate nodes

17

Element constructors

- An element constructor creates an XML element
- consists of a start tag and an end tag, enclosing an optional list of expressions that provide the content of the element
  - the start tag may also specify the values of one of more attributes
- typical use:
  - nested inside another expression that binds variables that are used in the element constructor

18

Example

- Generate an <emp> element containing an "empid" attribute and nested <name> and <job> elements. The values of the attribute and nested elements are specified elsewhere.

```
<emp empid = {$id}>
   <name> {$n} </name>
   <job> {$j} </job>
</emp>
```

19

Element constructors

- In an element constructor, curly braces {} delimit enclosed expressions, distinguishing them from literal text
- enclosed expressions are evaluated and replaced by their value, whereas material outside curly braces is simply treated as literal text
- an enclosed expression may evaluate to any sequence of nodes and/or simple values

20

Computed element constructors

- Generate an element with a computed name, containing nested elements named <description> and <price>

```
element {$tagname}{
   <description> {$d} </description>
   <price> {$p} </price>
}
```

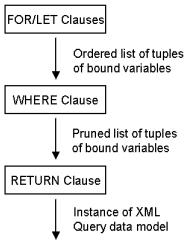
21

FLWOR expressions

- Constructed from **for, let, where, order by**, and **return** clauses
- ~SQL select-from-where
- clauses must appear in a specific order
  - 1. for/**let**, 2. where, 3. order by, 4. return
- a FLWOR expression binds values to one or more variables and then uses these variables to construct a result (in general, an ordered forest of nodes)

22

A flow of data in a FLWOR expression



23

Examples

- Assume: a document named "bib.xml"
- contains a list of <book> elements
- each <book> contains a <title> element, one or more <author> elements, a <publisher> element, a <year> attribute, and a <price> element

24

List the titles of books published by Addison Wesley after 1991

```
<bib>{
   for $b in document("http://www.bn.com/bib.xml")/bib/book
   where $b/publisher = "Addison Wesley"
           and $b/@year > 1991
   return
      <book year= "{$b/@year}">
         {$b/title}
      </book>
}</bib>
```

25

Result could be...

```
<bib>
<book year="1994">
  <title>TCP/IP Illustrated</title>
</book>
<book year="1992">
   <title>Advanced Programming in the Unix environment</title>
</book>
</bib>
```

26

for clauses

- A **for** clause introduces one or more variables, associating each variable with an expression that returns a list of nodes (e.g. a path expression)
- the result of a **for** clause is a list of tuples, each of which contains a binding for each of the variables
- each variable in a **for** clause can be thought of as iterating over the nodes returned by its respective expression

27

let clauses

- A **let** clause is also used to bind one or more variables to one or more expressions
- a **let** clause binds each variable to the value of its respective expression without iteration
- results in a single binding for each variable
- Compare:
  - for $x in /library/book -> many bindings (books)
  - let $x := /library/book -> single binding (a list of books)

28

for/let clauses

- A FLWOR expression may contain several **for** and **let** clauses
  - each of these clauses may contain references to variables bound in previous clauses
- the result of the **for/let** sequence:
  - an ordered list of tuples of bound variables (="tuple stream")
- the number of tuples generated by the **for/let** sequence:
  - the product of the cardinalities of the node-lists returned by the expressions in the **for** clauses

29

for/let clauses

```
let $s := (<one/>, <two/>, <three/>)
return <out>{$s}</out>
```

Result:

```
<out>
  <one/>
  <two/>
  <three/>
</out>
```

30

for/let clauses

```
for $s in (<one/>, <two/>, <three/>)
return <out>{$s}</out>
```

Result:

```
<out><one/></out>
<out><two/></out>
<out><three/></out>
```

31

for/let clauses

```
for $i in (1,2), $j in (3,4)
return  <tuple><i>{ $i }</i> <j>{ $j }</j></tuple>
```

Result:

```
<tuple><i>1</i><j>3</j></tuple>
<tuple><i>1</i><j>4</j></tuple>
<tuple><i>2</i><j>3</j></tuple>
<tuple><i>2</i><j>4</j></tuple>
```

32

where clause

- Each of the binding tuples generated by the **for** and **let** clauses can be filtered by an optional **where** clause
- only those tuples for which the condition in the **where** clause is true are used to invoke the **return** clause
- the **where** clause may contain several predicates connected by **and, or,** and **not**
  - predicates usually contain references to the bound variables

33

where clause

- Variables bound by a **for** clause represent a single node
  - -> scalar predicates, e.g. $p/color = "Red"
- Variables bound by a **let** clause may represent lists of nodes
  - -> list-oriented predicates, e.g. avg($p/price) > 100

34

order by clause

- an **order by** clause determines the order of the tuples in the tuple stream
- the order determines the order in which the **return** clause is evaluated
- if no **order by** clause is given, the order of the tuple stream is determined by the orderings of the sequences returned by the expressions in the **for** clauses

35

Make an alphabetic list of authors, within each author, make a list of books in alphabetic order

```
for $a in distinct-values(document("...bib.xml")//author)
order by $a
return
   <author>
        <name> {$a/text()} </name>
        <books>
          { for $b in document("...bib.xml")//book[author = $a]
            order by $b/title
            return $b/title }
        </books>
   </author>
```

36

return clause

- The **return** clause generates the output of the FLWOR expression
  - a node, an ordered forest of nodes, primitive value
- is executed on each tuple of the tuple stream
- contains an expression that often contains element constuctors, references to bound variables, and nested subexpressions

37

For each book at bib.xml and reviews.xml, list the title of the book and its price from each source

```
<books-with-prices>
 { for $b in document("…/bib.xml")//book,
       $a in document ("…/reviews.xml")//entry
   where $b/title = $a/title
   return
         <book-with-prices>
                { $b/title }
                <price-amazon>{ $a/price/text() }</price-amazon>
                <price-bn> { $b/price/text() }</price-bn>
         </book-with-prices>
 }
</books-with-prices>
```

38

Result:

```
<books-with-prices>
        <book-with-prices>
                <title>TCP/IP Illustrated </title>
                <price-amazon>65.95 </price-amazon>
                <price-bn>65.95 </price_bn>
        </book-with-prices> …
        <book-with-prices>
                <title>Data on the Web </title>
                <price-amazon>34.95</price-amazon>
                <price-bn>39.95</price_bn>
        </book-with-prices>
</books-with-prices>
```

39

Built-in functions

- A core library of built-in functions
- **document:** returns the root node of a named document
- all functions of the XPath core function library
- all the aggregation functions of SQL
  - avg, sum, count, max, min…
- **distinct-values**: eliminates duplicates from a list
- **empty:** returns **true** if and only if its argument is an empty list

40

List each publisher and the average price of its books

```
for $p in distinct-values(document("…bib.xml")//publisher)
let $a := avg(document("…bib.xml")//book[publisher = $p]/price)
return
        <publisher>{
                <name> {$p/text()} </name> ,
                <avgprice> {$a} </avgprice>
        }</publisher>
```

41

List the publishers who have published more than 100 books

```
<big_publishers>{
        for $p in distinct-values(document("…bib.xml")//publisher)
        let $b := document("bib.xml")//book[publisher = $p]
        where count($b) > 100
        return $p
}</big_publishers>
```

42

Operators in expressions

- Expressions can be constructed using infix and prefix operators; nested expressions inside parenthesis can serve as operands
- arithmetic and logical operators; collection operators (union, intersect, except)

43

Conditional expressions

- if-then-else
- conditional expressions can be nested and used wherever a value is expected
- assume: a library has many holdings (element <holding> with a "type" attribute that identifies its type, e.g. book or journal). All holdings have a title and other nested elements that depend on the type of holding

44

Make a list of holdings, ordered by title. For journals, include the editor, and for all others, include the author

```
for $h in //holding
order by $h/title
return
        <holding>
            {$h/title}
            {if ($h/@type = "Journal")
            then $h/editor
            else $h/author}
        </holding>
```

45

Quantifiers

- It may be necessary to test for existence of some element that satisfies a condition, or to determine whether all elements in some collection satisfy a condition
- -> existential and universal quantifiers

46

Find titles of books in which both sailing and windsurfing are mentioned in the same paragraph

```
for $b in  //book
where some $p in $b//para satisfies
        contains($p/text(), "sailing")
        and contains($p/text(), "windsurfing")
return $b/title
```

47

Find titles of books in which sailing is mentioned in every paragraph

```
for $b in //book
where every $p in $b//para satisfies
      contains($p/text(), "sailing")
return $b/title
```

48