XML Technologies

Doc. RNDr. Irena Holubova, Ph.D.

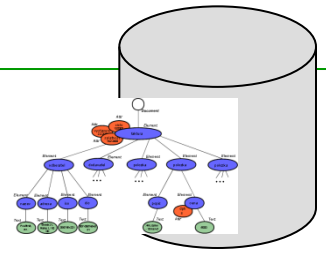
holubova@ksi.mff.cuni.cz

Web pages:
MFF: http://www.ksi.mff.cuni.cz/~holubova/NPRG036/
FEL: http://www.ksi.mff.cuni.cz/~holubova/A7B36XML/

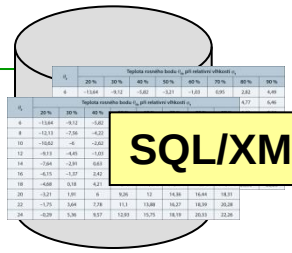
Outline

- Introduction to XML format, overview of XML technologies
- DTD
- XML data models
- Interfaces for XML data
- XPath
- XSLT
- XQuery, XQuery Update
- XML schema languages
- SQL/XML
- An overview of standard XML formats
- XML data persistence

XML Technologies



native
XML DB



relational
XML DB

SQL/XML

XHTML
XForms
UDDI
MathML
SOAP
SVG
OpenTravel
DocBook
OpenDocument
UBL

exploitation

XSLT

persistence

transformation

parsing

validation

SAX
DOM
StAX
LINQ

XML
data

DTD
XML Schema
RELAX NG
Schematron

querying

XPath
XQuery
XQuery Update

XML Query Languages

- Aims: querying, views, updates
- Since 1998 XML-QL, XQL, …
  - W3C specifications: XSLT 1.0, 2.0, 3.0, … XPath 1.0, 2.0, 3.0, XQuery 1.0, 3.0
    - XPath (1.0) – selecting of parts of the tree
    - XSLT – data transformations
    - XQuery – XML querying (user-oriented syntax)

XQuery

- Currently:
  - XQuery 1.0 – recommendation
    - http://www.w3.org/TR/xquery/
  - XQuery 3.0 – recommendation since 08 April 2014
    - http://www.w3.org/TR/xquery-30/
- The same data model as XPath 2.0
  - XPath 2.0 $\subset$ XQuery
  - Each XPath 2.0 query is also a query in XQuery
- XPath 1.0 and 2.0 (and hence XQuery 1.0) are not fully compatible
  - Different subsets of XML Infoset

XQuery

- Higher expressive power than XPath 2.0, XQL, etc.
- Clear semantics (XQuery Core model)
  - See later
- Exploitation of XML Schema
  - Description of structure
  - Data types
- Compatibility of data model with XML Infoset
- W3C: XML Query Use Cases
  - http://www.w3.org/TR/xquery-use-cases/

XQuery Example – Data

```
<?xml version="1.0"?>
<catalogue>
  <book year="2002">
    <title>The Naked Chef</title>
    <author>Jamie Oliver</author>
    <isbn>0-7868-6617-9</isbn>
    <category>cook book</category>
    <pages>250</pages>
  </book>
  <book year="2007">
    <title>Blue, not Green Planet</title>
    <subtitle>What is Endangered? Climate or Freedom?</subtitle>
    <author>Václav Klaus</author>
    <isbn>978-80-7363-152-9</isbn>
    <category>society</category>
    <category>ecology</category>
    <pages>176</pages>
  </book>
  …
```

```
  <book year="2006">
    <title>Jamie po italsku</title>
    <original>
      <title>Jamie's Italy</title>
      <translation>Vladimir Fuksa</translation>
    </original>
    <author>Jamie Oliver</author>
    <isbn>80-89189-18-0</isbn>
    <category>cook book</category>
    <pages>319</pages>
  </book>
  <book year="2007">
    <title>Nepříjemná pravda</title>
    <subtitle>Naše planeta v ohrožení - globální oteplování a co
    s ním můžeme udělat</subtitle>
    <original>
      <title>An incovenient Truth</title>
      <translation>Jitka Fialová</translation>
    </original>
    <author>Al Gore</author>
    <isbn>978-80-7203-868-8</isbn>
    <category>ecology</category>
    <pages>329</pages>
  </book>
</catalogue>
```

XQuery

- XQuery is a functional language
  - Query is an expression
  - Expressions can be combined
- XQuery query:
  - (Optional) declaration of namespaces
  - (Optional) definition of functions
  - Query itself

XQuery

- XPath expressions
  - //catalogue/book[author="Jamie Oliver"]
- Constructor
  - element book {element author}
- FLWOR expression
  - FOR ... LET ... WHERE ... ORDER BY ... RETURN
- Conditional expression
  - IF ... THEN ... ELSE

XQuery

- Quantifiers
  - EVERY $var IN expr SATISFIES expr
  - SOME $var IN expr SATISFIES expr
- Type operator
  - TYPESWITCH typeexpr CASE ... DEFAULT
- Operators and functions
  - x + y, z = x, func(x,y,z)
- Variables and constants
  - $x, "Obama", 256
- Comparison

XQuery – Constructors

- Direct constructors

```
<html>
  <body>
    <h1>Listing from doc("catalogue.xml")//book</h1>
    <h2>title: {doc("catalogue.xml")//book[1]/title}</h2>
    <h3>subtitle: {doc("catalogue.xml")//book[1]/subtitle}</h3>
    <h2>
      title: {fn:data(doc("catalogue.xml")//book[2]/title)}
    </h2>
    <h3>
      subtitle: {fn:data(doc("catalogue.xml")//book[2]/subtitle)}
    </h3>
  </body>
</html>
```

XQuery – Constructors

- Computed constructors

```
element html {
  element body {
    element h1 {"Listing from doc('catalogue.xml')//book"},
    element h2 {
      text{"title: "},
      doc("catalogue.xml")//book[1]/title
    }
    ...
  }
}
```

```
<html>
  <body>
    <h1>Listing from doc('catalogue.xml')//book</h1>
    <h2>title: <title>The Naked Chef</title></h2>
    <h3>subtitle: </h3>
    <h2>title: Blue, not Green Planet</h2>
    <h3>subtitle: What is Endangered? Climate or Freedom?</h3>
  </body>
</html>
```

XQuery – FLWOR

- Basic XQuery construction
  - Like SELECT-FROM-WHERE-… in SQL
- Clause for (for $var in expr) (FLWOR)
  - Evaluates expression expr whose result is a sequence
    - see XPath 2.0 data model
  - Iteratively assigned to variable $var
- Clause let (let $var := expr) (FLWOR)
  - Evaluates expression expr and assigns the result to variable $var
- Clause where (where expr) (FLWOR)
  - Filters sequences from clause for

XQuery – FLWOR

- Clause order by (order by expr) (FLWOR)
  - Sorts sequences filtered by clause where according to the given criterion
- Clause return (return expr) (FLWOR)
  - Concluding clause which constructs the result of the query from the selected, filtered and sorted sequences

XQuery – FLWOR

- For each book with more than 300 pages return the title and author sorted by year of edition

```
for       $book in doc("catalogue.xml")//book
where     $book/pages > 300
order by  $book/@year
return
  <book>
    {$book/title}
    {$book/author}
  </book>
```

XQuery – FLWOR

- For each book having an original title return the translated and original title and author

```
for       $book in doc("catalogue.xml")//book
where     $book/original
return
  <book>
    {$book/title}
    <originaltitle>
      {data($book/original/title)}
    </originaltitle>
    {$book/author}
  </book>
```

XQuery – FLWOR

- FLWOR expressions enable to transform the original structure of the data
  - e.g., transformation to XHTML and other formats
- Example:
  - XHTML table of books
  - Swapping of parent/child elements
    - book / author → author / list of books
  - Grouping
    - Grouping of books according to categories
  - Joining of data from different resources
    - We extend the books in the catalogue with reviews from another resource

XQuery – FLWOR

- Return an HTML table of cook books with columns title, author and (number of) pages

```
<table>
  <tr><th>title</th><th>author</th><th>pages</th></tr>
  {
    for       $book in doc("catalogue.xml")//book
    where     $book/category = "cook book"
    return
      <tr>
        <td>{data($book/title)}</td>
        <td>{data($book/author)}</td>
        <td>{data($book/pages)}</td>
      </tr>
  }
</table>
```

XQuery – FLWOR

- For each author return the list of their books

```
<authors>
{
  for $name in distinct-values(doc("catalogue.xml")//author)
  return
    <author>
      <name>{data($name)}</name>
      {
        for    $book in doc("catalogue.xml")//book
        where  $book/author = $name
        return
          <book>{$book/title}</book>
      }
    </author>
}
</authors>
```

XQuery – FLWOR

- Group the books into categories. For each category create a separate element with its name in an attribute.

```
<list-of-categories>
{
  for    $category in distinct-values(doc("catalogue.xml")//category)
  return
    <category name="{data($category)}">
      {
        for    $book in doc("catalogue.xml")//book
        where  $book/category = $category
        return
          <book>{$book/title}</book>
      }
    </category>
}
</list-of-categories>
```

XQuery – FLWOR

- For each book add a list of sold pieces from document sale.xml (inner join)

```
<books>
{
  for     $book in doc("catalogue.xml")//book,
          $sale in doc("sale.xml")//book
  where   $book/ISBN = $sale/ISBN
  return
    <book>
      {$book/title}
      {$book/author}
      {$sale/status}
    </book>
}
</books>
```

XQuery – FLWOR

- For each book add a list of reviews from document review.xml (outer join)

```
<books>{
  for     $book in doc("catalogue.xml")//book
  return
    <book>
      {$book/title}
      {$book/author}
      {
        for $review in doc("review.xml")//review
        where $review/ISBN = $book/ISBN
        return $review/text
      }
    </book>
}</books>
```

XQuery – Conditions

- Clause if (if expr)
  - Evaluates expression expr whose value is true/false
- Clause then (then expr)
- Clause else (else expr)

XQuery – Conditions

- For each book return its title and the first category. If it has multiple categories, add element <more-categories/>.

```
for     $book in doc("catalogue.xml")//book
return
  <book>
    {$book/title}
    {$book/category[1]}
    {
      if ( count($book/category) > 1 )
      then <more-categories/>
      else ()
    }
  </book>
```

XQuery – Quantifiers

- Clause every/some (every/some $var in expr)
  - Evaluates expression expr and requires that each/some of the sequences in its result satisfies the condition
- Clause satisfies (satisfies expr)
  - expr is the condition of the quantifier

XQuery – Quantifiers

- Return the authors from document authors.xml, who write only books which are translated (i.e. have an original)

```
for $author in distinct-values(doc("authors.xml")//author)
where every $author-book in
        for    $book in doc("catalogue.xml")//book
        where  $book[author = $author/name]
        return $book
      satisfies
        $author-book/original
return $author
```

XQuery – Functions

- Built-in functions
  - distinct-values, empty, name, ...
  - Aggregation functions max, min, avg, count, ...
  - Other: string, numeric and other data types
    - A huge number
  - Namespace fn
    - URI: http://www.w3.org/2005/xpath-functions
- User-defined functions
  - Defined using the XQuery syntax
  - Typed, recursive, …
  - Support for libraries

XQuery – Built-in Functions

- We already know some:
  - Document node related to the specified uri:

    ```
    fn:doc($uri as xs:string?) as document-node()?
    ```

  - Sequence of atomic values of the given sequence of items

    ```
    fn:data($arg as item()*) as xs:anyAtomicType*
    ```

  - Number of items in a sequence

    ```
    fn:count($arg as item()*) as xs:integer
    ```

  - Removing of duplicities (only for atomic values)

    ```
    fn:distinct-values($arg as xs:anyAtomicType*) as xs:anyAtomicType*
    ```

XQuery – User-defined Functions

- Syntax

```
declare function name(parameters) as type
```

where:

- **`name`** = name of the function
- **`parameters`** = list of parameters
  - Typed/untyped
- **`type`** = type of return value

XQuery – User-defined Functions

- Function returning names of books of a given author (having the given name and surname) sorted according to name of book. Each book can have multiple authors.

```
module namespace ksi="http://ksi.mff.cuni.cz/xquery/books";

declare function ksi:books-author($name, $surname) as element()*
{
  for $book in doc("catalogue.xml")//book
  where some      $author in $book/author
        satisfies $author/surname = $surname and
                  $author/name = $name
  order by $book/name
  return $book/name
};
```

XQuery – User-defined Functions

- Import of a library with assigning of a particular namespace prefix

```
import module namespace ksi =
        "http://ksi.mff.cuni.cz/xquery/books"
    at "file://home/novak/xquery/lib/books.xq";

<author>
  <name>Barack</name>
  <surname>Obama</surname>
  <publications>
    {ksi:books-author("Barack", "Obama")}
  </publications>
</author>
```

XQuery – User-defined Functions

- Function which recursively traverses the structure of a book and returns the number of subsections of a given book or section

```
module namespace ksi="http://ksi.mff.cuni.cz/xquery/books";

declare function ksi:subsections($book-or-section) as element()*
{
  for $subsection in $book-or-section/section
  return
    <section>
      {$subsection/name}
      <number>{fn:count($subsection/section)}</number>
      <subsections>{ksi:subsections($subsection)}</subsections>
    </section>
};
```

XQuery – User-defined Functions

- Import of a library with assigning of a particular namespace prefix

```
import module namespace ksi =
        "http://ksi.mff.cuni.cz/xquery/books"
    at "file://home/novak/xquery/lib/books.xq";

for $book in fn:doc("catalogue.xml")//book
return
    <book>
      {$book/name}
      <number>{fn:count($book/section)}</number>
      {ksi:subsections($book)}
    </book>
```

XQuery – Comparison
Value

- Operators
  - lt, gt, le, ge, eq, ne meaning "less than", "greater than", "less or equal", "greater or equal", "equal", "non equal"
- Algorithm:
  - Atomization
    - Atomic value
  - Implicit conversion to the same data type
  - Comparison of the operands

XQuery – Comparison
Value

- Untyped operands are implicitly converted to strings
- If any of the operands is converted to an empty sequence, the result is an empty sequence
- If any of the operands is converted to a sequence longer than 1, error

XQuery – Comparison
Value

- 1 le 2 => true
- (1) le (2) => true
- (1) le (2,1) => error
- (1) le () => ()
- <a>5 eq 5 => true
- $book/author eq "Jamie Oliver"
  => true if $book has exactly one subelement author with value "Jamie Oliver"

XQuery – Comparison
General

- Operators
  - <, >, <=, >=, =, !=
- Also for sequences
- Algorithm:
  - Atomization
    - The result are sequences of atomic values
  - Searching for an item from left and right operands which evaluate to true
    - If there exists such pair, true
    - Otherwise, false

XQuery – Comparison
General

- When searching a pair of items, again a conversion
  - Both untyped – conversion to xs:string
  - One untyped, other numeric – conversion to xs:double
  - One untyped, other typed, but other than numeric/string – conversion to the particular type

XQuery – Comparison
General

- 1 < 2 => true
- (1) < (2) => true
- (1) < (2,1) => **true**
- (1) < () => false
- (0,1) = (1,2) => true
- (0,1) != (1,2) => true
- $book/author = "Jamie Oliver"
  => true if $book has at least one subelement author with value "Jamie Oliver"

XQuery – Comparison
Node

- Operators is, << and >>
- Algorithm:
  - Evaluation of operands
  - If one of the operands is an empty sequence, the result is an empty sequence
  - If any of the operands returns a sequence longer than 1, error
  - Otherwise:
    - is returns true if both operands are nodes with the same identity
    - << returns true if the left operand precedes the right operand (in the document order)
    - >> returns true if the left operand follows the right operand (in the document order)

XQuery – Comparison
Node

```
/catalogue/book[isbn="0-7868-6617-9"]
                    is
/catalogue/book[title="Jamie Oliver"]
```

true if both the operands return the same node

XQuery – Comparison
Node

- Consider a conference program. Return the lectures which take place on the first day before the first coffee break.

```
let $day-program          := doc("program.xml")/program/day[1]
let $first-coffee-break := $day-program/break[@type="coffee"][1]
for $lecture              in $day-program/lecture
where  $lecture << $first-coffee-break
return $lecture
```

XQuery – Integrity Constraints

- XML Schema can be used as a tool for specification of various integrity constraints (ICs)
  - e.g. cardinalities, keys, data types, …
  - Version 1.1: element assert (using XPath)
- It does not provide a robust tool for specification of more complex ICs
  - e.g. "If an author does not write in Czech, each of their books must contain also a title in their original language and the name of translator."

XQuery – Integrity Constraints

- XQuery is a sufficiently powerful language for expressing ICs
  - Like the CHECK constraint in SQL
- The IC is expressed as a query which returns a warning if necessary, e.g.
  - If the data are OK
    - **<ok no="number of IC"/>**
  - If an IC is violated
    - **<error no="number of IC">warning text</error>**

XQuery – Integrity Constraints

- If an author does not write in Czech, each of their books must contain also a title in their original language and the name of translator (in element original).

```
let $authors := doc("authors.xml")//author[language != "cs"]
return
    if ( every $author in $authors
         satisfies every $author-book in
                        for    $book in doc("catalogue.xml")//book
                        where  $book[author = $author/name]
                        return $book
                   satisfies
                          $author-book/original )
    then <ok no="1001" />
    else <error no="1001" />
```

XQuery – Integrity Constraints

```
let $authors := doc("authors.xml")//author[language != "cs"]
let $wrong_authors
             := for $author in $authors
                where some $author-book in
                          for    $book in doc("catalogue.xml")//book
                          where  $book[author = $author/name]
                          return $book
                       satisfies
                           count($author-book/original) = 0
                return $author
return
  if ( count($wrong_authors) > 0)
  then for $author in $wrong_authors
       return <error no="1001">
                 Author {data($author)} has a book without the
                 original title and name of translator!
              </error>
  else <ok no="1001" />
```

XQuery – Integrity Constraints

```
let $wrong_authors := (
  for $author in doc("authors.xml")//author[language != "cs"]
  let $wrong_books :=
    for    $book in doc("catalogue.xml")//book
    where  $book[author=$author/name] and count($book/original)=0
    return $book/name
  where count($wrong_books) > 0
  return <author>{$author/name}
         <books>{$wrong_books}</books></author> )
return
  if ( count($wrong_authors) > 0 )
  then <error no="1001"> {
           for $author in $wrong_authors
           return <suberror>Author {$author/name} does not have
                the original title and translator for books
                { for $name in $author/books/name
                  return $name }.</suberror>
           } </error>
  else <ok no="1001" />
```

XQuery – Integrity Constraints

- Refined warning

```
<suberror>
  Author {$author/name} does not have
  the original title and translator for books
  {
    for $name in $author/books/name[position()<last()]
    return $name
  } and {
    $author/books/name[last()]
  }.
</suberror>
```

XQuery – Support for Schemas

- Support for schemas is an important extension of query languages
  - XQuery must be able to work with documents without a schema
  - XQuery must exploit the schema if it exists
  - The implementation can allow static typing and detect and report type errors
- The type system is based on XML Schema

XQuery – Support for Schemas

```
typeswitch($customer/billing-address)
   case $a as element(*, USAddress) return $a/state
   case $a as element(*, CanadaAddress) return $a/province
   case $a as element(*, JapanAddress) return $a/prefecture
   default return "unknown"
```

```
5       instance of xs:decimal
(5, 6)  instance of xs:integer+
.       instance of element()
```

XQuery Semantics

- XQuery contains a huge amount of redundancies
- XQuery Core defines a syntactic subset of XQuery with the same expressive power as XQuery, but without duplicities
  - The definition involves also rules for re-writing of queries into XQuery Core
- XQuery Core is useful mainly from the theoretical point of view
  - The queries are long and complex

http://www.w3.org/TR/xquery-semantics/

XQuery Core Example

```
for $k in /books/book,
    $r in /reviews/book
where $k/name = $r/name
return
  <book>
    { $k/name, $k/author, $r/content }
  </book>
```

```
for $b in (
  for $dot in $root return
    for $dot in $dot/child::books
      return $dot/child::book
) return
    for $r in (
      for $dot in $root return
        for $dot in $dot/child::reviews
          return $dot/child::book
    ) return
      if ( not( empty(
            for $v1 in (
              for $dot in $b return $dot/child::name
            ) return
              for $v2 in (
                for $dot in $r return $dot/child::name
              ) return
                if (eq($v1,$v2)) then $v1 else ()
          ) ) )
      then (
        element book {
          for $dot in $b return $dot/child::name,
            for $dot in $b return $dot/child::author,
              for $dot in $r return $dot/child::content
        } )
      else ()
```