XQuery – Continued.

csc343, Introduction to Databases
Nosayba El-Sayed

Slides from Diane Horton, with material from Ryan Johnson, Manos Papagelis, Jeff Ullman, Ramona Truta, and Renée Miller
Fall 2015
Announcements

• Welcome back from the Fall “Break”
• Q: How are prep exercises graded (7%)?
  • We will consider the best 7 prep marks you get, out of all preps in the course (~10)

CAREER MENTORSHIP PROGRAM

With a degree from a great department and the skills required for an in-demand field, you are ready to work... But have you thought about your career? Computer Science’s Career Mentorship Program will connect you with a successful alumnus/a who wants to help you chart your path to a rewarding and highly successful career. The program runs January to April 2016 with a launch event on Monday, January 18, 2016, where you will meet your mentor and other alumni mentors.

Apply by Friday, November 20
uoft.me/CS-Mentorship2015

Note: You must be available to attend the preparation session on Thursday, December 10, 2015 (10 am to 12 PM) in order to qualify. Contact ugliaison@cs.toronto.edu for more information.
XQuery

• FLWOR Example:

```
let $d := fn:doc("bank.xml")
for $tfq in $d//TFQuestion
where $tfq/@answer="True"
order by $qid
return $tfq/question
```

• Example:

```
<title>Facts about Canada</title>
<truth>
{ let $d := fn:doc("bank.xml")
  return $d//tf-question[@solution="true"]/question
}
</truth>
```
Generous comparison

• If A and B are sequences, \( A = B \) means
  \( \exists \ x \in A, \ y \in B \) such that \( x = y \).

• Examples:
  • \((1,2) = (2,3)\) is true.

  • Given that a “race” element contains multiple “results”,
  this path expression:
  \( \text{fn:doc("races.xml")//race[result < 3.50] } \)
  yields races that include \textit{any} result less than 3.50.
Strict comparison

• Alternative: The comparison operators
  \[ \text{eq, ne, lt, le, gt, ge} \]

  succeed only if both sequences have length one.

• Example:
  \[
  \text{fn:doc("races.xml")}
  \text{//race[sponsor eq "HarryRosen"]}
  \]

  is true if the LHS yields a sequence of length one that is "HarryRosen".
Eliminating duplicates

- Apply function `distinct-values` to a sequence.
- Subtlety:
  - It strips tags away from elements and compares the string values.
  - But it doesn’t restore the tags in the result.
- Example:
  ```
  let $d := fn:doc("races.xml")
  return distinct-values($d//result)
  ```
More kinds of expressions
Branching expressions

- Form: `if («E1») then «E2» else «E3»`
- All three parts are required.
- Value of the if expression is
  - `E2` if the EBV of `E1` is true, and
  - `E3` if the EBV of `E1` is false.
    (EBV = Effective Boolean Value)
- Example:
  
  `if ($q/@solution="True") then $q/question else ()`
Any type can be treated as boolean

• Like many languages, we can treat anything as boolean.
• The **effective boolean value** (EBV) of an expression is:
  • the value of the expression, if it is already of type boolean
  • otherwise it is
    • FALSE if the expression evaluates to 0, "", or ()
    • TRUE if not.
• Example:

```xml
let $d := fn:doc("races.xml")
return
  if ($d//result[@who="r1"]) then <yay/>
  else <nay/>
```
Quantifier expressions

• Form: some «variable» in «E1» satisfies «E2»

• Meaning
  • Evaluate E1, yielding a sequence.
  • Let the variable be each item in the sequence, and evaluate E2 for each.
  • The value of the whole expression is true if E2 has EBV true at least once.

• Form: every «variable» in «E1» satisfies «E2»

• Meaning is analogous.
Comparisons based on document order

- **Form:** «E1» «E2» and «E1» «E2»
- **Meaning:** comes before (or after) in the document.
- **Example:**
  ```
  let $d := fn:doc("races.xml")
  return
  $d//race[@name="WaterfrontMarathon"]
  «
  $d//race[@name="HarryRosen"]
  »
  ```

   Output, given our “races.xml” file:
   true()
Set operators

- **Form:**
  
  «E₁» union «E₂»
  
  «E₁» intersect «E₂»
  
  «E₁» except «E₂»

- **Meaning is analogous to SQL.**

- **Result does not include duplicates.**

- **Result appears in document order.**

- **All based on node comparisons, not values**
Set operators in XQuery - Examples

let $group1 := (<a/>,<b/>)
let $group2 := (<b/>,<c/>)
return $group1 union $group2
⇒ (<a/>,<b/>,<b/>,<c/>)

let $a := <a/>
let $b := <b/>
let $c := <c/>
return ($a,$b) union ($b,$c)
⇒(<a/>,<b/>,<c/>)

19
for $question$ in (doc("quiz.xml")//Question)
return if ($question/number(@weight) > 2)
   then <important>{data($question/@QID)}</important>
   else <unimportant>{data($question/@QID)}</unimportant>

Result:

<unimportant>N-15</unimportant>,
<unimportant>TF-01</unimportant>,
<important>MC-05</important>,
<unimportant>MC-08</unimportant>
Example

for $question$ in (doc("quiz.xml")//Question)
return if ($question[data(@weight) = "3"] or $question[number(@weight) = 1])
    then $question$
else ()

Result:

<Question QID="TF-01" weight="1"/>,
<Question QID="MC-05" weight="3"/>
Example

let $bdoc := doc("bank.xml")
for $mcq in $bdoc//MCQuestion
for $option in $mcq//Option
return
 (: ($mcq/Text, $option/Text) :)
</Option>

Result:
<Option oID="MC-01.a"><Text>What do you promise when you take the oath of citizenship?</Text><Text>To pledge your loyalty to Queen Elizabeth II</Text></Option>,
<Option oID="MC-01.b"><Text>What do you promise when you take the oath of citizenship?</Text><Text>To fulfill the duties of a Canadian</Text></Option>,
<Option oID="MC-01.c"><Text>What do you promise when you take the oath of citizenship?</Text><Text>To pledge your allegiance to the flag</Text></Option>,
<Option oID="MC-01.d"><Text>What do you promise when you take the oath of citizenship?</Text><Text>To pledge your loyalty to Canada from sea to sea</Text></Option>,
...
*Note*

Defining an element that has both a text (PCDATA) part but also subelements:

```xml
<!ELEMENT element (#PCDATA|subelement1|subelement2)*>
```
Summary of some XPath/XQuery Functionalities
Axes: Special Constructs

- Non-element axes
  - attribute: retrieve attributes of the context node
    - => e.g. `<book in-print="true">/attribute::*` returns ‘in-print’
  - namespace: retrieve node namespace(s)
    - => e.g. `<amazon:book-list>/namespace::*` returns ‘amazon’

- Selecting elements vs. text
- Example: `<foo>abc<bar>d</bar></foo>`
  - `foo/child::*` returns child elements only: `<bar>d</bar>`
  - `foo/child::text()` selects text children only: `abc`
  - `foo/child::node()` selects everything: `abc<bar>d</bar>`

- Element positions (1-based, in document order)
  - `elem::position()` returns position of elem w.r.t. its parent
  - `elem::last()` returns the number of nodes in elem
Short Forms

• Make queries more compact, easier to read
• * = all elements of current axis
• . = self::node()
• .. = parent::node()
• elem = child::elem •@ = attribute::
• // = /descendant-or-self::node()/
• [3] = [position()=3]
• [last()] = [position()=last()]
Absolute vs. Relative Paths

- Child of current context node: book/title
- At document root: /book/title
Predicates

• \[\$expr\] applies boolean predicate to a node set
  • Return subset of nodes for which \$expr is true

• Boolean values can be any of
  • Boolean constant: \texttt{true()} or \texttt{false()}
  • Numbers (false if -0, +0, or NaN)
  • Strings (false if zero-length)
  • Result of comparison (=, !=, <, >, etc.) => \texttt{/book-list/book[price < 50]}
  • Node set (true if exists/non-empty) => \texttt{/book-list/book[@special-offer]}
  • Compound expressions => A and B, A or B, not(A)
Nesting Path Steps and Predicates

• Path step: one segment of a path
  • e.g. /book-list/book/author/last-name has 5 path steps

• OK to chain path steps and/or predicates
  • /book-list/book[price < 50][npages > 100][3]
  • Order matters when position() is involved.

• Also OK to mix and match
  • /book-list/book[price < 50]/author[last-name=‘Asimov’]

• Full nesting also works
  • /book-list/book[author[last-name=‘Asimov’]]
  • Like SQL, often possible to simplify nested queries
    => /book-list/book[author/last-name=‘Asimov’]
Parentheses and Union Operator

- Occasionally need parenthesis for grouping
  - Often due to positional predicates: [1], [last()], etc.
  - => elem/preceding-sibling[1] != (elem/preceding-sibling)[1]
  - => //elem[1] != (//elem)[1]

- Union: combine results of 2+ XPath queries
  - Syntax: (a | b | ...)
  - e.g. title and publisher of all books written by Isaac Asimov
    - => //book[author/last-name=‘Asimov’]/(title | publisher)
  - e.g. books whose keyword or title mentions ‘robot’
    - => //book[(keyword | title)[contains(text(), ‘robot’)]]
Standard Functions

• Node-related
  • `count($node-set)` returns the cardinality of `$node-set`
  • `id($idarg)` returns the element having the specified ID (if any)
  • `name($node-set)` returns the tag name of `$node-set[1]`

• Number-related
  • `number($arg?)` convert $arg or . into a number
  • `sum($node-set)` converts the nodes to numbers and sums them
  • `floor`, `round`, and `ceil` all do what you’d expect
Standard Functions

• String manipulation
  • `string($arg?)` converts `$arg` or `. into a string
  • `starts-with($str, $prefix)`
  • `contains($haystack, $needle)`
  • `substring($str, $beg, $len?)` uses 1-based indexing!
  • `normalize-space($arg?)` turns “\n\t ab \n\t cd \n\t” into “ab cd”
  • `string-length($arg?)` and `concat($a,$b, ...)` do what you’d expect
Aaxes

• Other axes include:
  • parent
  • ancestor
  • ancestor-or-self
  • following-sibling
  • preceding-sibling

• See section 2.2 of the documentation for more:
  http://www.w3.org/TR/xpath/#axes