XQuery Query Language
Announcements

• Next week:
  • Fall break => No class on Tuesday
  • Thursday class is held as usual, as well as office hours

• For more midterm questions or remark requests, come to office hours

• For A2 questions, you’re welcome to come to office hours, or use Piazza
Intro

• XQuery extends XPath.
• It uses the same data model.
  • A document is a tree.
  • A query result is a sequence of items from the document.
• XQuery is an expression language.
  • Any XQuery expression can be an argument of any other XQuery expression.
Quick review of XPath

• Strengths
  • Compact syntax
  • Efficient XML tree traversal
  • Predicates filter out nodes we don’t want

• Weaknesses
  • Most joins impossible
  • No means of formatting results
  • No control flow (branching or loops)
  • Little/no ability to manipulate XML
  • No way to specify input!
How XQuery can be used?

- Extracting information from a database for use in a web service or an application integration
- Generating reports on data stored in an XML database
- Transforming XML data to XHTML to be published on the Web
- Searching textual documents on the Web
- Splitting up a large XML document to multiple XML documents
XQuery

- Address most weaknesses with XPath
  - without adding too much complexity
- Primary features
  - access methods (read XML from file, etc.)
  - control flow: if/then/else, iteration
  - variables
  - functions (both user-defined and library flavours)
  - XML transformation: make data presentable
  - sorting, more powerful predicates, set operations...

*Expressiveness: XPath << SQL << XQuery*
Key concepts of XQuery

• Template of sorts: mixed output and logic
  • Statically create the overall structure
  • Embed logic to handle input data

• All expressions return XML
  • Like in RA, outputs of one operation can be input to another
  • Returned value may be text, element, or node set

• “FLWOR” expressions
  • Allows iteration over node sets and other sequences

• Functions
  • Allows logic encapsulation, recursion

**NOTE:** XQuery syntax bleeds back into XPath
FLWOR expressions

- **F(or)**
  - Iterate over each item in a sequence
  - Multiple sequences separated by commas
- **L(et)**
  - Declares a variable and assigns it a value
  - Multiple declarations separated by commas
  - Bound once per iteration of every `for` above it
- **W(here), O(rder by)**
  - Stolen shamelessly from SQL...
- **R(eturn)**
  - The value that should be computed at each iteration
  - FLWOR is an expression, NOT a function call!
  - Overall value is a sequence of “returned” values

\[(\text{for} \mid \text{let})^+ \text{ where? order-by? return}\]
FLWOR expressions

• Example:

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

• The semantics of return is surprising:
  • It does not terminate the FLWOR expression!
  • It specifies the value produced by the current iteration.
  • The sequence of these is the result of the FLWOR expression.
Output behaviour of FLWOR

- In XPath, every node output at most once
  - Predicates just “mark” nodes which “pass”
- In FLWOR, node output with every `return`
  - Every node in a node set bound to the loop variable
  - Emit all which make it past the `where` clause
- Distinction matters for nested loops!
  - Cartesian product: `for $x in //book, $y in //book...`

Notes about the syntax

• Keywords are case-sensitive.
• Variables begin with $.
• Recall rule: \((\text{for} \mid \text{let})^+ \ \text{where}^? \ \text{order-by}^? \ \text{return}\)
• Remember that XQuery is an expression language.

• A FLWOR expression has a subexpressions.
  let $d := \text{fn:doc(“bank.xml”)}
  for $tfq in $d//TFQuestion
  where $tfq/@answer=“True”
  order by $qid
  return $tfq/question

• Any of these could itself be a FLWOR expression or other complex expression.
For vs let

• For is like
  ```python
  for x in [99, 42, 101, 5]
  ```
  • It iterates over the items in a sequence.
  • Each time, the variable gets a new value.

• Let is like
  ```python
  x = [99, 42, 101, 5]
  ```
  • No iteration occurs.
  • x gets one value, which is a sequence.
Sequences in XQuery

- Most expressions return sequences of nodes
  - LET $b = /bib/book $b is a sequence
  - $b/@isbn a sequence
  - $b/price sequence of n prices
  - $b/price * 0.7 sequence of n numbers
  - $b/price * $b/quantity sequence of n x m numbers

- Sequence literals also allowed
  - e.g. (1,2,3), also shortcut for ranges: (1 to 10)
  - empty sequence: ()

- Sequences combine easily, but flatten
  - (1, 2, (3, 4, 5), (), 6, 7) (1, 2, 3, 4, 5, 6, 7)
Examples
Order-by

• Form: `order by «expression»`
• We can optionally specify `ascending` or `descending`.
• The expression is evaluated for each assignment to variables.
• Its value determines placement of the FLWOR expression’s result in the output sequence.
Mixing static output and evaluated expressions

• Lets us construct new XML structures with our code.
• Example:

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

<lies>
  {
    let $d := fn:doc("bank.xml")
    return $d//tf-question[@solution="false"]/question
  }
</lies>
```
What’s evaluated and what’s not?

• The default: don’t evaluate.
  • Example:
    <title>$x$</title>
  • This evaluates to a title element with value “$x$”

• To override the default and force evaluation, surround with braces.
  • Example:
    <title>{{$x$}}</title>
Return has the opposite default

- Return’s default is to evaluate the expression.
  - Example:
    `return $x`
- To override the default and treat the value literally, surround with quotes.
  - Example:
    `return "$x"`
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.
  - 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
  eq  ne  lt  le  gt  ge
succeed only if both sequences have length one.

• Example:

  fn:doc("races.xml")
    //race[sponsor eq "HarryRosen"]

is true if the LHS yields a sequence of length one
that is "HarryRosen".
eq is picky about document location

- eq requires that two elements not only look alike, but have the same location.
- Example (actually returns true!! ??):

```xml
let $d := fn:doc("races.xml")
return
  $d//race[@name="HarryRosen"]
    /result[@who="r1"]

eq
  $d//race[@name="WaterfrontMarathon"]
    /result[@who="r1"]
```
Why??

- In the document tree, an element is really a pointer to a section of the document.
- It’s like `is` vs `==`.
- To compare based only on element values, not their location, use the `data` function.
Comparison of elements and literal values

• Example:
  
  \[
  \text{fn:doc("races.xml")//race[@name="RunForTheCure"]}
  \text{/result[@who="r12"]}
  \]
  
  \[< 3.0\]

  • result elements are whole elements, such as
    \[
    \text{<result who = "r12">2.50</result>}
    \]

  • But when making the comparison, the value of the element is used, if it is atomic.
Comparison of nodes

• Compare identity of two nodes based on position in document

• Operators:
  • \texttt{is}
  • \texttt{<<} (for preceding)
  • \texttt{>>} (for following)

• “is” only true if both sides are actually the same node
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"]
More kinds of expressions
Branching expressions

• Form:  \texttt{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.
• Example:

\begin{verbatim}
for $q$ in //question
  if ($q/@solution="True")
    then <trueQuestion>{$q/question}</trueQuestion>
  else ()
\end{verbatim}
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 boolean type
  • otherwise it is
    • FALSE if the expression evaluates to 0, “”, or ()
    • TRUE if not.

• Example:
  ```
  let $d := fn:doc("races.xml")
  return
    if ($d//result[@who="r1"])
      then <yay/>
    else <nay/>
  ```
Boolean operators and EBVs

• Because of EBVs, we can apply boolean operators
  and or not
to any expressions.

• Examples:
  not(3 eq 5 or 0)

• Also: true() and false() are functions that return
  the values TRUE and FALSE respectively.
Nesting expressions arbitrarily

• Remember that XQuery is an expression language.
  • any of these subexpressions can be an if, FLWOR, or any other complex expression.
  • The if itself can go anywhere an expression can.
Quantifier expressions

- **Form:** \texttt{some} «variable» in «E1» satisfies «E2»
- **Meaning**
  - Evaluate \texttt{E1}, yielding a sequence.
  - Let the variable be each item in the sequence, and evaluate \texttt{E2} for each.
  - The value of the whole expression is true if \texttt{E2} has EBV true at least once.
  - Same as ANY/SOME in SQL.
- **Form:** \texttt{every} «variable» in «E1» satisfies «E2»
- **Meaning** is analogous to ALL in SQL.
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!

  ⇒ Good for ints, strings; useless if you want to compare the actual elements

  ⇒ distinct-values((1, 2, 3, 1, 2)) = (1, 2, 3)

• Example:

```xml
let $d := fn:doc("races.xml")
return distinct-values($d//result)
```
Set operators

- Xpath defines only union ("|")
- Xquery adds union, intersect, except operators
- Form:
  
  «E1» union «E2»
  «E1» intersect «E2»
  «E1» except «E2»

- Meaning is analogous to SQL.
- Result does not include duplicates.
  - All based on node comparisons, not values
    - Attributes and children (recursively) must also match
  => Usually, “match” means “same node in the src doc”
- Result appears in document order.
Comparisons

let $\text{group1} := (<a/>,<b/>)$
let $\text{group2} := (<b/>,<c/>)$
return $\text{group1} \cup \text{group2}$
=> ($<a/>,<b/>,<b/>,<c/>)$

let $\text{a} := <a/>$
let $\text{b} := <b/>$
let $\text{c} := <c/>$
return ($\text{a},\text{b}$) $\cup$ ($\text{b},\text{c}$)
=> ($<a/>,<b/>,<c/>$)