XQuery: turning XPath into a real query language (and then some)

Introduction to databases
CSCC43 Winter 2011
Ryan Johnson

Quick review of XPath

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

• Weaknesses
  – Most joins impossible (self-joins possible but hard)
  – No means of formatting results
  – No control flow (branching or loops)
  – Little/no ability to manipulate XML
  – No way to specify input!

Why might we manipulate XML?

• Consider the XPath query
  => returns a list of complete book elements

• How to turn that into the following?
  <book_list>
    <book>
      <title>I, Robot</title>
      <publisher>Gnome Press</publisher>
    </book>
    ...
  </book_list>

• XPath union operator isn’t enough
  – Best we can do: mixed list of title and publisher elements
  => What if <!ELEMENT book (title*, publisher*, ...)?>

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 flavors)
  – XML transformation: make data presentable
  – sorting, more powerful predicates, set operations...

Expressiveness: XPath << SQL << XQuery

XPath can only return full (not “sparse”) subtrees
Key concepts of XQuery

- Template of sorts: mixed output and logic
  - Statically create the overall structure
  - Embed logic to “flesh out” using 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

Basic XQuery code format

```
<title>Useful information about Tor:</title>
<books-by-tor>
{ //book[publisher='Tor']/title }
</books-by-tor>
<authors-with-tor>
{ //book[publisher='Tor']/author/last-name }
</authors-with-tor>
```

Oops! Author list has duplicates...

FLWOR (“flower”) expressions

- XPath:
  ```
  //book[publisher='Tor'
      and author/last-name='Asimov'
    ]/author | title
  ```
- FLWOR:
  ```
  for $b in //book
  let $a := $b/author
  where $b/publisher = 'Tor'
      and $a/last-name='Asimov'
  order by $b/title
  return <book>{$b/title,$a}</book>
  ```

In what ways is the FLWOR superior to the XPath?

Characteristics of a FLWOR

- F(or)
  - Iterate over each item in a sequence
  - Multiple sequences separated by commas
- L(let)
  - Declares a variable and assigns it a value
  - Multiple declarations separated by commas
  - Bound once per iteration of every for above it
- W(where), O(der 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

(for | let)+ where? order-by? return
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...

Sequences in XQuery

- Most expressions return sequences of nodes
  - Ordered, e.g. document order or due to order by clause
- Sequence literals also allowed
  - e.g. (1,2,3)
  - 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)

If-then-else expressions

- Syntax is very C-like:
  if ($expr) then $expr else $expr
- BUT, like FLWOR, it is an expression!
  for $b$ in //book
  return
    if ($b/publisher = ‘Tor’) then <book>{$b/(title|author)}</book>
    else ()

Advanced predicates on node sets

- So far, two ways of predicking on node sets
  - Test for empty/non-empty
  - Iterate over their members and apply a predicate
- Two other techniques exist also
  - Explicit quantification
  - Single-object matching
Quantification

- XPath implicitly uses existential quantification
  - `//genre[.//author/last-name='Asimov']/@name` => Names every genre containing at least one book by Asimov
  - Tests whether `./author/last-name[n='Asimov']` is empty

- XQuery adds explicit control
  - Existential (∃): `some $x in $y satisfies $expr`
  - Universal (∀): `every $x in $y satisfies $expr`

Examples

- `//genre[some $n in .//author/last-name satisfies $n='Asimov']/@name`
- `transcript/semester[every $m in mark satisfies $m > 3.5]`
- `transcript/semester[some $m in mark satisfies $m < 2.0]`

Comparisons

- General comparisons
  - Apply to sequences of objects (same as XPath)
  - Operators: = != < > <= =>
  - A op B is true <=> ∃x∈A and ∃y∈B s.t. x op y is true

- Value comparisons
  - Compare the values of two objects
  - Operators: eq ne lt gt le ge
  - Error if both sides cannot be “atomized” to primitive types

- Node comparisons
  - Compare identity of two nodes based on position in document
  - Operators: is << >>
  - “is” only true if both sides are actually the same node

Set operations

- XPath defines only union ("|")
- XQuery adds `union`, `intersect`, `except` operators
  - Same meanings as in SQL

Also, duplicate elimination: `distinct-values()`

- All based on node comparisons, not values
  - Attributes and children (recursively) must also match

- Usually, “match” means “same node in the source doc”

User-defined functions

- Example
  ```
declare function count-nodes($e as element()) as integer {
return type
1 + sum(for $c in $e/* return count-nodes($c))
}
```

- Arguments can be typed
  - Parenthesis after type names
  - Cardinality controlled in the usual way: + ? *
  - Default type: item(*)

- Function body is an expression to evaluate
  - Ironically, functions don’t use the `return` keyword!
  - Recursion allowed

note the semicolon!
Duplicate elimination

- XQuery often produces duplicates
  
  ```xquery```
  ```
  //book[publisher='Tor']/author/last-name
  ```
- Solution: `distinct-values()`?
  - “Atomizes” inputs (elements become strings)
  => Good for ints, strings; useless for elements
- Solution: XPath’s `index-of()` function
  - `index-of($x, $y)` returns `position()` of each $x$ where $x=y$
  => every $z$ in $x$ where $z$ satisfies $y=z$
  => `index-of((a,b,c,b,a), a) = (1,5)`

Deduplication using `index-of()`

- Consider `index-of($x, $y)[1]`
  - Return the first (in document order) $z$ in $x$ satisfying $y=z$
- Solution: `$x[position()=index-of($x, .)[1]]`
  - Return every $z$ in $x$ which is the first in its set of matches
  - Abbreviated syntax: `$x[index-of($x, .)[1]]`
- WARNING: only works at document root!
  => Assign non-root sequences to a variable first

Node deduplication examples

- Author list example
  ```xml```
  ```xml```
- One-off deduplication
  ```xml```
- Even better: encapsulate it in a function!
  ```javascript```
  ```javascript```
- Can also define “duplicate” more narrowly
  ```xml```
  ```javascript```