XML Query Languages (1)

Prof. Weining Zhang

XQuery

- XQuery is a general purpose query language for XML data
- V 1.0 is a W3C working Draft, 2002
- XQuery is derived from the Quilt query language, which itself borrows from SQL, XQL and XML-QL
- It is an extension of XPath 2.0
  - For the common sub-language, they give the same result
- Based on XML data model & XML Schema types

XQuery Overview

- Syntax is similar to XPath 2.0
  - A functional language
  - Basic block is an expression
    - Many types of expressions exist
  - Strongly typed
    - Types are those defined in XML Schema
    - Type checking is done when binding values to variables
    - Result is a sequence of items (nodes or values)
    - Semantics is defined via the execution context

Expressions

- Path Expression
  - Basic form is same as in XPath
    - Step, Axes, node test, predicate, context, etc
  - Axes: less than those in XPath, 5 fwd, 1 bwd
  - Text() gives text content of an element without any sub-element
  - Support the “=>” operator for dereferencing IDREFs
    - Equivalent to the id() function of XPath, but simpler to use
    - Can be applied to a set of IDREFs to get a set of results

Expressions

- Sequence Expression
  - Construction: (1, (2, 3), 4), (10, 1 to 4)
  - Combining: $sqn1 union $sqn2; also intersection, etc.
- Arithmetic Expression
  - -3 div 2, $a/date - $b/date, etc.
- Comparison Expression
  - $a/name eq “bill”
- Logical Expression
  - 1 eq 1 and 3 idiv 0 = 1; also or, not, etc.
An FLWR Example

+ find all accounts with balance > 400, with each result enclosed in an <account-number> tag
  
  for $x$ in /bank-2/account
  let $acctno := $x/@account-number
  where $x/balance > 400
  return <account-number>
    $acctno
  </account-number>

An XQuery Example (cont.)

+ Let clause not really needed in this query, and selection can be done In XPath. Query
can be written as:
  for $x$ in /bank-2/account[balance>400]
  return <account-number>
    $x/@account-number
  </account-number>

+ Path expressions can be used in places other than for clause
  E.g. used in let clause, to bind variables to results of path expressions

For vs Let

- Ex: let $s = (<one/>, <two/>)
  return <out> $s </out>
  output: <out> <one/> <two/></out>

- Ex: for $s$ in <one/>, <two/> return <out> $s </out>
  output: <out>one/></out>
  <out>two/></out>

- Ex: for $a$ in (1, 2), $b$ in (3, 4)
generates tuples <$a$, $b$> in {<1, 3>, <1, 4>, <2, 3>, <2, 4>}

Functions

- Many built-in functions
  - The function distinct() can be used to removed duplicates in path expression results
  - The function document(name) returns root of named document
    E.g. document("bank-2.xml")/bank-2/account
  - Aggregate functions such as sum() and count() can be applied to path expression results
  - XQuery does not support groupby, but the same effect can be obtained by nesting FLWR expressions within a result clause

Sorting in XQuery

- Sortby clause can be used at the end of any expression. E.g. to return customers sorted by name
  for $c$ in /bank/customer
  return <customer> $c/* </customer>
  sortby(name)

Sorting in XQuery

- Can sort at multiple levels of nesting (sort by customer-name, and by account-number within each customer)
  
  <bank-1>
  for $c$ in /bank/customer
  return <customer>
    $c/*
    for $d$ in /bank/depositor[customer-name=$c/customer-name],
    $a$ in /bank/account[account-number=$d/account-number]
    return <account> $d/*
    sortby(account-number)
  </customer>

  sortby(customer-name)
  </bank-1>
User-Defined Functions

- Ex:
  function balances(xsd:string $c)
  returns list(xsd:numeric) {
    for $d in /bank/depositor[
      customer-name = $c,
      $a in /bank/account[account-number =
      $d/account-number]
    ]
    return $a/balance
  }

Joins

- Joins are specified in a manner very similar to SQL
  for $b in /bank/account,
  $c in /bank/customer,
  $d in /bank/depositor
  where $a/account-number = $d/account-number
  and $c/customer-name = $d/customer-name
  return <cust-acct> $c $a </cust-acct>

Joins

- The same query can be expressed with the selections specified as XPath selections:
  for $a in /bank/account
  $c in /bank/customer
  $d in /bank/depositor
  account-number = $a/account-number
  and
  customer-name = $c/customer-name
  return <cust-acct> $c $a </cust-acct>

Changing Nesting Structure

- The following query converts data from the flat structure for bank information into the nested structure used in bank-1
  <bank-1>
  for $c in /bank/customer
  return <customer>
  $c
  for $d in /bank/depositor[customer-name = $c/customer-name],
  $a in /bank/account[account-number = $d/account-number]
  return $a
  </customer>
  </bank-1>

Changing Nesting Structure

- $c/* denotes all the children of the node to which $c is bound, without the enclosing top-level tag
- Exercise: write a nested query to find sum of account balances, grouped by branch.

Other XQuery Features

- Types are optional for function parameters and return values
- Universal and existential quantification in where clause predicates
  - some $e in path satisfies $P
  - every $e in path satisfies $P
- Supports if-then-else clauses
- Has a number of expressions on sequence type: instance of, typeswitch, cast, etc.
XQL: XML Query Language

- Proposed to W3C, 1998, as an extension of XSL (used in templates)
- XQL is a declarative language, not for navigation
- Can fit in an URL
- Return results in document order without repeats of nodes
- Results may be different structures: nodes, documents, trees, etc.
  ▲ Closure is not guaranteed

Basic XQL Syntax

- Context
  ▲ Root context: /a/b, //a/b
  ▲ Current context: a/b, a//b
  ▲ Current context is the default
- Collection: a tag name represents the collection of all elements in the current context that have the tag
- Attributes are prefixed with “@”
- Filter [some_test_predicate]
  ▲ Applied to a collection & is TRUE if any element satisfies the test

Boolean Expressions

- author[(degree $or$ award) $and$ publication]
- author[degree $and$ $not$ publication]
- author[last-name = 'Bob']
- author[last-name Seq $'Bob' $]
- degree[@from != 'Harvard']
- degree[@from $notSeq$ 'Harvard']

Methods on collections

- Information methods: text(), NodeType(), ...
- Index() within parent, starting at 0
- Applied with “!”, ex. price[@intl!value() = 'canada']
- Index into a collection: author[first-name][2]

XQL Extensions

- Namespace: my:book[my:author], my: *
- Collection of attribute: @
- case insensitive comparisons: $ieq$, $ine$, $lt$, $le$, $gt$, $ge$
- Quantifier
  ▲ author[$any$ last-name = 'Bob']
  ▲ author[$all$ last-name != 'Bob']
- Set operators
  ▲ book[author/first-name = 'Bob'] $union$ magazine[price $lt$ 10]

Collection Methods

- p/textNode()[1] finds the second text node under parent p
- //comment()[1] finds the second comment anywhere in the document
- ancestor(book/author) finds the nearest ancestor author element under book
- author[0.2 $to$ 4, -1] finds the first, the third through fifth, and the last author elements