XPath query language

Introduction to databases
CSCC43 Spring 2012
Ryan Johnson

Thanks to Manos Papagelis, John Mylopoulos, Arnold Rosenbloom
and Renee Miller for material in these slides

Queries over hierarchical data

- Observe: all XML documents are trees
  - Each element has one “path” to the document root
  - Similar to a file system
- Idea: specify full or partial paths, with selection
  - /book-list/book/title
    => Returns the title of each book in the book-list
  - /book-list/book[title='SQL for Dummies']
    => Returns any book whose title is ‘SQL for Dummies’

Add bells and whistles, call the result X[ML]Path

Key concepts of XPath

- Axes
  - Different ways of “slicing” the tree
  - An axe defines a node-set relative to the current node (self, attributes, children, descendants, siblings, etc.)
- Path expression
  - Identifies [part of] a path from the document root
    => /book-list/book/title
- Predicates (selection)
  - Returned nodes must satisfy specified predicates
    => [title='SQL for Dummies']
- Nesting
  - Paths inside predicates
    => /book-list/book[author/last-name='Crichton']
  - Predicates inside paths
    => /book-list/book[author/last-name='Crichton']/title

Axes and context nodes

- Axis
  - What “dimension” of the tree to consider next
  - Filesystems provide three axes: self (/), parent (../), child (/)
  - XPath provides others, but defaults to child
- Context node
  - Where in the tree we are
    => Similar to current working directory of filesystem
  - Implied, or requested explicitly by specifying self axis
Example: XML Document Tree

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:  
  ```xml
  <foo>abc
  <bar><![/bar>
  </foo>
  ```
  - foo/child::* returns child elements only: <bar><![/bar>
  - foo/child::text() selects text children only: abc
  - foo/child::node() selects everything: abc<bar><![/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

Quick-reference: element axes

Quick reference: 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()]
Path expressions

• Context node
  – Where in the tree we are
  – Where the next expression will be applied
  – Like current working directory in a filesystem
  – Need to specify axis if next dimension to use is not child

• Document root
  – Special unnamed node which holds all others
  => Parent of the top-level node named in the XML file

• Absolute vs. relative paths
  – Child of current context node: book/title
  – At document root: /book/title
  – Anywhere in document: //book/title

Predicates

• \[\texttt{xexpr}\] applies boolean predicate to a node set
  – Similar to if/then statement
  – Returns subset of nodes for which \texttt{xexpr} is \texttt{TRUE}
  – Excludes subset of nodes for which \texttt{xexpr} is \texttt{FALSE}

• Expression 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 (=, \texttt{!=}, <, >, etc.)
  => /book-list/book[price < 50]
  – Node set (true if exists/non-empty)
  => /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 \texttt{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 (usually?) possible to simplify nested queries
  => /book-list/book[author[last-name='Asimov']]

Parentheses

• Occasionally need parenthesis for grouping
  – Often due to positional predicates: [1], \texttt{[last()]}, etc.
  => elem/preceding-sibling[1] != (elem/preceding-sibling)[1]
  => //elem[1] != (///elem)[1]
Union operator

- Union: combine results of 2+ XPath queries
  Syntax: (a | b | ...)
- Combine multiple queries
  e.g. All titles and all authors
  => (///title | //author)
- Combine multiple subqueries
  e.g. Books whose keyword or title mentions ‘robot’
  => //book[(keyword | title)[contains(text(), ‘robot’)]]

Union only allowed at start of path

Union and location steps

- Union not a valid location step in XPath 1.0
  - XPath 2.0 adds it, but not widely supported
  - SaxonHE: yes... libxml2, browser XSLT, etc.: no
- Examples:
  - (//nickname | //name)[.=“Joe”] OK
  - (//author | //editor)/last_name OK
  - //book/(author | editor)/last_name ERROR
  - //book/(author | publisher)/address ERROR
  => //book/*[self::title | self:::publisher]/ address OK
  => //book/*[self::author | self:::editor]/last_name OK

Yes, it’s a language design flaw

Quick reference: 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
- 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

Warning: above list is non-exhaustive!
Example: The DTD

```xml
<!DOCTYPE gamer-catalog [ 
  <!ELEMENT catalog (platform*, game*)>  
  <!ELEMENT platform (supports*, seller*)>  
    <!ATTLIST platform name ID #REQUIRED>  
    <!ATTLIST platform manf CDATA #IMPLIED>  
  <!ELEMENT supports ID #REQUIRED>  
  <!ELEMENT seller (#CDATA)>   <!-- contains: price -->  
    <!ATTLIST seller name ID #REQUIRED>  
  <!ELEMENT game (#CDATA)>   <!-- contains: price -->  
    <!ATTLIST game title ID #REQUIRED>  
    <!ATTLIST available IDREFS #REQUIRED> ]>
```

Example: The XML Document

```xml
<catalog>
  <platform name="XBox" manf="Microsoft">
    <supports>Halo</supports>
    <supports>Call of Duty</supports>
    ...  
    <seller name="Wal-Mart">249.99</seller>
  </platform>
  ...  
  <game title="Halo" available="Xbox">59.99</game>
  <game title="Call of Duty" available="PS3 XBox PC">45</game>
  ...
</catalog>

Tree representation of the XML Doc

Example XPATH Queries

- Path expressions
- Attributes in paths
- Paths that begin anywhere
- Paths that make use of wildcard *
- Selection Conditions
- Attributes in Selection
Path Expressions

XPATH: /catalog

<catalog>
  <platform name="XBox" manf="Microsoft">
    <supports>Halo</supports>
    <supports>Call of Duty</supports>
    ...
    <seller name="Wal-Mart">249.99</seller>
  </platform>
  ...
  <game title="Halo" available="Xbox">59.99</game>
  ...
</catalog>

Path Expressions (cont.)

XPATH: /catalog/game

<catalog>
  ...
  <platform name="XBox" manf="Microsoft">
    <supports>Halo</supports>
    <supports>Call of Duty</supports>
    ...
    <seller name="Wal-Mart">249.99</seller>
  </platform>
  ...
  <game title="Halo" available="Xbox">59.99</game>
  ...
</catalog>

Path Expressions (cont.)

XPATH: /catalog/platform/supports

<catalog>
  <platform name="XBox" manf="Microsoft">
    <supports>Halo</supports>
    <supports>Call of Duty</supports>
    ...
    <seller name="Wal-Mart">249.99</seller>
  </platform>
  ...
  <game title="Halo" available="Xbox">59.99</game>
  ...
</catalog>

Attributes in Paths

XPATH: /catalog/platform/@name

<catalog>
  <platform name="XBox" manf="Microsoft">
    <supports>Halo</supports>
    <supports>Call of Duty</supports>
    ...
    <seller name="Wal-Mart">249.99</seller>
  </platform>
  ...
  <game title="Halo" available="Xbox">59.99</game>
  ...
</catalog>

These supports elements followed by the supports elements of all the other platform elements

This game element followed by all the other game elements

This platform contributes "XBox" to the result; other @name values follow
Paths that Begin Anywhere

**XPATH:** //@name

```xml
<catalog>
  <platform name="XBox" manf="Microsoft">
    <supports>Halo</supports>
    <supports>Call of Duty</supports>
    ...
    <seller name="Wal-Mart">249.99</seller>
  </platform>
  ...
  <game title="Halo" available="Xbox">59.99</game>
  ...
</catalog>
```

Paths that use a wildcard *

**XPATH:** /catalog/platform/*

```xml
<catalog>
  <platform name="XBox" manf="Microsoft">
    <supports>Halo</supports>
    <supports>Call of Duty</supports>
    ...
    <seller name="Wal-Mart">249.99</seller>
  </platform>
  ...
  <game title="Halo" available="Xbox">59.99</game>
  ...
</catalog>
```

Selection Conditions

**XPATH:** /catalog/game[@<50]

```xml
<catalog>
  <platform name="XBox" manf="Microsoft">
    ...
  </platform>
  ...
  <game title="Call of Duty" available="...">44.99</game>
  <game title="Halo" available="Xbox">59.99</game>
  ...
</catalog>
```

Attributes in Selection

**XPATH:** /BARS/BAR/PRICE[@theBeer = "Miller"]

```xml
<BARS>
  <BAR name = "JoesBar">
    <PRICE theBeer = "Bud">2.50</PRICE>
    <PRICE theBeer = "Miller">3.00</PRICE>
  </BAR>
</BARS>
```

Now, this PRICE element is selected, along with any other prices for Miller.