XPath and XQuery

Database Systems: The Complete Book Ch. 12.1, 12.2

Ning Deng

November 15, 2017

Outline

Outline

Introduction

- XPath: a simple language for describing sets based on paths in a graph of semistructured data
- XQuery: an extension of XPath that adopts something of the style of SQL

XPath

Data Model

- Tree-based: a sequence of items:
  - nodes: document root, element, attribute
  - atomic values: integer, real, string...

- Every XPath query refers to a document
- **Document node**: constructed from an XML document by doc(*doc_uri*), where the document URI is provided in double quotes
- The document node represents the XML document itself, not the root element, the root element is a child of the document node

Path Expressions

- Describes a set of paths in a document
- returns a sequence of nodes
- evaluated in a context
- absolute (starting at document root) or relative
- consists of steps separated by /
- wildcards
- union (|), intersection, difference

Example XML

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<catalog>
   <book id="bk101">
      <author>Gambardella, Matthew</author>
      <title>XML Developer&amp;apos;s Guide</title>
      <genre>Computer</genre>
      <price>44.95</price>
      <publish_date>2000-10-01</publish_date>
      <description>An in-depth look at creating applications
      with XML.</description>
   </book>
   <book id="bk102">
      <author>Ralls, Kim</author>
      <title>Midnight Rain</title>
      <genre>Fantasy</genre>
      <price>5.95</price>
      <publish_date>2000-12-16</publish_date>
   </book>
</catalog>
```

Path Expressions

Simplest expression

Absolute path: $/T_1/T_2/.../T_n$, where each T_i is a tag

- Evaluation starts with a sequence of one node: the document node.
- Each T_i is evaluated in turn, starting with T_1
- To evaluate T_i, consider the sequence $\mathcal{S}$ of items obtained from processing all previous tags
- Examine the items of $\mathcal{S}$ **in order** and, for each one, find all subelements whose tag is T_i and append them to the output sequence, **in document order**

Relative Path Expressions: *relative* to the current node or sequence of nodes, e.g. the selector in last lecture

Example

doc("book.xml")/catalog/book/author

```
1 <author>Gambardella, Matthew</author>

2 <author>Ralls, Kim</author>
```

Attributes in Path Expressions

Simplest expression

$/T_1/T_2/.../T_n/@A$, where each T_i is a tag, and A is an attribute of T_n

- To evaluate the expression, first compute the expression $/T_1/T_2/\ldots/T_n/$
- For each element in the resulting sequence, if attribute A exists in its opening tag, its value is appended to the output sequence

Example

doc("/db/book.xml")/catalog/book/@id/string()

```
1 "bk101"

2 "bk102"
```

XPath Axes

- XPath provides a rich set of axes, or modes of navigation
- We have seen two axes: child(default) and attribute ("@")
- We can prefix a tag or attribute with an axis name and a double-colon, for examples:
  - $child :: /T_1/child :: T_2/\ldots/child :: T_n$, this is equivalent to $/T_1/T_2/.../T_n$
  - $child :: /T_1/child :: T_2/\ldots/child :: T_n/attribute :: A$, this is equivalent to $/T_1/T_2/.../T_n/@A$
- Other axes: **parent** (".."), **ancestor**(proper), **descendant**, **next-sibling**(to the right), **previous-sibling**(to the left), **self**("."), **descendant-or-self**("//")
- Semantics assumes that elements in the result of an XPath expression are reference to elements in the actual document

Axes

AxisName	Result
ancestor	Selects all ancestors (parent, grandparent, etc.) of the current node
ancestor-or-self	Selects all ancestors (parent, grandparent, etc.) of the current node and the current node itself
attribute	Selects all attributes of the current node
child	Selects all children of the current node
descendant	Selects all descendants (children, grandchildren, etc.) of the current node
descendant-or-self	Selects all descendants (children, grandchildren, etc.) of the current node and the current node itself
following	Selects everything in the document after the closing tag of the current node
following-sibling	Selects all siblings after the current node
namespace	Selects all namespace nodes of the current node
parent	Selects the parent of the current node
preceding	Selects all nodes that appear before the current node in the document, except ancestors, attribute nodes and namespace nodes
preceding-sibling	Selects all siblings before the current node
self	Selects the current node

Example

doc("/db/book.xml")//author

```
1 <author>Gambardella, Matthew</author>

2 <author>Ralls, Kim</author>
```

Other Expressions

- Wildcards: "*" for any tag or attribute, e.g. doc("/db/book.xml")/catalog/*
- Conditions: boolean expressions within square brackets
  - Operands can be path expressions
  - Path expression operands have existential semantics
  - Integer [i]: true only for the i-th child of the parent
  - Tag [T]: true only for elements having one or more subelements with tag T.
  - Tag [@A]: true only for elements having a value for attribute A.

Example

```
1)  <? xml version="1.0" encoding="utf-8" standalone="yes" ?>
2)  <Movies>
3)      <Movie title = "King Kong">
4)          <Version year = "1933">
5)              <Star>Fay Wray</Star>
6)          </Version>
7)          <Version year = "1976">
8)              <Star>Jeff Bridges</Star>
9)              <Star>Jessica Lange</Star>
10)         </Version>
11)         <Version year = "2005" />
12)     </Movie>
13)     <Movie title = "Footloose">
14)         <Version year = "1984">
15)             <Star>Kevin Bacon</Star>
16)             <Star>John Lithgow</Star>
17)             <Star>Sarah Jessica Parker</Star>
18)         </Version>
19)     </Movie>
20) </Movies>
```

Example

/Movies/Movie[//Star="Jessica Lange"]/@title

/Movies/Movie/Version[1]/@year

/Movies/Movie/Version[Star]

/Movies/Movie/Version[@year]

XPath Queries

Select the root element: **doc("abc.xml")/A**

Example

```
<A>
    <B></B>
    <C>John</C>
    <B></B>
    <D>
      <B></B>
    </D>
    <C>Mary</C>
</A>
```

XPath Queries

Select the text content of C elements: **doc("abc.xml")//C/text()**

Example

```
<A>
    <B></B>
    <C>John</C>
    <B></B>
    <D>
      <B></B>
    </D>
    <C>Mary</C>
</A>
```

XPath Queries

Select the C elements that have some text content:
doc("abc.xml")//C[text()]

Example

```
<A>
   <B></B>
   <C>John</C>
   <B></B>
   <D>
      <B></B>
   </D>
   <C></C>
</A>
```

XPath Queries

Select the B elements that are children of some D elements
doc("abc.xml")//D/B

Example

```
<A>
   <B></B>
   <C>John</C>
   <B></B>
   <D>
      <B></B>
   </D>
   <C>Mary</C>
</A>
```

XPath Queries

Select all elements that enclosed by the path A/C/D
doc("abc.xml")/A/C/D/*

Example

```
<A>
   <B></B>
   <C>John</C>
   <B></B>
   <D>
     <B></B>
   </D>
   <C>
     <D>
       <E>Mary</E>
     </D>
   </C>
</A>
```

XPath Queries

Select all elements: **doc("abc.xml")//***

Example

```
<A>
   <B></B>
   <C>John</C>
   <B></B>
   <D>
     <B></B>
   </D>
   <C>Mary</C>
</A>
```

XPath Queries

Select the first B child of every D elements: **doc("abc.xml")//D/B[1]**

Example

```
<A>
    <B></B>
    <C>John</C>
    <B></B>
    <D>
      <B></B>
      <B>123</B>
    </D>
    <C>Mary</C>
</A>
```

XPath Queries

Select the last B child of every D elements:
doc("abc.xml")//D/B[last()]

Example

```
<A>
   <B></B>
   <C>John</C>
   <B></B>
   <D>
     <B></B>
     <B>123</B>
   </D>
   <C>Mary</C>
</A>
```

XPath Queries

Select all id attributes: **doc("abc.xml")//@id**

Example

```
<A>
   <B id="B1"></B>
   <C>John</C>
   <B id="B2"></B>
   <D id="D1">
     <B></B>
   </D>
   <C>Mary</C>
</A>
```

XPath Queries

Select B elements having an id attribute: **doc("abc.xml")//B[@id]**

Example

```
<A>
   <B id="B1"></B>
   <C>John</C>
   <B id="B2"></B>
   <D id="D1">
     <B></B>
   </D>
   <C>Mary</C>
</A>
```

XPath Queries

Select B elements having some attribute: **doc("abc.xml")//B[@*]**

Example

```
<A>
   <B id="B1"></B>
   <C>John</C>
   <B id="B2"></B>
   <D id="D1">
     <B flag="1"></B>
   </D>
   <C>Mary</C>
</A>
```

XPath Queries

Select B elements without attributes: **doc("abc.xml")//B[not(@*)]**

Example

```
<A>
   <B id="B1"></B>
   <C>John</C>
   <B id="B2"></B>
   <D id="D1">
     <B></B>
   </D>
   <C>Mary</C>
</A>
```

XPath Queries

Select B elements with an id attribute with value "B1":
doc("abc.xml")//B[@id="B1"]

Example

```
<A>
   <B id="B1"></B>
   <C>John</C>
   <B id="B2"></B>
   <D id="D1">
     <B></B>
   </D>
   <C>Mary</C>
</A>
```

XPath Queries

Select elements that have 2 B elements:
doc("abc.xml")//*[count(B)=2]

Example

```
<A>
   <B id="B1"></B>
   <C>John</C>
   <D id="D1">
     <B></B>
     <B></B>
   </D>
   <C>Mary</C>
</A>
```

XPath Queries

Select B and C elements: **doc("abc.xml")//B|doc("abc.xml")//C**

Example

```
<A>
   <B id="B1"></B>
   <C>John</C>
   <D id="D1">
     <B></B>
     <B></B>
   </D>
   <C>Mary</C>
</A>
```

XPath Queries

Select the parents of some B element: **doc("abc.xml")//B/parent::***

Example

```
<A>
   <B id="B1"></B>
   <C>
       <D id="D1">
         <B></B>
         <B></B>
       </D>
   </C>
</A>
```

XPath Queries

Select the ancestors of some B element:
doc("abc.xml")//B/ancestor::*

Example

```
<A>
   <B id="B1"></B>
   <C>
      <D id="D1">
        <B></B>
        <B></B>
      </D>
   </C>
</A>
```

XPath Queries

Select the elements following B elements:
doc("abc.xml")//B/following::*

Example

```
<A>
   <B id="B1"></B>
   <C>
      <D id="D1">
        <B></B>
        <B></B>
      </D>
   </C>
</A>
```

XPath Queries

Select the intersection of B elements on certain paths:
doc("abc.xml")/A//C//B intersect doc("abc.xml")/A/C/B

Example

```
<A>
   <B id="B1"></B>
   <C>
      <B></B>
      <B></B>
      <D id="D1">
        <B></B>
        <B></B>
      </D>
   </C>
</A>
```

XPath Queries

Select the set difference of B elements on certain paths:
doc("abc.xml")/A//B except doc("abc.xml")/A/C/B

Example

```
<A>
   <B id="B1"></B>
   <C>
      <D id="D1">
        <B></B>
        <B></B>
      </D>
   </C>
<A>
```

XQuery

- Extension of XPath
- Standard for higher-level XML applications
- A functional language: any XQuery expression can be used in any place that an expression is expected
- The main query expression is the FLWOR expression
- FLWOR is XQuery's analogous of SQL's SPJ expressions

FLWOR

```
FLWORExpr::=(ForClause|LetClause)+
WhereClause?
OrderByClause?
ReturnClause
```

LET Clause

```
LetClause::= let $ var := expr
             (, $var2 := expr)*
```

- All LET variables start with a dollar sign
- Multiple simultaneous assignments are supported
- The result of expr is an ordered sequence of items
- The entire sequence is bound to the LET variable

Example

```
let $ bk := doc("book.xml"),
    $ bka := doc("book.xml")/catalog//author
```

FOR Clause

```
ForClause::= for $ var in expr
             (, $ var2 in expr)*
```

- All FOR variables start with a dollar sign
- Multiple simultaneous assignments are supported
- The result of expr is an ordered sequence of items
- Each item is bound to the FOR variable, in turn

Example

```
let $bk :=doc("book.xml")
for $b in $bk/catalog//author
    ...
```

WHERE Clause

```
WhereClause::= where expr
```

- Filters the item sequence, retaining some items and discarding others.

RETURN Clause

```
ReturnClause::=return expr
```

- Evaluated once for every item in the sequence
- The final result is an ordered sequence containing the results of these evaluations
- The RETURN clause is typically executed multiple times inside FOR loops

Example

```
let $bk :=doc("book.xml")
for $b in $bk/catalog//book
where $b/author="John"
return $b/title
```

ORDER BY Clause

```
OrderByClause::= (order by|stable order by) orderExpr
```

- Used to reorder the item sequence
- Keywords *ascending* and *descending* supported

Example

```
let $bk :=doc("book.xml")
for $b in $bk/catalog//book
where $b/author="John"
order by $b/@id ascending
return $b/title
```

Joins in XQuery

- Define two or more variables
- Enforce the join condition using WHERE clause
- Comparing elements is analogous to comparing objects in a language such as Java, it is a reference comparison. You normally want to compare primitive values, for which you can use the fn:data() function (atomization)

Example

Example

```
let $m := doc("movies.xml"),
    $s := doc("stars.xml")
for $s1 in $m/Movie//Star,
    $s2 in $s/Star
where data($s1)=data($s2/Name)
return $s2/Address/City
```

Comparison

- XQuery provides a set of comparison operators that only compare sequences consisting of a single item, and fail if either operand is a sequence of more than one item: **eq, ne, lt, gt, le, ge**

Example

```
 1)  <? xml version="1.0" encoding="utf-8" standalone="yes" ?>
 2)  <Stars>
 3)      <Star>
 4)          <Name>Carrie Fisher</Name>
 5)          <Address>
 6)              <Street>123 Maple St.</Street>
 7)              <City>Hollywood</City>
 8)          </Address>
 9)          <Address>
10)              <Street>5 Locust Ln.</Street>
11)              <City>Malibu</City>
12)          </Address>
13)      </Star>
             ... more stars
14)  </Stars>
```

Find all the stars that live at 123 Maple St., Malibu.

Example

First attempt:

```
let $stars := doc("stars.xml")
for $s in $stars/Stars/Star
where $s/Address/Street = "123 Maple St."
and $s/Address/City="Malibu"
return $s/Name
```

Unfortunately this does not work.

Example

Second attempt:

```
let $stars := doc("stars.xml")
for $s in $stars/Stars/Star
where $s/Address/Street eq "123 Maple St."
and $s/Address/City eq "Malibu"
return $s/Name
```

Example

Second attempt:

```
let $stars := doc("stars.xml")
for $s in $stars/Stars/Star
where $s/Address/Street eq "123 Maple St."
and $s/Address/City eq "Malibu"
return $s/Name
```

Unfortunately this does not work too.

Example

Second attempt:

```
let $stars := doc("stars.xml")
for $s in $stars/Stars/Star
where $s/Address/Street eq "123 Maple St."
and $s/Address/City eq "Malibu"
return $s/Name
```

Unfortunately this does not work too.
Exercise!

Eliminate Duplicates

- Built-in functions: **distinct-values**

Example

```
let $stars := distinct-values(
    let $movies := doc("movies.xml")
    for $m in $movies/Movies/Movie
    return $m/Version/Star)
return <Stars>{$stars}</Stars>
```

Any text is permissible between tags or as attributes in XQuery expressions. To include an expression, surround it with curly braces.

Quantification in XQuery

There are expressions that say "for all" and "there exists":

```
every $var in expr1 satisfies expr2

some $var in expr1 satisfies expr2
```

Example

```
let $stars := doc("stars.xml")
for $s in $stars/Stars/Star
where every $c in $s/Address/City satisfies
   $c ="Hollywood"
return $s/Name
```

Aggregation

- Built-in aggregate functions that take as input a sequence and output a scalar value
- avg(), count(), max(), min(), sum()
- group by *expr*
  In some software there's no construct for grouping. (Question: so how can we do grouping?

Aggregation

- Built-in aggregate functions that take as input a sequence and output a scalar value
- avg(), count(), max(), min(), sum()
- group by *expr*
  In some software there's no construct for grouping. (Question: so how can we do grouping? **distinct-values**)

Example

```
let $movies := doc("movies.xml")
for $m in $movies/Movies/Movie
where count($m/Version) >2
return $m
```

Branching

if-then-else expression in XQuery:

$$\text{if } (\textit{expr1}) \text{ then } \textit{expr2} \text{ else } \textit{expr3}$$

Note: this is not a statement: XQuery is a functional language, so every expression must return a sequence of items. This means that the **else** part is mandatory. If you do not wish to return anything, return the empty sequence: ().

Example

Example

```
let $m :=
doc("movies.xml")/Movies/Movie[@title="King Kong"]
for $v in $m/Version
return
    if ($v/@year=max($m/Version/@year))
    then <latest>{$v}</latest>
    else <old>{$v}</old>
```

Context Item Expression

- . : refers to the *context item*
- can be used inside a predicate [...]
- can be used inside a path step "/."

Example

```
let $m :=
doc("movies.xml")/Movies/Movie
for $v in $m/Version
return $v[./year=2010]
```