Applied Databases

Lecture 17
XPath

Sebastian Maneth

University of Edinburgh - March 20th, 2017
Assignment 2

Please check web page!!

→ java Searcher "star trek"

had a wrong order

This is the correct order (see web page)

totalHits 72
1049430907, SUPERMAN WITH GEN 13 AND OTHER PRESTIGE BOOKS, score: 1.6115568, price: 6.00
1047062670, Superman's Pal Jimmy Olsen # 81, score: 1.4560543, price: 1.20
1045823269, Superman Doomsday Hunter Prey tpb ,score: 1.4560543, price: 7.97
1048743351, Superman Lunchbox Hallmark Ornament, score: 1.3813344, price: 9.99
1048647703, SUPERMAN COMIC NO.199 - AUSTRALIAN ISSUE, score: 1.2355031, price: 1.99
1047692530, BATMAN OR SUPERMAN CHRISTMAS ORNAMENTS HOT!!, score: 1.1648434, price: 19.99
1047761329, Superman Domed Lunchbox/Carrying Case NEW!!, score: 1.1511121, price: 11.99
1048263344, SUPERMAN DAILY PLANET Magnet PICTURE FRAME, score: 1.0896113, price: 4.95
1047388061, SUPERMAN #405 NM "BATMAN" (1985), score: 1.069977, price: 6.99
1046936194, SUPERMAN METAL LUNCH BOX, score: 1.069977, price: 12.95
Each degree of latitude is approximately 69 miles (111 kilometers) apart. The range varies (due to the earth's slightly ellipsoid shape) from 68.703 miles (110.567 km) at the equator to 69.401 (111.699 km) at the poles. This is convenient because each minute (1/60th of a degree) is approximately one [nautical] mile.

About this result

What Is the Distance Between a Degree of Latitude ... - ThoughtCo
However, the earth is slightly elliptical in shape and that creates a small variation between the arc we make as we move from the equator to the north and south poles. Each degree of latitude is...
XML Querying with XPath
Tree-structured data
- XML
- HTML
- JSON

"shredding"

Relational tables (for SQL querying)

Inverted Files (for keyword search)
Tree-structured data
e.g. in
- XML
- HTML
- JSON

“shredding”

Relational tables  (for SQL querying)

Inverted Files  (for keyword search)

Sometimes: more intuitive / natural to query the tree directly
Tree-structured data
- e.g. in
- XML
- HTML
- JSON

Sometimes: more intuitive / natural to *query the tree directly*
Tree-structured data
e.g. in
- XML
- HTML
- JSON

Sometimes: more intuitive / natural to query the tree directly

→ need a query language for trees / XML!
 XPath

→ low-level query language to select nodes of an XML document

→ W3C Standard (1999)

→ **most important XML query language**: used in many technologies such as XQuery, XSLT, XPointer, XLink, Javascript, ...

→ Cave: newer versions are more expressive than 1.0
We study XPath 1.0 [current version: 3.0 (2014)]

Terminology: instead of “query” we often say **XPath expression**.

→ an expression is the primary construction of the XPath grammar; it matches the production Expr of the XPath grammar.
XPath

→ low-level query language to select nodes of an XML document

→ W3C Standard (1999)

→ most important XML query language: used in many technologies such as XQuery, XSLT, XPointer, XLink, Javascript, ...

→ Cave: newer versions are more expressive than 1.0
  We study XPath 1.0 [ current version: 3.0 (2014) ]

Terminology: instead of “query” we often say XPath expression.

→ an expression is the primary construction of the XPath grammar; it matches the production Expr of the XPath grammar.
→ find Director's name in the HTML
Tree structure of an IMDB movie page (HTML)

→ deep tree structure

→ **span-node** of Director’s name at depth > 50
Tree structure of an IMDB movie page (HTML)

- deep tree structure
- **span-node** for director’s name at depth > 50

XPath query selecting the span-node

```
//*[@id='title-overview-widget']/div[3]/div[1]/div[2]/span/a/span
```
Outline

1. XPath Data Model: **7 types of nodes**
2. Simple Examples
3. Location Steps and Paths
4. Value Comparison, and other Functions
XPath Query Q

XML document D

Evaluate Q on D (in XPath data model)

sequence of result nodes

Document D is modeled as a **tree**.

THERE ARE SEVEN TYPES OF NODES in the XPath Data Model:
Result Sequences

XPath Query Q

XML document D

Evaluate Q on D (in *XPath data model*)

→ ordered in *document order*
→ contains *no duplicates*

sequence of result nodes
Simple Examples

In abbreviated syntax.

Q1: /bib/book/year

Document:

```xml
<bib>
  <book>
    <author>Abiteboul</author>
    <author>Hull</author>
    <author>Vianu</author>
    <title>Foundations of Databases</title>
    <year>1995</year>
  </book>
  <book>
    <author>Ullmann</author>
    <title>Principles of Database and Knowledge Base Systems</title>
    <year>1998</year>
  </book>
</bib>
```
Simple Examples

In abbreviated syntax.

Q3: /a/b///d

“b-child of a-doc. element”

ALL d-nodes in this subtree
Simple Examples

In abbreviated syntax.

Q4: /*/c
Simple Examples

In abbreviated syntax.

Q5: //c
Simple Examples

In abbreviated syntax.

Q6: /*
→ important: there is always a (virtual) Root-node!
even if `<?xml ... >` is missing.

/a = a-child of Root-node
/a/../* = same node
→ important: there is always a (virtual) **Root-node**!
even if `<?xml ... >` is missing.

```
/a  = a-child of Root-node
/a/../*  = same node
/a/../../../a = “No Match”
```
→ important: there is always a (virtual) **Root-node**!
even if `<?xml ... >` is missing.

```
/a = a-child of Root-node
/a/../* = same node
/a/../../a = "No Match"
/a/.. = "No DOCTYPE Declaration,
Root is [Element :<a/>]"
```

Implementation-dependent
Abbreviations

In abbreviated syntax.

/a is abbreviation for /child::a

//a is abbreviation for /descendant-or-self::node()/child::a

. is abbreviation for self::node()

.. is abbreviation for parent::node()

→ Child and descendant-or-self are only 2 out of 12 possible axes.

An “Axis” is a sequence of nodes. It is evaluated relative to a context-node.

Other axes: → descendant
→ parent
→ ancestor-or-self
→ ancestor
→ following-sibling
→ preceding-sibling
→ attribute
→ following
→ preceding
→ self
Predicates (aka “Filters”)

In abbreviated syntax.

Q7: //c[./b]  

“has b-child” (context-nodes are all c-nodes…)

Filters [...] have

→ existential semantics

→ [./b] = “there exists a b-child”
Predicates (aka “Filters”)

In abbreviated syntax.

Q8: //c[./b]/d  "has b-child"

All d-children of the context-node(s)
Predicates (aka "Filters")

In abbreviated syntax.

Q9: //c[./b]/d/.. "has b-child"

select parent(s)
of context-node(s)

parent(s)
of the context-node(s)

Q9 selects c-nodes that "have a b-child AND a d-child"
Predicates (aka “Filters”)

In abbreviated syntax.

Q9: //c[./b]/d/.. “has b-child”

select parent(s) of context-node(s)

Q9 selects c-nodes that “have a b-child AND a d-child”

More direct way: //c[./b and ./d]
Predicates (aka “Filters”)

In abbreviated syntax.

```
//c[b and d]
```

A “Filter” evaluates to true/false

c-nodes that “have a b-child AND a d-child”
Predicates (aka “Filters”)

In abbreviated syntax.

```c
//c[b and d]
```

Evaluates to true/false

A “Filter”

Question

How to only select the other c-node?

Can use “not(...)” in a filter!

```c
//c[not(b)]
```

“does not have a b-child”
Examples: Predicates

In abbreviated syntax.

```c
//c[b and d]
```

---

A “Filter” evaluates to `true/false`

---

**Question**

How to only select the other c-node?

---

Many more possibilities, of course:

```c
//c[parent::b]
//c[../../b]
//c[../d]
```

---

Can use “not( ... )” in a filter!

```c
//c[not(b)]
```
Examples: Predicates

In abbreviated syntax.

```
// c[b and d] evaluates to true/false
```

A “Filter”

Question

How to only select the other c-node?

Many more possibilities, of course:

```
// c[parent::b]
// c[.../.../b]
// c[.../d]
```

Can use “not( ... )” in a filter!

```
// c[not(b)]
```

→ can you say “c-node that has only d-children”?
Examples: Predicates

In abbreviated syntax.

//c[b and d] evaluates to true/false

A “Filter”

Can use “not(...)” in a filter!

//c[not(b)]

-> can you say “c-node that has only d-children”?

YES! needs a bit of logic...

//c[not(child::*[not(self::d)]))

Question

How to only select the other c-node?

Many more possibilities, of course:

//c[parent::b]
//c[../../b]
//c[../d]
Location Steps & Paths

→ A Location Path is a sequence of Location Steps

→ A Location Step is of the form

\[
\text{axis} :: \text{nodetest} \ [ \text{Filter}_1 \ ] \ [ \text{Filter}_2 \ ] \ldots \ [ \text{Filter}_n \ ]
\]

Filters (aka predicates, (filter) expressions)
→ evaluate to true/false
→ XPath queries, evaluated with
  \[ \text{context-node} = \text{current node} \]

Boolean operators: \text{and, or}

Empty string/sequence are converted to \text{false}
Location Steps & Paths

→ A Location Path is a sequence of Location Steps

→ A Location Step is of the form

axis :: nodetest [ Filter_1 ] [ Filter_2 ] ... [ Filter_n ]

Filters (aka predicates, (filter) expressions)
evaluate to true/false

nodetest: * or node-name (could be expanded \( \rightarrow \) namespaces) or

Example  child::text()  "select all text node children of the context node"

→ the nodetest node() is true for any node.

attribute::*  "select all attributes of the context node"
Location Steps & Paths

→ A Location Path is a sequence of Location Steps

→ A Location Step is of the form

\[ \text{axis} :: \text{nodetest} \ [ \text{Filter}_1 \ ] \ [ \text{Filter}_2 \ ] \ldots \ [ \text{Filter}_n \ ] \]

Filters (aka predicates, (filter) expressions) evaluate to true/false

nodetest: * or node-name (could be expanded \( \rightarrow \) namespaces) or

12 Axes

Forward Axes:

→ self
→ child
→ descendant-or-self
→ descendant
→ following
→ following-sibling

In doc order

Backward Axes:

→ parent
→ ancestor
→ ancestor-or-self
→ preceding
→ preceding-sibling
→ attribute

\( \rightarrow \) text()
\( \rightarrow \) comment()
\( \rightarrow \) processing-instruction(In)
\( \rightarrow \) node()
Location Steps & Paths

Axis = a sequence of nodes  (is evaluated relative to context-node)

→ from context node, execute query:
   axis::*

Forward Axes:

→ self
→ child
→ descendant-or-self
→ descendant
→ following
→ following-sibling

In doc order

Backward Axes:

→ parent
→ ancestor
→ ancestor-or-self
→ preceding
→ preceding-sibling
→ attribute

reverse doc order
Location Steps & Paths

Axis = a sequence of nodes (is evaluated relative to context-node)

Forward Axes:
- self
- child
- descendant-or-self
- descendant
- following
- following-sibling

Backward Axes:
- parent
- ancestor
- ancestor-or-self
- preceding
- preceding-sibling
- attribute

→ from context node, execute query:

axis::*
Location Steps & Paths

Axis = a sequence of nodes  (is evaluated relative to context-node)

Forward Axes:
→ self
→ child
→ descendant-or-self
→ descendant
→ following
→ following-sibling
In doc order

Backward Axes:
→ parent
→ ancestor
→ ancestor-or-self
→ preceding
→ preceding-sibling
→ attribute
reverse doc order

→ from context node, execute query:
axis::*
Location Steps & Paths

**Axis** = a sequence of nodes  (is evaluated relative to **context-node**)
Location Steps & Paths

**Axis** = a sequence of nodes  (is evaluated relative to context-node)

![Diagram of node relationships]

→ from context node, execute query:

```
axis::*
```

**Forward Axes:**
- `self`
- `child`
- `descendant-or-self`
- `descendant`
- `following`
- `following-sibling`

In doc order

**Backward Axes:**
- `parent`
- `ancestor`
- `ancestor-or-self`
- `preceding`
- `preceding-sibling`
- `attribute`

reverse doc order
Location Steps & Paths

Axis = a sequence of nodes  (is evaluated relative to context-node)

Forward Axes:
- self
- child
- descendant-or-self
- descendant
- following
- following-sibling

In doc order

Backward Axes:
- parent
- ancestor
- ancestor-or-self
- preceding
- preceding-sibling
- attribute

reverse doc order

→ from context node, execute query:

axis::*
Location Steps & Paths

Axis = a sequence of nodes (is evaluated relative to context-node)

Forward Axes:

- self
- child
- descendant-or-self
- descendant
- following
- following-sibling

In doc order

Backward Axes:

- parent
- ancestor
- ancestor-or-self
- preceding
- preceding-sibling
- attribute

reverse doc order
Location Steps & Paths

Axis = a sequence of nodes  (is evaluated relative to context-node)

→ from context node, execute query:
  axis::*

Forward Axes:

→ self
→ child
→ descendant-or-self
→ descendant
→ following
→ following-sibling

In doc order

Backward Axes:

→ parent
→ ancestor
→ ancestor-or-self
→ preceding
→ preceding-sibling
→ attribute

reverse doc order
Location Steps & Paths

**Axis** = a sequence of nodes  (is evaluated relative to context-node)

Forward Axes:
- self
- child
- descendant-or-self
- descendant
- following
- following-sibling

Backward Axes:
- parent
- ancestor
- ancestor-or-self
- preceding
- preceding-sibling
- attribute

→ from context node, execute query:

\[
\text{axis}::* \n\]
Location Steps & Paths

Axis = a sequence of nodes  (is evaluated relative to context-node)

→ from context node, execute query:

axis::*

Forward Axes:
→ self
→ child
→ descendant-or-self
→ descendant
→ following
→ following-sibling

In doc order

Backward Axes:
→ parent
→ ancestor
→ ancestor-or-self
→ preceding
→ preceding-sibling
→ attribute

reverse doc order
Location Steps & Paths

Axis = a sequence of nodes (is evaluated relative to context-node)

→ from context node, execute query:

axis::*

Forward Axes:
→ self
→ child
→ descendant-or-self
→ descendant
→ following
→ following-sibling

In doc order

Backward Axes:
→ parent
→ ancestor
→ ancestor-or-self
→ preceding
→ preceding-sibling
→ attribute

reverse doc order
Location Steps & Paths

Axis = a sequence of nodes (is evaluated relative to context-node)

Forward Axes:
- self
- child
- descendant-or-self
- descendant
- following
- following-sibling

In doc order

Backward Axes:
- parent
- ancestor
- ancestor-or-self
- preceding
- preceding-sibling
- attribute

reverse doc order

→ from context node, execute query:
axis::*
Location Steps & Paths

Axis = a sequence of nodes  (is evaluated relative to context-node)

Forward Axes:
- self
- child
- descendant-or-self
- descendant
- following
- following-sibling

In doc order

Backward Axes:
- parent
- ancestor
- ancestor-or-self
- preceding
- preceding-sibling
- attribute

reverse doc order
A Location Path $\mathbf{P}$ is a sequence of Location Steps

$$a_1 :: n_1 \ [ F_1 \_1 ] \ [ F_1 \_2 ] \ ... \ [ F_1 \_n1 ]$$
$$a_2 :: n_2 \ [ F_2 \_1 ] \ [ F_2 \_2 ] \ ... \ [ F_2 \_n2 ]$$
$$a_m :: n_m \ [ F_m \_1 ] \ [ F_m \_2 ] \ ... \ [ F_m \_nm ]$$

$S_0 =$ initial sequence of context-nodes

(1) (to each) context-node $N$ in $S_0$, apply axis $a_1$: gives sequence $S_1$ of nodes
(2) remove from $S_1$ any node $M$ for which
    $\rightarrow$ test $n_1$ evaluates to false
    $\rightarrow$ any of filters $F_1 \_1, ..., F_1 \_n1$ evaluate to false.

Proceed similarly for $S_1$ and $a_2$, et cetera

Finally, obtain $Sm =$ result sequence of query $\mathbf{P}$. 
More Details

XPath Query Q

XML document D

Evaluate Q on D (in XPath data model)

sequence of result nodes

NOT correct (at least not for intermediate expr’s)

An expression evaluates to an object, which has one of the following four basic types

- node-set (an unordered collection of nodes w/o duplicates)
- boolean (true or false)
- number (a floating-point number)
- string (a sequence of UCS characters)
Attribute Axis

How to
- test attribute nodes

Examples

//attribute::*

Result:
b="1"
a="1"
a="2"
a="1.0"

Remember, these are just NODEs.

//attribute::*/. gives same result

And //attribute::a/.. gives

<b a="1"><d/></b><c a="2"><d/></c><d/></b>
<c a="2"><d/></c><d/></c>
<c a="1.0"><b/></c><d/></c>
Attribute Axis & Value Tests

How to
→ test attribute values

Examples

// *[attribute::a=1]

(selects the two red nodes)
Attribute Axis & Value Tests

How to
→ test attribute values

Examples

// *[attribute::a=1]
(selects the two red nodes)

Watch out

// *[attribute::a="1"]

// *[attribute::a="1.0"]

string comparison
Attribute Axis & Value Tests

How to
→ test attribute values

Examples

//*[attribute::a=1]
(selects the two red nodes)

Watch out

//*[attribute::a="1"]
//*[attribute::a="1.0"]

attribute:: is abbreviated by @

//@a=!"1" selects both c-nodes
//@a>1 selects only left c-node
//@a=//=@b selects what?? (hint: "=" is string comp. here)
Tests in Filters

- `or`
- `and`
- `=, !=`
- `<=, <, >=, >`

The operators are all left associative.
For example, `3 > 2 > 1` is equivalent to `(3 > 2) > 1`, which evaluates to `false`.

But, `3 > 2 > 0.9` evaluates to `true`.

For two strings `u, v`

\[
\begin{align*}
\text{if } & \quad \text{Always return } \textbf{false}! \\
\text{u<=v} & \\quad \Rightarrow \text{Unless both } u \text{ and } v \text{ are numbers.} \\
\text{u<v} & \\
\text{u>=v} & \\
\text{u>v}
\end{align*}
\]

\[\text{["1.0">="1"] evaluates to } \textbf{true}.\]
Text Nodes

How
→ test text nodes & values

//text()

Result:
foo
foo
Bar

//*[text()="foo"]

Result: the two red nodes

Question:

What is the result for
//*[text()=./b/text()]
Useful Functions (Strings)

The string-value of an element node is the concatenation of the string-values of all text node descendants in document order.

```
strtotime("foo")
strtotime("foobar")
strtotime("foofoobar")
```

```

```
Useful Functions (Strings)

The string-value of an element node is the concatenation of the string-values of all text node descendants in document order.

```javascript
/* [.=="foo"] */
/* [.=="foobar"] */

→ concaten(st_1, st_2, ..., st_n) = st_1 st_2 ... st_n
→ startswith("abcd", "ab") = true
→ contains("bar", "a") = true
→ substring-before("1999/04/01","/") = 1999.
→ substring-after("1999/04/01","19") = 99/04/01
→ substring("12345",2,3) = "234"
→ string-length("foo") = 3
```

What is the result to this:  /* [contains(.,"bar") ] */
Useful Functions (Strings)

The string-value of an element node is the concatenation of the string-values of all text node descendants in document order.

```xml
<element .="foo"/>
<element .="foobar"/>
```

→ `normalize-space(" foo bar a ") = "foo bar a"

→ `translate("bar","abc","ABC") = BAr`

returns the first argument string with occurrences of characters in the second argument string replaced by the character at the corresponding position in the third argument string.
Useful Functions (on Node Sets)

→ `count`
Counts number or results

/`a[count(//*[text()=//b/text()]]=2]`

What is the result?
Useful Functions (on Node Sets)

→ count
Counts number or results

/a[count(/*[text()=//b/text()] */) = 2]

What is the result?

Same result as:

/a[count(/*[text()="foo"] */) > count(/*[text()="bar"] */) ]
Useful Functions (on Node Sets)

count
Counts number or results

/a[count(//*[text()=./b/text()])=2]

What is the result?

Same result as:

/a[count(//*[text()="foo"])
> count(//*[text()="bar"])]

What is the result for:

//c[count(b)=0]

(same as //c[not(b)])
Useful Functions (on Node Sets)

→ `last()`
returns context-size from the evaluation context

→ `position()`
Returns context-position from the eval. context

```xml
//a is abbreviation for descendant-or-self::node()/child::a
```

```
/*[position()=2]*/
```
Useful Functions (on Node Sets)

- `last()`
  - Returns context-size from the evaluation context

- `position()`
  - Returns context-position from the evaluation context

```
// *[position()=2]
// *[position()=2 and ../../a]
```

```
  a = "1"
    b = "1"
      d
    d
  b
    a = "2"
      d
    c = "1.0"
      d
    d
  c
    b
      d
    d
  "foo"
  "foo"
  "bar"
```
Useful Functions (on Node Sets)

→ `last()`
returns context-size from the evaluation context

→ `position()`
Returns context-position from the eval. context

Which nodes?

//*[position()=2]

//*[position()=2 and ../..//a]
Useful Functions (on Node Sets)

→ `last()`
returns context-size from the evaluation context

→ `position()`
Returns context-position from the eval. context

Which node?
// *[position()=2]
// *[position()=2 and ..../..../a]
Useful Functions (on Node Sets)

→ `last()`
returns context-size from the evaluation context

→ `position()`
Returns context-position from the evaluation context

\[
\begin{align*}
&\text{a = "1"} \\
&\text{b = "1"} \\
&\text{c = "1.0"} \\
&\text{d = "2"}
\end{align*}
\]

//*[position()=2]

//*[position()=2 and ../..//a]

//*[position()=last()]
Useful Functions (on Node Sets)

→ `last()`
returns context-size from the evaluation context

→ `position()`
Returns context-position from the eval. context

```
// *[position()=2]
// *[position()=2 and ..//..//a]

// *[position()=last()-1]
```
Useful Functions (on Node Sets)

last()
returns context-size from the evaluation context

position()
Returns context-position from the evaluation context

//* [position()=2]
//* [position()=2 and .. ../../../a]

//* [position()=last()-1
and ./text()=“foo”]
Useful Functions (on Node Sets)

→ last()
returns context-size from the evaluation context

→ position()
Returns context-position from the eval. context

// *[position()=2]
// *[position()=2 and ../../a]

Useful:
child::*[self::*chapter or self::*appendix][position()=last()]
selects the last chapter or appendix child of the context node
**Useful Functions (on Node Sets)**

→ last()
returns context-size from the evaluation context

→ position()
Returns context-position from the eval. context

```xml
tree
  a
    b
      d
      c
        d
        b
d
b
  c
    a
      d
      b
d
```

```xml
// *[position()=2]
// *[position()=2 and ../../*[a]]
// *[position()=last()-1 and ./text()="foo"]

**// *[position()=1]// *[position()=2]// *[position()=2]**

→ allows absolute location of any node (a la Dewey)
Useful Functions (on Node Sets)

→ last()
returns context-size from the evaluation context

→ position()
Returns context-position from the eval. context

```
/*[position()=2]

/*[position()=2 and ..../..../a]

/*[position()=last()-1
and ./text()="foo"]

/*[position()=1]//*[position()=2]//*[position()=2]
```

Abbreviation:  *//[1]/*[2]/*[2]
Useful Functions (on Node Sets)

→ `last()`
  returns context-size from the evaluation context

→ `position()`
  Returns context-position from the eval. context

```xml
/*[position()=2]
/*[position()=2 and ../../a]
/*[position()=last()-1
  and ./text()="foo"]

/*[position()=1]//*[position()=2]//*[position()=2]
```

Abbreviation:  */*[1]/*[2]/*[2]  ➔ What is result for  //*[./*[2]/*[2]]
Useful Functions (on Node Sets)

→ `last()`
returns context-size from the evaluation context

→ `position()`
Returns context-position from the eval. context

How do you select the last 20 book-children of books?
Useful Functions (on Node Sets)

→ \texttt{last()}
returns context-size from the evaluation context

→ \texttt{position()}
Returns context-position from the eval. context

![Diagram of a tree structure with nodes labeled as books, book, and last 20. The text asks how to select the last 20 book-children of books.]

How do you select the last 20 book-children of books?

/\texttt{books/book[position()}>last()-20]
A mentally unstable Vietnam war veteran works as a night-time taxi driver in New York City where the perimeter of his world continually shrinks...
XPath Query Evaluation

How to implement?

How expensive? complexity?

What are the most difficult queries?
END
Lecture 17