Chapter 5
Query Languages: XPath

- Network Data Model: no query language
- SQL – only for a flat data model, but a “nice” language
  (easy to learn, descriptive, relational algebra as foundation, clean theory, optimizations)
- OQL: SQL with object-orientation and path expressions
- Lorel (OEM): extension of OQL
- F-Logic: navigation in a graph by path expressions with additional conditions
  descriptive, complex.

REQUIREMENTS ON AN XML QUERY LANGUAGE

- suitable both for databases and for documents
- declarative: binding variables and using them
  - rule-based, or
  - SQL-style clause-based (which is in fact only syntactic sugar)
- binding variables in the rule body/selection clause:
  suitable for complex objects
  - navigation by path expressions, or
  - patterns
- generation of structure in the rule head/generating clause

Evolution of XPath

- when defining a query language, constructs are needed for addressing and accessing individual elements/attributes or sets of elements/attributes.
- based on this *addressing mechanism*, a clause-based language is defined.

Early times of XML (1998)
different navigation formalisms of that kind:

- XSL Patterns (inside the stylesheet language)
- XQL (XML Query Language)
- XPointer (referencing of nodes/areas in an XML document)

used all the same basic idea with slight differences in the details:

- paths in UNIX notation
- conditions on the path

  /mondial/country[@car_code="D"]/city[population > 100000]/name

5.1 XPath – the Basics

1999: specification of the navigation formalism as *W3C XPath*.

- Base: UNIX directory notation

    in a UNIX directory tree: /home/dbis/Mondial/mondial.xml
    in an XML tree: /mondial/country/city/name

  Straightforward extension of the URL specification:
  http://.../dbis/Mondial/mondial.xml#mondial/country/city/name [XPointer until 2002]
  http://.../dbis/Mondial/mondial.xml#xpointer(mondial/country/city/name) [XPointer now]

- W3C: XML Path Language (XPath), Version 1.0
  `http://www.w3.org/TR/xpath`
- W3C: XPath 2.0 and XQuery 1.0
  `http://www.w3.org/TR/xquery`
- Tools: see Web page
  - XML (XQuery) database system "eXist"
  - lightweight tool "saxonXQ" (XQuery)

XPATH: NAVIGATION, SIMPLE EXAMPLES

XPath is based on the UNIX directory notation:

- /mondial/country
  addresses all country elements in MONDIAL,
  the result is a set of elements of the form

  <country code="..."> ... </country>

- /mondial/country/city
  addresses all city elements, that are direct subelements of country elements.
- /mondial/country//city
  adresses all city elements that are subelements (in any depth) of country elements.
- //city
  addresses all city elements in the current document.
- wildcards for element names:
  /mondial/country/*/city
  addresses all city elements that are grandchildren of country elements
  (different from /mondial/country//city !)

... and now systematically:

XPATH: ACCESS PATHS IN XML DOCUMENTS

- Navigation paths

  /step/step/.../step

  are composed by individual navigation steps,
- the result of each step is a set of nodes, that serve as input for the next step.
- each step consists of

  *axis::nodetest[condition]**

  - an axis (optional),
  - a test on the type and the name of the nodes,
  - (optional) predicates that are evaluated for the current node.
- paths are combined by the "/"-operator
- additionally, there are function applications
- the result of each XPath expression is a *sequence* of nodes or literals.

XPath: Axes

Starting with a *current node* it is possible to navigate in an XML tree to several "directions" (cf. xmllint's "cd"-command).

In each navigation step

path/*axis*::*nodetest*[*condition*]/*path*

the *axis* specifies in which direction the navigation takes place. Given the set of nodes that is addressed by *path*, for *each* node, the step is evaluated.

- Default: child axis: child::country ≡ country.
- Descendant axis: all sub-, subsub-, ... elements:
  country/descendant::city
  selects all city elements, that are contained (in arbitrary depth) in a country element.
  Note: *path*//city actually also addresses all these city elements, but "//" is *not* the exact abbreviation for "/descendant::" (see later).

XPath: Axes

... another important axis:

- attribute axis:
  attribute::car_code ≡ @car_code
  wildcard for attributes: attribute::* selects all attributes of the current context node.
- and a less important:
  self axis: self::city ≡ ./city
  selects the current element, *if* it is of the element type city.

for the above-mentioned axes there are the presented abbreviations. This is important for *XSL patterns* (see Slide 301):

XSL (match) patterns are those XPath expressions, that are built *without* the use of "axis::" (the abbreviations are allowed).

XPath: Axes

Additionally, there are axes that do not have an abbreviation:

- parent axis: //city[name="Berlin"]/parent::country
  selects the parent element of the city element that represents Berlin, *if* this is of the element type country.
  (*only* the parent element, not all ancestors!)
- ancestor: all ancestors:
  //city[name="Berlin"]/ancestor::country selects all country elements that are ancestors of the city element that represents Berlin (which results in the Germany element).
- siblings: following-sibling::..., preceding-sibling::...
  for selecting nodes on the same level (especially in ordered documents).
- straightforward: "descendant-or-self" and "ancestor-or-self".
  Note: The popular short form country//city is defined as country/descendant-or-self::node()/city.
  This makes a difference only in case of *context functions* (see Slide 216).

XPath: Axes for use in document-oriented XML

- following: all nodes after the context node in document order, excluding any descendants and excluding attribute nodes
- preceding: all nodes that are before the context node in document order, excluding any ancestors and excluding attribute nodes and namespace nodes

Note: For each element node x, the ancestor, descendant, following, preceding and self axes *partition* a document (ignoring attribute nodes): they do not overlap and together they contain all the nodes in the document.

Example:

Hamlet: what is the next speech of Lord Polonius after Hamlet said "To be, or not to be"?
(note: this can be in a subsequent scene or even act)

Exercise:

Provide equivalent characterizations of "following" and "preceding"

i) in terms of "preorder" and "postorder",

ii) in terms of other axes.

XPath: Nodetest

- The *nodetest* constrains the node type and/or the names of the selected nodes
- "*" as wildcard: //city[name="Berlin"]/child::*
  returns all children.
- test if something is a node: //city[name="Berlin"]/descendant::node()
  returns all descendant nodes.
- test if something is a node: //city[name="Berlin"]/descendant::element()
  returns all descendant *elements* (note: not the text nodes).
- test if something is a text node: //city[name="Berlin"]/descendant::text()
  returns all descendant *text nodes*.
  //city[name="Berlin"]/population/text()
  returns the text contents of the population element.
- test for a given element name:
  //country[name="Germany"]/descendant::element(population)
  or short form:
  //country[name="Germany"]/descendant::population
  returns all descendant *population elements*.

XPath: Tests

In each step

path/*axis*::*nodetest*[*condition*]/*path*

condition is a predicate over XPath expressions.

- The expression selects only those nodes from the result of *path*/*axis*::*nodetest* that satisfy *condition*. *condition* contains XPath expressions that are evaluated relative to the current *context node* of the respective step.

  //country[@car_code="D"]
  returns the country element whose car_code attribute
  has the value "D"
- When comparing an element with something, the text() method is applied implicitly:

  //country[name = "Germany"] is equivalent to
  //country[name/text() = "Germany"]
- If the right hand side of the comparison is a number, the comparison is automatically evaluated on numbers:

  //country[population > 1000000]

XPATH: TESTS (CONT'D)

- boolean connectives "and" and "or" in *condition*:

  //country[population > 100000000 and @area > 5000000]
  //country[population > 100000000 or @area > 5000000]

- boolean "not" is a *function*:

  //country[not (population > 100000000)]

- XPath expressions in *condition* have existential semantics:
  The *truth value* associated with an XPath expression is *true*, if its result set is non-empty:

  //country[inflation]
  selects those countries that have a subelement of type inflation.

  ⇒ formal semantics: a path expression has

  – a semantics as a result set, and

  – a truth value!

XPATH: TESTS (CONT'D)

- XPath expressions in *condition* are not only "simple properties of an object", but are path expressions that are evaluated wrt. the current context node:

  //city[population/@year='95']/name

- Such comparisons also have existential semantics:

  //country[.//city/name='Cordoba']/name
  returns the names of all countries, in which a city with name Cordoba is located.

  //country[not (.//city/name='Cordoba')]/name
  returns the names of those countries where no city with name Cordoba is located.

  Remark:
  Note that descendant::city (relative) and //city (absolute) have different effect:

  //country[//city/name='Cordoba']/name
  returns the names of *all* countries (the filter just checks if there is *some* city with name Cordoba in the document).

XPath: Evaluation Strategy

- Input for each navigation step: A set of nodes (*context*)
- each of these nodes is considered separately for evaluation of the current step
- and returns zero or more nodes as (intermediate) result.
  This intermediate result serves as context for the next step.
- finally, all partial results are collected and returned.

Example

- conditions can be applied to multiple steps

  //country[population > 10000000]
  //city[@is_capital and population > 1000000]
  /name/text()

  returns the names of all cities that have more than 1,000,000 inhabitants and that are the capital of a country that has more than 10,000,000 inhabitants.

Absolute and Relative Paths

So far, conditions were always evaluated only "local" to the current element on the main navigation path.

- Paths that start with a name are *relative* paths that are evaluated against the current context node (used in conditions):

  //city[name = "Berlin"]

- Semijoins: comparison with results of independent "subqueries":
  Paths that start with "/" or "//" are absolute paths:

  //country[population > //country[@car_code='B']/population]/name

  returns the names of all countries that have more inhabitants than Belgium

- conflict between "//" for absolute paths and for descendant axis:

  //country[.//city/name="Berlin"]
  (equivalent: //country[descendant::city/name="Berlin"])

  can be used for starting a relative path.

XPath: Functions

Input: a node/value or a set of nodes/values.
Result: in most cases a value; sometimes one or more nodes.

- dereferencing (see Slide 207)
- access to text value and node name (see Slide 210)
- aggregate functions count(*node_set*), sum (*node_set*)

  count(/mondial/country)

  returns the number of countries.
- context functions (see Slide 216)
- access to documents on the Web:

  doc("*file or url*")/*path*
  doc('http://www.dbis.informatik.uni-goettingen.de/index.html')//text()

  (for querying external HTML documents, consider use of namespaces as described on Slide 227 - nodetests work only with namespace!)
- see W3C document *XPath/XQuery Functions and Operators*

IDREF Attributes

- ID/IDREF attributes serve for expressing cross-references
- can be resolved by semi-joins:
  (similar to foreign keys in SQL)

  //city[@id = //organization[@abbrev="EU"]/@headq]

  SQL equivalent (uncorrelated subquery):

```
SELECT *
FROM city
WHERE (name, country, province) IN
    (SELECT city, country, province
     FROM organization
     WHERE abbrev = 'EU')
```

... not a really elegant way in a graph-based data model ...

XPath: Dereferencing

Access via "keys"/identifiers

The function id(*string**) returns all elements (of the current document) whose id's are enumerated in *string**:

- id("D") selects the element that represents Germany
  (country/@car_code is declared as ID)
- id(//country[car_code="D"]/@capital)
  yields the element node of type city that represents Berlin.

This notation is hard to read if multiple dereferencing is applied, e.g.

    id(id(id(//organization[abbrev='IOC']/@headq)/@country)/@capital)/name

Alternative syntaxes:

    //organization[abbrev='IOC']/id(@headq)/id(@country)/id(@capital)/name
    //organization[abbrev='IOC']/@headq/id(.)/@country/id(.)/@capital/id(.)/name

XPath: Dereferencing (Cont'd)

Analogously for multi-valued reference attributes (IDREFS):

- //country[@car_code="D"]/@memberships
  returns "org-EU org-NATO ..."
- id(//country[@car_code="D"]/@memberships)
  //country[@car_code="D"]/id(@memberships)
  returns the set of all elements that represent an organisation where Germany is a member.
- id(//organization[abbrev="EU"]/members/@country)
  //organization[abbrev="EU"]/members/id(@country)
  returns all countries that are members (of some kind) in the EU.

Aside: Dereferencing by Navigation [Currently not supported]

Syntax:

attribute::*nodetest*⇒*elementtype*

Examples:

- //country[car_code="D"]/@capital⇒city/name
  yields the element node of type city that represents Berlin.
- //country[car_code="D"]/@memberships⇒organization
  yields elements of type organization.
- Remark: this syntax is not supported by all XPath Working Drafts:
  - XPath 1.0: no
  - has originally be introduced by Quilt (2000; predecessor of XQuery)
  - XPath 2.0: early drafts yes, later no
  - announced to be re-introduced later ...

XPath: string() Function

The *function* string() returns the string value of a node:

- straightforward for elements with text-only contents:
  string(//country[name='Germany']/population)
  Note: for these (and only for these!) nodes, text() and string() have the same semantics.
- for attributes: //country[name='Germany']/string(@area)
  Note: an attribute node is a name-value pair, not only a string (will be illustrated when constructing elements later in XQuery)!
  free-standing attribute nodes as result cannot be printed!
- the string() function can also be appended to a path; then the argument is each of the context nodes: //country[name='Germany']//name/string()
- the string value of a subtree is the concatenation of all its text nodes:
  //country[@name='Germany']/string()
  Note: compare with //country[@name='Germany']//text() which lists all text nodes.
- string() *cannot* be applied to node sequences: string(//country[name='Germany']//name) results in an error message.
  (see W3C XPath and XQuery Functions and Operators).

XPath: Some more Details on Comparisons

- in the above examples, all predicate expressions like [name="Berlin"] or [@car_code="D"] always *implicitly* compare the string value of nodes, e.g., here the string values of <name>Berlin</name> or attribute: (car_code, "D").

Usage of Numbers

- comparisons using > and < and a number literal given in the query implicitly cast the string values as *numeric* values.

  //city[population > 200000]

  returns the all cities with a population higher than 200,000.

  //city[population > '200000']

  returns the all cities with a population *alphabetically* "bigger" than 200,000, e.g., 3500, but not 1,000,000!

  //city[population > //city[name="Munich"]/population]

  does *not* recognize that numerical values are meant:
  All cities with population lexically bigger than "1244676" are returned.

  //city[population > //city[name="Munich"]/population/number()]

  It is sufficient to apply the number() casting function (see later) to one of the operands.

XPath: Comparison between Nodes

Usage of Node Identity

- as seen above, the "=" predicate uses the string values of nodes.

  In most cases, this is implicitly correct:

  Consider the following query: "Give all countries whose capital is the headquarter of an organization":

  //country[id(@capital)=//organization/id(@headq)]/name

  Compares the overall string values of city elements, e.g., "Brussels 4.35 50.8 951580".

- but for empty nodes, the result is not as intended ...

Comparison by Node Identity: “a is b”

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mondial-simple SYSTEM "mondial-simple.dtd">
<mondial-simple>
 <country car_code="D" capital="Berlin"/>    <city name="Berlin"/>
 <country car_code="B" capital="Brussels"/>  <city name="Brussels"/>
 <organization name="EU" headq="Brussels"/>
</mondial-simple>
```

[Filename: XPath/node-comparison.xml]

- the query //country[id(@capital)=//organization/id(@headq)]/string(@car_code) yields “D” and “B”.
- Comparison by *node identity* is done by “is”:
  //country[id(@capital) is //organization/id(@headq)]/string(@car_code)
  (“is” is only provided since XPath 2.0.)
- Aside: “deep equality” of nodes can be tested with the predicate deep-equal(x, y).
  (by this, two subtrees can be checked to have the same structure+contents)

XPath: Predicates and Operations on Strings

- concat(string, string, string*)
- startswith(string, string)
  //city[starts-with(name,’St.’)]/name
- contains(string, string)
  //city[contains(name,’bla’)]/name
- substring-before(string, string, int?)
- substring-after(string, string, int?)
- substring(string, int, int): the substring consisting of i_2 characters starting with the i_1th position.

XPath: Name Function

- the function name() returns the element name of the current node:
  - name(//country[@car_code='Germany']) or //country[@car_code='Germany']/name()
  - //*[name='Monaco' and not (name()='country')] yields only the city element for Monaco.

XPath: IDREF Function

- the function idref(string*) returns all nodes that have an IDREF value that refers to one of the given strings (note that the results are attribute nodes):
  idref('D')/parent::*/name yields the name elements of all "things" that reference Germany.

XPath: Context Functions

- All functions retain the order of elements from the XML document (document order).
- the position() function yields the position of the current node in the current result set.

  /mondial/country[position()=6]

  Abbreviation: [x] instead of [position()=x]; [-1] yields the last node:

  /mondial/country[population > 1000000][6]

  selects the 6th country that has more than 1,000,000 inhabitants (in document order, not the one with the 6th highest population!)

  /mondial/country[6][population > 1000000]

  selects the 6th country, if it has more than 1,000,000 inhabitants.
- the last() function returns the position of the last elements of the current sub-results, i.e., the size of the result.

  //country[position()=last()]

XPath: Context Functions (Cont'd)

- consider again the "//" abbreviation (cf. Slide 197):
  - /mondial/descendant::city[18] selects the 18th city in the document,
  - /mondial/descendant-or-self::node()/city[18] selects each city which is the 18th child of its parent (country or province).
    (note that some implementations are buggy in this point ...)
- Example queries against mondial.xml and hamlet.xml.

XPath: Forward- and Backward Axes

- the result of each query is a *sequence of nodes*
- document order (and final results): forward
- context functions: forward or backward
- all axes enumerate results starting from the current node.
  - forward axes: child, descendant, following, following-sibling
  - backward axes: ancestor, preceding, preceding-sibling
    //table/preceding-sibling::h4//text()
    selects all preceding h4 elements (section headers).
    The result is -as always- output in document order
    //table/preceding-sibling::h4[1]//text()
    selects the last preceding section header (context function on backward axis)
  - undirected: self, parent, attribute (and namespace)
- only relevant for queries against document-oriented XML.

Extensions with XPath 2.0

- further string- and aggregate functions
- more complex path constructs (alternatives, parentheses)
  (//city|//country)[name='Monaco']
  /mondial/country/(city|(province/city))/name
- extended subscript operator:
  //country[population > 1000000][-3]
  //country[population > 1000000][5-10]
  //country[population > 1000000][1,5-10,-3]
- ANY and ALL semantics for *condition*:
  //country[ALL city/population > 1000000]
  //country[ANY city/population > 1000000]
  (countries where all/at least one city has more than 1000000 inhabitants)
- extending the language to more than usual navigation ...
- alignment of the whole XML world (XPath, XQuery) with datatypes (data model and XML Schema)

5.2 Aside: Namespaces

The names in an XML instance (i.e., tag names and the attribute names) actually consist of two parts:

- localpart + namespace (which can be empty, as in the previous examples)

Use of Namespaces

- a namespace is similar to a language: defining a set of names and sometimes having a DTD (if intended as an XML vocabulary).
- e.g. "mondial:city", "bib:book", "xhtml:tr" "dc:author", "xsl:template" etc.
- used for distinguishing coinciding element names in different application areas.
- each namespace is associated with a URI (which can be a "real" URL), and abbreviated by a *namespace prefix* in the document.
- e.g., associate the namespace prefix `xhtml` with url `http://www.w3.org/1999/xhtml`. these things will become clearer when investigating the RDF, RDFS, and Semantic Web Data Models.

USAGE OF NAMESPACES IN XML DOCUMENTS

- each element can have (or can be in the scope of) multiple *namespace declarations* (represented by a node in the data model, similar to an attribute node).
- namespace declarations are inherited to subelements
- the element/tag name and the attribute names can then use one of the declared namespaces.
  By that, every element can have one *primary namespace* and "knows" several others.

Alternatives:

1. node has no namespace (e.g. mondial),
2. document declares a default namespace (for all elements that do not get an explicit one (often in XHTML pages)),
3. elements have an explicit namespace (multiple namespaces allowed in a document; e.g. an XSL document that operates with XHTML markup and "mondial:" nodes).

- (2) and (3) are semantically equivalent.

... see next slides.

EXPLICIT NAMESPACE IN AN XML DOCUMENT

```
<xh:html xmlns:xh="http://www.w3.org/1999/xhtml">
 <xh:body>
  <xh:h3>Header</xh:h3>
  <xh:a xh:href="http://www.informatik.uni-goettingen.de">IFI</xh:a>
 </xh:body>
</xh:html>
```

[Filename: XML-DTD/xhtml-expl-namespace.xml]

```
declare namespace ht = "http://www.w3.org/1999/xhtml";
/ht:html//ht:a/string(@ht:href)
```

[Filename: XPath/xhtml-query.xq]

- Note: the namespace *must* be used in the query,
  i.e., "ht:html" is different from just "html"
- more accurate, it means something like `<{http://www.w3.org/1999/xhtml}html>...</...>` since not the chosen namespace prefix matters, but only the URI assigned to it.

Two Explicit Namespaces in an XML Document

- "Dublin Core" defines a vocabulary for metadata description of resources (here: of XML documents); cf. http://dublincore.org/documents/dces/

```
<xh:html xmlns:xh="http://www.w3.org/1999/xhtml"
         xmlns:dc="http://purl.org/dc/elements/1.1/">
 <dc:creator>John Doe</dc:creator>
 <dc:date>1.1.2000</dc:date>
 <xh:body> ... </xh:body> </xh:html>
```

[Filename: XML-DTD/xhtml-expl-namespaces.xml]

```
declare namespace ht = "http://www.w3.org/1999/xhtml";
declare namespace dc = "http://purl.org/dc/elements/1.1/";
/ht:html//dc:creator/text()
```

[Filename: XPath/xhtml-dc-query.xq]

- the document is *not* valid wrt. the XHTML DTD since it contains additional "alien" elements.
  (combination of languages is a problem in XML – this is better solved in RDF/RDFS)
- in RDF, dc:creator from above expands to the URI
  http://purl.org/dc/elements/1.1/creator.

Default Namespaces in an XML Document

- a Default Namespace can be assigned to an element (and inherited to all its subelements where it is not overwritten):

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:dc="http://purl.org/dc/elements/1.1/">
 <dc:creator>John Doe</dc:creator>
 <date xmlns="http://purl.org/dc/elements/1.1/">1.1.2000</date>
 <body> ... </body> </html>
```

[Filename: XML-DTD/xhtml-def-namespaces.xml]

```
declare namespace ht = "http://www.w3.org/1999/xhtml";
declare namespace dc = "http://purl.org/dc/elements/1.1/";
/ht:html/dc:date/text()
```

[Filename: XPath/xhtml-dc-def-query.xq]

Namespaces and Attributes

- Namespaces are *not* inherited to attributes in any case. Attributes *must* have explicit namespace prefixes

```
<ht:html xmlns:ht="http://www.w3.org/1999/xhtml">
 <ht:body>
  <ht:a ht:href="1">IFI</ht:a>
  <x:a xmlns:x="http://www.w3.org/1999/xhtml" href="2">IFI</x:a>
  <a xmlns="http://www.w3.org/1999/xhtml" href="3">IFI</a>
 </ht:body> </ht:html>
```

[Filename: XML-DTD/namespaces-attr.xml]

```
declare namespace ht = "http://www.w3.org/1999/xhtml";
/ht:html//ht:a/@ht:href/string()
```

[Filename: XPath/namespaces-attr-query.xq]

- only “1” is returned,
- the query /ht:html//ht:a/@href/string() returns “2” and “3”.

Declaring Namespaces in the DTD Document

- introduce default namespace in the DTD as attribute of the root element (e.g. in XHTML):

```
<!ELEMENT html (head, body)>
<!ATTLIST html
  xmlns  %URI;  #FIXED 'http://www.w3.org/1999/xhtml' >
```

- XHTML instance:

  <html *xmlns=“http://www.w3.org/1999/xhtml”*> <body> ... </body></html>

- introduce explicit namespaces as attribute of the root element (e.g. in XHTML):

  <!ELEMENT html (head, body)>
  <!ATTLIST html xmlns:xh %URI; #FIXED 'http://www.w3.org/1999/xhtml' >

  This is used with RDF/XML in the Semantic Web

Example: Querying XHTML in Presence of Namespaces

Sample XHTML files:

- DBIS Web pages:

```
declare namespace h = "http://www.w3.org/1999/xhtml";
doc('http://www.dbis.informatik.uni-goettingen.de/')//h:li/h:a/@href/string()
```

[Filename: XPath/web-queries.xq]

- DBIS WWW2002 paper: in the local exist at /db/xmlcourse/xlink.htm

```
declare namespace ht = "http://www.w3.org/1999/xhtml";
doc('/db/xmlcourse/xlink.htm')//ht:h1
```


5.3 XPath: The Limits

- addressing only sets of nodes
- not "give all pairs of ..."
- the highest mountain in Africa:

```
doc('mondial.xml')//mountain[
  id(id(located/@country)/encompassed/@continent)/name='Africa'
  and
  not (height <
   //mountain[
    id(id(located/@country)/encompassed/@continent)/name='Africa']/height)]
 /name
```

[Filename: XPath/highestmountain.xq]

... comparison only by semijoins in the condition.

- for *each* continent, give the highest mountain?
  not possible: two properties of the same object (height, continent) must be compared independently → requires variable binding

5.4 XPath: Conclusion

What can XPath do?

Comparison with relational operators

- selection: yes (selection of values and of (sub)structures)
- projection/reduction: no. Only complete nodes can be selected
- join/combination: no. Only semi-joins can be expressed in the conditions

Other functionality:

- correlated subqueries: inside the conditions as semijoins
- restructuring of the results: no
- only following a “main path” for navigating to nodes (including semijoins)

⇒ only a fragment of a query language for addressing nodes.
  - compared with SQL, XPath is only a unary “FROM” clause!
  - XQL (Software AG, 1998/1999) for some time followed (as one of the predecessors of XPath) an approach to add join variables and constructs for projection and restructuring/grouping to the path language.

IMPORTANCE OF XPATH IN THE XML-WORLD

- adressing mechanism for nodes in XML documents
- navigation in the tree structure
- serves as base for different concepts:
  - XQuery
  - XSL/XSLT: stylesheets, transformation language
  - other query languages
  - XML Schema
  - XPointer/XLink