

NIE-PDB: Advanced Database Systems

<http://www.ksi.mff.cuni.cz/~svoboda/courses/221-NIE-PDB/>

Lecture 3

XML Databases: XPath, XQuery

Martin Svoboda

martin.svoboda@fit.cvut.cz

4. 10. 2022

Charles University, Faculty of Mathematics and Physics

Czech Technical University in Prague, Faculty of Information Technology

Lecture Outline

XPath and XQuery

- Data model
- Query expressions
 - **Path** expressions
 - **Comparison** expressions
 - Direct and computed **constructors**
 - **FLWOR** expressions
 - **Conditional** expressions
 - **Quantified** expressions

Introduction

XPath = *XML Path Language*

- **Navigation and selection of nodes**
- Versions: 1.0 (1999), 2.0 (2010), 3.0 (2014), **3.1** (March 2017)
- W3C recommendation
 - <https://www.w3.org/TR/xpath-31/>

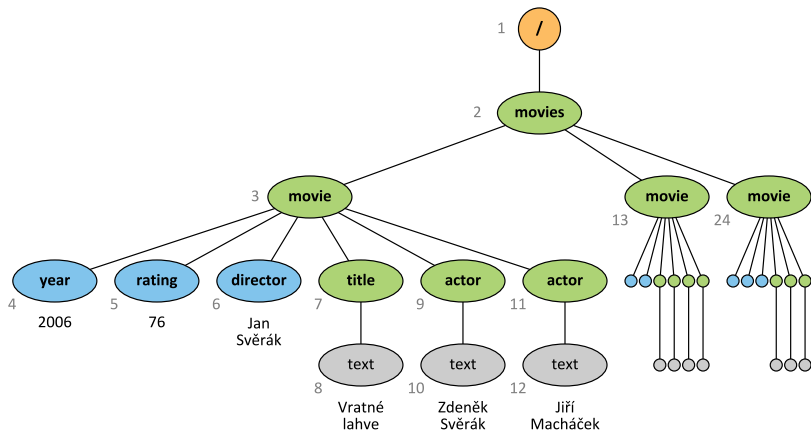
XQuery = *XML Query Language*

- **Complex queries and transformations**
- Contains XPath
- Versions: 1.0 (2007), 3.0 (2014), **3.1** (March 2017)
- W3C recommendation
 - <https://www.w3.org/TR/xquery-31/>

Sample Data

```
<?xml version="1.1" encoding="UTF-8"?>
<movies>
  <movie year="2006" rating="76" director="Jan Svěrák">
    <title>Vratné lahve</title>
    <actor>Zdeněk Svěrák</actor>
    <actor>Jiří Macháček</actor>
  </movie>
  <movie year="2000" rating="84">
    <title>Samotáři</title>
    <actor>Jitka Schneiderová</actor>
    <actor>Ivan Trojan</actor>
    <actor>Jiří Macháček</actor>
  </movie>
  <movie year="2007" rating="53" director="Jan Hřebejk">
    <title>Medvídek</title>
    <actor>Jiří Macháček</actor>
    <actor>Ivan Trojan</actor>
  </movie>
</movies>
```

Sample Data



Data Model

XDM = *XQuery and XPath Data Model* (XPath 2.0, XQuery 1.0)

- **XML tree** consisting of **nodes** of different kinds
 - Document, element, attribute, text, ...
- **Document order**
 - The order in which nodes appear in the XML file
 - I.e. nodes are numbered using a **pre-order depth-first traversal**
- Reverse document order

Query result

- Each query expression is evaluated to a **sequence**

Data Model

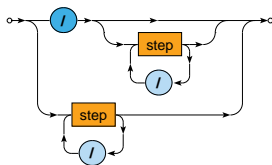
Sequence = ordered collection of **nodes** and/or **atomic values**

- Can be **mixed**
 - But usually just nodes, or just atomic values
- Are automatically **flattened**
 - E.g.: $(2, (), (4, 1, (3))), (1)) \Leftrightarrow (2, 4, 1, 3, 1)$
- Can be **empty**
 - E.g.: $()$
- Standalone items are treated as **singleton sequences**
 - E.g.: $1 \Leftrightarrow (1)$
- Can have **duplicate items**

Path Expressions

Path expression

- Allows for navigation within an XML tree
- Consists of **navigational steps**



- **Absolute** paths: start with /
 - Navigation starts at the **document node**
- **Relative** paths
 - Navigation starts at an implicitly specified context node

Path Expressions: Examples

Absolute path expressions

```
/
```

```
/movies
```

```
/movies/movie
```

```
/movies/movie/title/text()
```

```
/movies/movie/@year
```

Relative path expressions

```
actor/text()
```

```
@director
```

Path Expressions

Evaluation of a path expression P

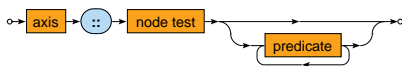
- ... with respect to the initial **context sequence** C

```
1 if  $P$  does not contain any step then
2   | return  $C$  (we already have the final result)
3 else (when  $P$  contains at least one step)
4   | let  $S$  be the first step and  $P'$  the remaining steps (if any)
5   | let  $C' = \langle \rangle$  be an empty sequence
6   | foreach context node  $u \in C$  do
7     | evaluate  $S$  with respect to  $u$  and add the selected
8     | items  $C'_u$  to  $C'$ 
9   | return evaluate  $P'$  with respect to  $C'$ 
```

Path Expressions: Steps

Navigational step

- Each step consists of up to 3 components

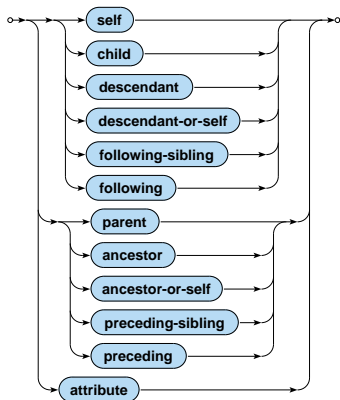


- **Axis**
 - **Relation of nodes** to be selected for a given context node u
- **Node test**
 - **Basic condition** these selected nodes must satisfy
- **Predicates**
 - **Advanced conditions** these nodes must further satisfy

Path Expressions: Axes

Axis

- Selects nodes that are reachable from a given context node



Path Expressions: Axes

child axis

- Selects **children** of a given context node
 - Note that attributes are not considered to be child nodes!
- Used as the **default axis** (when omitted)

```
/movies/child::movie
```

attribute axis

- Selects **attributes** of a given context node
 - Note that this is the only axis that can select attributes!

```
/movies/movie/attribute::year
```

self axis

- Selects just the current **context node**

Path Expressions: Axes

descendant(-or-self) axes

- Select all (non-attribute) **nodes in a subtree** of a given context node excluding / including itself

```
/descendant::actor/text()
```

parent axis

- Selects the **parent node** of a given context node

ancestor(-or-self) axes

- Select all **ancestors** of a given context node
 - I.e., the parent, the parent of the parent, and so on, until the document node, excluding / including the context node itself

Path Expressions: Axes

preceding-sibling and **following-sibling** axes

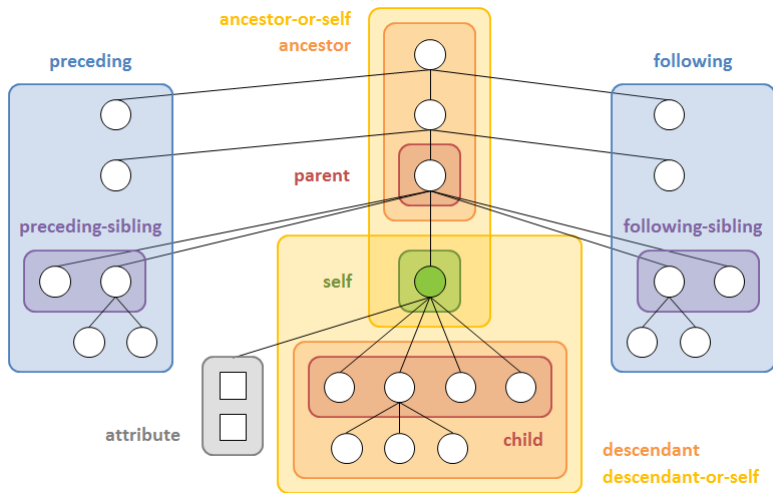
- Select all **siblings** of a given context node that occur before / after this context node in the document order

```
/descendant-or-self::movie/title/following-sibling::actor
```

preceding and **following** axes

- Select all (non-attribute) **nodes that occur before / after** a given context node in the document order, excluding nodes returned by the ancestor / descendant axis

Path Expressions: Axes



Path Expressions: Axes

Forward axes

- self, child, descendant(-or-self), following(-sibling)
- Nodes are returned in the **document order**

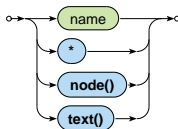
Reverse axes

- parent, ancestor(-or-self), preceding(-sibling)
- Nodes are returned in the **reverse document order**

Path Expressions: Node Tests

Node test

- Filters the nodes selected by the axis using a basic condition
 - Only **names and kinds** of nodes can be tested



name: elements / attributes with a given name

```
/movies
```

```
/movies/movie/attribute::year
```

Path Expressions: Node Tests

*****: all elements / attributes

```
/movies/*
```

```
/movies/movie/attribute::*
```

text(): all text nodes

```
/movies/movie/title/text()
```

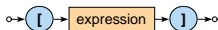
node(): all nodes

```
/movies/descendant-or-self::node()/actor
```

Path Expressions: Predicates

Predicates

- Additional filtering of the nodes based on advanced conditions



- When **multiple predicates** are provided...
 - They must all be satisfied
 - They are evaluated one by one, from left to right

Commonly used **conditions**

- Path existence tests, comparisons, position tests
- Logical expressions
- ...

Path Expressions: Predicates

Path existence tests

- Relative or absolute path expressions
 - Relative path expressions are evaluated with respect to the node for which a given predicate is tested
- Treated as true when evaluated to a **non-empty sequence**

```
/movies/movie[actor]
```

```
/movies/movie[actor]/title/text()
```

Comparisons

- General, value, or node comparison expressions

```
/descendant::movie[@year > 2000]
```

```
/descendant::movie[count(actor) ge 3]/title
```

Path Expressions: Predicates

Position tests

- Allow for filtering of items based on context positions
 - Numbered starting with 1
 - Always relative to the current context (intermediate result)
 - Base order is implied by the axis used

```
/descendant::movie/actor[position() = 1]
```

```
/descendant::movie[actor][position() = last()]
```

Logical expressions

- and, or, not connectives

```
/movies/movie[@year > 2000 and @director]
```

```
/movies/movie[@director][@year > 2000]
```

Path Expressions: Abbreviations

Omitted axis: the default `child` axis is assumed

```
/movies/movie/title
```

```
/child::movies/child::movie/child::title
```

Attributes: `@` \Leftrightarrow `attribute::`

```
/movies/movie/@year
```

```
/movies/movie/attribute::year
```

Descendants: `//` \Leftrightarrow `/descendant-or-self::node()/`

```
/movies//child::actor
```

```
/movies/descendant-or-self::node()/child::actor
```

Path Expressions: Abbreviations

Context item: `.` \Leftrightarrow `self::node()`

```
/movies/movie[.//actor]
```

```
/movies/movie[self::node()//actor]
```

Parent: `..` \Leftrightarrow `parent::node()`

Position tests: `[number]` \Leftrightarrow `[position() = number]`

```
/movies/movie/child::actor[2]
```

```
/movies/movie/child::actor[position() = 2]
```

```
/movies/movie[actor][last()]
```

```
/movies/movie[actor][position() = last()]
```


Path Expressions: Conclusion

Evaluation of path expressions

- Evaluated **from left to right, step by step**
 - Result of the entire expression is the **result of the last step**

Only one of the following can be returned...

- Sequence of **nodes**
 - Always sorted in the **document order**
 - **Duplicate nodes are removed**
 - Based on the identities of nodes
- Sequence of **atomic values**
 - The order as well as duplicate values are both preserved

⇒ the returned sequences will never be mixed

Comparison Expressions

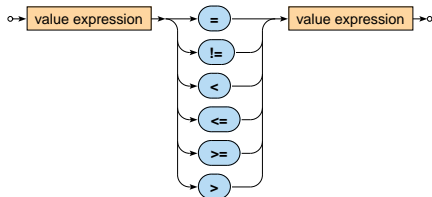
Comparisons

- **General** comparisons
 - Two sequences of values are expected to be compared
 - =, !=, <, <=, >=, >
 - E.g.: (0,1) = (1,2)
- **Value** comparisons
 - Two standalone values (singleton sequences) are compared
 - eq, ne, lt, le, ge, gt
 - E.g.: 1 lt 3
- **Node** comparisons
 - is – tests identity of nodes
 - <<, >> – test positions of nodes (preceding, following)
 - Similar behavior as in the case of value comparisons

Comparison Expressions

General comparisons (existentially quantified comparisons)

- Both the operands can be evaluated to **sequences of items** of any length



- The result is true if and only if **there exists at least one pair of individual items** satisfying a given relationship

Comparison Expressions: Examples

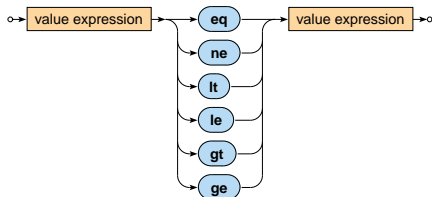
General comparisons

- $[(1) < (2)] = \text{true}$
- $[(1) < (1,2)] = \text{true}$
- $[(1) < ()] = \text{false}$
- $[(0,1) = (1,2)] = \text{true}$
- $[(0,1) \neq (1,2)] = \text{true}$

Comparison Expressions

Value comparisons

- Both the operands must be evaluated to singleton sequences



- Empty sequence** () is returned...
 - when at least one operand is evaluated to an empty sequence
- Type error** is raised...
 - when at least one operand is evaluated to a longer sequence

Comparison Expressions: Examples

Value comparisons

- $[(1) \text{ le } (2)] = \text{true}$
- $[(1) \text{ le } ()] = ()$
- $[(1) \text{ le } (1,2)] \Rightarrow \text{error}$
- $[(()) \text{ le } (1,2)] = ()$

Comparison Expressions

Value and general comparisons

- **Atomization of values** – applied automatically
 - Atomic values are preserved untouched
 - **Nodes are transformed to atomic values**
- In particular...
 - **Element** node is transformed to a string with concatenated text values it contains in the document order
 - E.g.: `<movie year="2006">Vratné lahve</movie>` is atomized to a string `Vratné lahve`
 - I.e., attribute values and element names are not included!
 - **Attribute** node is transformed to its value
 - **Text** node is transformed to its value

Comparison Expressions: Examples

Value and general comparisons

- `[[<a>5 eq 5]] = true`
- `[[<a>12 = <a>12]] = true`
- `[[3 lt 5]] = true`

Expressions

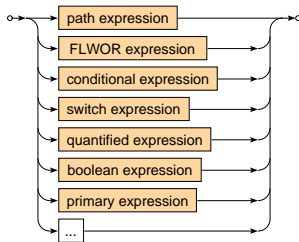
XQuery expressions

- **Path** expressions (traditional XPath)
 - Selection of nodes of an XML tree
- **FLWOR** expressions
 - `for ... let ... where ... order by ... return ...`
- **Conditional** expressions
 - `if ... then ... else ...`
- **Quantified** expressions
 - `some|every ... satisfies ...`

Expressions

XQuery expressions

- **Boolean** expressions
 - `and`, `or`, `not` logical connectives
- **Primary** expressions
 - Literals, variable references, function calls, **constructors**, ...
- ...



Node Constructors

Constructors

- Allow for **creation of new nodes** for elements, attributes, ...
 - I.e. nodes that do not exist in the original XML document

Direct constructor

- Well-formed XML fragment with embedded query expressions
 - E.g.: `<movies>{ count(//movie) }</movies>`

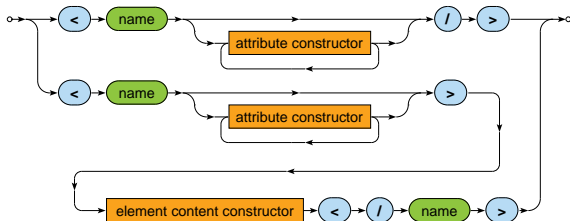
Computed constructor

- Special syntax
 - E.g.: `element movies { count(//movie) }`

Node Constructors

Direct constructor

- The entire expression must be a **well-formed XML fragment**
 - **Names of elements and attributes** must be fixed

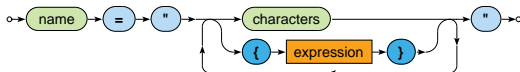


- **Embedded query expressions** can be used
 - However, only in attribute values and element content!

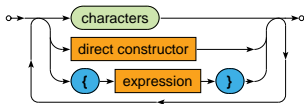
Node Constructors

Direct constructor

- Attribute



- Element content



- **Embedded query expressions**
 - Enclosed by curly braces {}
 - Escaping sequence: {{ and }}

Node Constructors: Example

Create a summary of all movies

```
<movies>
  <count>{ count(//movie) }</count>
  {
    for $m in //movie
    return
      <movie year="{ data($m/@year) }">{ $m/title/text() }</movie>
  }
</movies>
```

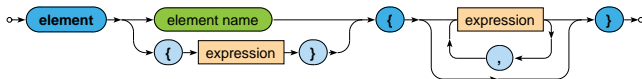
```
<movies>
  <count>3</count>
  <movie year="2006">Vratné lahve</movie>
  <movie year="2000">Samotáři</movie>
  <movie year="2007">Medvídek</movie>
</movies>
```

Node Constructors

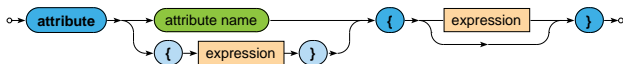
Computed constructor

- Names of elements and attributes can be dynamic

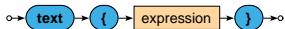
- Element** node



- Attribute** node



- Text** node



Node Constructors: Example

Create a summary of all movies

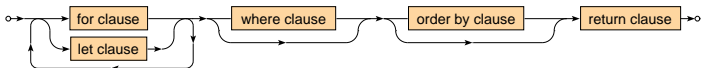
```
element movies {  
  element count { count(//movie) },  
  for $m in //movie  
  return  
    element movie {  
      attribute year { data($m/@year) },  
      text { $m/title/text() }  
    }  
}
```

```
<movies>  
  <count>3</count>  
  <movie year="2006">Vratné lahve</movie>  
  <movie year="2000">Samotáři</movie>  
  <movie year="2007">Medvídek</movie>  
</movies>
```


FLWOR Expressions

FLWOR expression (XQuery 1.0)

- Allow for advanced **iterations over sequences** of items



Clauses

- **for** – selection of items to iterate over
- **let** – bindings of auxiliary variables
- **where** – conditions to be satisfied
- **order by** – order in which the items are processed
- **return** – result to be constructed

FLWOR Expressions: Example

Find titles of movies with rating 75 and more

```
for $m in //movie
let $r := $m/@rating
where $r >= 75
order by $m/@year
return $m/title/text()
```

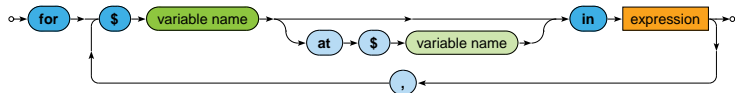
Samotáři

Vratné lahve

FLWOR Expressions: Clauses

For clause

- **Iterates over items** of one or more input sequences
 - These items are accessible via the introduced variables



- Optional **positional variable**
 - Allows to access the ordinal number of the current item
- When **multiple input sequences** are provided...
 - Then the behavior is identical to the usage of multiple consecutive single-variable for clauses
 - I.e., as if the for loops are embedded into each other

FLWOR Expressions: Clauses

Let clause

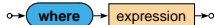
- Defines one or more auxiliary **variable assignments**



FLWOR Expressions: Clauses

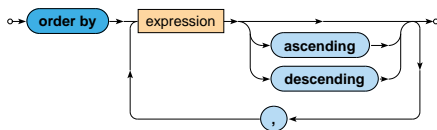
Where clause

- Allows to describe complex **filtering conditions**
- Items not satisfying the conditions are skipped



Order by clause

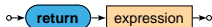
- Defines the **order in which the items are processed**



FLWOR Clauses

Return clause

- **Defines how the result sequence is constructed**
- Evaluated once for each suitable item



Various supported **use cases**

- Querying, joining, grouping, aggregation, integration, transformation, validation, ...

FLWOR Examples

Find titles of movies filmed in *2000* or later such that they have at most 3 actors and a rating above the overall average

```
let $r := avg(//movie/@rating)
for $m in //movie[@rating >= $r]
let $a := count($m/actor)
where ($a <= 3) and ($m/@year >= 2000)
order by $a ascending, $m/title descending
return $m/title
```

```
<title>Vratné lahve</title>
<title>Samotáři</title>
```

FLWOR Examples

Find movies in which each individual actor starred

```
for $a in distinct-values(//actor)
return <actor name="{ $a }">
  {
    for $m in //movie[actor[text() = $a]]
    return <movie>{ $m/title/text() }</movie>
  }
</actor>
```

```
<actor name="Zdeněk Svěrák">
  <movie>Vratné lahve</movie>
</actor>
<actor name="Jiří Macháček">
  <movie>Vratné lahve</movie>
  <movie>Samotáři</movie>
  <movie>Medvídek</movie>
</actor>
...
```


FLWOR Examples

Construct an HTML table with data about movies

```
<table>
  <tr><th>Title</th><th>Year</th><th>Actors</th></tr>
  {
    for $m in //movie
    return
      <tr>
        <td>{ $m/title/text() }</td>
        <td>{ data($m/@year) }</td>
        <td>{ count($m/actor) }</td>
      </tr>
  }
</table>
```

FLWOR Examples

Construct an HTML table with data about movies

```
<table>
  <tr><th>Title</th><th>Year</th><th>Actors</th></tr>
  <tr><td>Vratné lahve</td><td>2006</td><td>2</td></tr>
  <tr><td>Samotáři</td><td>2000</td><td>3</td></tr>
  <tr><td>Medvídek</td><td>2007</td><td>2</td></tr>
</table>
```

Conditional Expressions

Conditional expression



- Note that the else branch is compulsory
 - Empty sequence () can be returned if needed

Example

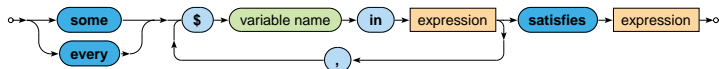
```
if (count(//movie) > 0)
then <movies>{ string-join(//movie/title, ", ") }</movies>
else ()
```

```
<movies>Vratné lahve, Samotáři, Medvídek</movies>
```

Quantified Expressions

Quantifier

- Returns true if and only if...
 - in case of some **at least one item**
 - in case of every **all the items**
- ... of a given sequence/s **satisfy the provided condition**



Quantified Expressions

Examples

Find titles of movies in which *Ivan Trojan* played

```
for $m in //movie
where
  some $a in $m/actor satisfies $a = "Ivan Trojan"
return $m/title/text()
```

Samotáři
Medvídek

Find names of actors who played in all movies

```
for $a in distinct-values(//actor)
where
  every $m in //movie satisfies $m/actor[text() = $a]
return $a
```

Jiří Macháček

Final Observations

XQuery

- **Keywords** must always be in **lowercase**
- XQuery is a **functional query language**
- Whenever `expression` is mentioned in any diagram, expression of any kind can be used (without any limitations)

Lecture Conclusion

XPath expressions

- Absolute and relative paths
- Axes, node tests, and predicates

XQuery expressions

- Constructors: direct, computed
- FLWOR expressions
- Conditional, quantified, comparison, ...