NDBI040: **Big Data Management and NoSQL Databases**

http://www.ksi.mff.cuni.cz/~svoboda/courses/2016-1-NDBI040/

Lecture 7

XML Databases: XQuery

Martin Svoboda
svoboda@ksi.mff.cuni.cz

22. 11. 2016

Charles University in Prague, Faculty of Mathematics and Physics
Czech Technical University in Prague, Faculty of Electrical Engineering

Lecture Outline

Native XML databases

- General introduction

XQuery and XPath

- Data model
- Expressions
  - Path expressions
  - FLWOR expressions
  - Conditional, quantified, switch and other expressions

Native XML Databases

Data model

- **XML documents**
  - Tree structure with nested elements, attributes, and text values (beside other less important constructs)
  - Documents are organized into collections

Query languages

- **XPath**: *XML Path Language* (navigation)
- **XQuery**: *XML Query Language* (querying)
- **XSLT**: *XSL Transformations* (transformation)

Representatives

- **Sedna**, **Tamino**, BaseX, eXist-db
- *Multi-model*: **MarkLogic**, OpenLink **Virtuoso**

Native XML Databases

Representatives

XQuery and XPath

XML Query Language
XML Path Language

Introduction

XPath = *XML Path Language*

- **Navigation in an XML tree,**
  selection of nodes by a variety of criteria
- Versions: 1.0 (1999), 2.0, **3.0**, 3.1 (2015, just draft)
- W3C recommendation
  - http://www.w3.org/TR/xpath-30/

XQuery = *XML Query Language*

- **Complex functional query language**
- Contains XPath
- Versions: 1.0 (2007), **3.0** (2014)
- W3C recommendation
  - http://www.w3.org/TR/xquery-30/

Data Model

XDM = *XQuery and XPath Data Model*

- **XML tree** consisting of **nodes** of different kinds
  - Document, element, attribute, text, …
- **Document order** / reverse document order
  - The order in which nodes appear in the XML file

The result of a query is a sequence

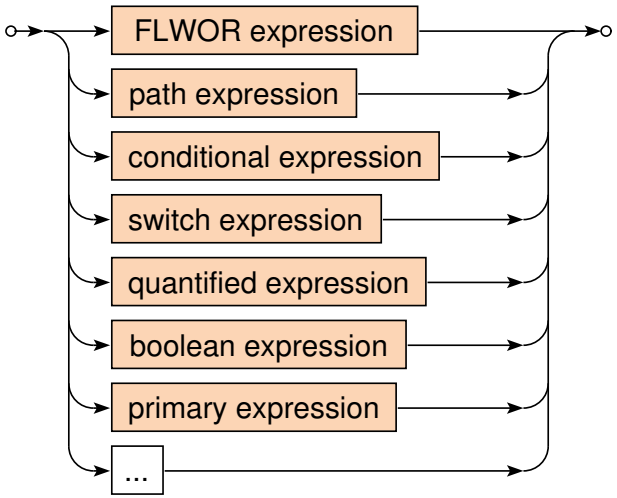
- Ordered collection of items
  - Flat, mixed, duplicate atomic values are allowed
- Item is an **atomic value** or a node

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>
```

Expressions

XQuery expressions



- **FLWOR** expressions
  - `for … let … where … order by … return …`
- **Conditional** expressions
  - `if … then … else …`

Expressions

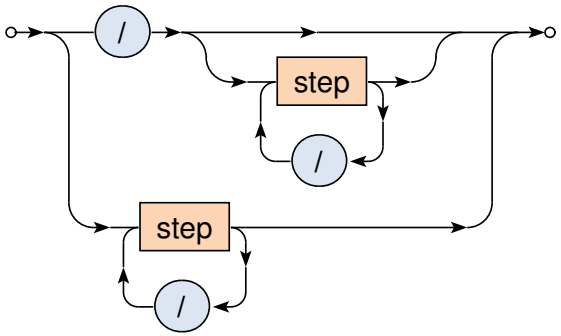
XQuery expressions

- **Switch** expressions
  - `switch … case … default …`
- **Quantified** expressions
  - `some|every … satisfies …`
- **Boolean** expressions
  - `and`, `or`, `not` logical connectives
- **Path** expressions
  - Selection of nodes of an XML tree
- **Primary** expressions
  - Literals, variable references, function calls, **constructors**, …
- …

Path Expressions

Path expression

- Describes navigation within an XML tree
- Consists of individual navigational steps



- **Absolute** paths = path expressions starting with /
  - Navigation always starts at the document node
- **Relative** paths
  - Navigation starts at an explicitly specified node / nodes

Path Expressions

Examples

Absolute paths

```
/
```

```
/movies
```

```
/movies/movie
```

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

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

Relative paths

```
actor/text()
```

```
@director
```

Path Expressions

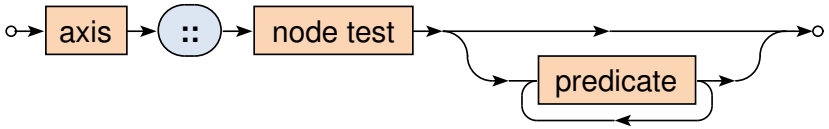
Evaluation of path expressions

- Let P be a **path expression**
- Let C be an initial **context set**
  - If P is **absolute**, then C contains just the document node
  - Otherwise (P is **relative**) C is given by the user or the context
- If P does not contain any step
  - Then C is the **final result**
- Otherwise (i.e when P contains **at least one step**)
  - Let S be the **first step**, P' the **remaining steps** (if any)
  - Let $C' = \{\}$
  - For each node $u \in C$:
    evaluate S with respect to u and add the result to C'
  - Evaluate P' with respect to C'

Path Expressions

Step

- Each step consists of (up to) 3 components



- **Axis**
  - Specifies the relation of nodes to be selected for a given node u
- **Node test**
  - Filters nodes selected by the given axis using basic tests
- **Predicates**
  - Filter the nodes again, this time using advanced conditions

Path Expressions: Axes

Axis

- Specifies the relation of nodes to be selected for a given node

Forward axes

- `self`, `child`, `descendant(-or-self)`, `following(-sibling)`
- The order of the nodes corresponds to the document order

Reverse axes

- `parent`, `ancestor(-or-self)`, `preceding(-sibling)`
- The order of the nodes is reversed

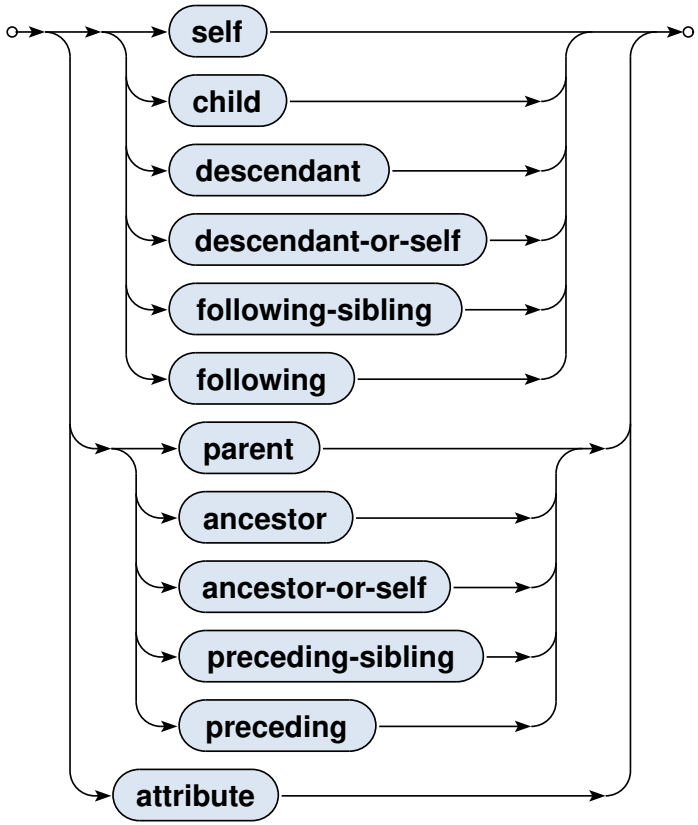
Attribute axis

- `attribute` – the only axis that selects attributes

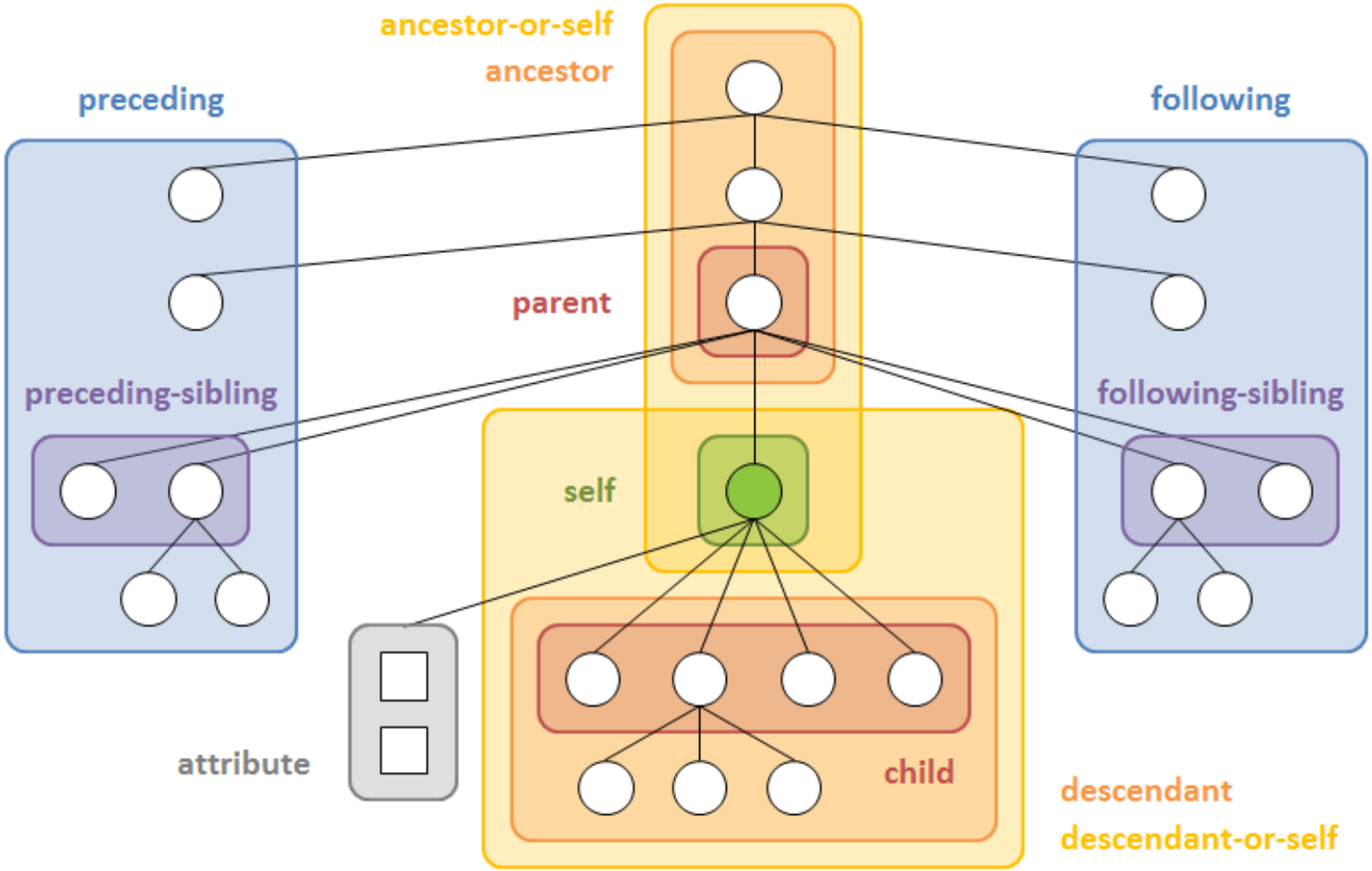
However, the final result of a step is always in document order

Path Expressions: Axes

Available **axes**



Path Expressions: Axes



Path Expressions

Examples

Axes

```
/child::movies
```

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

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

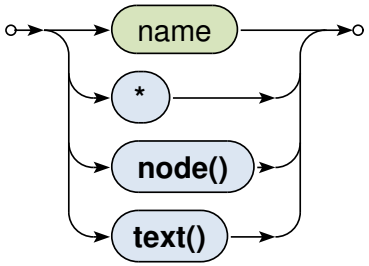
```
/descendant::movie/child::title
```

```
/descendant::movie/child::title/following-sibling::actor
```

Path Expressions: Node Tests

Node test

- Filters the nodes selected by the given axis using basic tests



Available node tests

- `name` – all elements / attributes with a given name
- `*` – all elements / attributes
- `node()` – all nodes (i.e. no filtering takes place)
- `text()` – all text nodes

Path Expressions

Examples

Node tests

```
/movies
```

```
/child::movies
```

```
/descendant::movie/title/text()
```

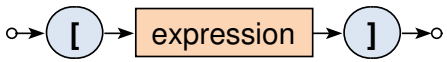
```
/movies/*
```

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

Path Expressions: Predicates

Predicate

- Performs additional filtering of the selected nodes using advanced conditions



Commonly used conditions

- Boolean expressions
- Path expressions
  - Return `true` when evaluated to a non-empty sequence
- Comparisons, position testing, …

When **multiple predicates** are provided, they must all be satisfied

Path Expressions

Examples

Predicates

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

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

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

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

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

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

Path Expressions: Abbreviations

Multiple (mostly syntax) **abbreviations** are provided

- …/… (i.e. no axis is specified) ⇔ …/`child::`…
- …/`@`… ⇔ …/`attribute::`…
- …/`.`… ⇔ …/`self::node()`…
- …/`..`… ⇔ …/`parent::node()`…
- …`//`… ⇔ …`/descendant-or-self::node()/`…
- …/…`[number]`… ⇔ …/…`[position() = number]`…

Path Expressions

Examples

Abbreviations

```
/movie/title
```

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

```
/movie/@year
```

```
/child::movie/attribute::year
```

```
//actor
```

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

```
/movie/actor[2]
```

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

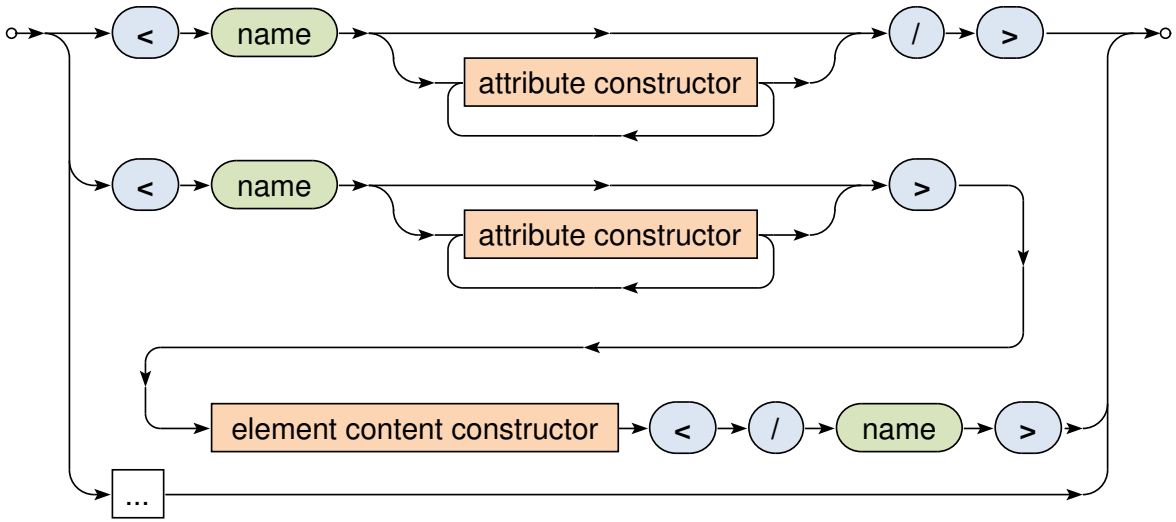
Constructors

Constructors

- Allow us to **create new nodes for elements, attributes, …**
- **Direct** constructor
  - Well-formed XML fragment with **nested query expressions**
  - **Names of elements and attributes are fixed**, their content can be dynamic
- **Computed** constructor
  - Special syntax
  - **Both names and content can be dynamic**

Constructors

Direct constructor

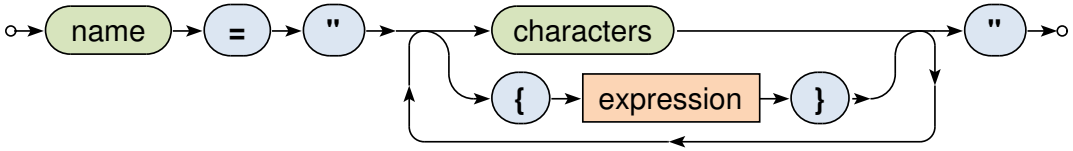


- Both **attribute value and element content** may contain an arbitrary number of **nested query expressions**
  - Enclosed by curly braces {}
  - Escaping sequences: {{ and }}

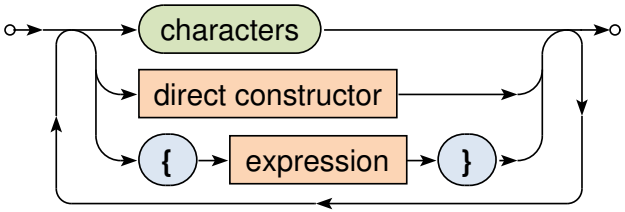
Constructors

Direct constructor

- Attribute



- Element content



Constructors

Example: Direct Constructor

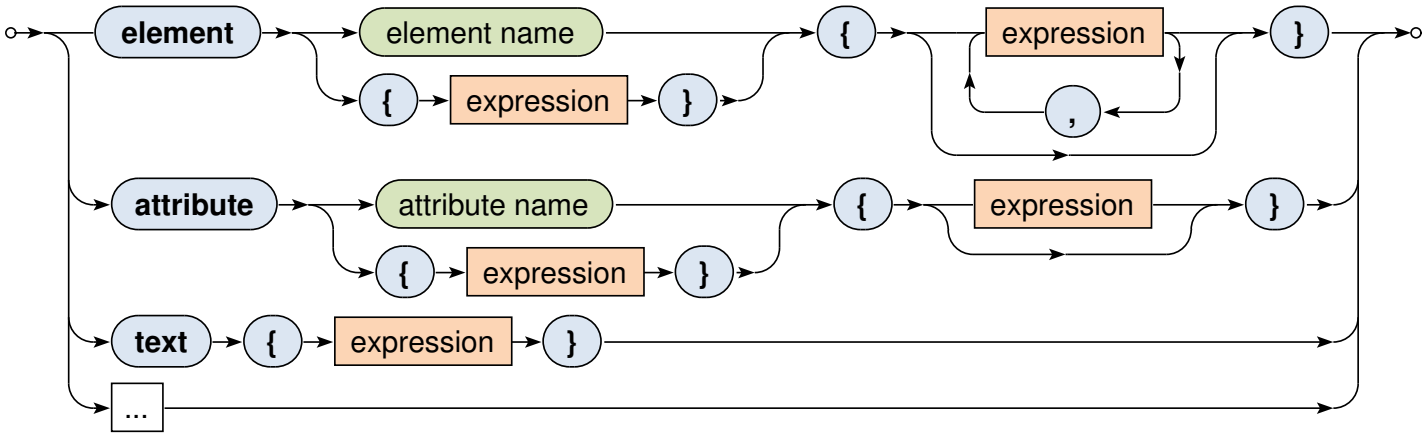
Create a summary of all the movies

```
<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>
```

Constructors

Computed constructor



Constructors

Example: Computed Constructor

Create a summary of all the movies

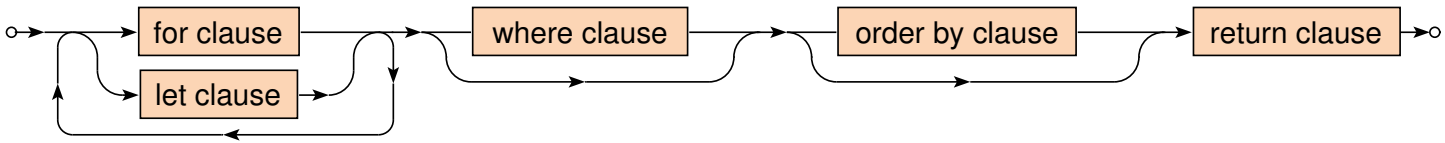
```
element movies {
  element count { count(//movie) },
  for $m in //movie
  return
    element movie {
      attribute year { data($m/@year) },
      text { $m/title/text() }
    }
}
```

```
<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

- Versatile construct allowing for **iterations over sequences**
- Generates one flat result sequence



Clauses

- `for` – sequence to be iterated
- `let` – binding of variables
- `where` – filtering conditions
- `order by` – ordering of the result
- `return` – construction of the result

FLWOR Expressions

Example

Find titles of all the movies with rating *75* and more

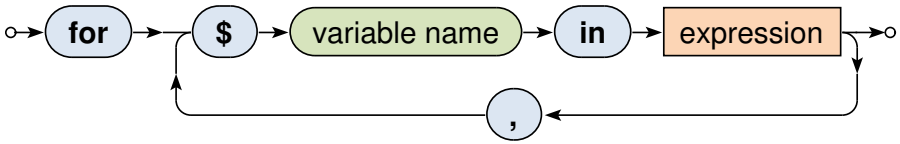
```
for $m in //movie
let $r := $m/@rating
where $r >= 75
order by $m/@year
return $m/title/text()
```

```
Samotáři
Vratné lahve
```

FLWOR Expressions: Clauses

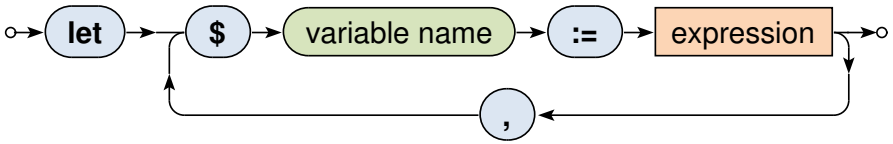
For clause

- Specifies a **sequence of values or nodes to be iterated over**
- Multiple sequences can be specified at once
  - Then the behavior is identical as when more single-variable for clauses would be provided



Let clause

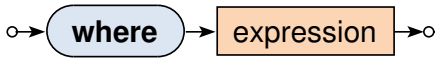
- Defines one or more auxiliary **variable assignments**



FLWOR Expressions: Clauses

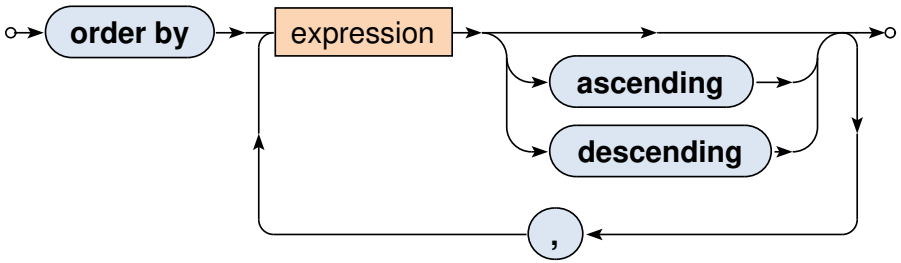
Where clause

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



Order by clause

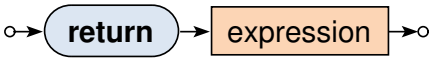
- Describes mutual **order of items in the result sequence**



FLWOR Expressions: Clauses

Return clause

- **Defines how the result sequence should be constructed**



Various supported **use cases**

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

FLWOR Expressions

Examples

Find titles of movies filmed in *2000* and later such that they have at most 3 actors and also 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 Expressions

Examples

Find all the movies in which each individual actor stared

```
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 Expressions

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 Expressions

Examples

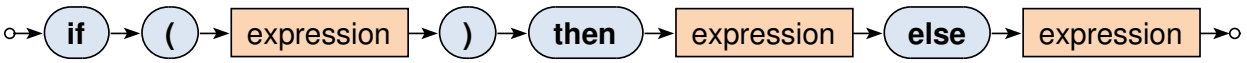
Construct an HTML table with data about movies

```
<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

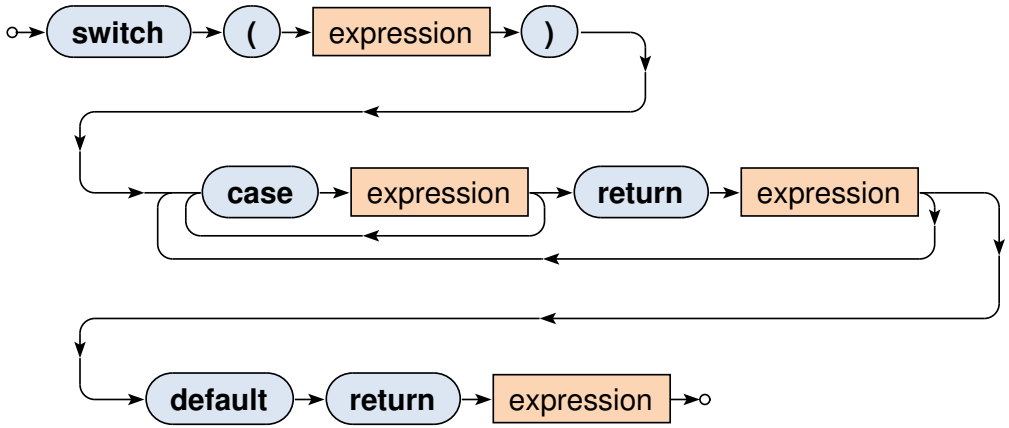
```
if (count(//movie) > 0)
then <movies>{ string-join(//movie/title, ", ") }</movies>
else ()
```

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

Switch Expressions

Switch

- **The first matching branch is chosen,**
  its `return` clause is evaluated
  and the result returned



- The `default` branch is compulsory
  and must be provided as the last option

Switch Expressions

Example

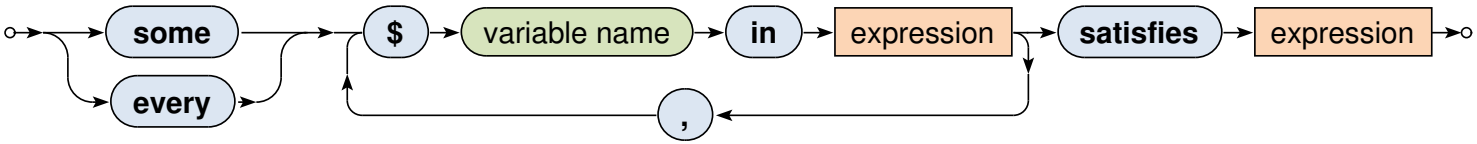
Return all movies with aggregated information about their actors

```
xquery version "3.0";
for $m in //movie
return
  <movie>
    { $m/title }
    {
      switch (count($m/actor))
      case 0 return <no-actors/>
      case 1 return <actor>{ $m/actor/text() }</actor>
      default return <actors>{ string-join($m/actor, ", ") }</actors>
    }
  </movie>
```

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 all actors that played in all movies

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

```
Jiří Macháček
```

Comparison Expressions

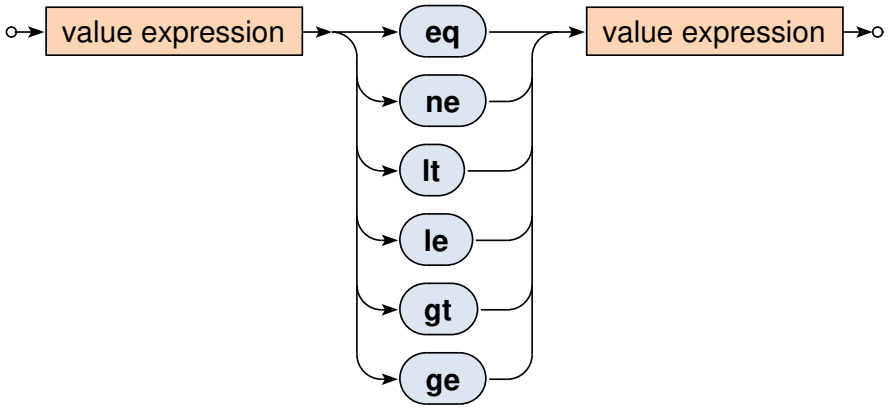
Comparisons

- **Value** comparisons
  - Two atomic values are expected to be compared
  - `eq`, `ne`, `lt`, `le`, `ge`, `gt`
- **General** comparisons
  - Two sequences of values are expected to be compared
  - =, !=, <, <=, >=, >
- **Node** comparisons
  - `is` – tests identity of nodes
  - <<, >> – test positions of nodes
  - Similar behavior as in case of value comparisons

Comparison Expressions

Value comparison

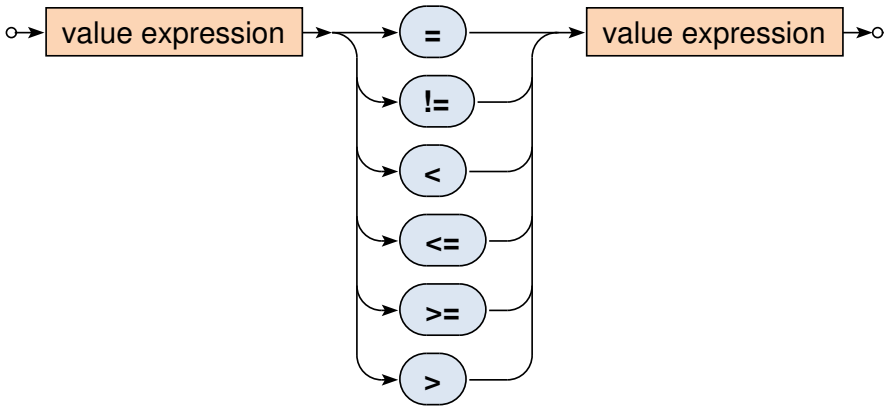
- **Both the operands are expected to be evaluated to single values** (or sequences with just one value)
  - Then these values are mutually compared in a standard way
- Empty sequence () is returned…
  - when at least one operand is evaluated to an empty sequence
- Error is risen…
  - when at least one operand is evaluated to a longer sequence



Comparison Expressions

General comparison (**existentially quantified** comparisons)

- **Both the operands can be evaluated to sequences of values of any length**
- The result is `true` if and only if there exists at least one pair of individual values satisfying the given relationship



Comparison Expressions

Atomization of values

- Takes place in case of both the **value and general comparisons**
- **Items (of a given sequence) are first atomized**
  - Atomic value is kept untouched
  - **Node is transformed into a string with concatenated text values** it contains (even indirectly)
    - Note that attribute values are not included!
- Corresponds to the effect of `data()` function

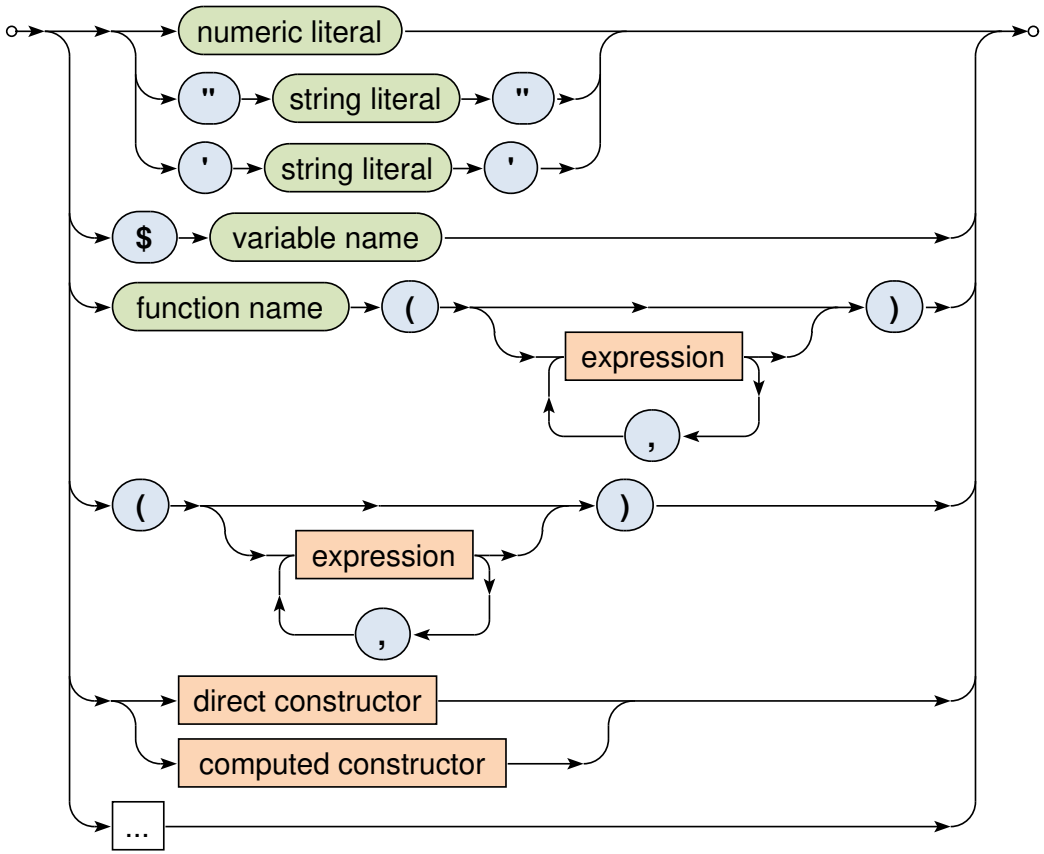
Comparison Expressions

Examples

- `1 le 2` ⇒ `true`
- `(1) le (2)` ⇒ `true`
- `(1) le (2,1)` ⇒ `error`
- `(1) le ()` ⇒ `()`
- `<a>5</a> eq <b>5</b>` ⇒ `true`
- `1 < 2` ⇒ `true`
- `(1) < (2)` ⇒ `true`
- `(1) < (2,1)` ⇒ `true`
- `(1) < ()` ⇒ `false`
- `(0,1) = (1,2)` ⇒ `true`
- `(0,1) != (1,2)` ⇒ `true`

Primary Expressions

Primary expression



Lecture Conclusion

XPath expressions

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

XQuery expressions

- Constructors: direct, computed
- FLWOR
- Conditional, quantified, …
- Comparison, arithmetic, …