XML Databases: XQuery

Big Data Management

Anis Ur Rahman
Faculty of Computer Science & Information Technology
University of Malaya

September 25, 2019

Borrowed from http://www.ksi.mff.cuni.cz/~svoboda/courses/171-NDBI040/

Lecture Outline

Native XML databases

- General introduction

XQuery and **XPath**

- Data model
- Query expressions
  - Path expressions
  - FLWOR expressions
  - Constructors, conditions, quantifiers, comparisons, ...

Outline

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 (March 2017)
- W3C recommendation
  - `https://www.w3.org/TR/xpath-31/`

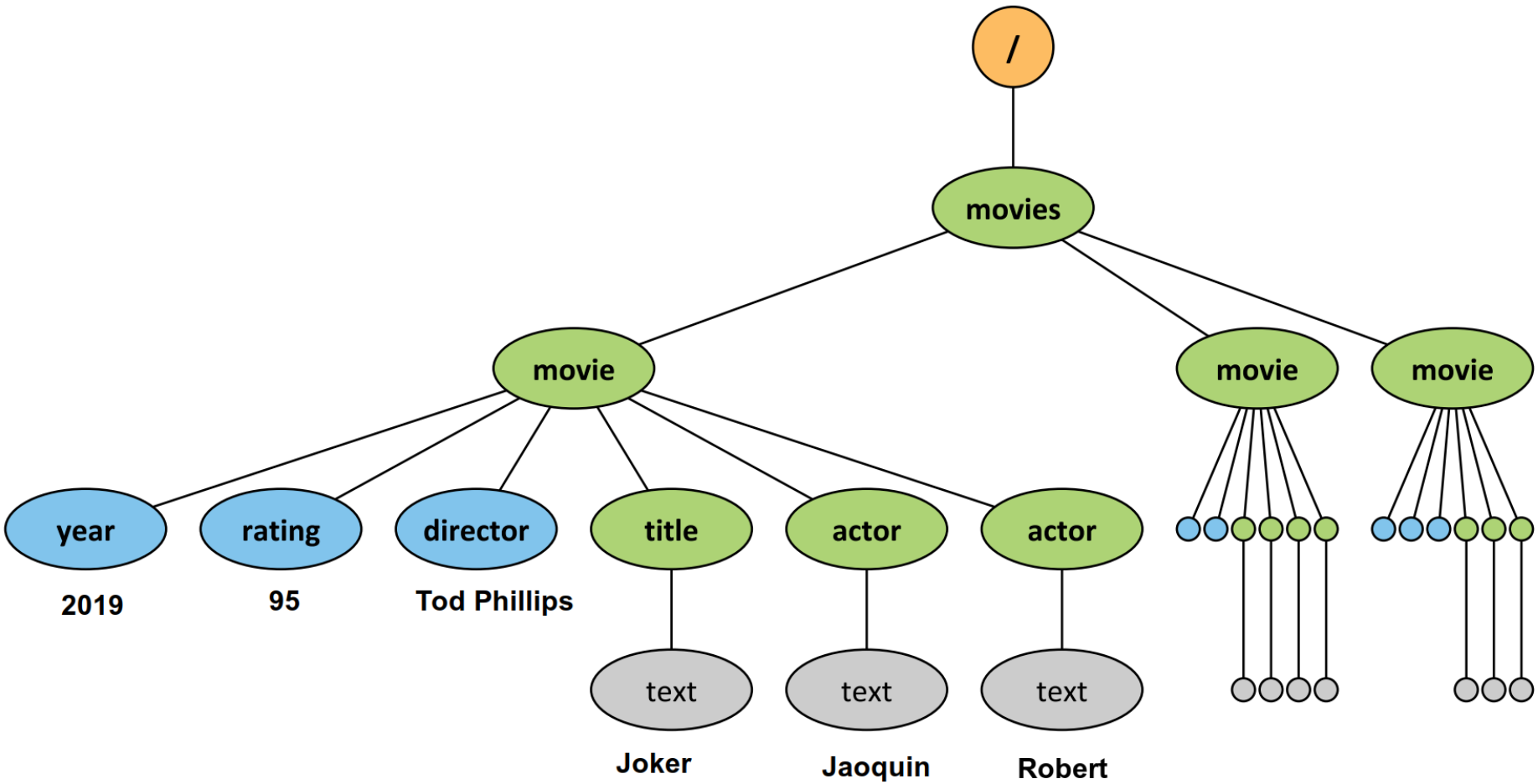
XQuery = XML Query Language

- Complex functional query language
- 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
<?xml version="1.1" encoding="UTF-8"?>
<movies>
  <movie year="2019" rating="92" director="Todd Phillips">
    <title>Joker</title>
    <actor>Joaquin Phoenix</actor>
    <actor>Robert De Niro</actor>
  </movie>
  <movie year="2009" rating="84">
    <title>Zombieland</title>
    <actor>Woody Harrelson</actor>
    <actor>Jesse Eisenberg</actor>
    <actor>Emma Stone</actor>
  </movie>
  <movie year="2013" rating="83" director="Richard Ayoade">
    <title>The Double</title>
    <actor>Jesse Eisenberg</actor>
    <actor>Mia Wasikowska</actor>
  </movie>
</movies>
```

Sample Data


/
movies
movie
movie
movie
year
rating
director
title
actor
actor
2019
95
Tod Phillips
text
text
text
Joker
Jaoquin
Robert

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
  - I.e. nodes are numbered using a **pre-order depth-first** traversal

Query result

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

Data Model

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

- Automatically **flattened**
  - E.g.: (1, (), (2, 3), (4)) ⇔ (1, 2, 3, 4)
- **Standalone** items are treated as **singleton** sequences
  - E.g.: 1 ⇔ (1)
- Can be **mixed**
  - But usually just **nodes**, or just atomic values
- **Duplicate** items are **allowed**

Expressions

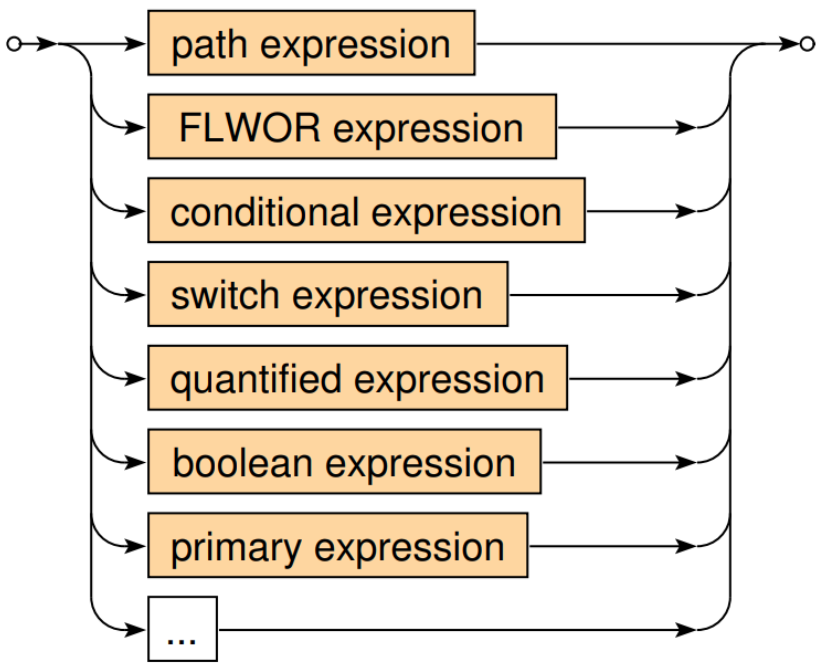
XQuery expressions

1. **Path** expressions (**traditional** XPath)
   - **Selection** of nodes of an XML tree
2. **FLWOR** expressions
   - for ... let ... where ... order by ... return ...
3. **Conditional** expressions
   - if ... then ... else ...
4. **Switch** expressions
   - switch ... case ... default ...
5. **Quantified** expressions
   - some|every ... satisfies ...

Expressions

XQuery expressions

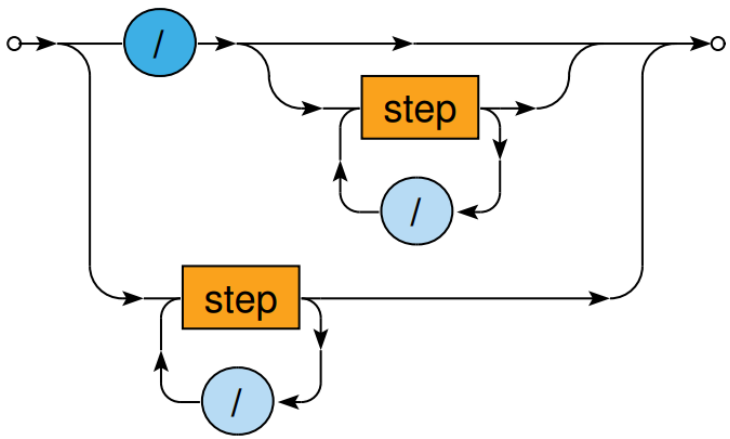
- Boolean expressions
  - and, or, not logical connectives
- Primary expressions
  - Literals, variable references, function calls, constructors, ...
- ...



Path Expressions

Path expression

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



1. **Absolute paths** = path expressions starting with /
   - Navigation **starts** at the **document node**
2. **Relative paths**
   - Navigation **starts** at an explicitly **specified node(s)**

Path Expressions: Examples

Absolute paths

```
1 /
```

```
1 /movies
```

```
1 /movies/movie
```

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

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

Relative paths

```
1 actor/text()
```

```
1 @director
```

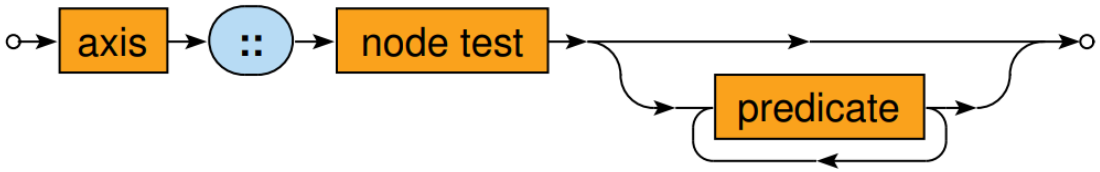
Path Expressions

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 (i.e. P is **relative**) C is **given by** the **user or context**
- If P does **not contain any step**
  - Then C is the **final result**
- Otherwise (i.e when P **contains at least one step**)
  1. Let S be the **first step**, P the **remaining steps** (if any)
  2. Let $C' = \{\}$
  3. 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) **three** components



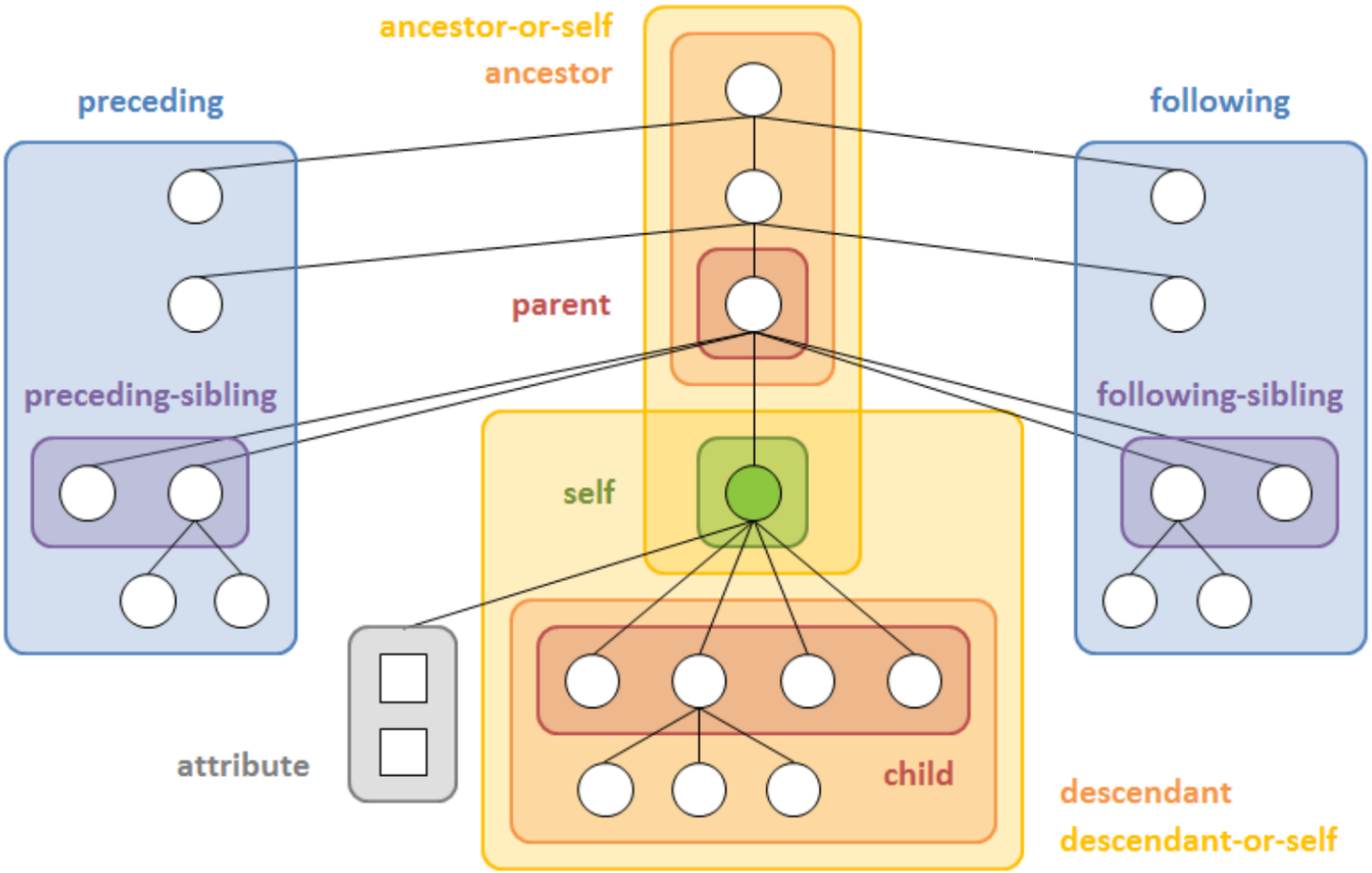
1. **Axis.** Specifies the **relation** of nodes to be selected for a given node u
2. **Node test.** **Basic condition** the selected nodes must further satisfy
3. **Predicates.** **Advanced conditions** the selected nodes must further satisfy

Path Expressions: Axes

Axis. Specifies the relation of nodes to be **selected** for a given node

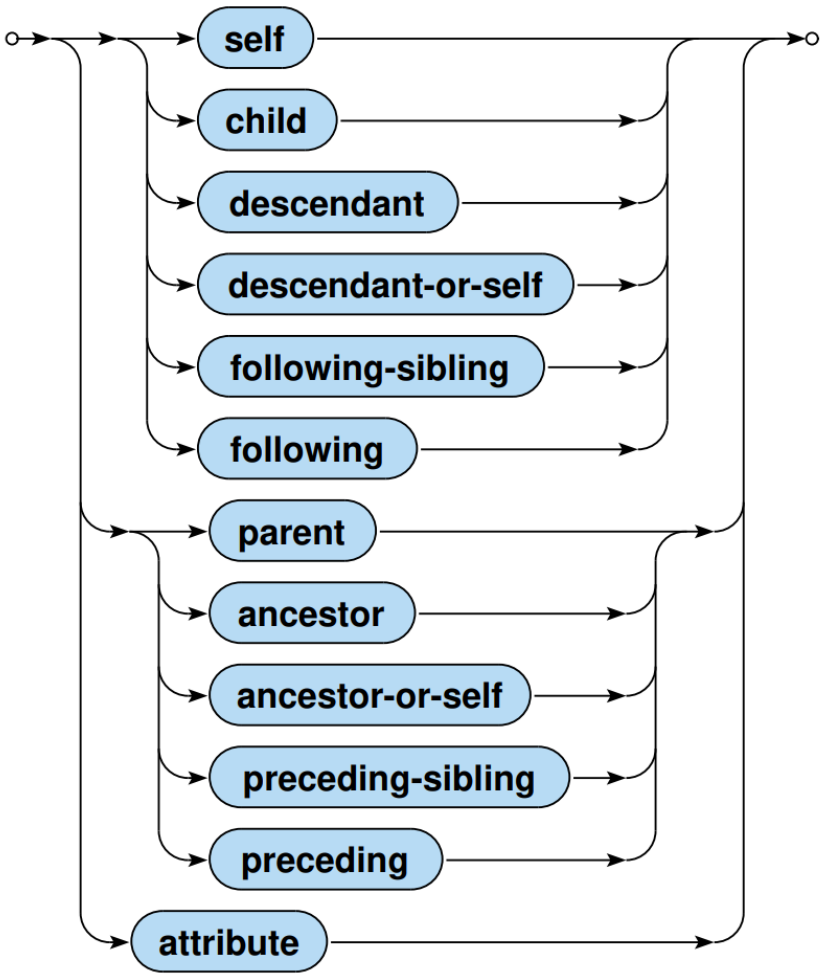
1. **Forward axes**
   - self, child, descendant(-or-self), following(-sibling)
   - The order of the nodes **corresponds** to the document order
2. **Reverse axes**
   - parent, ancestor(-or-self), preceding(-sibling)
   - The order of the nodes is reversed
3. **Attribute axis.**
   - attribute – the only axis that **selects attributes**

Path Expressions: Axes



Path Expressions: Axes

Available axes



Path Expressions: Examples

Axes

```
1 /child::movies
```

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

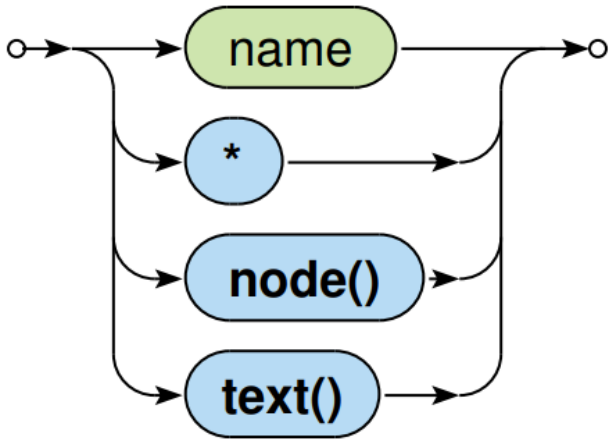
```
1 /child::movies/child::movie/attribute::year
```

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

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

Path Expressions: Node Tests

Node test. Filters the **nodes selected** by the 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

```
1 /movies
```

```
1 /child::movies
```

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

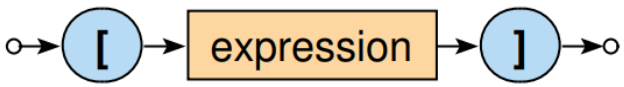
```
1 /movies/*
```

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

Path Expressions: Predicates

Predicate

- Further **filters** the nodes using **advanced conditions**



Commonly used conditions

1. **Comparisons**
2. **Path expressions**
   - Handled as **true** when evaluated to a **non-empty sequence**
3. **Position testing**
   - **Based on** the **order** as defined by the axis, starting with 1
4. **Boolean expressions, ...**
   - When **multiple** predicates are provided, they must **all be satisfied**

Path Expressions: Examples

Predicates

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

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

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

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

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

```
1 /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

```
1 /movie/title
```

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

```
1 /movie/@year
```

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

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

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

```
1 //actor
```

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

Path Expressions: Conclusion

Path expressions

- Absolute/relative

Step components

- Axis
- Node test
- Predicates

Path expression result

- Result of the **entire** path expression is the **result** of its **laststep**
- Nodes are **ordered** in the document order
- **Duplicate** nodes are **removed** (based on the identity of nodes)

Constructors

Constructors. Allows to **create** new **nodes** for elements, attributes, ...

1. **Direct** constructor
   - **Well-formed XML fragment** with nested query expressions

```
1 <movies>{ count(//movie) }</movies>
```

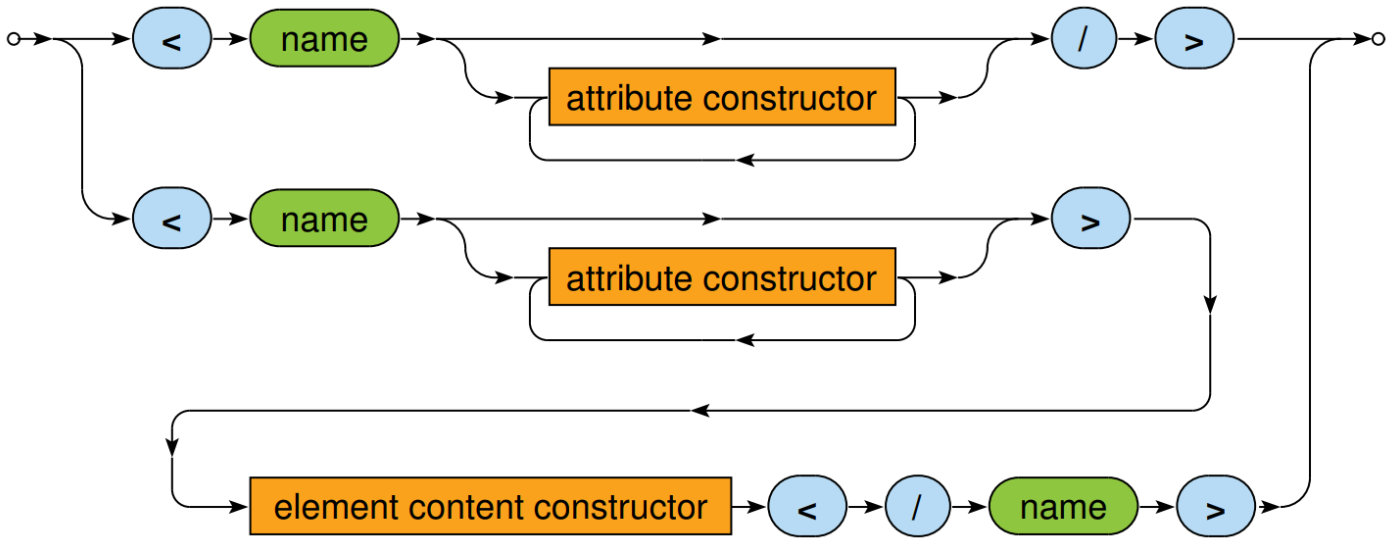
   - **Names** of elements and attributes must be **fixed**, their **content** can be **dynamic**
2. **Computed** constructor
   - Special syntax

```
1 element movies { count(//movie) }
```

   - Both **names** and **content** can be **dynamic**

Constructors

Direct constructor

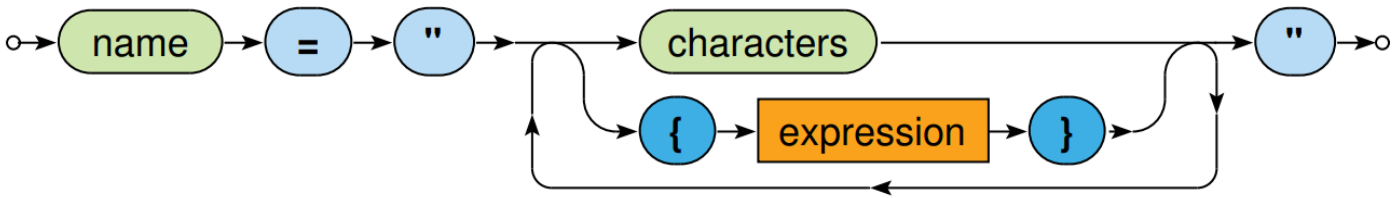


- Both **attribute value** and **element content** may contain an **arbitrary** number of **nested query expressions**
  - **Enclosed** by **curly** braces {}
  - Escape sequences: {{ and }}

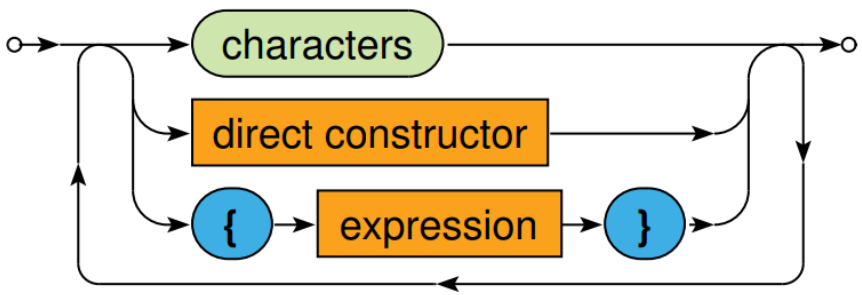
Constructors

Direct constructor

1. Attribute



2. Element content



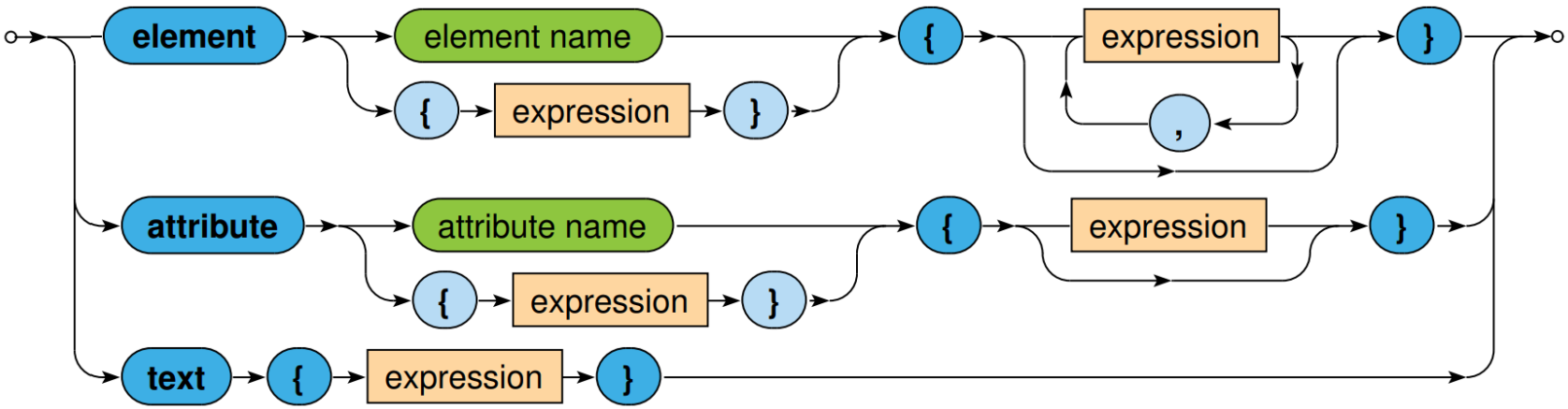
Direct Constructor: Example

Create a summary of all movies

```
1 <movies>
2   <count>{ count(//movie) }</count>
3   {
4     for $m in //movie
5     return
6       <movie year="{ data($m/@year) }">{ $m/title/text() }</↩
            movie>
7   }
8 </movies>
```

```
1 <movies>
2   <count>3</count>
3   <movie year="2019">Joker</movie>
4   <movie year="2009">Zombieland</movie>
5   <movie year="2013">The Double</movie>
6 </movies>
```

Computed constructor



Computed Constructor: Example

Create a summary of all movies

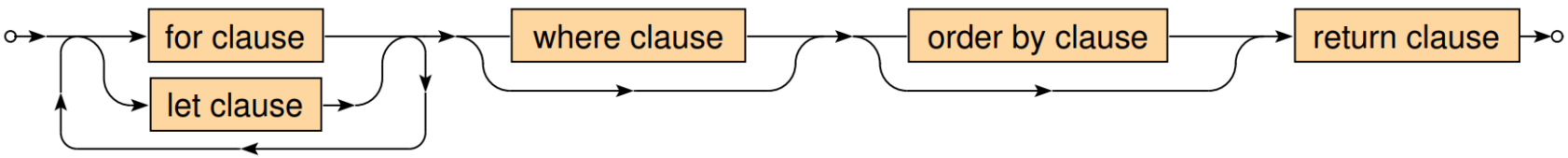
```
element movies {
  element count { count(//movie) },
  for $m in //movie
  return
    element movie {
      attribute year { data($m/@year) },
      text { $m/title/text() }
    }
}
```

```
<movies>
<count>3</count>
<movie year="2019">Joker</movie>
<movie year="2009">Zombieland</movie>
<movie year="2013">The Double</movie>
</movies>
```

FLWOR Expressions

FLWOR expression

- Versatile **construct** allowing for **iterations** **over sequences**



Clauses

1. **for** – **selection** of items to be **iterated over**
2. **let** – **bindings** of auxiliary variables
3. **where** – **conditions** to be satisfied (by a given item)
4. **order by** – **order** in which the items are processed
5. **return** – **result** to be constructed (for a given item)

FLWOR Expressions: Example

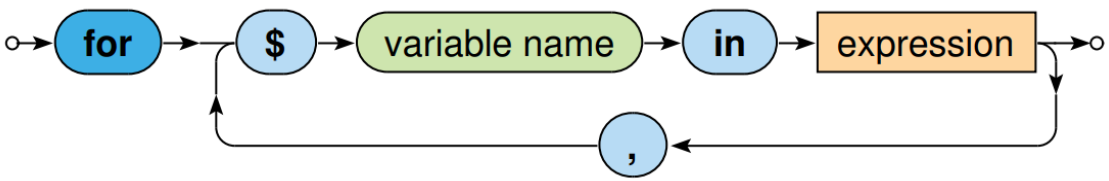
Find titles of movies with rating 85 and more

```
1 for $m in //movie
2 let $r := $m/@rating
3 where $r >= 85
4 order by $m/@year
5 return $m/title/text()
6 Joker
7 Zombieland
```

FLWOR Expressions: Clauses

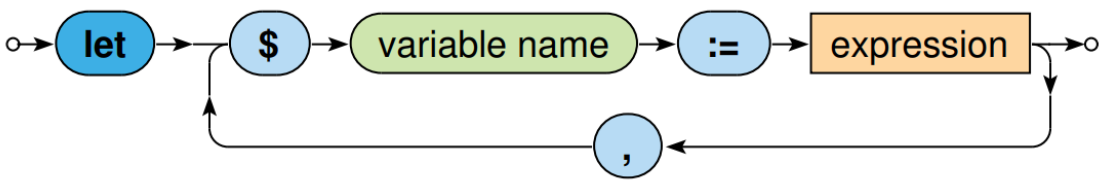
For clause

- 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

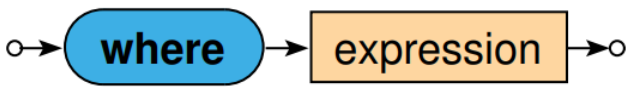
- Defines one or more **auxiliary variable assignments**



FLWOR Expressions: Clauses

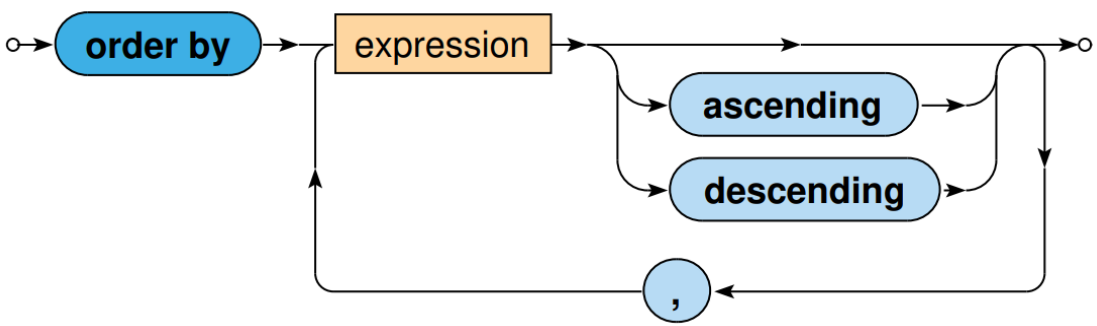
Where clause

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



Order by clause

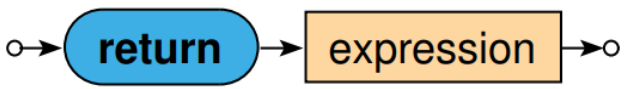
- Defines the order in which the items are **processed order**



FLWOR Expressions: 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 Expressions: Examples

Find titles of movies filmed in 2010 or later such that they have at most 3 actors and a rating above the overall average

```
1 let $r := avg(//movie/@rating)
2 for $m in //movie[@rating >= $r]
3 let $a := count($m/actor)
4 where ($a <= 3) and ($m/@year >= 2010)
5 order by $a ascending, $m/title descending
6 return $m/title
```

```
1 <title>Joker</title>
2 <title>The Double</title>
```

FLWOR Expressions: Examples

Find movies in which each individual actor stared

```
1  for $a in distinct-values(//actor)
2  return <actor name="{ $a }">
3  {
4  for $m in //movie[actor[text() = $a]]
5  return <movie>{ $m/title/text() }</movie>
6  }
7  </actor>
```

```
1  <actor name="Joaquin Phoenix">
2    <movie>Joker</movie>
3  </actor>
4  <actor name="Jesse Eisenberg">
5    <movie>Zombieland</movie>
6    <movie>The Double</movie>
7  </actor>
8  ...
```

FLWOR Expressions: Examples

Construct an HTML table with data about movies

```
1  <table>
2  <tr><th>Title</th><th>Year</th><th>Actors</th></tr>
3  {
4    for $m in //movie
5    return
6      <tr>
7        <td>{ $m/title/text() }</td>
8        <td>{ data($m/@year) }</td>
9        <td>{ count($m/actor) }</td>
10     </tr>
11 }
12 </table>
```

FLWOR Expressions: Examples

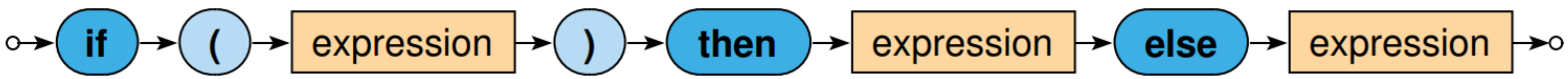
Construct an HTML table with data about movies

```
<table>
  <tr><th>Title</th><th>Year</th><th>Actors</th></tr>
  <tr><td>Joker</td><td>2019</td><td>2</td></tr>
  <tr><td>Zombieland</td><td>2009</td><td>3</td></tr>
  <tr><td>The Double</td><td>2013</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

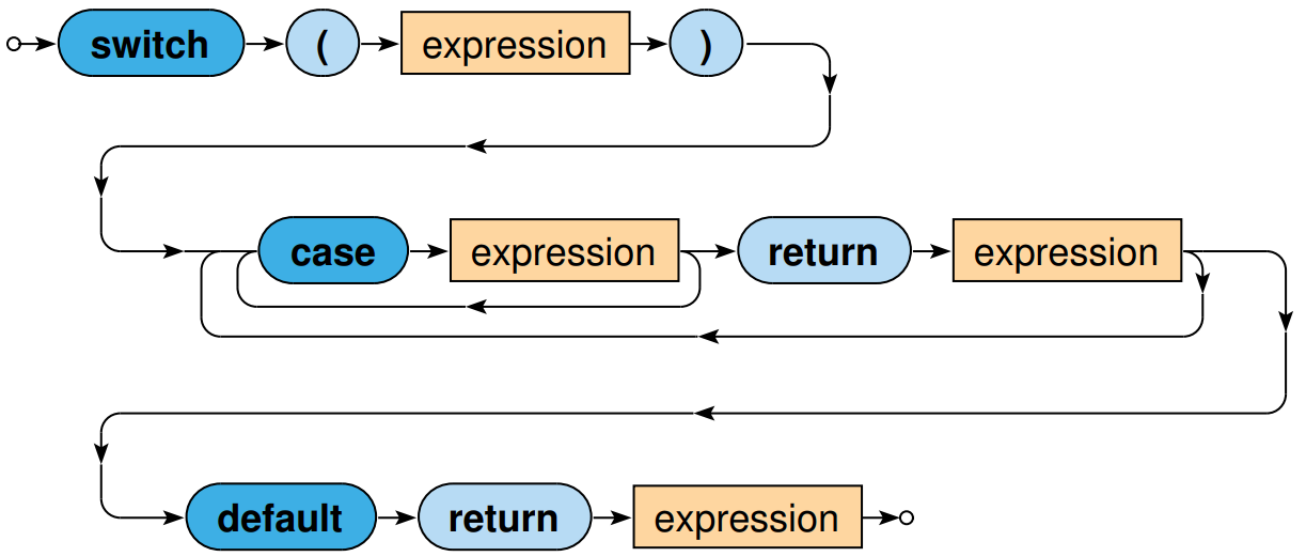
```
1 if (count(//movie) > 0)
2 then <movies>{ string-join(//movie/title, ", ") }</movies>
3 else ()
```

```
1 <movies>Joker, Zombieland, The Double</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

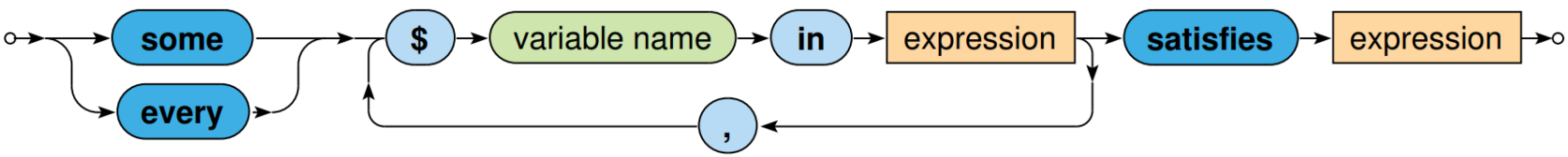
Return movies with aggregated information about their actors

```
1  xquery version "3.0";
2  for $m in //movie
3  return
4    <movie>
5      { $m/title }
6      {
7        switch (count($m/actor))
8        case 0 return <no-actors/>
9        case 1 return <actor>{ $m/actor/text() }</actor>
10       default return <actors>{ string-join($m/actor, ", ") }</↩
             actors>
11     }
12   </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 Jesse Eisenberg played

```
1 for $m in //movie
2 where
3   some $a in $m/actor satisfies $a = "Jesse Eisenberg"
4 return $m/title/text()
```

```
1 Zombieland
2 The Double
```

Find names of actors who played in all movies

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

```
1 ()
```

Comparison Expressions

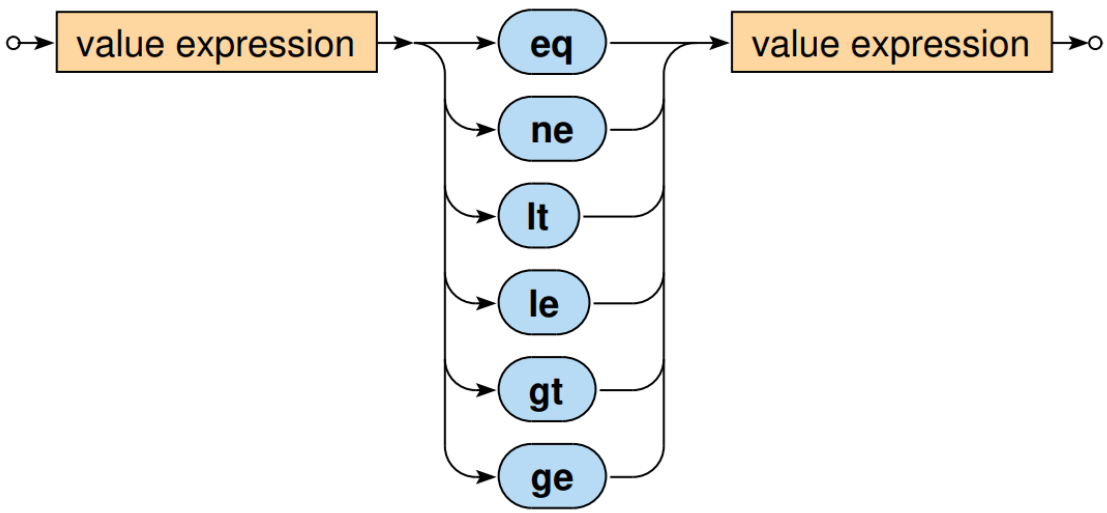
Comparisons

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

Comparison Expressions

Value comparison

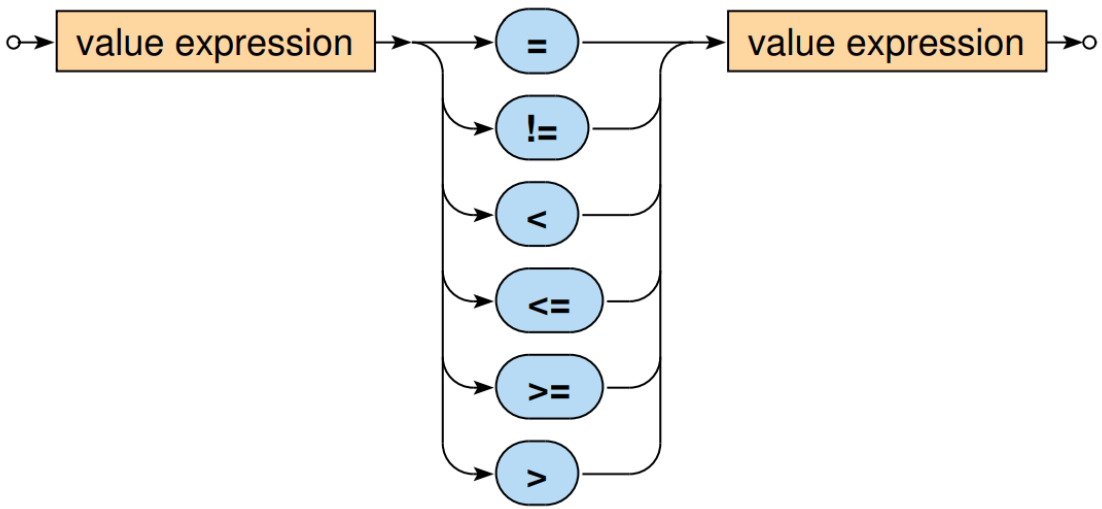
- Both the **operands** are expected to be evaluated to **singleton** sequences
  - Then these values are **mutually compared** in a standard way
- **Empty** sequence () is returned...
  - when **at least one operand** is evaluated to an **empty sequence**
- **Type error** is raised...
  - when **at 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

Value and general comparisons

- **Atomization** of values – takes place **automatically**
  - **Atomic values** are **preserved** untouched
  - Nodes are first **transformed** into **strings with concatenated text values** they contain (even indirectly)
    - E.g.

```
1 <movie year="2019">Joker</movie>
```

    - is atomized to a string

```
1 Joker
```

    - Note that **attribute values** are **not included**!

Comparison Expressions

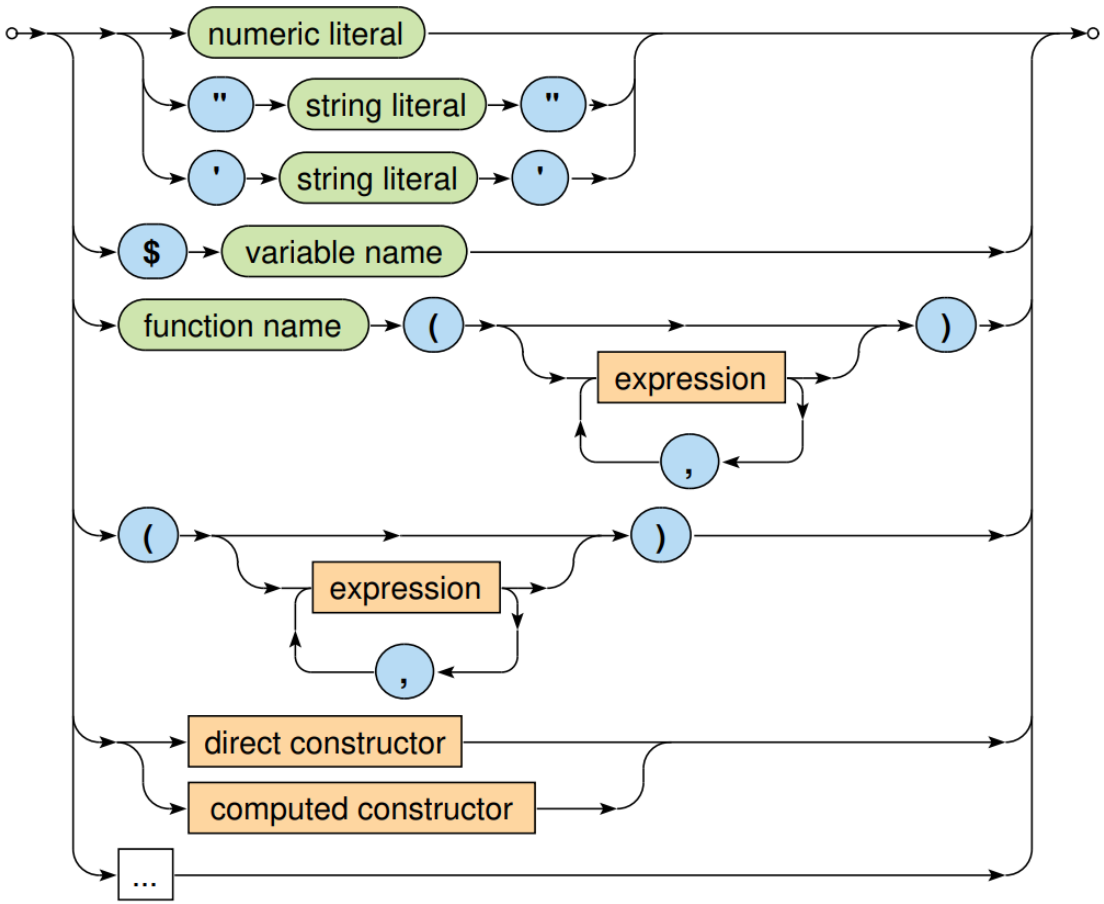
Examples

1. 1 le 2 $\Rightarrow$ true
2. (1) le (2) $\Rightarrow$ true
3. (1) le (1,2) $\Rightarrow$ error
4. (1) le () $\Rightarrow$ ()

6. 1 < 2 $\Rightarrow$ true
7. (1) < (1,2) $\Rightarrow$ true
8. (1) < () $\Rightarrow$ false
9. (0,1) = (1,2) $\Rightarrow$ true
10. (0,1) != (1,2) $\Rightarrow$ true

Primary Expressions

Primary expression



Lecture Conclusion

XPath expressions

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

XQuery expressions

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