NDBI001: **Query Languages I**

http://www.ksi.mff.cuni.cz/~svoboda/courses/231-NDBI001/

Lecture

# XML Databases: XQuery

**Martin Svoboda**
martin.svoboda@matfyz.cuni.cz

19. 12. 2023

**Charles University**, Faculty of Mathematics and Physics

# Lecture Outline

**XPath and XQuery**

- Query expressions
  - Direct and computed **constructors**
  - **FLWOR** expressions
  - **Conditional** expressions
  - **Quantified** expressions

# XQuery

XML Query Language

# 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

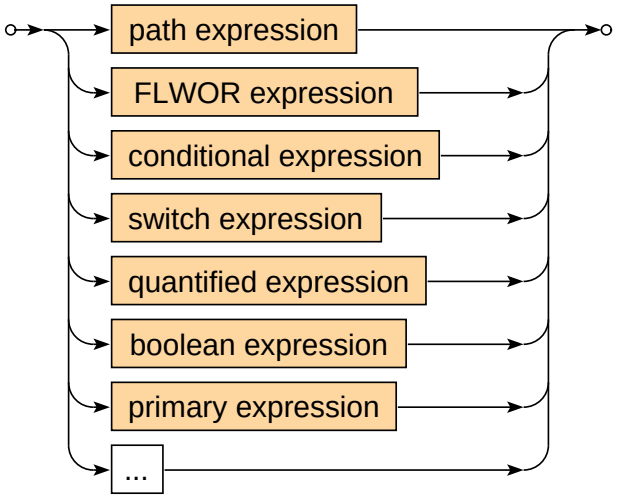
**XQuery expressions**

- **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>`

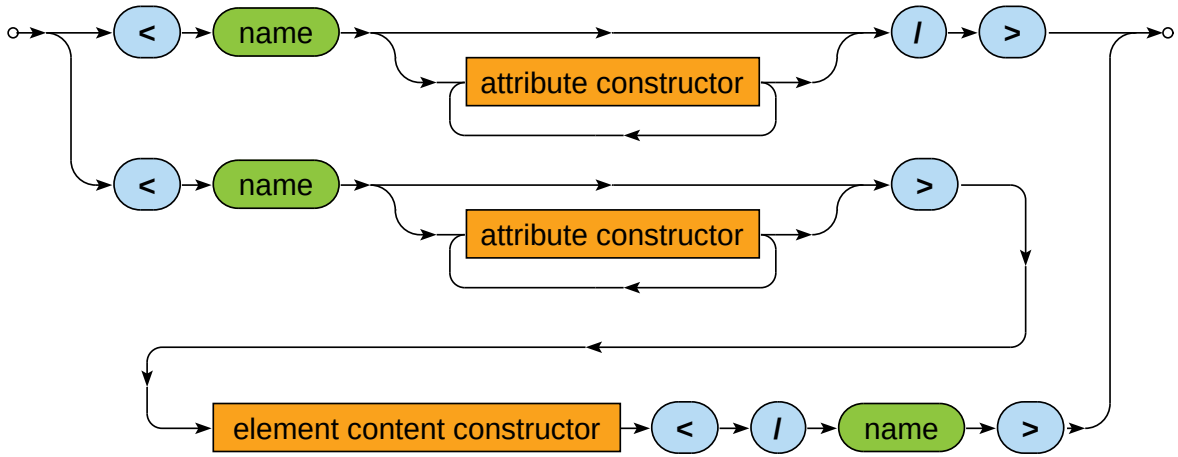
## Computed constructor

- 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

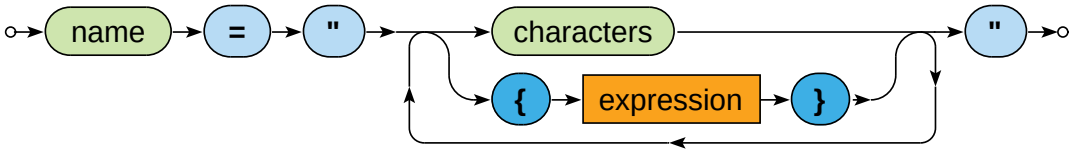



- **Embedded query expressions** can be used
  - However, only in attribute values and element content!

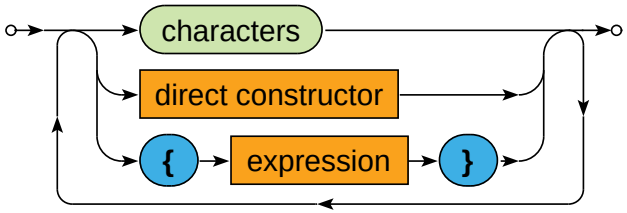
# Node Constructors

## Direct constructor

- Attribute




- Element content




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

# Node Constructors: Example

Create a summary of all movies

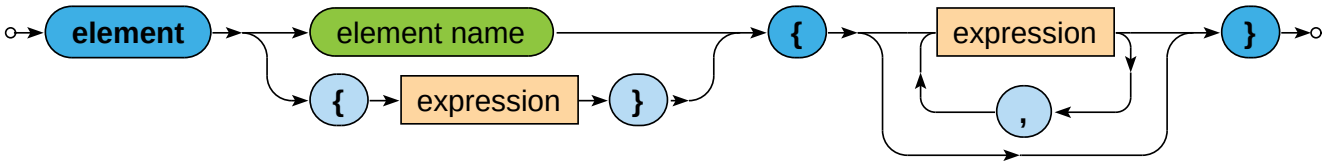
```
<movies>
  <count>{ count(//movie) }</count>
  {
    for $m in //movie
    return
      <movie year="{ data($m/@year) }">{ $m/title/text() }</movie>
  }
</movies>
```

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

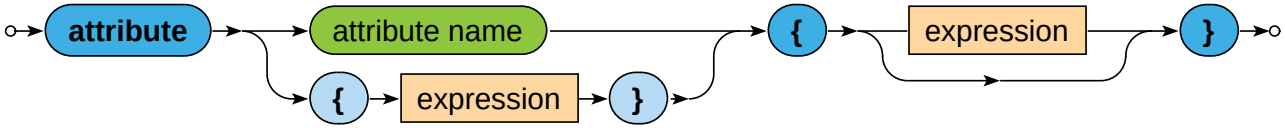
# Node Constructors

## Computed constructor

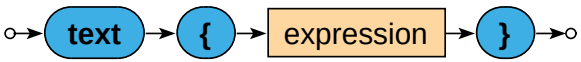
- Names of elements and attributes can be dynamic
- **Element** node




- **Attribute** node




- **Text** node



# Node Constructors: Example

Create a summary of all movies

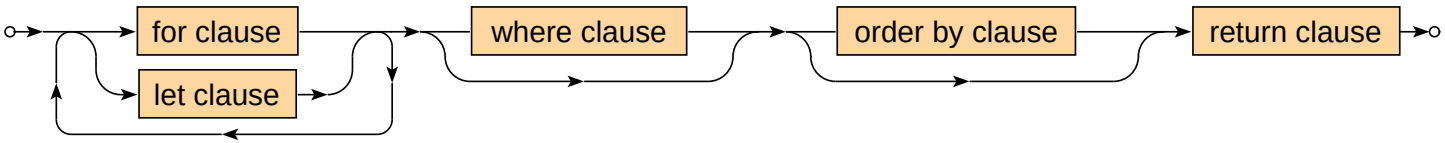
```
element movies {
  element count { count(//movie) },
  for $m in //movie
  return
    element movie {
      attribute year { data($m/@year) },
      text { $m/title/text() }
    }
}
```

```
<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

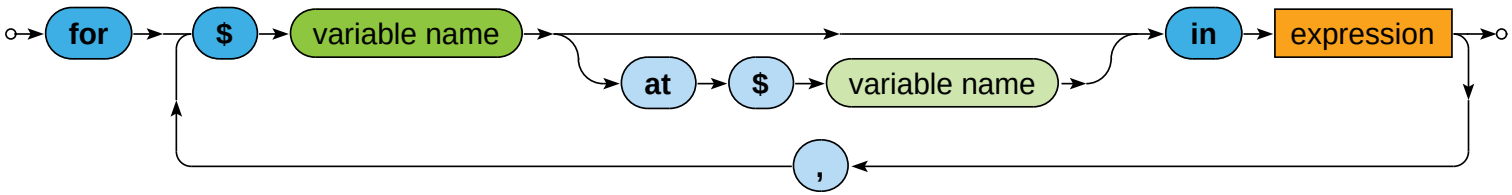
```
for $m in //movie
let $r := $m/@rating
where $r >= 75
order by $m/@year
return $m/title/text()
```

```
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

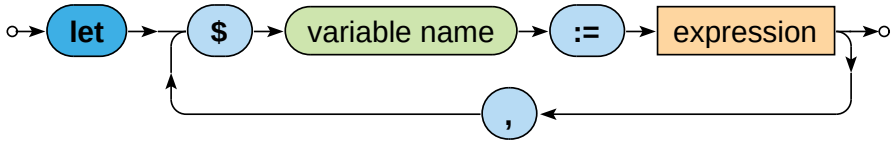



- Optional **positional variable**
  - Allows to access the ordinal number of the current item
- When **multiple input sequences** are provided…
  - Then the behavior is identical to the usage of multiple consecutive single-variable for clauses
    - I.e., as if the for loops are embedded into each other

# FLWOR Expressions: Clauses

**Let** clause

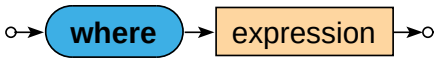
- Defines one or more auxiliary **variable assignments**



# FLWOR Expressions: Clauses

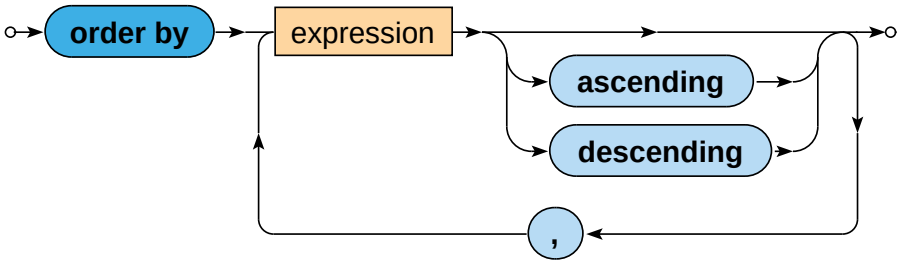
**Where** clause

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




**Order by** clause

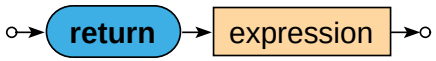
- Defines the **order in which the items are processed**



# 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 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 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>
```

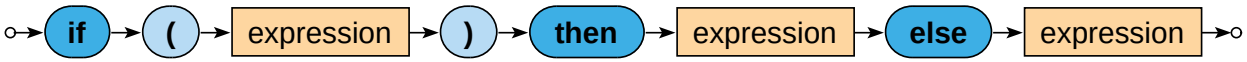
# FLWOR Examples

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

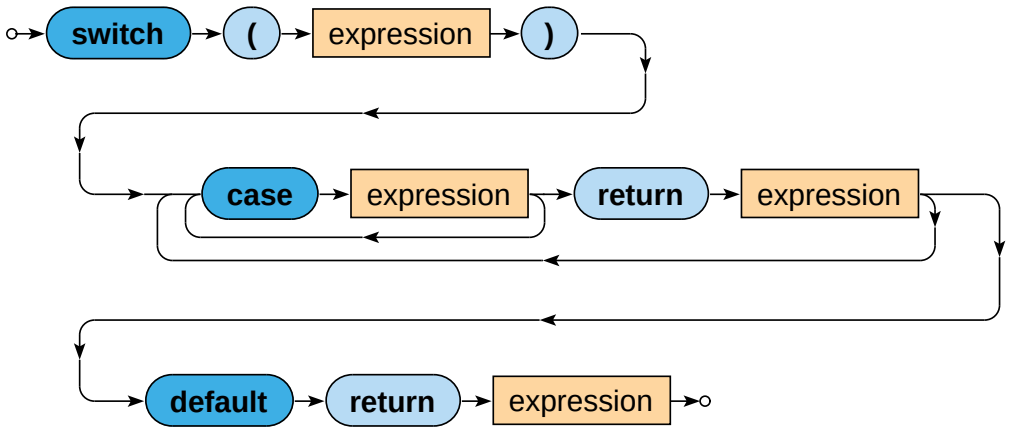
```
if (count(//movie) > 0)
then <movies>{ string-join(//movie/title, ", ") }</movies>
else ()
```

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

# 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

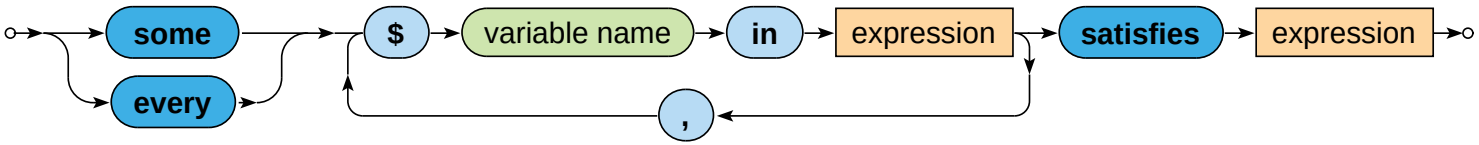
Return movies with aggregated information about their actors

```
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
```

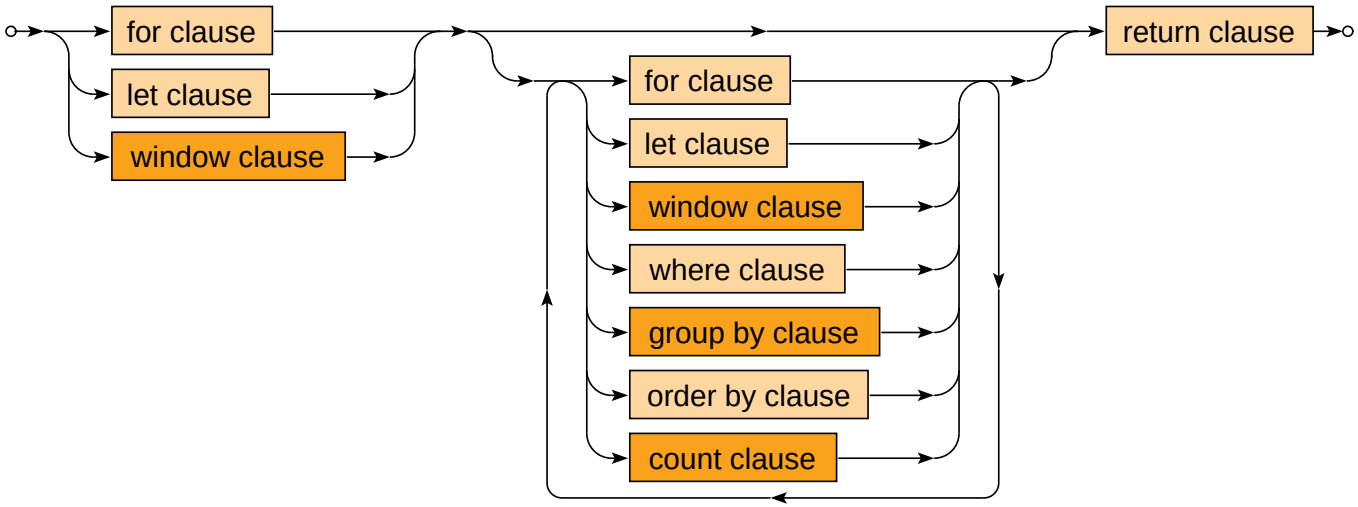
Find names of actors who played in all movies

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

```
Jiří Macháček
```

# FLWOR Expressions

Extended **FLWOR expression** (XQuery 3.0)



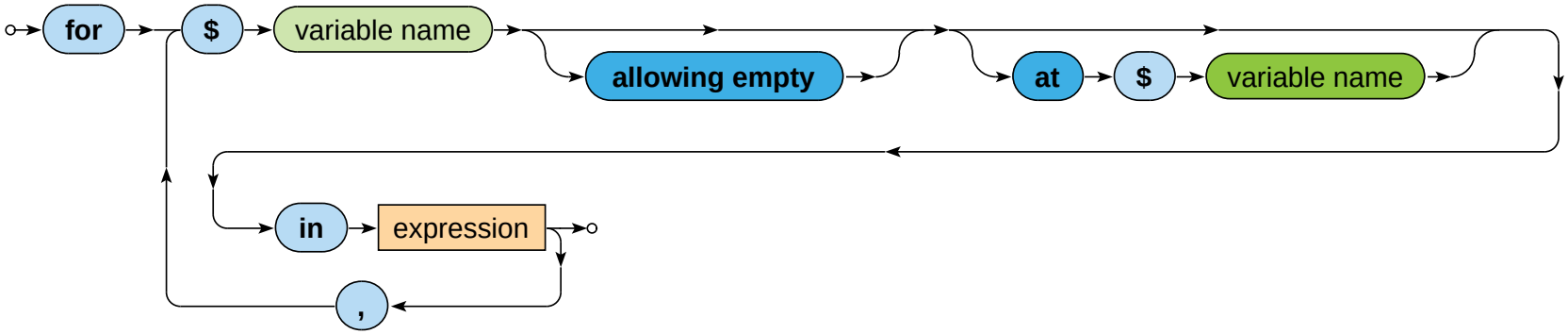

**Clauses**

- `window` – sliding or tumbling windows to iterate over
- `group by` – equality-based groupings of input items
- `count` – positional numbers of tuples in a stream

# FLWOR For Clauses

**For** clause

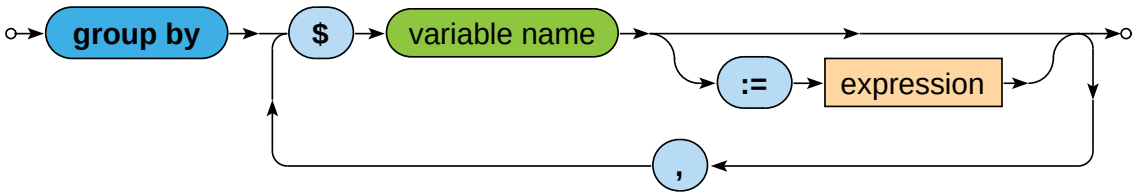
- Optional `allowing empty`
  - One `()` item is considered instead of an empty sequence
  - Suitable for outer joins
    - Does not eliminate one item when the other would be missing
- Positional variable
  - Allows to access the ordinal number of the current item



# FLWOR Group By Clauses

**Group by** clause

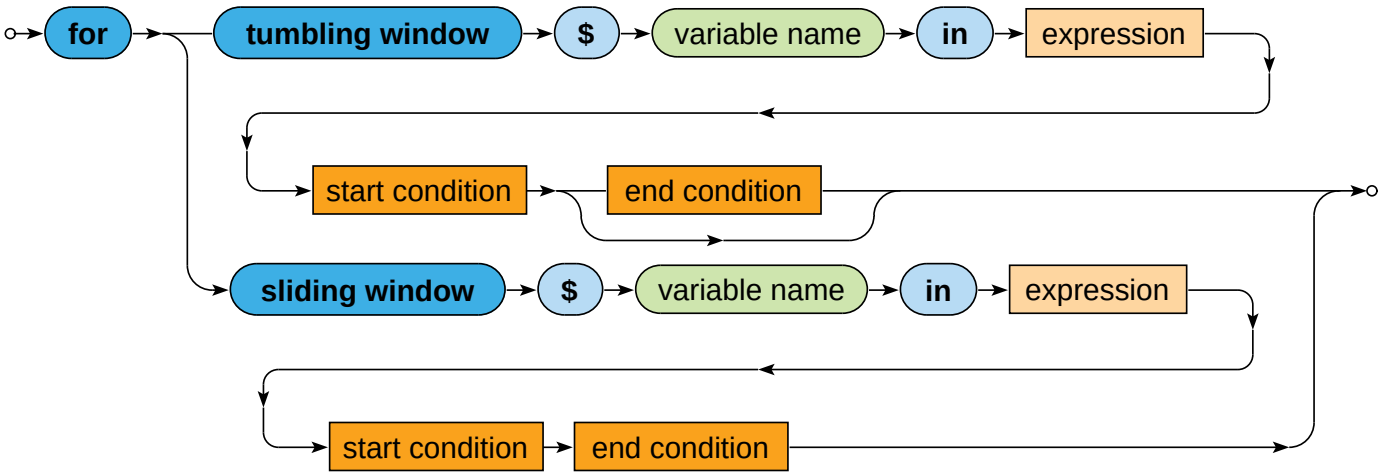
- Performs **equality-based grouping** defined by one or more grouping variables
  - Only singleton values are permitted for these variables
    - Otherwise a runtime error is raised
  - Each input item will appear only in one output group
- **Non-grouping variable** is rebound to a sequence of all the matching items from a given group



# FLWOR Window Clauses

**Window** clause

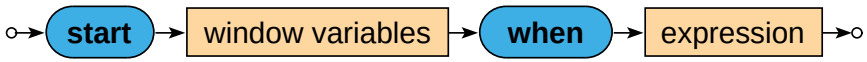
- Allows to iterate over the generated windows
  - Two modes: **tumbling** and **sliding**
- Window = **sequence of consecutive items** from the input
  - Accessible via the main variable
  - Contains the start item, end item, and all items between them



# FLWOR Window Clauses

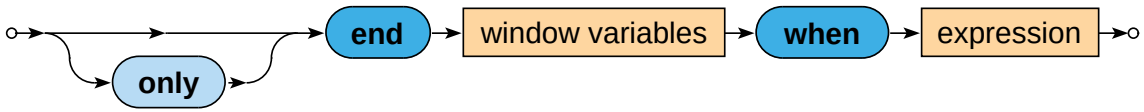
Window **start condition**

- Start item is an item that satisfies a given condition




Window **end condition**

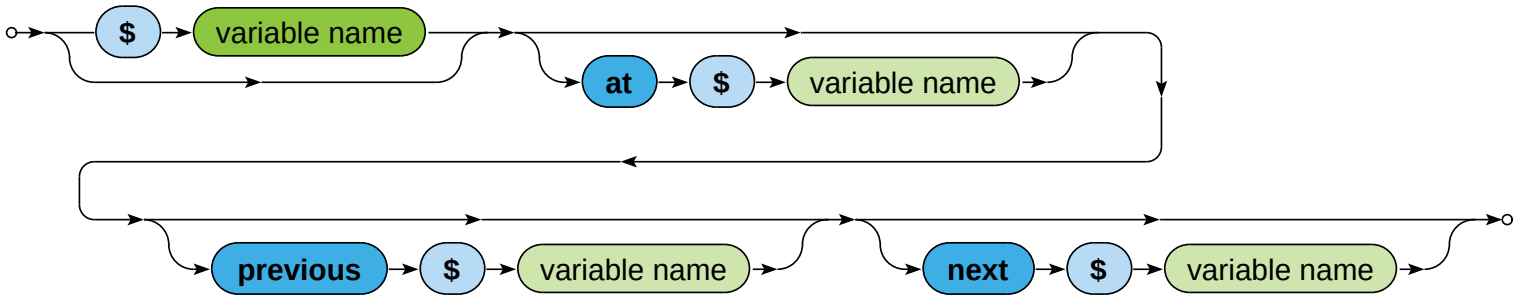
- End item is the first item (beginning with the start item) that satisfies a given condition
- When such an item cannot be found…
  - Then the last item is the very last input item
  - But only in case the `only` keyword is not specified
  - Otherwise such a window is not generated at all



# FLWOR Window Clauses

**Window variables** (all of them are optional)

- Bound to the first/last item
- `at`: bound to the ordinal position of the first/last item
- `previous`: bound to the item that precedes the first/last item
- `next`: bound to the item that follows the first/last item



# FLWOR Window Clauses

**Tumbling** window

- Search for the start item of the next window begins with the item that follows the end item of the previous window (or at the very beginning)
- $\Rightarrow$ **windows never overlap**
  - Input item may never be found in multiple windows
- When the end condition is missing…
  - All start items are first detected
  - Each window is terminated by the item that precedes the next starting one (or by the last input item at the very end)

# FLWOR Window Clauses

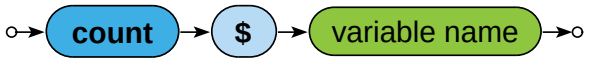
**Sliding** window

- Every item that satisfies the start condition becomes the starting item of a new window
- $\Rightarrow$ **windows may overlap**
  - Input item may be found in multiple windows

# FLWOR Count Clauses

**Count** clause

- Allows to access the ordinal number of the current tuple in a stream



# 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, …