Linguagem de Definição de Dados

Análise e Desenvolvimento de Sistemas

Paulo Maurício Gonçalves Júnior

Instituto Federal de Educação, Ciência e Tecnologia de Pernambuco

23 de agosto de 2023

Parte V

XPath

Introdução

- XPath é utilizado para selecionar partes de um documento XML.
- Projetada para ser utilizada embutida por outras linguagens, como XSLT e XQuery.
- A forma mais comum de utilizar XPath é passar uma expressão XPath e um ou mais documentos XML para uma *engine* XPath: ela avaliará a expressão e retornará o resultado, via
  - API de uma linguagem de programação;
  - Programa em linha de comando;
  - Indiretamente, embutida em outra linguagem;

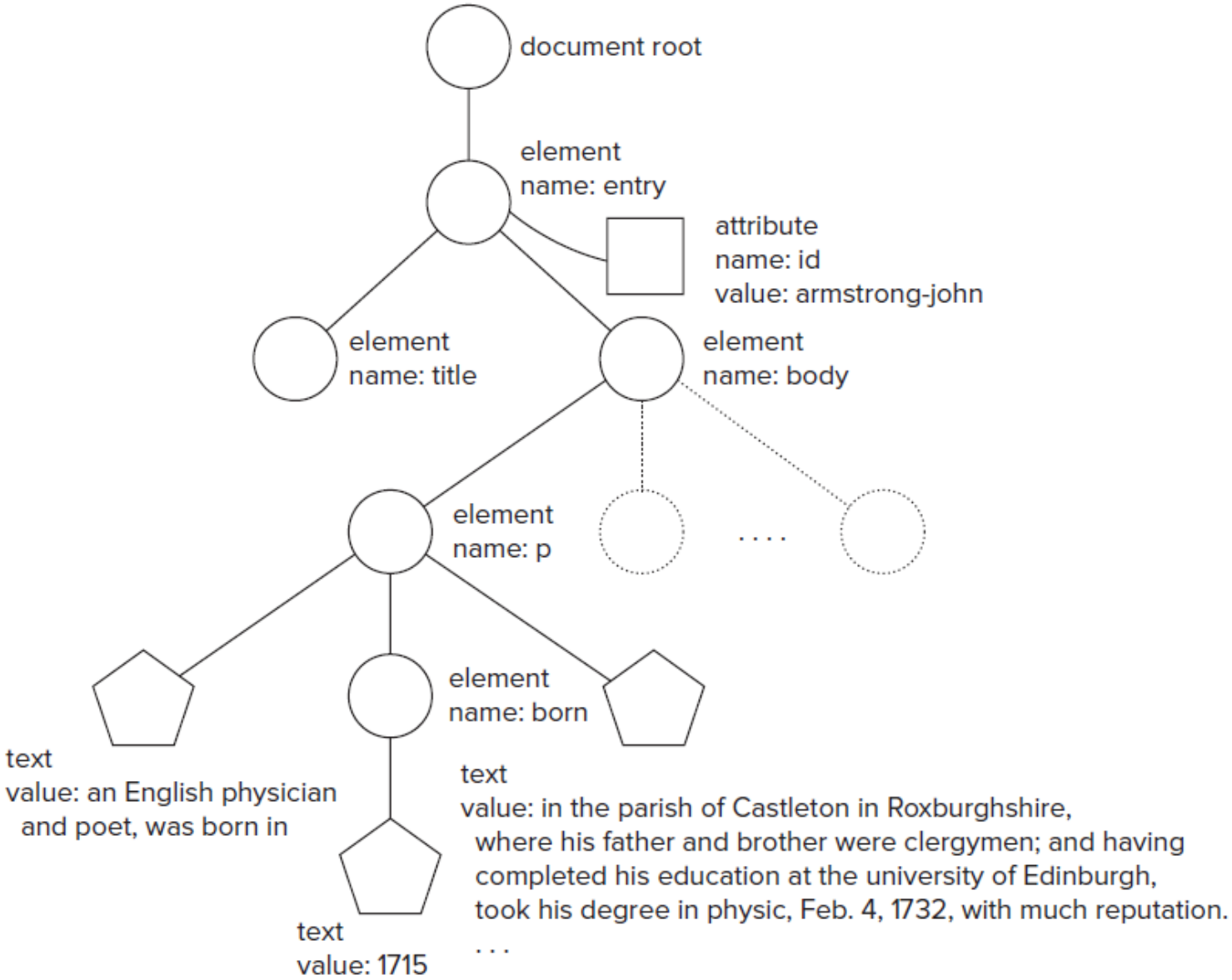
Representação em memória I

- Um documento XML é representado em memória geralmente como uma árvore.
- Por exemplo, o seguinte trecho de XML

```
<entry id="armstrong-john">
  <title>Armstrong, John</title>
  <body>
    <p>, an English physician and poet, was born in <born>1715</born>
    in the parish of Castleton in Roxburghshire, where his father and
    brother were clergymen; and having completed his education at the
    University of Edinburgh, took his degree in physics, Feb. 4, 1732,
    with much reputation. . . .
    </p>
  </body>
</entry>
```

- pode ser representado em memória da seguinte forma

Representação em memória II



Tipos de nós

- Cada item XML em uma árvore DOM como um elemento, atributo, texto é chamado de nó. Os tipos de nós que podem ser retornados são:
  - Raiz (documento)
  - Elementos
  - Atributos
  - Texto
  - Comentários
  - Instruções de processamento
  - Espaço de nomes (namespace)
- Nós DOM também possuem propriedades. Por exemplo, um elemento possui uma propriedade chamada `tagName`.
- Também é possível retornar uma lista de nós, que comumente é percorrido usando um iterador.

Exemplo de XPath

- Obter o ano de nascimento no exemplo anterior: `/entry/body/p/born`
- Se existirem várias entradas e você só quer o de um autor específico, podemos escrever: `/book/entry[@id = "armstrong-john"]/body/p/born`

Entendendo contextos

- XPath pode navegar através do documento e avaliar uma expressão em relação a qualquer nó da árvore. Ele é chamado de *item de contexto*. Ele pode ser setado em três lugares diferentes:
  - Começando com /, indicando que a pesquisa começa na raiz do documento.
  - Antes da avaliação pela linguagem ou ambiente que XPath esteja embutido (XSLT e XQuery).
  - Através do uso de predicados: `/book/entry[@id="armstrong-john"]`. O contexto para o predicado cada vez é um elemento `<entry>` diferente.

Tipos de nós e testes I

- Além de elementos, atributos e texto, podemos selecionar instruções de processamento, comentários, dentre outros.

Teste	**Tipos de nós encontrados**
`node()`	Encontra qualquer nó
`text()`	Nó de texto
`processing-instruction()`	Instrução de processamento
`comment()`	Comentários
`prefix:name`	Nome qualificado
`name`	Elemento com um dado nome
`@attr`	Atributo
`*`	Qualquer elemento
`element``(name, type)`	Elemento com um dado nome e tipo XML Schema
`attribute``(name, type)`	O mesmo acima para atributos

- Podemos usar qualquer um desses testes de nó em um caminho.

Tipos de nós e testes II

- Exemplo: `/entry/body/title/text()` que seleciona os nós de texto dentro de um elemento `<title>`.
- Precisamos dos parênteses para informar que queremos o nó de texto e não um elemento `<text>`.

Entendendo predicados

- Podemos aplicar predicados a qualquer lista de nós. Serão retornados os nós que forem avaliados como verdadeiro ou não-zero.
- Selecionar todos os elementos `<entry>` que tenham atributo `birthplace` igual a Toulouse:

```
/dictionary/entry[@birthplace="Toulouse"]
```

- A expressão dentro do predicado pode ser qualquer expressão XPath.

Predicado posicional

- Retorna um nó baseado na sua posição em relação ao seu elemento pai.
- `/book/chapter[2]` seleciona o segundo capítulo de um livro.
- O índice começa em 1.
- `/dictionary/entry[position() = 2]` é o mesmo que `/dictionary/entry[2]`
- `//table/tbody/tr[1]` != `(//table/tbody/tr)[2]`.
  - A primeira expressão seleciona todos os elementos `<tr>` que são primeiros filhos de elementos `<tbody>`.
  - A segunda expressão seleciona todos os elementos `<tr>` que são descendentes diretos de um elemento `<tbody>`, lista eles na ordem que aparece no documento e retorna o primeiro.

Contexto em predicados I

- `current()` se refere ao contexto inicial da expressão não ao elemento corrente.
- No contexto de um elemento `<def>`: `//*[@use = current()/@id]` != `//*[@use = @id]`
  - A primeira encontra cada elemento com um atributo chamado `use` cujo valor é igual ao atributo `id` do elemento corrente `<def>`.
  - A segunda retorna todo elemento cujos atributos `use` e `id` possuam o mesmo valor.
- Valor booleano efetivo: valor utilizado para identificar se um valor será retornado ou não
- Identifica se o predicado é verdadeiro (adiciona um nó a lista de resultado) ou falso.
  - Uma sequência vazia é falsa.

Contexto em predicados II

- Qualquer sequência cujo primeiro valor é um nó é verdadeiro.
- Um valor booleano, `true()` ou `false()`, retorna seu valor.
- Uma string é falsa se seu tamanho for zero, e verdadeiro caso contrário. Aplica-se a valores dos tipos xs:string , xs:anyURI , xs:untypedAtomic, e tipos derivados deles usando XML Schema.
- Valores numéricos são falsos se forem zero ou NaN, e verdadeiro caso contrário.

Passos e eixos XPath

- Eixo é uma direção, para cima ou para baixo.
- Um passo move ao longo do eixo escolhido.
  - `/book/chapter` é a forma resumida de `/child::book/child::chapter`
  - / é o passo.
- Eixos em XPath:

Atalho	**Nome completo**	**Significado**
`name`	`child::`	Eixo padrão
`//`	`descendant::`	Seleciona nós descendentes ao nó de contexto
`@`	**`attribute`**`::`	Seleciona um atributo do nó de contexto
	`self::`	Seleciona o nó atual
	`descendant-or-self::`	Nó atual e todos os seus descendentes

Passos e eixos XPath

Atalho	Nome completo	Significado
	`following-sibling::`	Elementos de mesmo nível do nó de contexto
	`following::`	Elementos que aparecem após o nó de contexto
`..`	`parent::`	Pai do nó de contexto
	`ancestor::`	Nós de nível superior ao nó de contexto
	`preceding-sibling::`	Oposto de `following-sibling::`
	`preceding::`	Oposto de `following::`
	`ancestor-or-self::`	Oposto de `descendant-or-self::`
	`namespace``::`	Aula de XSLT

- `//a[preceding-sibling::a/@href = ./@href]` seleciona os elementos `<a>` em qualquer posição no documento e então usa um predicado para escolher apenas aqueles elementos que possuem um elemento precedente de mesmo nível que também seja `<a>` que possua o mesmo valor para o atributo `href`.

Expressões XPath

- Exemplos de expressões em XPath:
  - `count(//p)`
  - `2+2`
  - `1 + count(//p) + string-length(@id)`
  - `10 idiv 4`
  - `(1, 2, 3)`
  - `(1 to 100)[. mod 5 eq 0]`
    - ★ << e >>: $A << B$ se A e B são nós e A ocorre antes de B
    - ★ `to`: 3 `to` 20 retorna a sequência de inteiros $3, 4, 5, \ldots, 19, 20$

Igualdade em XPath

- O operador = opera em sequências ou lista de nós. Se A e B são duas sequências, $A = B$ significa "existe pelo menos um valor que aparece em A e B".
  - `(1, 2, 3) = (2, 4, 6, 8)`: verdadeiro
  - `("a", "hello", "b") = ("c", "Hello", "A")`: falso
  - `3 = (1, 2, 3)`: verdadeiro
- `eq` pode ser usado para comparar valores individuais.
- `is` pode ser usado para comparar se duas variáveis se referem ao mesmo nó (e não se possuem o mesmo conteúdo).
- `deep-equal` pode ser usado para comparar ramos inteiros de uma árvore com relação às suas estruturas e conteúdos.

Comparações

- Existem duas formas de comparação:
  - Comparação geral: =, !=, <, <=, >, >=
  - Comparação de valor: **eq**, **ne**, **lt**, **le**, **gt**, **ge**
- Esta expressão retorna verdadeiro se qualquer atributo `q` tem um valor maior que 10: `$bookstore//book/@q > 10`
- Esta expressão retorna verdadeiro se existe apenas um atributo `q` retornado pela expressão, e seu valor é maior que 10. Se mais de um `q` é retornado, um erro ocorre: `$bookstore//book/@q gt 10`

Variáveis em Expressões XPath

- XPath é uma linguagem declarativa, significando que você descreve o resultado desejado e o computador procura o resultado.
- Não existe iteração nem mudança de valor de variáveis.
- Na realidade, em XPath 1, não é possível setar variáveis, vindo todas da linguagem hospedeira.
- Variáveis começam com $, podendo conter listas de nós, sequencias, ou tipos atômicos.
  - `$e/entry[@born le $year and @died ge $year]`

Novas Expressões em XPath 2.0

for

- XPath 2 introduziu as expressões **for**, **let**, **where**, **order by**, **return**, **if**, **some**, and **every**, bem como casts.
- O formato da expressão **for** é simples. Vejamos alguns exemplos:

```
for $i in (1 to 20)
return $i * 5

for $lastname in (//person/name/last), $i in (1 to string-length(
    $lastname))
return substring($lastname, 1, $i)

string-join(for $i in (1 to 20) return string($i * 5), ", ")

for $x at $i in doc("books.xml")/bookstore/book/title
return <book>{$i}. {data($x)}</book>
```

Novas Expressões em XPath 2.0

let e if

- O formato da expressão **let** também é simples:

```
let $v := 6 return $v + 1
```

- Também é possível declarar mais de uma variável:

```
let $entry := //entry[@id = "galileo"], $born := $entry/@born,
$died := $entry/@died return $died - $born
```

- XPath 2 possui a sequencia **if**/**then**/**else**, onde o **else** é obrigatório.
- `if (name() eq "entry") then "yes" else "no"`
- `if (true()) then population/$i else population/0`: não funciona, pois o **else** é inválido.

Expressões de cast e tipo

- **instance of**: Verifica se um valor é de um tipo

```
if ($input instance of element(sock)) then $input else ()
```

- **cast**: Converte um valor de um tipo para outro

```
//entry[(@born cast as xs:integer) gt 1700]
```

- **castable**: Verifica se um valor pode ser convertido para outro

```
if ($entry/@born castable as xs:integer) then $entry/@born cast as
     xs:integer else 42
```

- **treat as**: Similar a uma *assertion*. Se um valor não for de um tipo, um erro ocorre

```
math:sqrt($x treat as xs:double)
```

Funções

- XPath possui funções embutidas para auxiliar na expressividade da linguagem.
- A função `doc()` lê um documento XML externo e retorna o nó documento dele.
- Funções sobre strings: `substring()`, `substring-before()`, `substring-after()`, `translate()`, `matches()`, `replace()`, `concat()`, `string-length()`, dentre outras.
- Funções numéricas: `sum()`, `floor()`, `ceiling()`, `round()`, `abs()`, `round-half-to-even()`, dentre outras.
- Podemos definir nossas próprias funções:

```
let $addTax := function($a as xs:double) {
$a * 1.13
} return $addTax(/invoice/amounts/total)
```

Operações sobre conjuntos

Operação	**Expressão XPath**
. ocorre dentro de $a	`count($a|.) = count($a)`
Todos os nós em $a e $b	`$a	$b` ou `$a union $b`
Apenas os nós que estão em $a e $b	`$a intersect $b`
Nós em $a mas não em $b	`$a except $b`

Parte VI

XQuery

Introdução

- Enquanto XSLT usa XPath, XQuery estende XPath.
- XPath e XQuery são construídos sob o mesmo modelo abstrato de dados, XDM. Ou seja, ele não opera sob documentos XML e sim sobre árvores abstratas chamadas instâncias de modelo de dados.
- Versões recentes da SQL permitem embutir expressões XQuery no meio de comandos SQL.

FLWOR

- As principais funcionalidades de XQuery estão nas expressões FLWOR, em funções e em módulos.
- FLWOR significa **for**, **let**, **where**, **order by** e **return**. Exemplos:

```
for $boy in doc("students.xml")/students/boy
where $boy/eye-color = "yellow"
return $boy/name

for $dude in doc("chalmers-biography-extract.xml")//entry
where xs:integer($dude/@died) lt 1600
order by $dude/@died
return $dude/title

for $a in 1 to 5, $b in ("a", "b", "c")
return <e id="{$b}{$a}"/>
```

Anatomia de uma expressão XQuery

- Expressões em XQuery podem começar com uma declaração de versão, seguida de um prólogo e do corpo da consulta.
- Declaração de versão é opcional:

```
xquery version "1.0" encoding "utf-8";
```

Prólogo I

- O prólogo é o local para definições e propriedades. Pode-se definir funções, prefixos de namespaces, importar esquemas, definir variáveis, dentre outras.
- Declaração de espaço de nomes

```
declare namespace fobo = "http://www.fromoldbooks.org/ns/";
```

- Importando XML Schemas: usados para se referir aos tipos, para fins de validação

```
import schema fobo="http://www.fromoldbooks.org/Search/";
import schema "http://www.exmple.org/" at "http://www.example.org/
    xsdfiles/";
import schema fobo="http://www.fromoldbooks.org/Search/" at "http
    ://www.fromoldbooks.org/Search/xml/search.xsd", "http://www.
    fromoldbooks.org/Search/xml/additional.xsd";
```

  - O primeiro associa o espaço de nomes ao prefixo mas não informa onde encontrar o esquema.

Prólogo II

- O segundo associa um espaço de nomes a uma URI onde está um esquema.
- O terceiro informa o prefixo, o espaço de nomes e onde obter o esquema (neste caso, mais de um local).

- Importando módulos: uma coleção de definições em XQuery.

```
import module namespace fobo="http://www.example.org/ns/" at "
    fobo-search.xqm";
import module "global-defs.xqm";
```

- Quando você importa um módulo você tem acesso a suas funções e variáveis públicas.
- Módulos são escritos em XQuery começando com uma declaração.

```
module namespace w = "http://www.example.org/wikidates";
```

- Variáveis: não é possível mudar seu valor um vez atribuído.

Prólogo III

```
declare variable $socks := "black";
declare variable $sockprice as xs:decimal := 3.6;
declare variable $argyle as element(*) := <sock>argyle</sock>;
```

- Funções

```
declare variable $james := <person><name>James</name><socks>argyle
    </socks></person>;
declare function local:get-sock-color($person as element(person))
    as xs:string {
  xs:string($person/socks)
};
local:get-sock-color($james)
```

Corpo da Consulta I

- Expressões XPath só retornam ponteiros para a árvore do documento.
- Para retornar elementos sem seus descendentes ou criar novos elementos não existentes no documento, precisamos usar XQuery.
- Podemos para isso usar construtores de elementos. Eles podem ser *diretos* ou *computados*.
- Diretos:

```
let $isaac := <entry id="newton-isaac" born="1642" died="1737"><
    title>Sir Isaac Newton</title></entry>
return $isaac/title
```

- Computados:

Corpo da Consulta II

```
declare namespace svg = "http://www.w3.org/2000/svg";
let $width := 30, $height := 20, $isaac := <entry id="newton-isaac
    " born="1622" died="1736"><title>Sir Isaac Newton</title></
    entry>, $box := element svg:box {
  attribute width { $width },
  attribute height { $height },
  attribute x { $isaac/@born },
  attribute y { math:sin(xs:integer($isaac/@died)) }
}, $p := element text { fn:concat("His name was ", data($isaac/
    title), ".")}
return ($box, $p)
```

FLWOR

for

- Permite iterar sobre um conjunto de nós ou sequência de valores

```
for $var [as xs:integer] [allowing empty] [at $pos] in expr
```

- `as` permite converter o valor da variável
- `at` permite criar uma variável com a posição da variável na coleção
- Exemplos:

```
for $entry as element(entry) at $n in //entry
return <li>{$n}. {$entry/@id}</li>

for $a in (1, 2, 3), $b in (4, 5)
return $a + $b

for $a allowing empty in ()
return 42
```

FLWOR

let

- Permite atribuir um valor a uma variável

```
let $var [as type] := expression
```

- Exemplos:

```
let $x as xs:decimal := math:sin(0.5)
return $x

for $a in (1, 2, 3)
let $b := $a * $a
return <r>{$b}</r>
```

FLWOR

where

- Permite filtrar tuplas
- Exemplo:

```
for $a in (1 to 50), $b in (2, 3, 5, 7)
where $a mod $b eq 0
return $a * $b
```

FLWOR

order by

- Permite ordenar tuplas baseado no valor de uma expressão

```
[stable] order by expression [direction] [collation "URI"]
```

- Direção:

```
ascending|descending [ empty (greatest|least) ]
```

- `stable` permite manter os itens na mesma ordem se eles possuem a mesma chave
- `collation` permite comparação de strings, para por exemplo, permitir que letras com acento e sem acento sejam tratadas como idênticas
- Exemplos:

```
order by $b ascending empty least
order by $b descending empty greatest
order by $b stable ascending
order by $e
```

FLWOR

count

- Permite contar a quantidade de iterações
- Exemplos:

```
for $boy at $boypos in ("Simon", "Nigel", "David"), $game at
    $gamepos in ("pushups", "situps")
  count $count
  return
    <tuple n="{$count}">
      boy {$boypos} is {$boy}, item {$gamepos}: {$game}
    </tuple>
```

FLWOR

group by

- Permite agrupar os dados a partir de uma variável
- Exemplos:

```
for $e in /dictionary/entry[@birthplace]
  let $d := $e/@birthplace
  group by $d
  order by $d
  return
    if (count($e) eq 1) then () else
    <group birthplace="{$e[1]/@birthplace}">
    {
      for $person in $e
      return
        <person id="{$person/id}" born="{$person/@born}" died="{
    $person/@died}">{data($person/title)}</person>
    }
    </group>
```

FLWOR I

window

- Permite iterar sobre janelas de valores da sequência principal
- A quantidade de janelas e os itens da janela são determinados pelo tipo de janela bem como restrições específicas.
- A primeira linha informa o tipo de janela seguido por uma variável representando a janela.

```
for tumbling window $w in ('A', 'B', 'BB', 'BBB', 'C', 'CC', 'CCC
    ', 'CCCC')
```

- Em seguida informamos a restrição de início: define quando a janela começa

```
for tumbling window $w in ('A', 'B', 'BB', 'BBB', 'C', 'CC', 'CCC
    ', 'CCCC')
start $s when string-length($s) = 1
```

FLWOR II

window

- Em seguida informamos a restrição de fim: define quando a janela termina (opcional)

```
for tumbling window $w in ('A', 'B', 'BB', 'BBB', 'C', 'CC', 'CCC
    ', 'CCCC')
start $s when string-length($s) = 1
end $e when string-length($e) = 2
```

- Tipos de janela:
  - **tumbling**: janelas não se sobrepõem
  - **sliding**: janelas podem se sobrepor
- Exemplos:

FLWOR III

window

```
for tumbling window $w in ('A', 'B', 'BB', 'BBB', 'C', 'CC', 'CCC
    ', 'CCCC')
  start $s when string-length($s) = 1
  end $e when string-length($e) = 2
  return <window>
      {
       for $i in $w
       return <item>{$i}</item>
      }
     </window>
```

- Variáveis:
  - `at`: adicionando ao `start` ou `end` gera um índice do valor da sequência

FLWOR IV

window

```
<windows>
 {
  for tumbling window $w in ('A', 'B', 'C', 'D', 'E', 'F', 'G
    ')
  start at $s when true()
  end at $e when $e - $s eq 2
  return <window>
       {
        for $i in $w
        return <item>{$i}</item>
       }
      </window>
 }
</windows>
```

- **only**: só gerar uma janela se a condição estiver satisfeita

FLWOR V

window

```
<windows>
 {
  for tumbling window $w in ('A', 'B', 'C', 'D', 'E', 'F', 'G
    ')
  start at $s when true()
  only end at $e when $e - $s eq 2
  return <window>
       {
        for $i in $w
        return <item>{$i}</item>
       }
      </window>
 }
</windows>
```

- **next**: obtém o próximo item da sequência
- **previous**: obtém o item anterior da sequência

FLWOR VI

window

```
<windows>
  {
    for tumbling window $w in ( 'A', 'B', 'B', 'A', 'A', 'C',
    'D', 'D', 'D', 'E', 'A',  'A')
    start $s next $s-next when  $s = $s-next
    end $e previous $e-previous next $e-next when $e-previous
    = $e and $e-next != $e
    return <window>
        {
              for $i in $w
              return <item>{$i}</item>
        }
       </window>
   }
</windows>
```

FLWOR

try/catch

- Permite tratar um erro quando ele ocorre

```
for $i in (2, 0.2, 0.0, 4)
return
  try {
    12 div $i
  } catch * {
    42
  }
```

FLWOR

switch

```
for $word in ("the", "an", "a", "apple", "boy", "girl")
return (" ",
switch (substring($word, 1, 1))
  case "a" return upper-case($word)
  case "t" return $word
  case "b" case "B" return <b>{$word}</b>
  default return $word)
```