Consulta a dados XML - XQuery

Vanessa Braganholo

Linguagem de Consulta para XML

- 1996 – padronização do XML
- Cada vez mais dados disponíveis neste formato
  - Precisam ser consultados!

- Várias propostas surgiram na academia

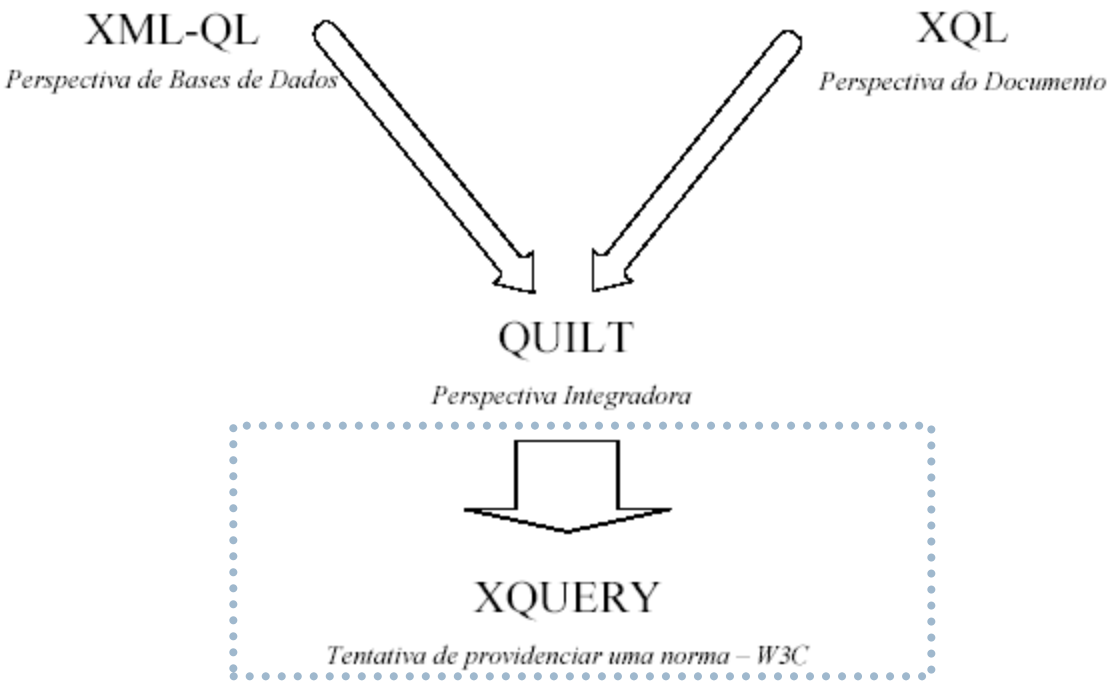
Linguagens de Consulta para XML

- Diferentes abordagens :
  - Orientadas para BD (XML-QL, Lorel, Yetl)
    - Sintaxe semelhante a consultas de BD relacionais
  - Orientadas para documentos (XQL)
    - Paradigma Funcional
      - Se baseiam em expressões de caminhos
      - Eficientes na realização de consultas em profundidade
  - Mistas (QUILT, XQuery)

Padronização?

- W3C define grupo de estudos para propor uma linguagem de consulta padrão
- Define requisitos para esta nova linguagem
  - http://www.w3.org/TR/xquery-requirements

Linguagens de Consulta XML - Evolução



XQL (XML Query Language)

- Compacta, fácil de expressar e ler
- Simples para casos usuais
- “Embutível” em programas, scripts, URLs
- Os resultados das consultas não retornam documentos com estruturas diferentes do original
- Não permite a consulta de várias fontes

XQL

- Processamento de consultas sobre documentos XML
  - Utiliza a idéia de contexto
    - Delimitado pelas expressões de caminhos
    - Conjunto de nós (DOM)
  - Resultado preserva ordem, hierarquia e identidade dos objetos
    - XML-QL permite skolem (auto-numeração)
- Suporta função de agregação
- Não suporta inserção, exclusão, atualização, etc

XQL: Sintaxe

- Imita a sintaxe de navegação da URI
- Notação
  - / : contexto raiz
  - ./ : contexto corrente
  - // : descendente recursivo a partir da raiz
  - .// : descendente recursivo a partir do nó corrente
  - @ : atributo
  - * : qualquer elemento

Exemplo

```
<?xml version='1.0'?>
<!-- This file represents a fragment of a book store inventory database -->
<bookstore specialty='novel'>
  <book style='autobiography'>
    <title>Seven Years in Trenton</title>
    <author>
      <first-name>Joe</first-name>
      <last-name>Bob</last-name>
      <award>Trenton Literary Review Honorable Mention</award>
    </author>
    <price>12</price>
  </book>
<my:book style='leather' price='29.50' xmlns:my='http://www.placeholder-
    name-here.com/schema/'>
    <my:title>Who's Who in Trenton</my:title>
    <my:author>Robert Bob</my:author>
  </my:book>
</bookstore>
```

XQL: Exemplos (1)

- ./author ≡ author
- /bookstore
- book[bookstore/@specialty = @style]
- author/first-name
- author/*
- *[@specialty]

XQL: Exemplos (2)

- book[@style]
- author[first-name][2]
- book[excerpt]/author[degree]
- book[excerpt][title] ≡ book[excerpt and title]
- author[name = …] ≡ author[name eq …]
- author[. = ‘Bob’] ≡ author[text() = ‘Bob’]
- author[first-name!text() = ‘Bob’]
- degree[index() lt 3] ≡ degree[index() < 3]

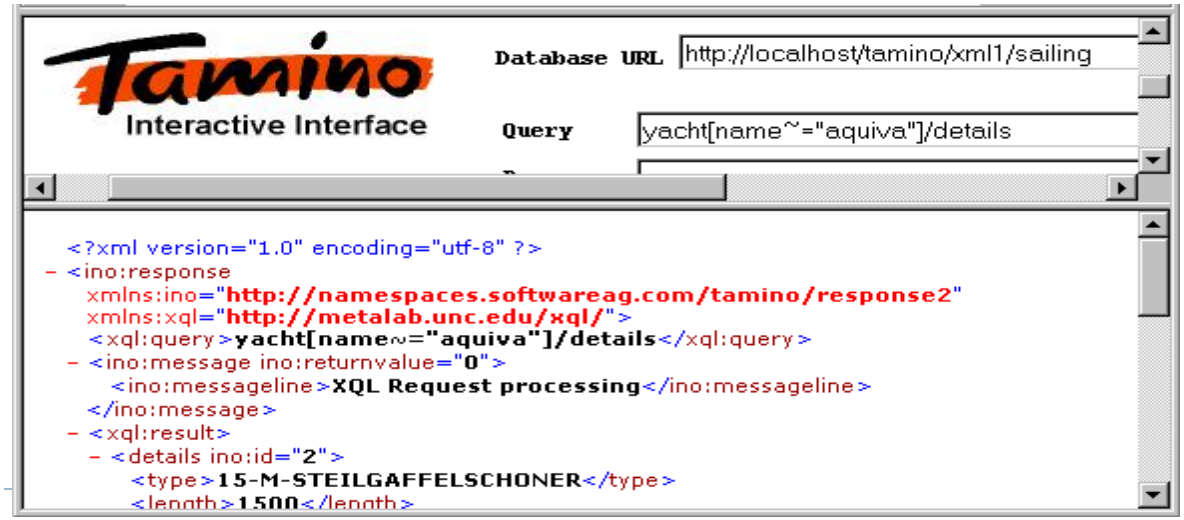
XQL: Exemplos (3)

- author[publications!count() > 10]
- books[pub_date < date(‘1995-01-01’)]
- books[pub_date < date(@first)]
- bookstore/(book | magazine)
- //comment()[1]
- ancestor(book/author)
- author[0, 2 to 4, -1]

Tamino - Interface Interativa

- Primeiro SGBD XML nativo
- XQL como linguagem de consulta

```
<?xml version="1.0" encoding="iso-8859-1" ?>
- <ino:response
    xmlns:ino="http://namespaces.softwareag.com/tamino/response2"
    xmlns:xql="http://metalab.unc.edu/xql/">
    <xql:query>yacht[name~="aquiva"]/details</xql:query>
  - <ino:message ino:returnvalue="0">
      <ino:messageline>XQL Request processing</ino:messageline>
    </ino:message>
  - <xql:result>
    - <details ino:id="2">
        <type>15-M-STEILGAFFELSCHONER</type>
```



XQuery

XQuery

- W3C XML Query Language
  - Linguagem declarativa similar a SQL
- Derivada diretamente do Quilt e XPath
- Mas aproveita características de muitas outras:
  - **XQL:**
    - sintaxe baseada em expressões de caminho, adequadas para hierarquias
  - **SQL:**
    - cláusulas baseadas em palavras reservadas tipo (SELECT-FROM-WHERE)
  - **OQL:**
    - noção de uma linguagem funcional, composta de expressões que podem ser encadeadas
- http://www.w3.org/TR/xquery/
  - Status: Recomendação em Janeiro de 2007

Estrutura da Linguagem

- Expressões FLWOR
- Expressões XPath
- Expressões Condicionais (análogo ao IF-THEN-ELSE das linguagens de programação)
- Construtores de Elementos
- Quantificador Existencial e Universal
- Cast de Tipos

XQuery é capaz de

- Gerar respostas com estrutura diferente do documento consultado
- Consultar vários documentos
- Gerar texto puro ou fragmentos de documentos XML

XQuery

- XQuery x XPath
  - XPath não produz resultados de consultas em uma estrutura diferente da existente no documento
  - XPath não permite realizar junções entre dados de dois documentos XML

- XQuery x XSLT
  - XSLT é mais adequado à tranformação e XQuery à consulta

Onde XQuery pode ser usada ?

- Ferramentas GUI
- Linhas de comando
- Programas escritos em Java, C++, e outras linguagens que necessitam extrair dados de documentos XML

XQuery (XML) ≈ SQL (Banco de Dados)

Exemplo JXQI - Java XQuery API

```
// import
public class ExecutaXQuery {
   public static void main(String[] args) throws Exception {
      XQueryContext ctx = new XQueryContext();
      Reader strm = new FileReader(args[0]);
      PreparedXQuery xq = ctx.prepareXQuery(strm);
      XQueryResultSet rset = xq.executeQuery(false);
      while (rset.next()) {
         XMLNode node = rset.getNode();
         // node.print
          …
      }
   }
}
```

XQuery – Estrutura de uma consulta

- Uma consulta XQuery pode ser dividida em três partes:
  - Declarações de Schema *
  - Definição de funções *
  - Expressões de consulta

* Opcional

XQuery – Estrutura – Exemplo

```
# Parte 1: Namespace e Declarações de Schema
namespace xsd = "http://www.w3.org/2000/10/XMLSchema"

# Parte 2: Definição de funções
define function factorial (xsd:integer $n) returns xsd:integer {
        if  ($n eq 0)
        then 1
        else $n * factorial($n - 1)
        }
# Parte 3: Expressões de consulta
<Results>
        <Description>Factorial of 10</Description>
        <Value>{factorial(10)}</Value>
</Results>
```

XQuery - Exemplos

Exemplos

- Os exemplos serão realizados sobre o documento de empregados
- Arquivo emps.xml

```
<? xml version="1.0" ?>
<empregados>
  <empregado cod="E01" dept="D01">
     <nome>João</nome>
     <inicial-meio>S.</inicial-meio>
     <sobrenome>Santos</sobrenome>
  </empregado>
  <empregado cod="E02" dept="D01">
     <nome>Ana</nome>
     <sobrenome>Ferraz</sobrenome>
  </empregado>
</empregados>
```

Construtor de Elemento

```
<emp-dept>
    {for $e in doc('emps.xml')//empregado
     return $e/nome
     }
</emp-dept>
```

Constrói no resultado um elemento emp-dept, que não existe no documento de origem

Construtor de Elemento

```
<emp-dept>
    {for $e in doc('emps.xml')//empregado
     return $e/nome
     }
</emp-dept>
```

$e/nome também é um construtor de elemento. Para entendê-lo, primeiro é preciso entender como a consulta é processada.

Exemplo de funcionamento

```
<emp-dept>
    {for $e in doc('emps.xml')//empregado
     return $e/nome
     }
</emp-dept>
```

1. Elemento é construído na saída

<emp-dept>

Exemplo de funcionamento

```
<emp-dept>
    {for $e in doc('emps.xml')//empregado
     return $e/nome
     }
</emp-dept>
```

2. A chave { indica que o próximo trecho precisa ser processado

<emp-dept>

Exemplo de funcionamento

```
<emp-dept>
    {for $e in doc('emps.xml')//empregado
     return $e/nome
     }
</emp-dept>
```

3. Expressão **for** liga a variável **$e** aos elementos empregado do documento emps.xml. O **for** itera sobre os elementos , um de cada vez, e a cláusula **return** é executada em cada iteração

Exemplo de funcionamento

```
<emp-dept>
    {for $e in doc('emps.xml')//empregado
     return $e/nome
     }
</emp-dept>
```

4. $e é ligada ao primeiro empregado

$e

```
<? xml version="1.0" ?>
<empregados>
  <empregado cod="E01" dept="D01">
     <nome>João</nome>
     <inicial-meio>S.</inicial-meio>
     <sobrenome>Santos</sobrenome>
  </empregado>
  <empregado cod="E02" dept="D01">
     <nome>Ana</nome>
     <sobrenome>Ferraz</sobrenome>
  </empregado>
</empregados>
```

Exemplo de funcionamento

```
<emp-dept>
    {for $e in doc('emps.xml')//empregado
     return $e/nome
     }
</emp-dept>
```

5. Cláusula $e/nome constrói o elemento nome no resultado

```
<emp-dept>
   <nome>João</nome>
```

Exemplo de funcionamento

```
<emp-dept>
    {for $e in doc('emps.xml')//empregado
     return $e/nome
     }
</emp-dept>
```

6. $e é ligada ao segundo empregado

$e

```
<? xml version="1.0" ?>
<empregados>
  <empregado cod="E01" dept="D01">
     <nome>João</nome>
     <inicial-meio>S.</inicial-meio>
     <sobrenome>Santos</sobrenome>
  </empregado>
  <empregado cod="E02" dept="D01">
     <nome>Ana</nome>
     <sobrenome>Ferraz</sobrenome>
  </empregado>
</empregados>
```

Exemplo de funcionamento

```
<emp-dept>
    {for $e in doc('emps.xml')//empregado
     return $e/nome
     }
</emp-dept>
```

7. Cláusula $e/nome constrói o elemento nome no resultado

```
<emp-dept>
   <nome>João</nome>
   <nome>Ana</nome>
```

Exemplo de funcionamento

```
<emp-dept>
    {for $e in doc('emps.xml')//empregado
     return $e/nome
     }
</emp-dept>
```

8. A marca emp-dept é fechada

```
<emp-dept>
   <nome>João</nome>
   <nome>Ana</nome>
</emp-dept>
```

Cláusula FOR

- Faz parte de uma cláusula mais complexa FLWOR
  - **F**OR
  - **L**ET
  - **W**HERE
  - **O**RDER BY
  - **R**ETURN

Analogia com SQL

for ⇔ SQL from
where ⇔ SQL where
return ⇔ SQL select
let (sem equivalência SQL) para variáveis temporárias, principalmente para execução de agregações

- FOR/LET associam valores às variáveis
- WHERE filtra o resultado vindo das cláusulas FOR/LET
- RETURN gera a saída da consulta

Exemplo WHERE/ORDER BY

```
<emp-dept>
{
    for $e in doc('emps.xml')//empregado
    where $e/@dept='D01'
    order by $e/nome
    return $e/nome
}
</emp-dept>
```

Todos os empregados são ligados a **$e** (um de cada vez), mas a cláusula **return** só é executada para os que satisfazem a condição **$e/@dept="D01"**

Exemplo WHERE/ORDER BY

```
<emp-dept>
{
    for $e in doc('emps.xml')//empregado
    where $e/@dept='D01'
    order by $e/nome
    return $e/nome
}
</emp-dept>
```

Além disso, os resultados são ordenados por **$e/nome**

Exemplo WHERE/ORDER BY

```
<emp-dept>
{
    for $e in doc('emps.xml')//empregado
    where $e/@dept='D01'
    order by $e/nome
    return $e/nome
}
</emp-dept>
```

Resultado

```
<emp-dept>
    <nome>Ana</nome>
    <nome>João</nome>
</emp-dept>
```

```
<? xml version="1.0" ?>
<empregados>
  <empregado cod="E01" dept="D01">
      <nome>João</nome>
      <inicial-meio>S.</inicial-meio>
      <sobrenome>Santos</sobrenome>
  </empregado>
  <empregado cod="E02" dept="D01">
      <nome>Ana</nome>
      <sobrenome>Ferraz</sobrenome>
  </empregado>
</empregados>
```

Expressão XPath

- doc(“emps.xml”)//empregado[@cod=“E01”]/nome

- Equivalente em XQuery

```
for $e in doc(“emp.xml”)//empregado
where $e/@cod = “E01”
 return $e/nome
```

Exemplo CONSULTA ANINHADA

```
<departamentos>
   {for $d in distinct-values(doc('emps.xml')//empregado/@dept)
    return
    <departamento>
       <codigo>{$d}</codigo>
       <empregados>
       {for $e in doc('emps.xml')//empregado
        where $e/@dept=$d
        return
        <empregado>
           {$e/nome}
           {$e/sobrenome}
       </empregado>
       }
       </empregados>
    </departamento>
}
</departamentos>
```

distinct-values seleciona apenas os departamentos distintos

Exemplo CONSULTA ANINHADA

```
<departamentos>
  {for $d in distinct-values(doc('emps.xml')//empregado/@dept)
   return
   <departamento>
     <codigo>{$d}</codigo>
     <empregados>
     {for $e in doc('emps.xml')//empregado
      where $e/@dept=$d
      return
      <empregado>
        {$e/nome}
        {$e/sobrenome}
      </empregado>
     }
     </empregados>
   </departamento>
}
</departamentos>
```

O for **$e** será executado uma vez para cada valor em **$d** (como um for aninhado em linguagem de programação)

Resultado Exemplo

```
<departamentos>
  {for $d in distinct-values(doc('emps.xml')//empregado/@de
   return
   <departamento>
     <codigo>{$d}</codigo>
     <empregados>
     {for $e in doc('emps.xml')//empregado
      where $e/@dept=$d
      return
      <empregado>
        {$e/nome}
        {$e/sobrenome}
      </empregado>
     }
     </empregados>
   </departamento>
}
</departamentos>
```

```
<? xml version="1.0" ?>
<empregados>
  <empregado cod="E01" dept="D01">
     <nome>João</nome>
     <inicial-meio>S.</inicial-meio>
     <sobrenome>Santos</sobrenome>
  </empregado>
  <empregado cod="E02" dept="D01">
     <nome>Ana</nome>
     <sobrenome>Ferraz</sobrenome>
  </empregado>
</empregados>
```

Resultado

```
<departamentos>
  <departamento>
    <codigo>D01</codigo>
    <empregados>
      <empregado>
        <nome>João</nome>
        <sobrenome>Santos</sobrenome>
      </empregado>
      <empregado>
        <nome>Ana</nome>
        <sobrenome>Ferraz</sobrenome>
      </empregado>
    </empregados>
  </departamento>
</departamentos>
```

Vamos consultar XML?

- Os arquivos necessários para os exercícios estão no site da disciplina
  - São 3 arquivos: bids.xml, users.xml, itens.xml
  - Os arquivos estão descritos aqui: http://www.w3.org/TR/xquery-use-cases/#rdb
- Usem o XML Exchanger Lite para executar as consultas

Exercício 1

- Selecionar itemno e description dos itens que foram oferecidos pelo usuário U01 (offered_by = “U01”).
- Salve a consulta num arquivo chamado exercicio1.xq

Exercício 2

- Monte uma consulta que traga todos os usuários, com name e rating, onde o rating=“B". As tags devem aparecer no resultado em português. Salve num arquivo chamado exercicio2.xq

```
<result>
  <usuario>
    <nome>Tom Jones</nome><nota>B</nota>
  </usuario>
  <usuario>
    <nome>Jack Sprat</nome><nota>B</nota>
  </usuario>
  <usuario>
    <nome>Rip Van Winkle</nome><nota>B</nota>
  </usuario>
</result>
```

Exercício 3

Selecionar todos os bids cujo itemno="1001". Retorne o resultado no seguinte formato:

```
<result>
  <bid><user><userid>U02</userid></user><bid>35</bid></bid>
  <bid><user><userid>U04</userid></user><bid>40</bid></bid>
  <bid><user><userid>U02</userid></user><bid>45</bid></bid>
  <bid><user><userid>U04</userid></user><bid>50</bid></bid>
  <bid><user><userid>U02</userid></user><bid>55</bid></bid>
</result>
```

Exercício 4

- Monte uma consulta que retorna todos os items (com sua estrutura completa) que tem reserve_price=25 e foram oferecidos por U02.

Produto cartesiano

- Operadores for na mesma cláusula FLWOR funcionam como um produto cartesiano
- Exemplos usando dois documentos: emps.xml e dept.xml

emps.xml

```
<? xml version="1.0" ?>
<empregados>
  <empregado cod="E01" dept="D01">
     <nome>João</nome>
     <inicial-meio>S.</inicial-meio>
     <sobrenome>Santos</sobrenome>
  </empregado>
  <empregado cod="E02" dept="D01">
     <nome>Ana</nome>
     <sobrenome>Ferraz</sobrenome>
  </empregado>
</empregados>
```

dept.xml

```
<? xml version="1.0" ?>
<departamentos>
  <departamento cod="D01">
     <nome>Vendas</nome>
     <local>3º. andar</local>
  </departamento>
  <departamento cod="D02">
     <nome>Financeiro</nome>
     <local>4º. andar</local>
  </departamento>
</departamentos>
```

Resultado

```
<resultado>
  {for $d in doc('dept.xml')//departamento),
       $e in doc('emps.xml')//empregado
   return
   <dep-emp>
     <departamento>{$d/nome/text()}</departamento>
     <empregado>{$e/nome/text()}</empregado>
   </dep-emp>
   }
</resultado>
```

```
<resultado>
 <dep-emp>
  <departamento>Vendas</departamento>
  <empregado>João</empregado>
 </dep-emp>
 <dep-emp>
  <departamento>Vendas</departamento>
  <empregado>Ana</empregado>
 </dep-emp>
 <dep-emp>
  <departamento>Financeiro</departamento>
  <empregado>João</empregado>
 </dep-emp>
 <dep-emp>
  <departamento>Financeiro</departamento>
  <empregado>Ana</empregado>
 </dep-emp>
</resultado>
```

emps.xml

```
<empregados>
  <empregado cod="E01" dept="D01">
     <nome>João</nome>
     <inicial-meio>S.</inicial-meio>
     <sobrenome>Santos</sobrenome>
  </empregado>
  <empregado cod="E02" dept="D01">
     <nome>Ana</nome>
     <sobrenome>Ferraz</sobrenome>
  </empregado>
</empregados>
```

dept.xml

```
<departamentos>
  <departamento cod="D01">
     <nome>Vendas</nome>
     <local>3º. andar</local>
  </departamento>
  <departamento cod="D02">
     <nome>Financeiro</nome>
     <local>4º. andar</local>
  </departamento>
</departamentos>
```

Junção

- Um produto cartesiano se transforma em junção se adicionarmos uma cláusula WHERE à consulta

Exemplo JUNÇÃO

```
<resultado>
   {for $d in doc('dept.xml')//departamento,
         $e in doc('emps.xml')//empregado
    where $d/@cod=$e/@dept
    return
    <dep-emp>
      <departamento>{$d/nome/text()}</departamento>
      <empregado>{$e/nome/text()}</empregado>
    </dep-emp>
    }
</resultado>
```

Resultado

```
<resultado>
  {for $d in doc('dept.xml')//departamento,
       $e in doc('emps.xml')//empregado
   where $d/@cod=$e/@dept
   return
   <dep-emp>
     <departamento>{$d/nome/text()}</departamento>
     <empregado>{$e/nome/text()}</empregado>
   </dep-emp>
   }
</resultado>
```

```
<resultado>
 <dep-emp>
  <departamento>Vendas</departamento>
  <empregado>João</empregado>
 </dep-emp>
 <dep-emp>
  <departamento>Vendas</departamento>
  <empregado>Ana</empregado>
 </dep-emp>
</resultado>
```

emps.xml

```
<empregados>
  <empregado cod="E01" dept="D01">
     <nome>João</nome>
     <inicial-meio>S.</inicial-meio>
     <sobrenome>Santos</sobrenome>
  </empregado>
  <empregado cod="E02" dept="D01">
     <nome>Ana</nome>
     <sobrenome>Ferraz</sobrenome>
  </empregado>
</empregados>
```

dept.xml

```
<departamentos>
  <departamento cod="D01">
     <nome>Vendas</nome>
     <local>3º. andar</local>
  </departamento>
  <departamento cod="D02">
     <nome>Financeiro</nome>
     <local>4º. andar</local>
  </departamento>
</departamentos>
```

XQuery

- O resultado de uma consulta XQuery pode não ser um documento XML!
- Exemplo: um resultado que não tem uma raiz

Exemplo fragmento (1)

```
for $e in doc("emps.xml")//empregado/nome
return $e
```

```
<? xml version="1.0" ?>
<empregados>
  <empregado cod="E01" dept="D01">
     <nome>João</nome>
     <inicial-meio>S.</inicial-meio>
     <sobrenome>Santos</sobrenome>
  </empregado>
 <empregado cod="E02" dept="D01">
     <nome>Ana</nome>
     <sobrenome>Ferraz</sobrenome>
  </empregado>
</empregados>
```

```
<nome>João</nome> ,
<nome>Ana</nome>
```

Exemplo fragmento (2)

```
for $e in doc("emps.xml")//empregado
return $e
```

```
<? xml version="1.0" ?>
<empregados>
  <empregado cod="E01" dept="D01">
     <nome>João</nome>
     <inicial-meio>S.</inicial-meio>
     <sobrenome>Santos</sobrenome>
  </empregado>
  <empregado cod="E02" dept="D01">
     <nome>Ana</nome>
     <sobrenome>Ferraz</sobrenome>
  </empregado>
</empregados>
```

Fragmento 1

```
<empregado cod="E01" dept="D01">
     <nome>João</nome>
     <inicial-meio>S.</inicial-meio>
     <sobrenome>Santos</sobrenome>
</empregado>,
```

Fragmento 2

```
<empregado cod="E02" dept="D01">
     <nome>Ana</nome>
     <sobrenome>Ferraz</sobrenome>
</empregado>
```

Exemplo fragmento (3)

- Para retornar mais de um elemento, sem raiz envolvendo cada retorno, usar parênteses

```
for $e in doc(“emps.xml")//empregado
return  ($e/nome, $e/sobrenome)
```

- O parênteses é necessário para que $e/sobrenome esteja dentro do escopo do **for**, caso contrário, o processador dirá que $e não foi declarada

Exercício 5

- Faça a junção de items e bids por itemno
- Retorne no resultado:
  - userid
  - description
  - itemno

```
<bids>
 <bid>
  <userid>U02</userid>
  <description>Red Bicycle</description>
  <itemno>1001</itemno>
 </bid>
 <bid>
  <userid>U04</userid>
  <description>Red Bicycle</description>
  <itemno>1001</itemno>
 </bid>
 <bid>
  <userid>U02</userid>
  <description>Red Bicycle</description>
  <itemno>1001</itemno>
 </bid>
...
</bids>
```

Exemplo LET

```
let $e := doc("emps.xml")//empregado
return $e
```

```
<? xml version="1.0" ?>
<empregados>
  <empregado cod="E01" dept="D01">
     <nome>João</nome>
     <inicial-meio>S.</inicial-meio>
     <sobrenome>Santos</sobrenome>
  </empregado>
  <empregado cod="E02" dept="D01">
     <nome>Ana</nome>
     <sobrenome>Ferraz</sobrenome>
  </empregado>
</empregados>
```

```
<empregado cod="E01" dept="D01">
     <nome>João</nome>
     <inicial-meio>S.</inicial-meio>
     <sobrenome>Santos</sobrenome>
 </empregado>,
<empregado cod="E02" dept="D01">
     <nome>Ana</nome>
     <sobrenome>Ferraz</sobrenome>
 </empregado>
```

Notem que o retorno também é em forma de fragmentos!!

Exemplo LET x FOR

```
let $e := doc("emps.xml")//empregado/nome
return
<emps>
{$e}
</emps>
```

```
<emps>
  <nome>João</nome>
  <nome>Ana</nome>
</emps>
```

```
for $e in doc("pedido.xml")//empregado/nome
return
<emps>
 {$e}
</emps>
```

```
<emps>
   <nome>João</nome>
</emps>,
<emps>
   <nome>Ana</nome>
</emps>
```

Operações de Agregação

- Existem 5 funções de agregação em Xquery
  - SUM
  - COUNT
  - AVG
  - MAX
  - MIN
- Devem ser usadas com o operador **LET**

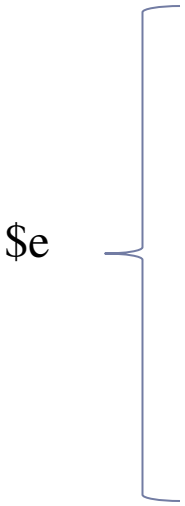
Exemplo

```
<num-emp>
   {let $e := doc(’emps.xml’)//empregado
    return
    count($e)
    }
</num-emp>
```

Ao contrário do **for**, o **let** se liga a todos os elementos de uma só vez

Exemplo

```
<num-emp>
   {let $e := doc('emps.xml')//empregado
    return
    count($e)
    }
</num-emp>
```




```
<? xml version="1.0" ?>
<empregados>
  <empregado cod="E01" dept="D01">
     <nome>João</nome>
     <inicial-meio>S.</inicial-meio>
     <sobrenome>Santos</sobrenome>
  </empregado>
  <empregado cod="E02" dept="D01">
     <nome>Ana</nome>
     <sobrenome>Ferraz</sobrenome>
  </empregado>
</empregados>
```

Exemplo

<num-emp>2</num-emp>

```
<num-emp>
   {let $e := doc('emps.xml')//empregado
    return
    count($e)
    }
</num-emp>
```

$e

```
<? xml version="1.0" ?>
<empregados>
  <empregado cod="E01" dept="D01">
     <nome>João</nome>
     <inicial-meio>S.</inicial-meio>
     <sobrenome>Santos</sobrenome>
  </empregado>
  <empregado cod="E02" dept="D01">
     <nome>Ana</nome>
     <sobrenome>Ferraz</sobrenome>
  </empregado>
</empregados>
```

Expressões de Quantificação

- Expressão **some**
  - Se algum elemento de uma coleção satisfaz determinada condição
  - Quantificador existencial
- Expressão **every**
  - Se todos os elementos da coleção satisfazem esta condição
  - Quantificador universal

DOC XML

```
<bib>
  <book>
    <author>
      <first>John</first>
      <last>Stevens</last>
    </author>
    <author>
      <first>Mary</first>
      <last>Jane</last>
    </author>
    <title>ABC</title>
  </book>
 <book>
    <author>
      <first>John</first>
      <last>Stevens</last>
    </author>
    <author>
      <first>Charles</first>
      <last>Stevens</last>
    </author>
    <title>XXX</title>
  </book>
</bib>
```

Expressões de Quantificação

- Listar os títulos dos livros que possuem **ALGUM** autor com sobrenome *Stevens*

```
for $b in doc("bib.xml")/bib/book
where some $a in $b/author/last
satisfies $a = "Stevens"
return $b/title
```

```
<bib>
  <book>
    <author>
      <first>John</first>
      <last>Stevens</last>
    </author>
    <author>
      <first>Mary</first>
      <last>Jane</last>
    </author>
    <title>ABC</title>
  </book>
 <book>
    <author>
      <first>John</first>
      <last>Stevens</last>
    </author>
    <author>
      <first>Charles</first>
      <last>Stevens</last>
    </author>
    <title>XXX</title>
  </book>
</bib>
```

```
for $b in doc("bib.xml")/bib/book
where some $a in $b/author/last
satisfies $a = "Stevens"
return $b/title
```

Expressões de Quantificação

- Listar os títulos dos livros que possuem **TODOS** os autores com sobrenome *Stevens*

```
for $b in doc("bib.xml")/bib/book
where every $a in $b/author/last
satisfies $a = "Stevens"
return $b/title
```

```
<bib>
  <book>
    <author>
      <first>John</first>
      <last>Stevens</last>
    </author>
    <author>
      <first>Mary</first>
      <last>Jane</last>
    </author>
    <title>ABC</title>
  </book>
 <book>
    <author>
      <first>John</first>
      <last>Stevens</last>
    </author>
    <author>
      <first>Charles</first>
      <last>Stevens</last>
    </author>
    <title>XXX</title>
  </book>
</bib>
```

```
for $b in doc("bib.xml")/bib/book
where every $a in $b/author/last
satisfies $a = "Stevens"
return $b/title
```

Exercício 6

(a) Faça uma consulta que retorne a média de bids de todos os usuários (usar o documento bids)

(b) Faça uma consulta que retorne a média de bids de cada usuário (dica: usar um for para pegar cada usuário e depois um let)

Para pegar só os usuários diferentes:

```
for $u in distinct-
   values(doc("bids.xml")/bids/bid_tuple/userid)
```

Exercício 6 (cont.)

(c) Faça uma consulta que retorna a descrição do item com reserve_price mais caro

(d) Faça uma consulta que retorna a descrição dos itens que possuem um bid maior que seu reserve_price

(e) Faça uma consulta que retorna o número de bids de cada item

doc e collection

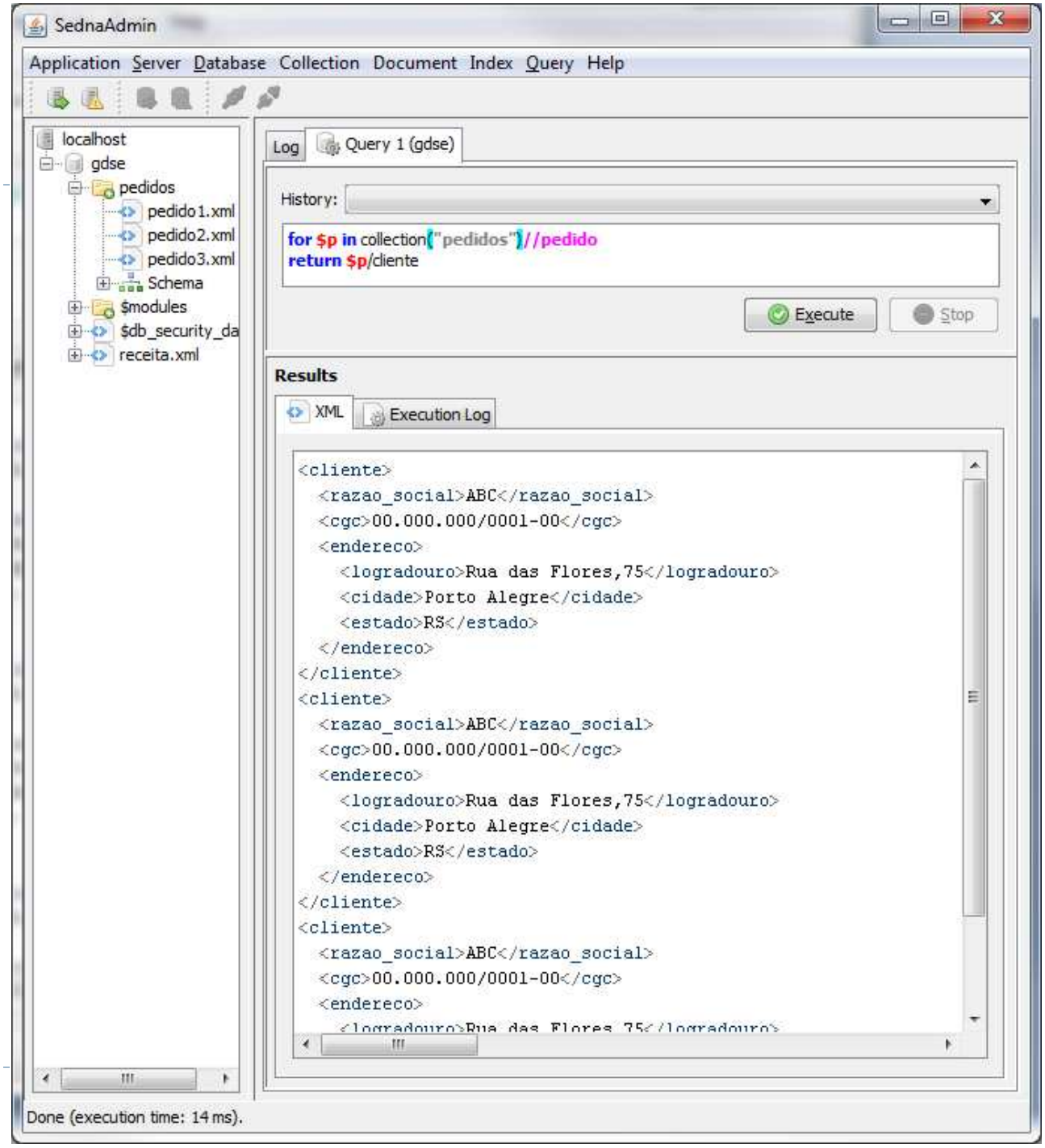
- Função **doc** é usada para referenciar o documento a ser consultado
- Função **collection** é usada para consultar vários documentos de uma só vez
  - Coleções (collections) são usadas em bancos de dados XML Nativos
  - Funcionam como um diretório onde se colocam vários documentos XML

Exemplo

- Assumindo que existe uma coleção chamada **empregados**

```
<emp-dept>
   {for $e in collection('empregados')//empregado
    return
    $e/nome
   }
</emp-dept>
```

Sedna


SednaAdmin
Application Server Database Collection Document Index Query Help
localhost
gdse
pedidos
pedido1.xml
pedido2.xml
pedido3.xml
Schema
$modules
$db_security_da
receita.xml
Log
Query 1 (gdse)
History:
for $p in collection("pedidos")//pedido
return $p/cliente
Execute
Stop
Results
XML
Execution Log
<cliente>
<razao_social>ABC</razao_social>
<cgc>00.000.000/0001-00</cgc>
<endereco>
<logradouro>Rua das Flores,75</logradouro>
<cidade>Porto Alegre</cidade>
<estado>RS</estado>
</endereco>
</cliente>
<cliente>
<razao_social>ABC</razao_social>
<cgc>00.000.000/0001-00</cgc>
<endereco>
<logradouro>Rua das Flores,75</logradouro>
<cidade>Porto Alegre</cidade>
<estado>RS</estado>
</endereco>
</cliente>
<cliente>
<razao_social>ABC</razao_social>
<cgc>00.000.000/0001-00</cgc>
<endereco>
Done (execution time: 14 ms).

Coleções no XML Spy

- Existem duas formas de usar coleções no XML Spy
  1. Usando um catálogo
  2. Usando caracteres coringa no caminho da coleção

Usando um arquivo de catálogo

- Referenciar um arquivo XML que contém um catálogo onde os arquivos da coleção são referenciados
- O formato do catálogo deve ser este:

```
<collection>
        <doc href="pedido1.xml" />
        <doc href="pedido2.xml" />
        <doc href="pedido3.xml" />
</collection>
```

(salvar este arquivo como catalogo.xml, e colocá-lo no mesmo diretório dos arquivos de pedido)

Usando um arquivo de catálogo

- Exemplo de consulta

```
<result>
   {for $i in collection("pedidos\catalogo.xml")//item
    return
    $i/produto
   }
</result>
```

(neste exemplo, o arquivo catalogo.xml foi colocado dentro do sub-diretório pedidos)

Usando caracteres coringa

```
<result>
   {for $i in collection("pedidos\*.xml")//item
    return
    $i/produto
   }
</result>
```

Alguns Aplicativos

- XML Spy (interface gráfica)
- Galax (Open Source)
- FatDog XQEngine
- Microsoft XQuery Demo
- Qexo - GNU Kawa
- QuiP – Software AG
  - developer.softwareag.com/tamino/quip

Outras Linguagens de consulta

- Lorel (Lightweight Object REpository Language)
- YATL
- Xtract
- XMLQuery
- XML Query Engine
- XML-GL
- Quilt
- XML-QL

XQuery Update Facility

Atualização de Documentos XML

- Primeira proposta que vendia a idéia de atualizações:
  - Publicada em 2001, num dos congressos mais importantes da área de Banco de Dados (SIGMOD): I. Tatarinov, Z. Ives, A. Halevy, and D. Weld. **Updating XML**. Maio de 2001

Linguagem para atualização

- XQuery passou a permitir atualizações recentemente
- http://www.w3.org/TR/xquery-update-10/ (W3C Recommendation de 17 de Março de 2011)

Funcionalidades:

- Excluir nodos
- Inserir nodos em uma determinada posição
- Substituir um nodo
- Modificar o valor de um nodo
- Substituir um nodo
- Modificar propriedades do nodo (valor, tipo, conteúdo, etc.)
- Mover nodos
- Permitir atualizações condicionais
- Permitir iterar sobre nodos para atualizá-los
- Permitir validação (esquema)
- Permitir que operações sejam compostas (o resultado de uma atualização é entrada de outra)
- Permitir atualizações parametrizadas

Sintaxe…

- Os exemplos a seguir foram extraídos do site do W3C

Inserção

- Inserir um elemento year depois do publisher do primeiro book

```
insert node <year>2005</year> after
  fn:doc("bib.xml")/books/book[1]/publisher
```

Exclusão

- Excluir o último autor do primeiro livro

delete node
fn:doc("bib.xml")/books/book[1]/author[last()]

Substituição

- Substituir a editora do primeiro livro pela editora do segundo livro

replace node
fn:doc("bib.xml")/books/book[1]/publisher **with**
fn:doc("bib.xml")/books/book[2]/publisher

Modificação

- Aumentar o preço do primeiro livro em 10%

replace value of node
fn:doc("bib.xml")/books/book[1]/price **with**
fn:doc("bib.xml")/books/book[1]/price * 1.1

Renomeação

- Renomear o primeiro elemento author do primeiro book para principal-author

```
rename node
  fn:doc("bib.xml")/books/book[1]/author[1] as
  "principal-author"
```

Renomeação

- Renomear o primeiro elemento author do primeiro book para o Qname que está na variável $newname

```
rename node
  fn:doc("bib.xml")/books/book[1]/author[1] as
  $newname
```

FLWOR

- Expressões de atualização podem ser adicionadas na cláusula **return** da expressão **FLWOR**

Transformação

- Retornar uma seqüência contendo todos os elementos employee que possuem Java como skill, excluindo o sub-elemento salary

```
for $e in //employee[skill = "Java"]
return
    copy $je := $e
    modify delete node $je/salary
    return $je
```

Transformação

- Copiar um nodo, modificar a cópia e retornar o nodo original e o modificado

```
let $oldx := /a/b/x
return
   copy $newx := $oldx
   modify
      (rename node $newx as "newx",
       replace value of node $newx with $newx * 2)
   return ($oldx, $newx)
```

eXist e Sedna

- Tanto o eXist quanto o Sedna suportam atualizações usando uma sintaxe diferente:
  - http://xmldb-org.sourceforge.net/xupdate/