XML Schema (Parte 2)

Vanessa Braganholo

Curso baseado em mini-cursos apresentados no SBBD. Autores: Carlos Heuser, Carina Dorneles e Vanessa Braganholo

Conteúdo Misto

```
<xs:complexType name="tEndereco“ mixed=“true”>
  <xs:sequence>
    <xs:element name="rua" type="xs:string"
                minOccurs="0" maxOccurs="1"/>
    <xs:element name="numero" type="xs:integer"
                minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
</xs:complexType>
<xs:element name=“endereco” type=“tEndereco”/>
```

Na instância XML...

```
<endereco>Meu endereço é <rua>Rua das Flores</rua> número
   <numero>34</numero></endereco>
```

Diferença em relação à DTD

- Os elementos de conteúdo misto respeitam as cardinalidades definidas para os sub-elementos.
- No exemplo anterior, endereço pode ter no máximo 1 rua e 1 número

- Na DTD, éramos obrigados a colocar um choice com repetição

Cardinalidade também pode ser colocada no sequence, choice e all...

```
<xs:complexType name="tEndereco“>
  <xs:sequence maxOccurs=“unbounded”>
    <xs:element name="rua" type="xs:string"
                minOccurs="0" maxOccurs="1"/>
    <xs:element name="numero" type="xs:integer"
                minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
</xs:complexType>
<xs:element name=“endereco” type=“tEndereco”/>
```

Este exemplo é equivalente ao anterior?

```
<xs:complexType name="tEndereco“>
  <xs:sequence>
    <xs:element name="rua" type="xs:string"
                minOccurs="0" maxOccurs=“unbounded"/>
    <xs:element name="numero" type="xs:integer"
                minOccurs="0" maxOccurs=“unbounded"/>
  </xs:sequence>
</xs:complexType>
<xs:element name=“endereco” type=“tEndereco”/>
```

Derivação de Tipos Simples

- Tipos simples podem ser derivados de tipos simples através de uma técnica chamada restrição
- Um tipo simples é usado com base, e sobre ele são aplicadas *facetas* ou expressões regulares

Facetas – MinInclusive e MaxInclusive

- Estabelecem valores mínimos e máximos

```
<xs:simpleType name="tNumero">
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="0"/>
    <xs:maxInclusive value="99999"/>
  </xs:restriction>
</xs:simpleType>

<xs:element name="numero" type="tNumero"/>
```

Facetas - Enumeration

- Limita um tipo simples a um conjunto de valores distintos

```
<xs:simpleType name=“tFigura">
  <xs:restriction base="xs:string">
    <xs:enumeration value = "jpeg"/>
    <xs:enumeration value = "gif"/>
    <xs:enumeration value = "bmp"/>
    <xs:enumeration value = "tiff"/>
    <xs:enumeration value = "wmf"/>
  </xs:restriction>
</xs:simpleType>

<xs:attribute name="tipo" type=“tFigura“/>
```

Expressões Regulares

```
<xs:simpleType name="tCep">
  <xs:restriction base="xs:string">
    <xs:pattern value="\d{5}-\d{3}"/>
  </xs:restriction>
</xs:simpleType>

<xs:element name="cep" type="tCep"/>
```

- Na instância XML:

```
<cep>24220-290</cep>
```

Expressões Regulares

- Uma lista completa do tipo de expressões regulares que podem ser aplicadas está disponível em http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/datatypes.html#dt-regex

- Veja que este endereço apresenta a **definição formal** de expressões regulares em XML Schema

Exercício 1

- Crie um documento XML que contenha um elemento do tipo tFigura, conforme definido no slide 7

Exercícios 2 e 3

2. Crie um tipo tCnpj para aceitar somente CNPJs com o seguinte formato:

<cgc>00.000.000/0001-00</cgc>

3. Crie um tipo tCpf para aceitar somente CPFs com o seguinte formato:

<cpf>000.000.000-00</cpf>

Derivação de Tipos Complexos

- Tipos complexos podem ser derivados por restrição ou por extensão
- Restrição
  - semelhante a restrição de tipos simples, mas ao invés de restringir valores, ela restringe elementos (por exemplo, cardinalidade)
- Extensão
  - utilizada para "aumentar" um tipo
  - o novo tipo derivado possuirá tudo que o tipo base possuía, mais outros elementos e atributos definidos na extensão

Derivação por Extensão

```
<xs:complexType name="tEndereco">
  <xs:sequence>
    <xs:element name="rua" type="xs:string"
                minOccurs="0" maxOccurs="1"/>
    <xs:element name="numero" type="xs:integer"
                minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
</xs:complexType>
```

- Extensão ao tipo complexo apresentado acima:

```
<xs:complexType name="tEnderecoEstendido">
  <xs:complexContent>
    <xs:extension base="tEndereco">
      <xs:sequence>
        <xs:element name="bairro" type="xs:string"
                    minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Derivação de Tipos
Reflexo nas Instâncias

- É possível declarar um elemento no esquema como sendo do tipo mais genérico, e, na instância, usar um tipo mais específico...

Derivação de Tipos
Reflexo nas Instâncias

No esquema...

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:complexType name="tPessoa">
   <xs:sequence>
     <xs:element name="nome" type="xs:string"/>
     <xs:element name="endereco" type="tEndereco"/>
   </xs:sequence>
 </xs:complexType>
 <xs:element name="pessoa" type="tPessoa"/>
 ...
 <!-- Declarações dos tipos tEndereco e tEnderecoEstendido, como nas
   transparências anteriores -->
</xs:schema>
```

Derivação de Tipos
Reflexo nas Instâncias

Na instância...

```
<pessoa xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceschemaLocation="pessoa.xsd">
  <nome>Jose da Silva</nome>
  <endereco xsi:type="tEnderecoEstendido">
    <rua>Rua das Flores</rua>
    <numero>34</numero>
    <bairro>Inga</bairro>
  </endereco>
</pessoa>
```

Atributo em elemento simples
Também são declarados usando extensão...

- Mesmo sendo de tipo simples, é necessário declarar o elemento como um **complexType**

```
<xs:element name=“preco">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:decimal">
        <xs:attribute name=“moeda" type="xs:string"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

Atributo em elemento simples
Exemplo

- Na instância XML...

<preco moeda = “Real”>95.3</preco>

Derivação por Restrição

```
<xs:complexType name="tEndereco">
 <xs:sequence>
   <xs:element name="rua" type="xs:string" minOccurs="0" maxOccurs="1"/>
   <xs:element name="numero" type="xs:integer" minOccurs="0" maxOccurs="1"/>
 </sequence>
</complexType>
```

- Restrição ao tipo complexo apresentado acima:

```
<xs:complexType name="tEnderecoObrigatorio">
 <xs:complexContent>
  <xs:restriction base="tEndereco">
   <xs:sequence>
    <xs:element name="rua" type="xs:string" minOccurs="1" maxOccurs="1"/>
    <xs:element name="numero" type="xs:integer" minOccurs="1" maxOccurs="1"/>
   </xs:sequence>
  </xs:restriction>
 </xs:complexContent>
</xs:complexType>
```

Exercício 4

- Crie um esquema para representar Pessoa, sendo que Pessoa pode ser Física ou Jurídica. Use extensão de tipos complexos.
- Valide os dois documentos XML abaixo contra o esquema criado:

```
<pessoa>
  <nome>João</nome>
  <endereco>Rua das Flores,45</endereco>
  <cpf>999.999.999-01</cpf>
</pessoa>
```

```
<pessoa>
  <nome>ABC LTDA</nome>
  <endereco>Rua das Flores,45</endereco>
  <cnpj>99.999.999/0001-01</cnpj>
</pessoa>
```

Unicidade

- Permite especificar que o valor de um elemento ou atributo deve ser único em um determinado **escopo**

- Elemento **unique**, declarado dentro de um **element**
  - Este elemento será o escopo onde a unicidade será testada
  - **Subelementos:**
    - **Selector** – indica uma lista de elementos que serão testados um a um
    - **Field** – indica o elemento ou atributo que tem que ser único dentro do escopo

Unicidade

```
<xs:element name="items" type="tItems">
    <xs:unique name="codProd">
        <xs:selector xpath="item"/>
        <xs:field xpath="@cod"/>
    </xs:unique>
</xs:element>
```

Escopo

Um nome para esta restrição de unicidade

Lista de elementos a serem testados um a um

Atributo que tem que ser único dentro de **items** (escopo)

Na instância XML...

```
...
<items>
   <item cod="c1">
      ...
   </item>
   <item cod="c2">
      ...
   </item>
   <item cod="c3">
      ...
   </item>

</items>
...
```

key e keyref

- **key** também tem que ser única – a diferença é que o valor pode ser referenciado por keyref

- Declaração de **key** igual a declaração de **unique**, só que usando o elemento **key** ao invés de **unique**

```
<xs:element name=“produtos" type=“tProdutos">
    <xs:key name=“chaveCodProd">
      <xs:selector xpath=“produto"/>
      <xs:field xpath="codigo"/>
   </xs:key>
</xs:element>
```

keyref

```
<xs:element name="items" type="tItems">
    <xs:keyref name="chaveEstrCodProd"
                    refer="chaveCodProd">
      <xs:selector xpath="item"/>
      <xs:field xpath="@cod"/>
   </xs:keyref>
</xs:element>
```

Nome da restrição de chave declarada anteriormente

Na instância XML

```
...
<produtos>
   <produto><codigo>c1</codigo>...</produto>
   <produto><codigo>c2</codigo>...</produto>
   <produto><codigo>c3</codigo>...</produto>
   <produto><codigo>c4</codigo>...</produto>
   <produto><codigo>c5</codigo>...</produto>
</produtos>
<items>
   <item cod="c4">
      ...
   </item>
   <item cod="c2">
      ...
   </item>
   <item cod="c3">
      ...
   </item>

</items>
...
```

Exercício 5

- Faça um XML Schema que valide o documento XML abaixo. Ele contém uma lista de produtos. O XML Schema deve garantir que o código do produto seja único.

```
<produtos>
    <produto>
      <codigo>1</codigo>
      <nome>caneta</nome>
    </produto>
    <produto>
      <codigo>2</codigo>
      <nome>caderno</nome>
    </produto>
    <produto>
      <codigo>3</codigo>
      <nome>borracha</nome>
    </produto>
</produtos>
```

Para pesquisar...

- Verifique a especificação do W3C e descubra o que é e como usar os seguintes conceitos:
  - include
  - import