XML para Concursos Públicos

Aula 01 – Características Gerais

➢Do que trata este Módulo?

Provas de TI
Tecnologia da Informação para Concursos

Iremos estudar as características mais importantes da Linguagem XML, sempre focando as questões mais cobradas em Concursos Públicos.

Abordaremos: diferenças com outras linguagens de marcação (HTML); como montar e estruturar de um documento XML; como pode ser feita a validação de um documento XML.

Não é preciso pré-requisitos de outras Linguagens de Programação ou conhecimento de lógica etc, basta a VONTADE de aprender!

Mas é importante deixarmos claro que não iremos estudar Web Services ou outras tecnologias que usam o XML como base. Isto é assunto para outros módulos!

➢Bibliografia

- XML Bible – Autor: Elliotte Rusty Harold – 2ª Edição

- Aprendendo XML – Autor: Erik T. Ray – 1ª Edição

- Treinamento Avançado em XML – Autor: Rogério Amorim de Faria – 1ª Edição

- Site www.w3schools.com

➢Composição das aulas

- Aula 01: Características Gerais
- Aula 02: Elementos e Atributos
- Aula 03: Estrutura da XML
- Aula 04: DTD (*Document Type Definition*)
- Aula 05: Resolução de Exercícios de DTD
- Aula 06: *XML Schema* – Elementos Simples
- Aula 07: *XML Schema* – Elementos Compostos
- Aula 08: Resolução de Exercícios de *XML Schema*
- Aula 09: Namespaces

ENTÃO VAMOS COMEÇAR LOGO!

➢O que é XML?

É uma “sigla” para *e**X**tensible **M**arkup **L**anguage*, ou seja, é uma linguagem de marcação **extensível** usada para definição de conteúdo (informação).

É considerada EXTENSÍVEL, pois não existe um conjunto pré-definido de marcadores (como a linguagem HTML). Você pode criar seus próprios marcadores.

A ideia é estruturar um conteúdo qualquer, em partes que sejam o mais autoexplicativas possível (inteligível para humanos e máquinas). Por isso, diz-se que a XML é uma linguagem para trabalhar METADADOS (dados que explicam dados).

Você poderia estruturar informações em vários formatos inteligíveis. O mais comum para algumas pessoas é pensar em termos de TABELAS. Por exemplo, uma lista de produtos num supermercado com seus respectivos preços.

MAS em XML é DIFERENTE: usamos o conceito de HIERARQUIA para estruturarmos as informações que queremos passar!

➢As Origens do XML

É importante sabermos como se originou o XML para entender melhor suas características.

SGML (*Structured Generalized Markup Language*): é considerada a primeira linguagem de marcação. Foi lançada em 1986 para padronizar a estrutura de documentos científicos.

- Introduziu o conceito de marcação completa, ou seja, um marcador antes do conteúdo e outro marcador no fim deste conteúdo.
- Utiliza o conceito de "atributo" para auxiliar na definição de um conteúdo.
- O usuário é livre para criar os marcadores que necessitar. Não existe um conjunto pré-definido de marcadores.

HTML(*HyperText Markup Language*): foi criada pensando nos documentos na Internet, baseada na SGML, só que inovando em alguns aspectos.

- Simplificou e melhorou o suporte aos hyperlinks.
- Criou uma série de marcadores para definição de conteúdo para a Web.

➢Nascimento do XML

Apesar de inovadoras, as linguagens SGML e HTML possuíam alguns "problemas" a serem resolvidos.

A SGML era extremamente complicada de se implementar, e a HTML é muito complacente com os erros de linguagem.

Daí surgiu a necessidade de se criar uma linguagem de marcação que atendesse a SGML, só que de forma mais simplificada, e que ao mesmo tempo fosse mais rigorosa e "extensível" que a linguagem HTML.

Em 1998, a W3C (*World Wide Web Consortium*, a organização que padroniza a Web) recomendou o uso do XML como padrão para estruturação de conteúdos (informação).

➢Diferenças XML x HTML

Provas de TI
Tecnologia da Informação para Concursos

Vejamos algumas diferenças entre XML e HTML, que costumam ser cobradas pelas bancas de Concursos.

HTML	XML
Usada para exibição de conteúdo de páginas Web.	Usada para definição de conteúdos quaisquer.
Conjunto pré-definido de marcadores.	Não existe conjunto pré-definido de marcadores. O usuário pode criar seus próprios marcadores.
Os Marcadores não são *case sensitive*.	Os Marcadores SÃO *case sensitive*.
Pode apresentar erros de sintaxe, que mesmo assim seu conteúdo é exibido.	Não pode haver erros de sintaxe!

Tais diferenças se fundamentam no fato de que a HTML é voltada para a formatação de conteúdo para a Web, enquanto que a XML é mais voltada para a padronização e troca de informações quaisquer entre vários sistemas.

➢Detalhes do XML

Atualmente, a linguagem XML possui duas versões: a 1.0 e a 1.1.

A versão 1.0, originalmente lançada em 1998, está em sua quinta revisão, lançada em 2008. Uma revisão são apenas pequenas funcionalidades que não afetam as que já existiam.

A versão 1.1 já apresenta algumas funcionalidades mais avançadas. A própria W3C só recomenda o seu uso se você REALMENTE necessitar de tais funcionalidades.

O XML, assim como o HTML, pode usar o CSS (*Cascading Style Sheets* ou Folha de Estilos em Cascata) para definir o *layout* de apresentação dos dados. Nas próximas aulas veremos como configurar um documento XML para que o mesmo possa usar tais Folhas de Estilos.

➢Mais características do XML

A linguagem XML possui duas características marcantes: ser “Bem-formado” e “Passível de Validação”.

Bem-formado: um documento XML precisa ser “bem-formado”, ou seja, sem apresentar quaisquer erros de sintaxe. Veremos nas próximas aulas, como fazer para considerar um documento XML como “bem-formado”.

Passível de Validação: um documento XML pode ser validado, ou seja, podemos testar se o seu conteúdo é válido, usando duas ferramentas: **DTD** (*Document Type Definition*) ou ***XML Schema*** (Esquema XML)**.** Veremos também nas próximas aulas como utilizar tais ferramentas.

➢Resolução de Questões

Questão 1 (FCC - 2012 - TST - Técnico Judiciário – Programação)

A linguagem XML

a) é considerada uma linguagem de marcação que tem uma biblioteca de tags muito rica e finita, a ponto de atender a todos os segmentos de negócios ligados a indústria, comércio e serviços.
b) foi concebida para trabalhar com metadados, que descrevem os dados do documento XML.
c) permite realizar diretamente no código diferentes formatações para exibir os dados de forma personalizada aos usuários.
d) cria uma DTD - Dados para Transferência de Documentos - que define a estrutura do documento XML.
e) está na versão 5.0 já que a XML 4.0 estava obsoleta e, gradativamente, sendo substituída pela WML.

➢Resolução de Questões

Questão 2 (CESPE - 2010 - TRE-BA - Técnico Judiciário - Programação de Sistemas)

Em relação à linguagem XML, julgue o próximo item.

XML pode ser utilizado como linguagem padrão para a integração de fonte de dados de diferentes formatos.

( ) CERTO ( ) ERRADO

➢Resolução de Questões

Questão 3 (FCC - 2011 - TRT - 14ª Região (RO e AC) - Técnico Judiciário - Tecnologia da Informação)

Em relação aos conceitos de HTML, CSS e XML, é INCORRETO afirmar:

a) os elementos contidos nas tags XML são case sensitive.
b) em CSS, as declarações de propriedade terminam com ponto-e-vírgula; a propriedade e o valor são separados por dois pontos.
c) os elementos contidos nas tags HTML não são case sensitive.
d) XML permite elementos definidos pelo usuário, enquanto em HTML os elementos são pré-definidos.
e) HTML e XML constituem dois tipos linguagens de marcação de apresentações.

➢Resolução de Questões

Questão 4 (UFG - 2010 - UFG - Analista de TI - Desenvolvimento de Sistemas)

A linguagem de marcação extensível (ou XML) é recomendada pelo World Wide Web Consortium como padrão internacional para representação e intercâmbio de informação estruturada na Internet. Em comparação a outras linguagens de marcação existentes, como a HTML, uma vantagem da linguagem XML é:

a) possuir vocabulário de tags predefinido, portanto, fácil de usar.
b) permitir a representação de diversos tipos de estruturas de dados, como listas, registros e árvores.
c) ser pouco verbosa, portanto, ter pouco impacto sobre a velocidade de transmissão de informação.
d) ter como foco a formatação e exibição de dados.

➢Resolução de Questões

Questão 5 (FCC - 2011 - TRT - 14ª Região (RO e AC) - Analista Judiciário - Tecnologia da Informação)

Ambas identificam elementos em uma página e ambas utilizam sintaxes similares. A grande diferença entre elas é que uma descreve a aparência e as ações em uma página na rede enquanto a outra não descreve nem aparência e nem ações, mas sim o que cada trecho de dados é ou representa, ou seja, descreve o conteúdo do documento. Uma tag esquecida na escrita de uma delas ou um atributo sem aspas torna o documento inutilizável, enquanto que na outra isso é tolerado.
Pelas características comparadas, o texto acima refere-se a

a) HTML e XML.
b) UML e XML.
c) PHP e Java.
d) Oracle Forms e UML.
e) Java e CSS.

➢Resolução de Questões

Questão 6 (FCC - 2010 - TRT - 8ª Região - Analista Judiciário - Tecnologia da Informação)

Sobre as tags HTML e XML, é correto afirmar:

a) Tags HTML são case sensitive, isto é, fazem distinção entre letras maiúsculas e minúsculas.
b) Tags XML não são case sensitive, isto é, não fazem distinção entre letras maiúsculas e minúsculas.
c) As tags XML são pré-definidas pelo W3C, devendo o autor utilizá-las quando da elaboração do documento.
d) As tags HTML não são pré-definidas, podendo o autor do documento criá-las livremente no momento da elaboração de seu documento.
e) A forma de fazer comentários em um documento HTML e em um documento XML são idênticas.

➢Final da Aula 1

GABARITO:

1 - B

2 - CERTO

3 - E

4 - B

5 - A

6 - E

Até a próxima aula!

XML para Concursos Públicos

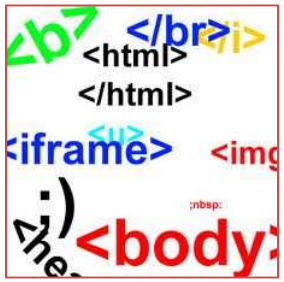
Aula 02 – Elementos e Atributos

➢Os Marcadores (*Tags*)

Como XML é uma linguagem de marcação, ela é composta de “Marcadores”, também chamados “*tags*”.

Cada *tag* é delimitado pelos sinais de “menor que” (<) e “maior que” (>). Por exemplo, `<empresa>` é uma *tag* XML.

GERALMENTE as *tags* vêm em duplas, sendo que a *tag* que inicia possui um determinado nome, e sua par, que finaliza, possui uma barra normal antes do seu nome. Por exemplo: `<html>` é uma *tag* inicial; e `</html>` é uma *tag* final.



MAS LEMBRE-SE: As *tags* não precisam vir sempre em duplas, elas podem vir sozinhas também, conforme veremos mais para frente!

➢Elementos XML

Um Documento XML é composto de um ou mais Elementos XML.
Existem dois tipos de Elementos: Não Vazio e Vazio.

```
<?xml?>
```

Um Elemento XML Não Vazio é composto de: uma *tag* de abertura, um conteúdo e sua respectiva *tag* de fechamento. Por exemplo: `<nome>Márcio</nome>`

Mas também existem os Elementos Vazios, que possuem apenas a *tag* de abertura, sem nenhum conteúdo. Para indicar que se trata de um elemento vazio, devemos colocar uma barra no final da *tag*. Por exemplo:

```
<tag_vazia />
```

➢Nomenclatura de Marcadores

Provas de TI
Tecnologia da Informação para Concursos

Existem regras para a nomenclatura dos Marcadores (*tags*):

Podemos usar Letras, Números e outros caracteres, tais como: o underline, o traço, e o ponto. O símbolo de dois pontos "**:**" só deve ser usado em *namespaces*.

NÃO podemos começar com um número ou um caracter de pontuação como o ponto.

Não podemos usar espaço!

Não podemos usar a palavra reservada "XML" e suas variantes em maiúsculas e minúsculas, por exemplo: xml, XML, Xml etc.

Outro detalhe importante é que não há um limite de caracteres para nomenclatura dos elementos. Mas não é recomendado "escrever um livro" ao se dar nomes a *tags*.

Enfim, o mais importante é que você dê um nome que faça muito sentido com o conteúdo que você quer definir.

➢Aninhamento de Elementos

Um Elemento XML pode conter outros Elementos, a isto damos o nome de “Aninhamento”. Daí dizermos que a linguagem XML organiza seus conteúdos de modo “hierárquico”.

Exemplo:

```
<pessoa>
  <nome>Márcio</nome>
  <sobrenome>Vilanova</sobrenome>
</pessoa>
```

No exemplo, temos um elemento <pessoa> que contém dois elementos: <nome> e <sobrenome>.

Não existem limites para níveis de aninhamentos de Elementos, você pode definir quantos achar necessários. Por exemplo:

```
<agenda>
  <contatos>
    <contato>
      <nome>Maria</nome>
      <telefone>12345678</telefone>
    </contato>
  </contatos>
</agenda>
```

No exemplo, temos quatro níveis de aninhamento: <agenda>, <contatos>, <contato> e <nome>.

O elemento <telefone> está no mesmo nível de aninhamento que o <nome>.

➢Mais sobre Aninhamento

Em XML, não podemos QUEBRAR a ordem dos Aninhamentos!

No exemplo anterior, temos:

```
<agenda>
  <contatos>
    <contato>
      <nome>Maria</nome>
      <telefone>12345678</telefone>
    </contato>
  </contatos>
</agenda>
```

Repare que sempre fechamos um Elemento, quando queremos criar outro Elemento no mesmo nível de aninhamento: *tags* <nome> e <telefone>.

Mas se alterarmos o XML para:

```
<agenda>
  <contatos>
    <contato>
      <nome>Maria<telefone>
      </nome>12345678</telefone>
    </contato>
  </contatos>
</agenda>
```

➢Entidades

Em XML, Entidade é uma unidade de armazenamento virtual de informação, que pode ser usado em vários Documentos XML.

```
<?xml?>
```

É usado como se fosse um "atalho" para determinados conteúdos. Por exemplo, se necessitar que um conteúdo seja inserido repetidamente em vários Documentos XML, bastaria incluir uma referência a Entidade (que armazena este conteúdo).

Nas próximas aulas, iremos aprender como CRIAR estas Entidades (usando DTD). Neste aula, iremos apenas aprender como USAR estas Entidades em nossos Documentos XML.

Aqui temos um exemplo do uso de Entidade:

```
<!ENTITY MeuEmail "marcio@passei.com.br">
```

Criamos uma Entidade chamada "`MeuEmail`", cujo conteúdo é "`marcio@passei.com.br`".

```
<contato>Marcio &MeuEmail;</contato>
```

Para usar a Entidade, usamos o símbolo **&** no início, e o **;** (ponto-e-vírgula) no final para referenciá-la.

```
<contato>Marcio marcio@passei.com.br</contato>
```

Ao ser processado, o Documento XML ficará assim.

➢Aplicações das Entidades

As Entidades podem servir para várias propósitos. Mas existem dois bastante comuns:

O primeiro e mais óbvio é servir de abreviação para trechos longos e repetitivos, conforme já vimos no exemplo anterior.

O segundo propósito mais comum é a representação de caracteres especiais, que podem ser: OU caracteres específicos de certos idiomas (ç, ã, é, etc); OU caracteres que não podem ser incluídos diretamente no Documento XML, pois já fazem parte da especificação da Linguagem XML.

Os caracteres que não podem ser incluídos diretamente no Documento XML são chamados de “Entidades Pré-definidas”, que são as cinco Entidades citadas a seguir:

&	o caracter & (o “e” comercial).
<	o caracter de “menor que”.
>	o caracter de “maior que”.
'	caracter de apóstrofo ou aspas simples.
"	caracter de aspas duplas.

Exemplos:

`Tom & Jerry`	→	`Tom & Jerry`
`If x < 2 then`	→	`If x < 2 then`

➢Atributos

Vimos que um Elemento XML é composto de um conteúdo delimitado por *tags*.
Mas podemos incluir informações adicionais a este conteúdo através dos Atributos.

Os **Atributos** são definidos dentro da *tag* de abertura e servem como uma informação acerca do conteúdo, ou seja, é como se fosse um METADADO.

Exemplos:

```
<pessoa rg="123456-8">José das Quantas</pessoa>
<carro placa="ABC-1234">Maverick V8</carro>
```

Os **Atributos** sempre são definidos em pares: **um nome** associado a **um valor**.

As regras de nomenclatura para os atributos é a mesma para os nomes das tags.

Os VALORES dos atributos SEMPRE devem estar delimitados OU por aspas simples OU por aspas duplas. Não misturar um tipo de aspa com outro!

➢Cuidados com Atributos

Podemos definir zero ou um ou vários atributos para um Elemento XML. Não há limites. Mas a boa prática é: não abuse!

Mas aí resta uma dúvida: quando devemos usar atributos?
Resposta: use os atributos SOMENTE quando desejar incluir um metadado (uma informação sobre o conteúdo).

Exemplos de Elementos XML com a mesma informação, mas estruturados de forma distinta:

```
<curso id="XML01">XML para Concursos</curso>
```

```
<curso>
  <id>XML01</id>
  <nome>XML para Concursos</nome>
</curso>
```

No primeiro exemplo, temos um atributo chamado **id**. E no segundo exemplo, este atributo virou um elemento chamado **id**.

➢Resolução de Questões

Questão 1 (CESPE - 2012 - ANAC - Analista Administrativo - Área 4)

Um arquivo XML possui atributos e elementos. No exemplo 1, que se segue, sexo é atributo e, no exemplo 2, sexo é elemento, provendo, em ambos os exemplos, a mesma informação.

Exemplo 1:

```
<pessoa sexo="F">
  <nome>Dhara</nome>
  <sobrenome>Silva</sobrenome>
</pessoa>
```

Exemplo 2:

```
<pessoa>
  <sexo>F</sexo>
  <nome>Dhara</nome>
  <sobrenome>Silva</sobrenome>
</pessoa>
```

( ) CERTO ( ) ERRADO

➢Resolução de Questões

Questão 2 (FCC - 2010 - TCE-SP - Agente da Fiscalização Financeira - Informática - Suporte de Web)

Existem caracteres que, por serem reservados, não podem ser utilizados em documentos XML. A linguagem XML oferece substitutos para estes casos. Tais substitutos começam por “&” e terminam por “;” (referência à entidade). Os caracteres reservados para a linguagem XML e suas referências são:

a) **>** < **<** > **&** & **'** ' **"** "
b) **<** < **>** > **&** & **'** ' **"** "
c) **<** < **>** > **"** & **'** ' **&** "
d) **<** < **>** > **&** & **"** ' **'** "
e) **<** < **&** > **>** & **'** ' **"** "

➢Resolução de Questões

Questão 3 (CESPE - 2010 - TRE-BA - Analista Judiciário - Análise de Sistemas)

Acerca do XML, julgue o item a seguir.
A sintaxe básica para um elemento XML pode ser corretamente representada pela instrução a seguir.

```
<nome_do_elemento>Texto</nome_do_elemento>
```

( ) CERTO ( ) ERRADO

➢Resolução de Questões

Questão 4 (CESPE - 2010 - Banco da Amazônia - Análise de Sistemas)
Com base na estrutura do documento XML apresentado abaixo, julgue o item.
As tags <autor>, <titulo>, <ano>, <preco>, <lancamento> e <oferta> são atributos da entidade <livro>.

( ) CERTO ( ) ERRADO

```
<?xml version="1.0"?>
<livraria>
   <livro isbn="9788598078359">
      <autor id="064">Stephenie Meyer</autor>
      <titulo>Lua Nova</titulo>
      <ano>2008</ano>
      <preco>35.00</preco>
      <lancamento/>
   </livro>
   <livro isbn="9788599296554">
      <autor id="095">Dan Brown</autor>
      <titulo>O Símbolo Perdido</titulo>
      <ano>2009</ano>
      <preco>25.00</preco>
      <oferta/>
   </livro>
</livraria>
```

➢Resolução de Questões

Questão 5 (FCC - 2008 - MPE-RS - Técnico em Informática - Área Sistemas)

Em XML pode-se definir um atributo, como informação adicional ao elemento, conforme o exemplo abaixo:

a) <funcionario> <sexo> masculino ...
b) <funcionario sexo=masculino> ...
c) <funcionario sexo="masculino"> ...
d) <funcionario> <sexo> "masculino" ...
e) <funcionario <sexo>="masculino"> ...

➢Resolução de Questões

Questão 6 (FCC - 2008 - MPE-RS - Técnico em Informática - Área Sistemas)

Sobre a sintaxe XML, considere:

I. Um elemento <CALCULA> deve ter sempre uma tag de fechamento <FIMCALCULA>.
II. Uma <Lista> *tag* é diferente da *tag* <lista>.
III. Um elemento <A> aberto no interior do elemento <B> pode ser fechado fora do elemento <B>.

Está correto o que consta APENAS em:

a) III
b) II
c) II e III
d) I e III
e) I e II

➢Final da Aula 2

GABARITO:
1 – CERTO
2 – Letra B
3 – CERTO
4 – ERRADO
5 – Letra C
6 – Letra B

Até a próxima aula!

XML para Concursos Públicos

Aula 03 – Estrutura da XML

➢Documento XML “Bem Formado”

Estudamos nas aulas anteriores sobre conceitos e os marcadores, que representam a base da Linguagem XML. Agora iremos abordar como estruturar um Documento XML.

Todo Documento XML deve estar “Bem Formado” para que seja aceito pelos programas que o utilizam. Lembre-se que a XML difere da HTML pois a XML não admite erros nas construções das tags, enquanto a HTML admite tais erros.

Existem algumas regras que o documento XML deve atender para que seja considerado “Bem Formado”, são elas:

1 – Deve haver um único Elemento Raiz, ou seja, um nó pai.

2 – Todos os Elementos XML devem estar corretamente aninhados.

3 – Todos os marcadores devem estar fechados.

4 – Todos os Atributos devem estar delimitados por aspas (simples ou duplas).

5 – Os Marcadores são CASE SENSITIVE.

➢Estrutura do Documento XML

Para construir um Documento XML, devemos atentar para algumas características. A seguir temos um exemplo:

```
<?xml version="1.0" ?>
<!DOCTYPE documento >
<documento>
  <titulo>Meu primeiro XML</titulo>
  <conteudo>
    Aqui vou descrever o conteúdo!
  </conteudo>
</documento>
```

A primeira linha é a chamada "Declaração XML", onde podemos colocar algumas informações sobre o documento.

Abaixo da declaração inicial podemos incluir outras declarações de comando XML, que são iniciadas com os símbolos <! .

Depois das declarações, podemos incluir os Elementos XML, sempre tendo apenas um Elemento Raiz. Repare que no exemplo, o Elemento Raiz é o <documento>, e todos os outros Elementos são descendentes (filhos) deste Elemento Raiz. Ou seja, todos os outros Elementos estão num nível de aninhamento ABAIXO do Elemento Raiz.

➢A Declaração `?xml`

A declaração **`?xml`** no início de um documento XML é opcional. Mas caso exista, esta declaração DEVE ser **o primeiro texto** a aparecer no documento.

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<documento>
  <titulo>Meu primeiro XML</titulo>
  <conteudo>
    Aqui vou descrever o conteúdo!
  </conteudo>
</documento>
```

<?xml?>

O que devemos saber sobre esta declaração XML:

Ela é opcional, mas caso seja usada, o atributo **`version`** é obrigatório. Como estamos estudando a versão 1.0 da XML, sempre usaremos este atributo assim: `version="1.0"`.

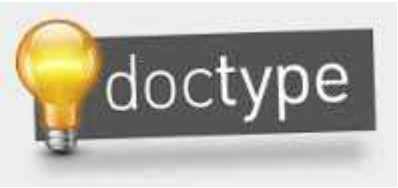
O atributo **`encoding`** é opcional. Ele informa em qual conjunto de caracteres o documento foi construído. Este valor pode ser "utf-8", "utf-16", "ISO-8859-1", entre outros. Este atributo, se usado, deve vir LOGO após o atributo **`version`** obrigatoriamente.

O atributo **`standalone`** é opcional, e seu valor padrão é "`yes`". Este atributo indica se o documento XML necessita de recursos externos. Ele deve ser usado após o atributo **`encoding`**.

➢A Declaração **!DOCTYPE**

A declaração **!DOCTYPE** logo após a declaração **?xml** define qual o Elemento Raiz ou *Root Tag* do documento XML.

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<!DOCTYPE documento >
<documento>
  <titulo>Meu primeiro XML</titulo>
  <conteudo>
    Aqui vou descrever o conteúdo!
  </conteudo>
</documento>
```



Esta declaração DOCTYPE não é obrigatória.

A sintaxe usada para esta declaração não é igual à usada no resto do documento XML. Esta declaração DOCTYPE faz parte da DTD (*Document Type Definitions*), que é uma ferramenta que podemos usar para "validar" documentos XML. Mas isto iremos estudar mais adiante em outra aula.

O que é importante notar é que estas declarações (**?xml** e **!DOCTYPE**) não possuem *tags* de fechamento. Isto é correto pois tais declarações não fazem parte do conteúdo em si, apenas são METADADOS (informações sobre as informações) do documento.

➢Comentários

Provas de TI
Tecnologia da Informação para Concursos

Também é possível inserir comentários num documento XML com o intuito de fornecer explicações a respeito do mesmo.

```
<?xml version="1.0">
<!-- Este é um comentário de uma linha -->
<documento>
  <titulo>Meu primeiro XML</titulo>
<!-- Este é um comentário
   de duas linhas -->
</documento>
```

A inserção de comentários num documento XML é da mesma forma que na linguagem HTML. Utilizamos os símbolos **<!--** para iniciar o comentário, e para finalizar usamos os símbolos **-->** .

É importante que um comentário não seja inserido DENTRO de um marcador, pois isso pode fazer com que a própria *tag* seja fechada erroneamente. Por exemplo:

```
<documento>
  <titulo <!-- Comentário errado -->>Meu primeiro
XML</titulo>
</documento>
```

Ao colocar o comentário, o símbolo **>** do comentário fecha a *tag* de abertura **titulo**.

➢Espaços em branco

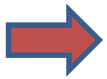
Num documento XML podemos tratar os “espaços em branco” de forma diferenciada.

Em primeiro lugar devemos dizer o que são “espaços em branco”. Segundo a especificação 1.0 da XML, podem ser: o próprio espaço, uma quebra de linha ou um caracter de tabulação.

Por padrão, as aplicações que leem XML desconsideram tais “espaços em branco”. Por exemplo:

```
<documento>
  <titulo>Meu primeiro XML</titulo>
  <conteudo> 1a linha
   2a linha
  </conteudo>
</documento>
```

Documento XML original



```
<documento>
  <titulo>Meu primeiro XML</titulo>
  <conteudo>1a linha 2a linha</conteudo>
</documento>
```

Documento XML interpretado pela aplicação

Ao ser lido por uma aplicação, repare que a quebra de linha do elemento <conteudo> foi desconsiderado.

Para controlar este efeito, a XML possui um atributo padrão chamado **xml:space**, cujos valores podem ser:

default: a aplicação pode fazer o padrão.

preserve: a aplicação deve preservar os espaços!

➢Resolução de Questões

Questão 1 (FCC - 2012 - MPE-AP - Analista Ministerial - Tecnologia da Informação)

Um documento XML bem formatado é aquele que apresenta uma sintaxe XML correta. Sobre as regras de sintaxe em documentos XML bem formatados é correto afirmar:

a) Os elementos XML não podem ter mais que um atributo e o valor desse atributo pode estar vazio.
b) Não é necessário que um documento XML tenha um elemento raiz.
c) Os elementos XML não precisam ser fechados por tag, exceto o que define a versão da XML usada.
d) Tags XML são case sensitive e os valores dos atributos devem aparecer entre aspas.
e) Elementos XML não precisam ser aninhados corretamente, sendo assim, o primeiro que abre sempre será o primeiro que fecha.

➢Resolução de Questões

Questão 2 (CESPE - 2011 - TRE-ES - Técnico - Programação de Sistemas - Específicos)

```
<pessoa> <nome> Fulano de Tal </nome>
  <matricula>123456</matricula>
  <cargo> Coordenador</cargo>
  <departamento>
     <sala>1001</sala><ramal>1234</departamento>
</pessoa>
```

Considerando a estrutura XML acima, armazenada no arquivo **Funcionario.xml**, julgue o próximo item.

No prólogo de um arquivo XML, existe o atributo **`standalone`** o qual, com valor padrão **`yes`**, é de escrita obrigatória, o que indica que o documento não pode ser analisado no lado servidor.

( ) CERTO ( ) ERRADO

➢Resolução de Questões

Questão 3 (CESPE - 2011 - TRE-ES - Técnico - Programação de Sistemas - Específicos)

```
<pessoa> <nome> Fulano de Tal </nome>
  <matricula>123456</matricula>
  <cargo> Coordenador</cargo>
  <departamento>
     <sala>1001</sala><ramal>1234</departamento>
</pessoa>
```

Considerando a estrutura XML acima, armazenada no arquivo **Funcionario.xml**, julgue o próximo item.

Com essa estrutura, ao se abrir o arquivo **Funcionario.xml** em um navegador, será mostrado um erro de processamento de recurso.

( ) CERTO ( ) ERRADO

➢Resolução de Questões

Questão 4 (CESPE - 2010 - TRE-BA - Analista Judiciário - Análise de Sistemas)

Acerca do XML, julgue o item a seguir.

A instrução a seguir está sintaticamente correta e permite o uso de algarismos romanos para codificação de números.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

( ) CERTO ( ) ERRADO

➢Resolução de Questões

Questão 5 (CESGRANRIO - 2008 - Petrobrás - Analista de Sistemas Júnior - Engenharia de Software)

Um tag XML válido do ponto de vista sintático é

a) <nome do cliente>Carlos da Silva</nome do cliente>
b) <_endereco tipo="residencial">Rua das Flores, 1234</_endereco>
c) <telefone numero=12345678 />
d) <*preferencial* />
e) <profissão>Professor</profissão>

➢Resolução de Questões

Questão 6 (CESGRANRIO - 2009 - BNDES - Análise de Sistemas – Desenvolvimento)
Considere as afirmativas a seguir sobre tecnologias de desenvolvimento para aplicações na Internet.
I - Ao contrário da linguagem Java, Javascript é uma linguagem de programação fracamente tipada que pode ser usada em páginas HTML na construção de aplicativos disponibilizados para visualização na Internet.
II - XML é uma metalinguagem capaz de descrever linguagens de marcação, utilizada também como elemento de integração entre sistemas.
III - Um exemplo de XML bem formado é:

```
<nome_destinatario>Pessoa A</nome_destinatario>
<nome_remetente>Pessoa B</nome_remetente>
<endereco_destino complemento="ap 101">Rua G,702
</endereco_destino>
<endereco_remetente complemento="ap 402">Rua K,
14</endereco_remetente>
<cabecalho></cabecalho>
<conteudo>Conteúdo da carta aqui</conteudo>
```

Est(ao) correta(s) a(s) afirmativa(s):
a) II, apenas
b) III, apenas
c) I e II, apenas
d) I e III, apenas
e) I, II e III

➢Final da Aula 3

GABARITO:
1 – Letra D
2 – ERRADO
3 – CERTO
4 – ERRADO
5 – Letra B
6 – Letra C

Até a próxima aula!

XML para Concursos Públicos

Aula 04 – DTD (*Document Type Definitions*)

➢Documento XML Válido

Já estudamos o que é um Documento XML “Bem Formado”.
Agora vamos ver o que é um Documento XML “Válido”.

Documentos XML “Bem Formados” obedecem algumas regras de sintaxe que vimos na aula anterior. Mas isso não quer dizer que tais Documentos XML estejam “Válidos”.

Para ser considerado “Válido”, o Documento XML, além de estar “Bem Formado”, deve obedecer regras estipuladas em uma das duas ferramentas abaixo:

DTD (*Document Type Definition*): regras que definem quais elementos, atributos e quantidade dos mesmos devem existir num Documento XML.

XML Schema: Desenvolvido pela W3C como uma alternativa ao DTD. Usa a própria sintaxe XML para descrever as regras, além de implementar melhorias visando suprir algumas deficiências da DTD.

LEMBRE-SE: Apesar de importante, esta validação é **opcional**!

➢Por que validar XML?

A XML é flexível e independente de plataformas, por isso é usado como ferramenta de interface entre sistemas implementados por tecnologias distintas.

Daí surge uma necessidade de se validar um documento XML, para certificar-se que um documento XML criado por um sistema possa ser entendido por outro sistema.

A SGML, precursora da XML, já apresentava uma ferramenta de validação, que era a DTD (*Document Type Definitions*). Por isso, foi adotada no início da XML para especificar as regras de validação de Documentos XML.

➢Tipos de DTD

Existem 2 tipos de DTDs: Internas e Externas.

DTD Interna: é definida juntamente com o documento XML. É útil apenas quando as regras são muito específicas para este documento.

DTD Externa: é definida numa entidade externa ao documento XML. É o mais usado, pois um mesmo DTD pode servir para uma grande variedade de documentos XML. Desta forma, se for necessária uma mudança no DTD, bastaria alterarmos o DTD e não todos os documentos XML que o utilizam.

Um mesmo documento XML pode ter várias DTDs associadas. Assim, podemos ter uma mistura de DTD Interna com DTD Externa num mesmo documento XML. Alguns autores se referem a isso como “DTD Mista”, mas este termo não é muito usado.

➢Estrutura da DTD Interna

Considere o documento XML a seguir. Existem várias formas de se estruturar uma DTD. Vamos iniciar com uma DTD Interna.

```
<agenda>
  <contato cod="1">
    <nome>André</nome>
    <fone>12345678</fone>
  </contato>
  <contato cod="2">
    <nome>Maria</nome>
    <fone>87654321</fone>
  </contato>
</agenda>
```

Documento XML

```
<!DOCTYPE agenda >
```

DTD

```
<?xml version="1.0" ?>
<!DOCTYPE agenda >
<agenda>
  <contato cod="1">
    <nome>André</nome>
    <fone>12345678</fone>
  </contato>
  <contato cod="2">
    <nome>Maria</nome>
    <fone>87654321</fone>
  </contato>
</agenda>
```

Documento XML com uma DTD Interna

Um DTD inicia com a expressão **<!DOCTYPE** , que já vimos em aulas anteriores. É usado para indicar qual o elemento raiz do documento. No nosso exemplo, o elemento raiz é agenda.

➢Estrutura da DTD Externa

DTD Externas ficam armazenadas em arquivos externos ao documento XML.

```
<agenda>
  <contato cod="1">
    <nome>André</nome>
    <fone>12345678</fone>
  </contato>
  <contato cod="2">
    <nome>Maria</nome>
    <fone>87654321</fone>
  </contato>
</agenda>
```

Documento XML

```
<!ELEMENT agenda (contato) >
```

Arquivo "agenda.dtd"

```
<?xml version="1.0" ?>
<!DOCTYPE agenda SYSTEM "agenda.dtd" >
<agenda>
  <contato cod="1">
    <nome>André</nome>
    <fone>12345678</fone>
  </contato>
  <contato cod="2">
    <nome>Maria</nome>
    <fone>87654321</fone>
  </contato>
</agenda>
```

Documento XML com uma DTD Externa

Na DTD Externa usamos a expressão **<!DOCTYPE** , depois a palavra **SYSTEM** e a URL do arquivo que contém a DTD.

No arquivo, usamos a expressão **<!ELEMENT** para informar qual o elemento raiz, mas devemos fazer a mesma referência na expressão **<!DOCTYPE** no documento XML.

➢Sintaxe da DTD

As DTDs usam uma sintaxe diferente da XML.

Para definição de Elementos XML usamos a expressão **<!ELEMENT** :

```
<!ELEMENT nome_da_tag categoria >  OU
<!ELEMENT nome_da_tag ( expressao_regular ) >
```

```
<!ELEMENT agenda (contato) >
<!ELEMENT contato (nome,fone) >
<!ELEMENT nome (#PCDATA) >
<!ELEMENT fone (#PCDATA) >
```

categoria e ***expressao_regular*** podem assumir vários valores, que iremos estudar ainda.

Para definição de atributos usamos a expressão **<!ATTLIST** :

```
<!ATTLIST nome_da_tag nome_do_atributo tipo_do_atributo valor_default>
```

```
<!ATTLIST contato cod CDATA “1”>
```

Neste exemplo, estamos definindo um atributo chamado **cod** do marcador **contato**. Seu tipo é CDATA e seu valor padrão é 1. Vamos estudar depois o que significa CDATA, e como definir as outras expressões para lidar com os atributos.

A seguir, vamos entender como usar as Expressões Regulares para definirmos Elementos, e como definir cada um dos atributos, usando como base o documento XML que ilustramos neste aula.

➢Categoria de Elementos

Podemos definir Elementos através de Categorias, conforme abaixo:

```
<!ELEMENT nome_da_tag categoria >
```

```
<!ELEMENT linha EMPTY >

<!ELEMENT observacoes ANY >
```

Exemplos de DTD

EMPTY permite a criação dos Elementos Vazios, ou seja, sem conteúdo.

ANY permite a criação de Elementos que podem ter qualquer tipo de conteúdo.

É IMPORTANTE LEMBRAR QUE:

Estas duas categorias (**EMPTY** e **ANY**) raramente são usadas pois permitem uma liberdade de definição de dados que pode não ser desejável.

Um exemplo de Elemento Vazio é da HTML, o marcador **`img`**, que permite a inserção de imagens em documentos HTML.

➢Definição de Elementos

Podemos definir Elementos através de Expressões Regulares entre parênteses.

```
<!ELEMENT nome_da_tag ( expressao_regular ) >
```

```
<!ELEMENT agenda (contato) >
<!ELEMENT contato (nome,fone) >
<!ELEMENT nome (#PCDATA) >
<!ELEMENT fone (#PCDATA) >
```

Exemplo de DTD

Uma expressão regular pode ser um Elemento.

Uma expressão regular pode ser uma sequência de Elementos separados por vírgula.

Uma expressão regular pode ser uma indicação de que o Elemento só pode conter texto puro, chamado em XML de **#PCDATA** .

É IMPORTANTE LEMBRAR QUE:

Ao definirmos uma sequência de Elementos, eles devem aparecer NA ORDEM em que são escritos na Expressão Regular. No nosso exemplo, o elemento **nome** deve vir ANTES do elemento **fone**.

#PCDATA significa *Parsed Character Data*, que é todo texto que deve ser interpretado (“parseado”) por uma aplicação que lê o documento XML.

➢Multiplicidade de Elementos

As Expressões Regulares utilizam alguns símbolos importantes para definição de multiplicidade de elementos, que são descritos abaixo:

`<!ELEMENT nome_da_tag ( expressao_regular ) >`

`expressao*` : zero ou mais repetições de `expressao`
`expressao+` : uma ou mais repetições de `expressao`
`expressao?` : zero ou UMA ocorrência de `expressao`
`expressao1 | expressao2` : ou `expressao1` ou `expressao2`
`expressao1 , expressao2` : `expressao1` e `expressao2`, nesta ordem
`expressao1.expressao2` : `expressao1` concatenado com `expressao2`

Vejamos alguns exemplos de multiplicidade:

`<!ELEMENT contato (nome)>` → O elemento **`contato`** DEVE possuir UM elemento **`nome`**.

`<!ELEMENT agenda (contato*)>` → O elemento **`agenda`** pode ter zero ou mais elementos **`contato`**.

`<!ELEMENT contato (nome+)>` → O elemento **`contato`** DEVE ter um ou mais elementos **`nome`**.

`<!ELEMENT contato (fone?)>` → O elemento **`contato`** DEVE zero ou UM elemento **`fone`**.

➢Definição de Atributos

Já vimos que para definir atributos usamos a expressão **<!ATTLIST** :
<!ATTLIST *nome_da_tag nome_do_atributo tipo_do_atributo valor_default*>

```
<!ATTLIST contato cod CDATA “1”>
```

Exemplo de DTD

```
<contato cod=“2” />
```

Elemento XML que atende a DTD

Neste exemplo, estamos definindo um atributo chamado **cod** do Elemento **contato**. Seu tipo é **CDATA**, ou seja, *Character Data*, e seu valor padrão é 1. *Character Data* é todo o texto que NÃO é interpretado (*parseado*) pela aplicação que lê o documento XML. Ele é apenas um METADADO, e não o dado propriamente dito.

O valor default pode assumir um valor definido entre aspas, como no nosso exemplo acima, ou pode assumir também os seguintes valores:

`#REQUIRED` : O atributo DEVE ser declarado no elemento, mas não há um valor padrão pré-definido.

`#IMPLIED` : O atributo PODE ser declarado no elemento, mas não é obrigatório.

`#FIXED`: O atributo só pode assumir o valor declarado entre aspas, e não pode ser alterado.

➢Tipos de Atributos: Identificadores

Existem vários tipos de Atributos, além do **CDATA**, a seguir são explicados aqueles tipos que lidam com identificadores.

ID : é um identificador único no documento XML. Deve ser usado quando se deseja criar elementos únicos num documento.

```
<empresa cod="EMP1" />
```

Elemento XML

```
<!ATTLIST empresa cod ID #REQUIRED>
```

DTD para criar um atributo do tipo ID obrigatório

IDREF : é uma referência a um identificador único no documento XML. Deve ser usado quando se deseja criar elementos que são relativos entre si.

```
<produto codEmp="EMP1" />
```

Elemento XML

```
<!ATTLIST produto codEmp IDREF>
```

DTD para criar um atributo que é referência a um ID

Mas LEMBRE-SE: atributos do tipo ID não podem ser repetidos num mesmo documento XML. E um atributo IDREF depende SEMPRE da existência de algum atributo ID. Logo, atente para os exemplos a seguir considerando as duas regras de atributo definidas acima:

```
<empresa cod="EMP1" />
<empresa cod="EMP2" />
<produto codEmp="EMP3" />
<empresa cod="EMP1" />
```

Erro, pois não há um elemento XML cujo identificador seja "EMP3". Este atributo DEPENDE da existência de um ID.

Erro, pois o ID "EMP1" já foi usado no primeiro elemento do documento.

➢Outros tipos de Atributos

NMTOKEN : o valor do atributo deve ser um nome válido em XML, ou seja, só pode conter letras, números, ou os símbolos de traço ou underline ou ponto ou dois pontos. NÃO PODE CONTER ESPAÇOS EM BRANCO!

```
<empresa cod="EMP_1" />
```

Elemento XML

```
<!ATTLIST empresa cod NMTOKEN>
```

DTD correspondente

Lista de Valores : neste caso, os valores do atributo devem ser um dos definidos numa lista de possíveis valores. Os valores devem ser separados pelo caracter **|**.

```
<!ATTLIST telefone tipo (casa | trabalho | celular) "celular">
```

DTD para atributo do tipo Lista de Valores

ENTITY: o valor do atributo deve ser representado por uma Entidade que também deve ser declarada na DTD.

```
<!ENTITY autor1 "Marcio Vilanova">

<!ATTLIST texto autor ENTITY>

<texto autor="&autor1;" />
```

Neste exemplo, declaramos uma entidade chamada **autor1**. Na 2ª linha definimos a regra DTD para o atributo **autor** ser uma entidade. E na 3ª linha, usamos a entidade no atributo.

➢Atributos de múltiplos valores

Os tipos de atributos IDREF, NMTOKEN e ENTITY só podem aceitar um valor para o atributo. Mas existem outros tipos equivalentes que permitem que você defina múltiplos valores.

IDREFS: é usado para uma lista de IDREF, separados por espaço.

```
<!ATTLIST produto fornecedoras IDREFS>
```

DTD para atributo com uma lista de valores IDREFs

```
<produto fornecedoras="EMP1 EMP2" />
```

Elemento XML cujo atributo é do tipo IDREFS

Analogamente, existem os tipos **NMTOKENS** (para uma lista de valores do tipo NMTOKEN) e **ENTITIES** (para uma lista de valores do tipo ENTITY).

Os tipos de atributo mais usados são o **CDATA** e a Lista de Valores. Os outros raramente são usados e caem muito pouco em Concursos Públicos. Portanto, não perca muito tempo em estudar esta variedades de atributos. Concentre-se em entender o que significa um atributo!

➢Final da Aula 4

Provas de TI
Tecnologia da Informação para Concursos

Na próxima aula veremos mais exemplos de DTDs analisando e resolvendo Questões!

Até a próxima aula!

XML para Concursos Públicos

Aula 05 – Resolução de Questões de DTD

➢Resolução de Questões

Questão 1 (FCC - 2012 - TRF - 2ª REGIÃO - Técnico Judiciário – Informática)

Analise a DTD abaixo, presente em um arquivo XML:

```
<?xml version="1.0" encoding="ISO-8859-1">
<!DOCTYPE loja[
<!ELEMENT loja(produto+)>
.............................
<!ELEMENT nome(#PCDATA)>
<!ELEMENT quantidade (#PCDATA)>
<!ELEMENT peso (#PCDATA)>
]>
```

A instrução que preenche a lacuna ....., declarando que o elemento **produto** poderá conter apenas uma ocorrência do elemento **nome** e o elemento **quantidade** ou o elemento **peso**, é:

a) <!ELEMENT produto (nome+,(quantidade|peso))>
b) <!ELEMENT produto (nome,(quantidade&peso))>
c) <!ELEMENT produto (nome?,(quantidade&peso))>
d) <!ELEMENT produto (nome*,(quantidade||peso))>
e) <!ELEMENT produto (nome,(quantidade|peso))>

➢Resolução de Questões

Questão 2 (FCC - 2012 - MPE-PE - Técnico Ministerial – Informática)

Para que um documento XML seja considerado válido, ele precisa ter um conjunto de instruções que define a estrutura do documento, ou seja, quais elementos e atributos são permitidos. Esse conjunto de instruções (que pode ser declarado dentro de um documento XML ou em um arquivo à parte) é denominado:

a) Extensible Stylesheet Language Transformations.
b) Document Type Definition.
c) XML Description Elements and Attributes.
d) Cascading Style Sheets.
e) Standard Description Library.

➢Resolução de Questões

Questão 3 (FCC - 2012 - TRT - 11ª Região (AM) - Técnico Judiciário - Tecnologia da Informação)

Observe a Document Type Definition (DTD):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE loja [
<!ELEMENT loja (produto+)>
<!ELEMENT produto (nome, distribuidor+, cor?, marca*,
        (quantidade|peso))>
<!ELEMENT nome (#PCDATA)>
<!ELEMENT distribuidor (#PCDATA)>
<!ELEMENT cor (#PCDATA)>
<!ELEMENT marca (#PCDATA)>
<!ELEMENT quantidade (#PCDATA)>
<!ELEMENT peso (#PCDATA)>
]>
```

O uso do asterisco no elemento **`marca`** indica:

a) nenhuma ou uma ocorrência do elemento.
b) nenhuma ou muitas ocorrências do elemento.
c) uma ou muitas ocorrências do elemento.
d) apenas uma ocorrência do elemento.
e) que a ocorrência do elemento é obrigatória.

➢Resolução de Questões

Questão 4 (CESPE - 2011 - TJ-ES - Analista Judiciário - Análise de Sistemas – Específicos)

Acerca de desenvolvimento de aplicações para Web, julgue o próximo item.

Um DTD (document type definition) permite definir o conteúdo válido para um documento XML. A sintaxe abaixo, de uma linha do DTD, define que cada elemento livro tem obrigatoriamente um título, um resumo opcional e múltiplos capítulos.

```
<!ELEMENT livro (titulo, resumo?, capitulos+)>
```

( ) CERTO ( ) ERRADO

➢Resolução de Questões

Questão 5 (Prova: UFG - 2010 - UFG - Analista de TI - Desenvolvimento de Sistemas)

Analise o trecho da Document Type Definition (DTD) a seguir.

```
<!DOCTYPE jornal [
<!ELEMENT jornal (artigo+)>
<!ELEMENT artigo (manchete, corpo)>
<!ELEMENT manchete (#PCDATA)>
<!ELEMENT corpo (#PCDATA)>
<!ATTLIST artigo autor CDATA #REQUIRED>
<!ATTLIST artigo editor CDATA #IMPLIED>
<!ATTLIST artigo data CDATA #IMPLIED>
]>
```

A Document Type Definition (DTD) permite a definição de regras descritas na forma de expressões regulares, que indicam que padrão de subelementos e atributos podem ocorrer dentro de um elemento XML. O trecho de DTD apresentado determina que o elemento

a) `jornal` é definido para conter 0 ou mais subelementos `artigo`.

b) `jornal` é definido para conter os subelementos `corpo` e `manchete`, nesta ordem, ambos armazenando dados de texto.

c) `artigo` é definido para conter um atributo `autor` que, por sua vez, é opcional.

d) `artigo` é definido para conter um atributo `editor` que, por sua vez, é opcional.

➢Resolução de Questões

Questão 6 (CESPE - 2010 - MPU - Técnico de Informática)

Julgue o próximo item acerca de XML (Extensible Markup Language).

Todo arquivo XML deve possuir um arquivo DTD correspondente.

( ) CERTO ( ) ERRADO

➢Resolução de Questões

Questão 7 (CESPE - 2010 - TRE-MT - Técnico Judiciário - Programação de Sistemas)

Acerca de XML (eXtensible Markup Language) e DTD (Document Type Definition), assinale a opção correta.

a) Uma DTD deve ser declarada dentro de um documento XML, pois não é possível utilizar uma referência externa para um documento separado.
b) PCDATA é um tipo de texto que não é processado pelo analisador (parser).
c) CDATA é um tipo de texto que é processado pelo analisador (parser).
d) Um documento XML pode ser considerado válido quando é um documento bem formado e obedece as regras definidas em uma DTD.
e) Em XML e DTDs, elementos fornecem informação adicional a respeito de atributos.

Resposta: Letra D.

➢Resolução de Questões

Questão 8 (NCE-UFRJ - 2005 - BNDES - Profissional Básico - Análise de Sistemas – Suporte)
Observe o esquema DTD (registro.dtd) na figura a seguir:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT funcionario (nome, filiacao,telefone*)>
<!ELEMENT nome (#PCDATA)>
<!ELEMENT filiacao EMPTY>
<!ATTLIST filiacao
   pai CDATA #REQUIRED
   mae CDATA #REQUIRED
>
<!ELEMENT telefone (#PCDATA)>
<!ATTLIST telefone preferido (sim|nao) "nao">
```

O documento XML que atende ao esquema é:

a)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE funcionario SYSTEM "C:\Lixo\Registro.dtd">
<funcionario>
    <filiacao pai="Saulo Silva" mae="Sandra Silva"/>
    <nome>Pedro Silva</nome>
    <telefone>22222222</telefone>
    <telefone preferido="sim">99999999</telefone>
</funcionario>
```

➢Resolução de Questões

b)

```
<?xml version=“1.0” encoding=“ISO-8859-1”?>
<!DOCTYPE funcionario SYSTEM “C:\Lixo\Registro.dtd”>
<funcionario>
    <nome>Pedro Silva</nome>
    <telefone>22222222</telefone>
    <telefone preferido=“sim”>99999999</telefone>
</funcionario>
```

c)

```
<?xml version=“1.0” encoding=“ISO-8859-1”?>
<!DOCTYPE funcionario SYSTEM “C:\Lixo\Registro.dtd”>
<funcionario>
    <nome>Pedro Silva</nome>
    <filiacao>
        <pai>Saulo Silva</pai>
        <mae>“Sandra Silva”</mae>
    </filiacao>
    <telefone>22222222</telefone>
    <telefone preferido=“sim”>99999999</telefone>
</funcionario>
```

➢Resolução de Questões

d)

```
<?xml version=“1.0” encoding=“ISO-8859-1”?>
<!DOCTYPE funcionario SYSTEM “C:\Lixo\Registro.dtd”>
<funcionario>
    <filiacao pai=“Saulo Silva” mae=“Sandra Silva”/>
    <nome>Pedro Silva</nome>
    <telefone preferido=“”>22222222</telefone>
    <telefone preferido=“sim”>99999999</telefone>
</funcionario>
```

e)

```
<?xml version=“1.0” encoding=“ISO-8859-1”?>
<!DOCTYPE funcionario SYSTEM “C:\Lixo\Registro.dtd”>
<funcionario>
    <nome>Pedro Silva</nome>
    <filiacao pai=“Saulo Silva” mae=“Sandra Silva”/>
</funcionario>
```

➢Resolução de Questões

Questão 9 (FCC - 2012 - TCE-AM - Analista de Controle Externo - Tecnologia da Informação)
Considere o documento XML bem formatado a seguir:

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE loja[
<!ELEMENT loja (entrega+,produto+)>
<!ELEMENT entrega (tempo+)>
<!ATTLIST entrega codigoEntrega ID  #REQUIRED>
<!ELEMENT produto (#PCDATA)>
<!ATTLIST produto tipoEntrega IDREF  #IMPLIED>
<!ELEMENT tempo (#PCDATA)>
]>
<loja>
   <entrega codigoEntrega="motoboy">
      <tempo>2 dias</tempo>
   </entrega>
   <entrega codigoEntrega="correios">
      <tempo>1 dia</tempo>
   </entrega>
   <produto tipoEntrega="correios">Impressora</produto>
</loja>
```

Sobre o documento apresentado, é correto afirmar:
a) O atributo `tipoEntrega` é obrigatório nos elementos `produto`.
b) O sinal de mais (+) na descrição dos elementos `entrega`, `produto` e `tempo` indica que poderá haver no documento nenhuma ou muitas ocorrências desses elementos.
c) Se um novo elemento `produto` for inserido, ele deverá ter como conteúdo do atributo `tipoEntrega` o valor motoboy ou correios.
d) Não é válido, pois há mais de uma ocorrência do elemento `entrega`.
e) O atributo `codigoEntrega` é obrigatório, porém, poderá estar vazio.

➢Final da Aula 5

GABARITO:
1 – Letra E
2 – Letra B
3 – Letra B
4 – CERTO
5 – Letra D
6 – ERRADO
7 – Letra D
8 – Letra E
9 – Letra C

Até a próxima aula!

XML para Concursos Públicos

Aula 06 – *XML Schema*

➢Validação com “Esquemas XML”

Já estudamos que podemos validar um documento XML através de DTDs. Agora veremos “Esquemas XML”.

Apesar de bastante úteis para validar documentos XML que contenham apenas texto, os DTDs possuem algumas desvantagens:

- DTDs não permitem a definição de tipos mais específicos (como números, datas etc), eles permitem apenas o tipo **#PCDATA**, que é um texto qualquer.
- A sintaxe das DTDs é diferente da sintaxe XML. Isso gera um trabalho a mais para os *parsers*. Além de termos de aprender uma nova sintaxe...
- Não permitem o uso de *namespaces* (que iremos estudar mais em outra aula).
- Como consequência, a W3C criou outra forma de validação: o ***XML Schema***. E o recomenda para a validação de documentos XML.

LEMBRE-SE: Apesar de importante, esta validação TAMBÉM é **opcional**!

➢Vários Esquemas XML

Existem vários Esquemas XML para validação.

Ao se deparem com as limitações dos DTDs, alguns desenvolvedores criaram outras formas de validação através de “Esquemas”. Daí surgiram o “RELAX”, “Schematron” entre outras.

A W3C, com o objetivo de padronizar os esquemas XML, criou seu próprio e o denominou “***XML Schema***”. E este é o mais cobrado em Concursos Públicos e será o que nós estudaremos! Algumas bancas chegam, inclusive, a chamar o “XML Schema” de simplesmente “esquema”.

Mas LEMBRE-SE: não confunda “***XML Schema***”, que é o esquema XML recomendado pela W3C, com “esquemas XML” diversos. A W3C criou um nome que pode confundir os mais leigos.

➢Exemplo simples de *XML Schema*

Considere o documento XML a seguir, e o seu respectivo esquema *XML Schema*:

```
<?xml version="1.0"
    encoding="utf-8"?>
<contato>
    <nome>Artur</nome>
    <tel>1234-7654</tel>
</contato>
```

Documento XML

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="contato">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="nome" type="xs:string"/>
      <xs:element name="tel" type="xs:string"/>
     </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML Schema

➢Sintaxe do *XML Schema*

O *XML Schema* usa uma sintaxe idêntica da XML. Vamos analisar o esquema a seguir:

Para iniciar um esquema *XML Schema*, também usamos a declaração **xml** como em qualquer documento XML: `<?xml version="1.0"?>`

```
<?xml version="1.0"?>
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="contato">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="nome" type="xs:string"/>
      <xs:element name="tel" type="xs:string"/>
     </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

O Elemento Raiz do esquema deve ser <**xs:schema**>. Aqui deve ser usada uma *namespace* que aponta para a definição do esquema. No caso do *XML Schema* é "www.w3.org/2001/XMLSchema". Por isso que todas as *tags* do esquema começam com "**xs:**".
Veremos numa aula mais adiante o que vem a ser um *namespace*.

O último marcador fecha o marcador inicial <**xs:schema**>.

➢Definição de Elementos

Definimos elementos usando a tag <xs:element> com a seguinte sintaxe:
<xs:element name="nome do elemento" type="tipo">

Em *XML Schema*, existem dois tipos de Elementos: os **Simples** e os **Complexos** (que também são chamados de *Compostos* por alguns autores).

Os **Elementos Simples** possuem valores de texto ou número ou data, mas não possuem atributos e nem outros Elementos aninhados abaixo dele.

```
<contato>
  <nome>Marcio</nome>
  <idade>18</idade>
  <tel tipo="cel">9999-0000</tel>
</contato>
```

No exemplo, os Elementos <nome> e <idade> são Simples, pois não possuem Elementos abaixo, e nem tampouco atributos.

Os **Elementos Complexos ou Compostos** possuem atributos ou outros Elementos aninhados abaixo dele, além de poder misturar texto com elementos e até mesmo sem conteúdo.

```
<contato>
  <nome>Marcio</nome>
  <idade>18</idade>
  <tel tipo="cel">9999-0000</tel>
</contato>
```

No exemplo, os Elementos <contato> e <tel> são Complexos, pois o primeiro possui outros Elementos abaixo, e o segundo possui um atributo.

➢Elementos Simples

Os Elementos Simples podem assumir vários tipos, também chamados *built in*, que já fazem parte da especificação *XML Schema*. Os mais usados são:

xs:string -> texto puro
xs:decimal -> números com casas decimais
xs: boolean -> valores booleanos (*true* ou *false*)

xs:date -> para datas
xs:time -> para horas
xs:integer -> número inteiro

Assim percebemos como o ***XML Schema*** é superior aos DTDs justamente pelos definição de tipos, já que os DTDs só definem tipos de texto, como o #PCDATA.

É possível também definir conteúdos padrão e fixos para os Elementos Simples:

Para definir um conteúdo padrão usamos o atributo ***default***:

```
<xs:element name="versao" type="xs:string" default="rascunho">
```

Para definir um conteúdo fixo usamos o atributo ***fixed***:

```
<xs:element name="versao" type="xs:string" fixed="1.0">
```

➢Restrições (Elementos / Atributos)

Provas de TI
Tecnologia da Informação para Concursos

Ao contrário do DTD, o XML Schema permite restrições de valores em Elementos e Atributos. Para isso podemos usar os elementos **`xs:simpleType`** e **`xs:restriction`** em conjunto.

Exemplo para restringir valores decimais a um determinado intervalo:

```
<xs:element name="pH">
  <xs:simpleType>
    <xs:restriction base="xs:decimal">
      <xs:minInclusive value="0.0" />
      <xs:maxInclusive value="14.0" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Definimos um elemento "pH" que pode conter valores decimais entre 0.0 e 14.0 inclusive.

Note que usamos o elemento <**`xs:restriction`**> para iniciar uma restrição, e o seu atributo base informa que a restrição se aplica a um númeral não inteiro (decimal).

Exemplo para restringir valores de texto a uma lista de valores:

```
<xs:element name="estadoCivil">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="solteiro" />
      <xs:enumeration value="casado" />
      <xs:enumeration value="separado" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Definimos um elemento "estadoCivil" que pode conter possuir APENAS três valores: "solteiro", "casado" e "separado".

Para este caso, usamos o elemento **`xs:enumeration`** para criar uma lista de valores restritos.

➢Mais sobre Restrições

Existem várias formas de se impor restrições, mas vejamos mais uma que pode ser útil e que possivelmente pode vir a cair em alguma prova que cobre *XML Schema*.

Uma forma importante de restrição é a de permitir a entrada somente de alguns caracteres. Para isso, usamos o elemento **`xs:pattern`**.

```
<xs:element name="resposta">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[A-E]" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Aqui definimos um elemento **`resposta`** que só pode ter um valor, que pode ser ou **A** ou **B** ou **C** ou **D** ou **E**.

Outros exemplos para o **`pattern`**:
`[A-Z][A-Z]` : duas letras maiúsculas
`[a-zA-Z]`: uma letra qualquer
`[abc]` : uma letra desde que seja **a** ou **b** ou **c**

É possível definir zero ou mais ocorrências de um caracter usando o caracter *****, da seguinte forma:

```
<xs:pattern value =“([a-zA-Z])*” />
```

Analogamente, você pode definir uma ou mais ocorrências com o caracter **+ .** Lembra-se das DTDs?

Em relação às restrições, LEMBRE-SE: podem ser aplicadas tanto a ELEMENTOS quanto a ATRIBUTOS!

➢Restrições de Tamanho

É possível determinar o tamanho, em caracteres, que os dados dos elementos simples podem ter através de **`xs:length`.**

```
<xs:element name=“CPF">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:length value=“11" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Definimos um elemento “CPF” que deverá ter obrigatoriamente 11 caracteres e só poderá aceitar números.

Também é possível definir um tamanho mínimo e máximo de caracteres através dos elementos **`xs:minLength`** e **`xs:maxLength`**.

```
<xs:element name=“identificador">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:minLength value=“1" />
      <xs:maxLength value=“10" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Definimos um elemento “identificador” que deve conter de 1 até 10 caracteres.

Você não precisa usar as duas restrições ao mesmo tempo. Você poderia, por exemplo, usar só a **`xs:maxLength`** para definir um tamanho máximo para o “Identificador”.

➢Final da Aula 6

Na próxima aula veremos as definições de Elementos Compostos no *XML Schema*!

Até a próxima aula!

XML para Concursos Públicos

Aula 07 – *XML Schema* – Elementos Compostos

➢Definição de Atributos

Vimos que os Elementos Compostos podem conter atributos, que são definidos de forma análoga aos Elementos, mas usando a *tag* **`xs:attribute`**:

```
<xs:attribute name="id" type="xs:integer" />
```

Neste exemplo, estamos definindo um atributo cujo nome é **`id`** e seu valor deve ser um número inteiro.

Quando um Elemento **`xs:attribute`** está abaixo de um Elemento **`xs:element`**, dizemos que é um atributo local, afetando apenas um elemento.

```
<xs:element name="contato" type="xs:string">
  <xs:attribute name="id" type="xs:integer" />
</xs:element>
```

Note que o elemento **`xs:attribute`** está aninhado abaixo do elemento **`xs:element`**. Isso que dizer que no documento XML o elemento <**`contato`**> possui um atributo de nome **`id`**.

Podemos definir atributos opcionais ou obrigatórios usando o atributo **`use`**:

```
<xs:attribute name="email" type="xs:string"
use="optional"/>
<xs:attribute name="id" type="xs:integer"
use="required"/>
```

Nestes exemplos, o atributo `email` é opcional (optional); e o `id` é obrigatório (required).

➢Elementos Compostos

Provas de TI
Tecnologia da Informação para Concursos

Vimos que Elementos Compostos podem possuir outros Elementos, que por sua vez, podem ser Simples ou Compostos. Para defini-los usamos o elemento <**xs:complexType**>.

No exemplo a seguir criamos um Elemento Composto que contém dois Elementos Simples.

```
<carro>
  <marca>DeLorean</marca>
  <modelo>DMC-12</modelo>
</carro>
```

Elemento XML Composto

Neste exemplo, o *XML Schema* define um Elemento Composto chamado **carro**.

```
<xs:element name="carro">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="marca" type="xs:string" />
      <xs:element name="modelo" type="xs:string" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Esquema XML

O elemento **xs:sequence** define que o elemento **carro** deve possuir dois elementos : **marca** e **modelo** NESTA ORDEM.

Os dois elementos (**marca** e **modelo**) são simples e do tipo texto.

Neste caso, os elementos simples **marca** e **modelo** só estão associados ao elemento composto **carro** e a nenhum outro elemento composto do documento XML.

➢Mais Elementos Compostos

No exemplo anterior, os elementos **marca** e **modelo** só estariam vinculados ao elemento **carro**. Mas é possível disponibilizar estes dois Elementos Simples para outros Elementos Compostos do nosso Documento XML.

```
<meusVeiculos>
  <carro>
    <marca>DeLorean</marca>
    <modelo>DMC-12</modelo>
  </carro>
  <moto>
    <marca>Harley</marca>
    <modelo>FatBoy</modelo>
  </moto>
</meusVeiculos>
```

Documento XML

Neste exemplo, vemos que os Elementos Compostos **carro** e **moto** possuem os mesmos Elementos Simples.

```
<xs:complexType name="veiculo">
  <xs:sequence>
    <xs:element name="marca" type="xs:string" />
    <xs:element name="modelo" type="xs:string" />
  </xs:sequence>
</xs:complexType>
```

Esquema XML para definir um Tipo Composto.

Desta forma, criamos um Tipo Composto englobando estes dois Elementos Simples. E assim este tipo composto ficaria disponível para os outros elementos composto do nosso documento XML.

➢Mais Elementos Compostos

```
<meusVeiculos>
  <carro>
    <marca>DeLorean</marca>
    <modelo>DMC-12</modelo>
  </carro>
  <moto>
    <marca>Harley</marca>
    <modelo>FatBoy</modelo>
  </moto>
</meusVeiculos>
```

Documento XML

```
<xs:element name="meusVeiculos">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="carro" type="veiculo" />
      <xs:element name="moto" type="veiculo" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:complexType name="veiculo">
  <xs:sequence>
    <xs:element name="marca" type="xs:string" />
    <xs:element name="modelo" type="xs:string" />
  </xs:sequence>
</xs:complexType>
```

Esquema XML para o Documento XML

Note que precisamos definir o tipo complexo apenas uma vez, e o usamos duas vezes: uma no elemento **`carro`** e outra no elemento **`moto`**.

➢Herança em Tipos Compostos

Existe o conceito de **Herança** nos tipos compostos, ou seja, é possível criar um Tipo herdando todos os elementos de um Tipo, além de adicionar novos elementos. Para isso usamos os elementos **`xs:complexContent`** (para indicar a herança) e o **`xs:extension`** (para informar qual o tipo que será usado como base).

```
<xs:complexType name="veiculo">
  <xs:sequence>
    <xs:element name="marca" type="xs:string" />
    <xs:element name="modelo" type="xs:string" />
  </xs:sequence>
</xs:complexType>
```

Tipo Composto Base

Como já vimos, o Tipo Composto **`veiculo`** só possui dois elementos simples: **`marca`** e **`modelo`**.

```
<xs:complexType name="veiculoPlus">
  <xs:complexContent>
    <xs:extension base="veiculo">
      <xs:sequence>
        <xs:element name="ano" type="xs:integer" />
        <xs:element name="placa" type="xs:string" />
      </xs:sequence>
    </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

Tipo Composto que herdou do tipo **`veiculo`**.

Já o tipo **`veiculoPlus`** possui todos os elementos do tipo **`veiculo`** e mais dois outros elementos: **`ano`** e **`placa`**.

➢Indicadores

Os Indicadores auxiliam na definição de mais características dos elementos XML. Existem dois tipos: de Ordem e de Ocorrência.

Os Indicadores de Ordem podem ser:

- **`all`** : este define que os elementos filhos podem aparecer em qualquer ordem. Isto é o "contrário" de usarmos o elemento **`xs:sequence`**, que especifica uma ordem obrigatória. Por exemplo:

```
<xs:complexType name="veiculo">
  <xs:all>
    <xs:element name="marca" type="xs:string" />
    <xs:element name="modelo" type="xs:string" />
  </xs:all>
</xs:complexType>
```

Aqui, o Tipo Composto **`veiculo`** os elementos filhos **`marca`** e **`modelo`** podem aparecer em qualquer ordem.

- **`choice`**: este define que apenas um dos elementos filhos deve ocorrer.

```
<xs:complexType name="veiculo">
  <xs:choice>
    <xs:element name="marca" type="xs:string" />
    <xs:element name="modelo" type="xs:string" />
  </xs:choice>
</xs:complexType>
```

Aqui, o Tipo Composto **`veiculo`** só admite UM dos dois elementos filhos **`marca`** ou **`modelo.`** Nunca os dois ao mesmo tempo.

- **`sequence`**: este define que apenas os elementos filhos devem aparecer todos na ordem especificada. Este é o modo que nós usamos nos exemplos anteriores. Para exemplos, retorne alguns slides nesta aula!

➢Indicadores de Ocorrência

Os Indicadores de Ocorrência definem justamente a quantidade de vezes que um determinado Elemento Filho pode ocorrer no Elemento Pai.

Os Indicadores de Ocorrência podem ser:

- **minOccurs** : este define a quantidade mínima de vezes que os elementos filhos podem aparecer no Elemento Pai. Por exemplo:

```
<xs:complexType name="poker">
  <xs:sequence>
    <xs:element name="jogador" type="TJogador" minOccurs="2" />
  </xs:sequence>
</xs:complexType>
```

Neste exemplo, o elemento **jogador** deve ocorrer pelo menos duas vezes.

- **maxOccurs** : este define a quantidade máximade vezes que os elementos filhos podem aparecer no Elemento Pai. Por exemplo:

```
<xs:complexType name="poker">
  <xs:sequence>
    <xs:element name="dealer" type="TJogador" maxOccurs="1" />
  </xs:sequence>
</xs:complexType>
```

Neste exemplo, o elemento **dealer** deve ocorrer no máximo uma vez.

Lembre-se: Tais indicadores podem ocorrer juntos no mesmo elemento. E quando não existirem explicitamente, assumem-se os valores padrão igual a "1".

➢Elementos com Texto Apenas

Existem casos em que desejamos que alguns Elementos contenham apenas Texto (podendo ser número, símbolos etc) e Atributos, mas que NUNCA tenham Elementos Filhos. Nestes casos, usamos o elemento **`xs:simpleContent`**.

Porém, devemos sempre usar em conjunto OU **`xs:restriction`** (caso queira restringir valores) OU **`xs:extension`** (caso queira estender um tipo *built in* ou um tipo criado por nós).

```
<xs:element name="nomeFilme">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="pais" type="xs:string"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

Aqui definimos um elemento **`nomeFilme`** que estende o tipo *built in* **`xs:string`**.

Ele só aceita texto e um atributo, ou seja, não pode conter elementos filhos.

Fragmento de um XML Schema usando **`xs:extension`**.

```
<filme titulo="Wayne's World">
  <nomeFilme pais="Brasil">Quanto mais idiota melhor</nomeFilme>
</filme>
```

Fragmento de um arquivo XML que atende ao XML Schema acima.

➢Elementos Vazios

Vimos que podemos ter Elementos Vazios na XML, contendo apenas ATRIBUTOS. Tais elementos são considerados Elementos Compostos. Existem duas formas para se definir estes tipos de Elementos.

```
<xs:element name="arquivo">
  <xs:complexType>
    <xs:complexContent>
      <xs:restriction base="xs:string"/>
        <xs:attribute name="src" type="xs:string"/>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

1º Modo: usando xs:complexContent.

Um elemento vazio chamado "**arquivo**" que possui apenas um atributo, e nem elemento filho.

Usamos o elemento **<xs:complexContent>** apenas para indicar uma restrição de atributo.

```
<xs:element name="arquivo">
  <xs:complexType>
    <xs:attribute name="src" type="xs:string"/>
  </xs:complexType>
</xs:element>
```

2º Modo: usando apenas xs:complexType.

Poderíamos ter feito a mesma definição, só que de uma forma mais compacta, sem usar o elemento **xs:complexContent**.

➢Conteúdo Misto

Apesar de não muito recomendado, é possível termos Conteúdo Misto, que é texto misturado com outros Elementos Filhos. Para isso, usamos o atributo booleano **`mixed`**.

```
<xs:element name="lembrete">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="dia" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Fragmento de um XML Schema.

Neste exemplo, o elemento **`lembrete`** admite tanto texto quanto o elemento **`dia`**.

```
<lembrete>
Hoje é <dia>27/10/2013</dia> e estou estudando
muito!
</lembrete>
```

Arquivo XML que é atendido pelo XML Schema acima.

➢Grupos de Substituição

É possível definir que certos elementos em nosso arquivo XML possam ser substituídos por outros elementos. Isso é útil quando lidamos com diferentes idiomas num mesmo sistema.

Para isso usamos os Grupos de Substituição, que em *XML Schema*, são definidos pelo elemento **`xs:substitutionGroup`**.

```
<xs:element name="titulo" type="xs:string"/>
<xs:element name="title" substitutionGroup="titulo"/>
<xs:element name="ano" type="xs:integer"/>
<xs:element name="year" substitutionGroup="ano"/>

<xs:complexType name="midia">
  <xs:sequence>
    <xs:element ref="titulo"/>
    <xs:element ref="ano"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="filme" type="midia"/>
<xs:element name="movie" substitutionGroup="filme"/>
```

Fragmento de um XML Schema.

Os elementos simples **`titulo`** e **`ano`** podem ser substituídos respectivamente por **`title`** e **`year`**.

E o elemento composto **`filme`** pode ser substituído pelo elemento **`movie`**.

```
<filme>
  <titulo>Ben-Hur</titulo>
  <ano>1959</ano>
</filme>
```

```
<movie>
  <titulo>Ben-Hur</titulo>
  <ano>1959</ano>
</movie>
```

Dois fragmento de arquivos XML atendidos pelo XML Schema acima.

Importante: o atributo "**`ref`**" usado no XML Schema ao lado faz referência a outro elemento definido no mesmo esquema (elementos **`titulo`** e **`ano`**).

➢Conteúdo não previsto

Mesmo com todo o controle que o XML Schema prevê, é possível determinar que conteúdos (elementos e atributos) não previstos, possam aparecer nos documentos XML.

Para o caso de Elementos, usamos o elemento <**any**>:

```
<xs:complexType name="veiculo">
  <xs:sequence>
    <xs:element name="marca" type="xs:string" />
    <xs:element name="modelo" type="xs:string" />
    <xs:any />
  </xs:sequence>
</xs:complexType>
<xs:element name="ano" type="xs:integer"/>
```

Aqui o elemento composto **veiculo** aceita um elemento **marca**, depois um elemento **modelo** e depois pelo menos um outro elemento qualquer.

Mas lembre-se: este "outro elemento qualquer" deve estar definido em algum lugar no seu XML Schema para que você possa usá-lo. Por exemplo:

```
<veiculo>
  <marca>Ferrari</marca>
  <modelo>308 GTS</modelo>
  <ano>1977</ano>
</veiculo>
```

VÁLIDO, pois o elemento **ano** foi definido no XML Schema.

```
<veiculo>
  <marca>Ferrari</marca>
  <modelo>308 GTS</modelo>
  <dono>Marcio</dono>
</veiculo>
```

INVÁLIDO, pois o elemento **dono** não foi definido no XML Schema.

➢Atributo não previsto

Para o caso de ATRIBUTOS, usamos o elemento <**anyAttribute**> para criar atributos não previstos num XML Schema. Por exemplo:

```
<xs:element name="carro">
<xs:complexType>
  <xs:sequence>
    <xs:element name="marca" type="xs:string" />
    <xs:element name="modelo" type="xs:string" />
  </xs:sequence>
   <xs:anyAttribute />
</xs:complexType>
</xs:element>
<xs:attribute name="ano" type="xs:integer"/>
```

Aqui o elemento composto **carro** aceita a definição de um atributo qualquer, além de aceitar um elemento **marca**, depois um elemento **modelo**.

Mas lembre-se: este "outro atributo qualquer" deve estar definido em algum lugar no seu XML Schema para que você possa usá-lo. Por exemplo:

```
<veiculo ano="1977">
  <marca>Ferrari</marca>
  <modelo>308 GTS</modelo>
</veiculo>
```

VÁLIDO, pois o atributo **ano** foi definido no XML Schema.

```
<veiculo dono="Marcio">
  <marca>Ferrari</marca>
  <modelo>308 GTS</modelo>
</veiculo>
```

INVÁLIDO, pois o atributo **dono** não foi definido no XML Schema.

➢Final da Aula 7

Provas de TI
Tecnologia da Informação para Concursos

Na próxima aula veremos Resoluções de Questões envolvendo *XML Schema*!

Até a próxima aula!

XML para Concursos Públicos

Aula 08 – Resolução de Questões de *XML Schema*

➢Resolução de Questões

Questão 1 (CESGRANRIO - 2011 - FINEP - Analista - Desenvolvimento de Sistemas)

Um DTD (Document Type Definition) é um conjunto de regras usado para definir uma linguagem de marcação XML particular. Caso um documento XML não seja aderente às regras definidas em um DTD, ele não será um documento válido em relação a essa linguagem. Que outra tecnologia XML pode ser usada para definir a estrutura de um documento XML?

a) XQuery
b) XML Schema
c) XML Document Object Model
d) XPath
e) XSLT

➢Resolução de Questões

Questão 2 (CESPE - 2010 - TRT - 21ª Região (RN) - Analista Judiciário - Tecnologia da Informação)

Com relação a interoperabilidade de sistemas, SOA e web services, arquitetura e-ping e padrões XML, julgue o item seguinte.

A XSD - *XML schema definition* - permite definir elementos e atributos que podem aparecer em um documento XML, tal como um DTD (document type definition).

( ) CERTO ( ) ERRADO

➢Resolução de Questões

Questão 3 (CESPE - 2010 - Banco da Amazônia – Técnico Científico - Análise de Sistemas)

Com base na estrutura do documento XML apresentado ao lado, julgue os próximos itens.

```
<?xml version="1.0"?>
<livraria>
   <livro isbn="9788598078359">
      <autor id="064">Stephenie Meyer</autor>
      <titulo>Lua Nova</titulo>
      <ano>2008</ano>
      <preco>35.00</preco>
      <lancamento/>
   </livro>
   <livro isbn="9788599296554">
      <autor id="095">Dan Brown</autor>
      <titulo>O Símbolo Perdido</titulo>
      <ano>2009</ano>
      <preco>25.00</preco>
      <oferta/>
   </livro>
</livraria>
```

Usando-se um XML Schema, validam-se os metadados e os dados de um documento XML.

( ) CERTO ( ) ERRADO

➢Resolução de Questões

Questão 4 (CESPE - 2009 - TRE-MA – Técnico Judiciário - Programação de Sistemas)

No código ao lado, a primeira linha está sintaticamente incorreta. Nessa situação, assinale a opção que contém essa linha de código sintaticamente correta.

a) <“xml version=“1.0””>
b) <//xml version=“1.0”//>
c) <!’xml version=“1.0”’!>
d) <@xml version=“1.0”@>
e) <?xml version=“1.0”?>

```
<xml version="1.0">
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"
elementFormDefault="qualified">
<xs:element name="note">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="to" type="xs:string"/>
      <xs:element name="from" type="xs:string"/>
      <xs:element name="heading" type="xs:string"/>
      <xs:element name="body" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

➢Resolução de Questões

Questão 5 (CESPE - 2010 - TRE-MT - Técnico Judiciário - Programação de Sistemas)

A respeito de XML e XML Schema, assinale a opção correta.

a) No formato **`xmlns:xs`**, o XML Schema pode utilizar declaração de escopo de nomes (*namespace*).
b) O elemento < **`xml schema`** > é o elemento raiz de todos os esquemas definidos em XML Schema.
c) Elementos complexos em XML Schema não podem ser vazios, nem conter só texto. Devem conter sempre ao menos um outro elemento.
d) Em XML Schema, restrições são utilizadas para definir valores aceitáveis para atributos e não para elementos.
e) XML Schema oferece suporte a tipos de dados predefinidos, não permitindo a criação de novos tipos de dados.

➢Resolução de Questões

Questão 6 (ESAF - 2009 - ANA - Analista Administrativo - Tecnologia da Informação – Desenvolvimento)

Um tipo interno definido pelo XMLSchema é o

a) Name.
b) StringBuffer.
c) Object.
d) Decimal.
e) Address.

➢Resolução de Questões

Questão 7 (FGV - 2008 - Senado Federal - Analista de Sistemas)
Considere as figuras A e B a seguir.

A

```
<?xml version="1.0"?>
<OrdemdeCompra  xmlns="http:/xyz.org/oc.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xyz.org oc.xsd"
datacompra="20-12-2000">
    <enddestino  pais="BRASIL">
        <nome>Luis Potata</nome>
        <rua>Rua Torta 423</rua>
        <cidade>Cintra</cidade>
        <estado>SP</estado>
        <cep>90952-023</cep>
    </enddestino>
    <endpagamento pais=" BRASIL">
        <nome>Julia Pombal</ nome>
        <rua>Rua Silvano 30</rua>
        <cidade>Pirara</cidade>
        <estado>PA</estado>
        <cep>76889-043</cep>
    </endpagamento>
    <comentario>Esta compra é urgente!</comentario>
   (...)
</OrdemdeCompra>
```

B

```
<xs:schema xmlns:xs=http://www.w3.org/2001/XMLSchema
targetNamespace="http:/xyz.org/oc.xsd"
xmlns="http://xyz.org/oc.xsd"
elementFormDefault="qualified">
 <xs:element name="OrdemdeCompra"
type="TipoOrdemdeCompra"/>
 <xs:element name="comentario" type="xs:string"/>
 <xs:complexType name=" TipoOrdemdeCompra ">
  <xs:sequence>
   <xs:element name="enddestino"  type="endereco"/>
   <xs:element name="endpagamento"  type="endereco"/>
   <xs:element ref="comentario" minOccurs="0"/>
   <xs:element name="itens"  type="Itens"/>
  </xs:sequence>
  <xs:attribute name="datacompra" type="xs:date"/>
 </xs:complexType>
 <xs:complexType name="endereco">
  <xs:sequence>
   <xs:element name="nome"   type="xs:string"/>
   <xs:element name="rua" type="xs:string"/>
   <xs:element name="cidade"   type="xs:string"/>
   <xs:element name="estado"  type="xs:string"/>
   <xs:element name="cep"    type="xs:decimal"/>
  </xs:sequence>
  <xs:attribute name="pais" type="xs:NMTOKEN"
     fixed="BRASIL"/>
 </xs:complexType>
(...)
</xs:schema>
```

Onde (...) representa outros elementos aqui não apresentados.

➢Resolução de Questões

Em relação ao documento XML Schema é correto afirmar que:

a) se **`minOccurs="0"`** fosse removido da declaração do elemento **`comentario`**, então o documento XML deveria ter pelo menos uma ocorrência desses elementos.
b) o elemento **`comentario`** é um tipo simples por não ter um atributo **`type`** associado.
c) os elementos **`nome, rua, cidade, estado`** e **`cep`** não poderiam ser declarados diretamente como subelementos dos elementos **`endpagamento`** e **`enddestino`** em lugar da declaração através do type **`endereco.`**
d) a declaração deveria anteceder a declaração do tipo complexo **`TipoOrdemdeCompra`**.
e) no documento XML, os elementos em que **`minOccurs="0"`** não podem ter qualquer ocorrência.

➢Resolução de Questões

Questão 8 (FGV - 2008 - Senado Federal - Analista de Sistemas)
Considere as figuras A e B a seguir.

A

```
<?xml version="1.0"?>
<OrdemdeCompra  xmlns="http:/xyz.org/oc.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xyz.org oc.xsd"
datacompra="20-12-2000">
    <enddestino  pais="BRASIL">
        <nome>Luis Potata</nome>
        <rua>Rua Torta 423</rua>
        <cidade>Cintra</cidade>
        <estado>SP</estado>
        <cep>90952-023</cep>
    </enddestino>
    <endpagamento pais=" BRASIL">
        <nome>Julia Pombal</ nome>
        <rua>Rua Silvano 30</rua>
        <cidade>Pirara</cidade>
        <estado>PA</estado>
        <cep>76889-043</cep>
    </endpagamento>
    <comentario>Esta compra é urgente!</comentario>
   (...)
</OrdemdeCompra>
```

B

```
<xs:schema xmlns:xs=http://www.w3.org/2001/XMLSchema
targetNamespace="http:/xyz.org/oc.xsd"
xmlns="http://xyz.org/oc.xsd"
elementFormDefault="qualified">
 <xs:element name="OrdemdeCompra"
type="TipoOrdemdeCompra"/>
 <xs:element name="comentario" type="xs:string"/>
 <xs:complexType name=" TipoOrdemdeCompra ">
  <xs:sequence>
   <xs:element name="enddestino"  type="endereco"/>
   <xs:element name="endpagamento"  type="endereco"/>
   <xs:element ref="comentario" minOccurs="0"/>
   <xs:element name="itens"  type="Itens"/>
  </xs:sequence>
  <xs:attribute name="datacompra" type="xs:date"/>
 </xs:complexType>
 <xs:complexType name="endereco">
  <xs:sequence>
   <xs:element name="nome"   type="xs:string"/>
   <xs:element name="rua" type="xs:string"/>
   <xs:element name="cidade"   type="xs:string"/>
   <xs:element name="estado"  type="xs:string"/>
   <xs:element name="cep"    type="xs:decimal"/>
  </xs:sequence>
  <xs:attribute name="pais" type="xs:NMTOKEN"
     fixed="BRASIL"/>
 </xs:complexType>
(...)
</xs:schema>
```

Onde (...) representa outros elementos aqui não apresentados.

➢Resolução de Questões

Questão 8 (FGV - 2008 - Senado Federal - Analista de Sistemas)

Considerando AINDA as duas Figuras A e B,

É correto afirmar que, no documento XML:

a) o elemento **`rua`** poderia anteceder o elemento **`nome`** em **`enddestino`**.
b) o elemento **`comentario`** pode aparecer mais de uma vez.
c) **`http:/xyz.org/oc.xsd`** é o *namespace* padrão.
d) o atributo **`datacompra`** não deveria estar dentro da tag de abertura do elemento **`OrdemdeCompra`**.
e) os elementos **`enddestino`** e **`endpagamento`** não podem ter um atributo com mesmo nome.

➢Resolução de Questões

Questão 9 (FUNRIO - 2013 - MPOG - Analista de Tecnologia da Informação)

XML Schema é uma linguagem baseada no formato XML para definição de regras de validação ("esquemas") em documentos no formato XML, que provê recursos como namespaces e datatypes. Esta linguagem é uma alternativa ao

a) CSS.
b) SOAP.
c) JSP.
d) DOM.
e) DTD.

➢Final da Aula 8

GABARITO:

1 – Letra B

2 – CERTO

3 – CERTO

4 – Letra E

5 – Letra A

6 – Letra D

7 – Letra A

8 – Letra C

9 – Letra E

Até a próxima aula!

XML para Concursos Públicos

Aula 09 – *Namespaces*

➢O que são *Namespaces*?

Os ***Namespaces*** são recursos da linguagem XML a fim de possibilitar a existência de *tags* de mesmo nome num mesmo documento XML.

Considere os dois documentos XML a seguir:

```
<agenda>
  <contato id="1">
    <nome>Marcio</nome>
    <tel>1234-5678</tel>
  </contato>
</agenda>
```

```
<agenda>
  <reuniao id="1">
    <data>21/06/2013</data>
    <lugar>sala verde</lugar>
  </reuniao>
</agenda>
```

Os dois documentos possuem *tags* <agenda>. Se tivéssemos que adicionar os dois num documento único, haveria conflito dos nomes das *tags*. Uma forma de resolver isso, seria a inclusão de um "prefixo" nos nomes das *tags*.

Adicionando prefixos nas *tags*, poderíamos juntar os dois fragmentos XML:

```
<tel:agenda>
  <tel:contato id="1">
    <tel:nome>Marcio</tel:nome>
    <tel:tel>1234-5678</tel:tel>
  </tel:contato>
</tel:agenda>
```

```
<com:agenda>
  <com:reuniao id="1">
    <com:data>21/06/2013</com:data>
    <com:lugar>sala verde</com:lugar>
  </com:reuniao>
</com:agenda>
```

Estes dois prefixos **`tel`** e **`com`** chamamos de ***Namespaces***.

➢Como definir *Namespaces*

Para definirmos uma Namespace, usamos a sintaxe a seguir:

`xmlns:`*`prefixo`*`=“URI”`

onde:

`prefixo` é o nome do prefixo que se deseja associar ao *namespace*. O recomendado é usar um prefixo bem sucinto e de fácil entendimento, geralmente uma sigla.

`URI` é sigla para *Uniform Resource Identifier*, ou seja, é um texto que identifica uma origem de um determinado recurso na Internet. Falaremos de *URI* mais adiante.

Exemplo de definição de *Namespace*:

```
<documento>
<tel:agenda
   xmlns:tel=“http://www.primmer.com.br/agenda”>
  <tel:contato id=“1”>
    <tel:nome>Marcio</tel:nome>
    <tel:tel>1234-5678</tel:tel>
  </tel:contato>
</tel:agenda>
</documento>
```

É importante notar que ao definirmos o *namespace* num elemento, todos os elementos abaixo dele já ficam associados a ele.

Lembre-se que o **`xmlns`** deve ser um atributo de um elemento no documento XML.

➢Mas o que é um *URI*?

Ao trabalharmos com *Namespaces*, usamos o conceito de *URI*.

A URI que usamos nas *Namespaces* **geralmente** aponta para um endereço na Internet contendo informações sobre esta *Namespace*. Porém, não é necessário que este endereço realmente exista na Internet ou Intranet.

Mas o conceito de URI é mais amplo. Ela define uma forma de identificar um recurso qualquer na Internet, seja uma imagem, um site, um vídeo etc.

Existem basicamente dois tipos de URI, que são:

URL (***Uniform Resource Locator***): é o endereço e o método para localizar o recurso. Por exemplo: http://www.w3.org/html/
É o link para acessar as definições da linguagem HTML. E devemos usar o protocolo HTTP.

URN (***Uniform Resource Name***): é o nome usado para identificar o recurso. Por exemplo: "HTML4.0"
É um nome único que usamos para identificar a versão 4.0 da linguagem HTML.

Ou seja, enquanto a URN simplesmente dá um nome a um recurso, a URL nos diz onde e como encontrá-lo. Mas não se preocupe com URN, pois raramente são usadas, até mesmo em provas de Concursos Públicos!

➢ *Namespaces* padrão

Há opção de usarmos uma *namespace* padrão numa parte do documento XML. Para isso, definimos a *namespace* com o atributo **`xmlns`** no atributo pai, e todos os elementos filhos deste não precisam usar o prefixo. Por exemplo:

```
<documento xmlns="padrao.com.br/doc">
  <autor>Marcio</autor>
  <texto>Aqui é um texto</texto>
</documento>
```

COM *namespace* padrão

```
<doc:documento xmlns:doc="padrao.com.br/doc">
  <doc:autor>Marcio</doc:autor>
  <doc:texto>Aqui é um texto</doc:texto>
</doc:documento>
```

SEM *namespace* padrão

No primeiro exemplo, usamos uma *namespace* padrão. No segundo exemplo, usamos uma *namespace* comum com seu prefixo. Os dois exemplos representam o mesmo conteúdo, só que o primeiro exemplo é mais enxuto, pois não precisamos escrever os prefixos em todas as *tags*, apenas na *tag* pai.

Outro exemplo interessante: a mistura entre *namespace* padrão e as definidas em prefixos.

```
<documento xmlns="padrao.com.br/doc"
    xmlns:meu="meupadrao.com.br/doc">
  <autor>Marcio</autor>
  <meu:texto>Aqui é um texto</meu:texto>
</documento>
```

Os elementos **`documento`** e **`autor`** pertencem ao *namespace* padrão "`padrao.com.br`". Mas o elemento **`texto`** pertence ao *namespace* "`meupadrao.com.br`".

➢ *Namespaces* padrão em atributos

IMPORTANTE: *Namespaces* padrão NÃO se aplicam a atributos!

Se você declarar uma namespace padrão em seu documento XML, e caso queira que um atributo esteja relacionado a esta *namespace*, você obrigatoriamente deve usar o prefixo correspondente.

No exemplo a seguir temos dois atributos, um deles associado a *namespace* padrão, e outro não.

```
<documento
  xmlns="padraoB.com.br"
  xmlns:padB="padraoB.com.br"
  idioma="portuguesBR"
  padB:versao="1">
  <cabecalho>Estudo de Namespaces</cabecalho>
  <autor>Marcio</autor>
</documento>
```

O atributo **`idioma`** não está associado a nenhum *namespace*, já este atributo não possui um prefixo.

Já o atributo **`versao`**, está associado ao *namespace* "padraoB.com.br", pois este atributo possui o prefixo **`padB`**.

➢ *Namespaces* de destino

Ao montarmos um *XML Schema*, podemos criar novas *namespaces*, onde as nossas regras de validação irão ficar armazenadas. Para definir tais *namespaces*, usamos o atributo **`targetNamespace`** no elemento raiz **`xs:schema`**.


```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://padrao.com.br/Regras1">
  <xs:element name="documento">
    <xs:sequence>
      <xs:element name="autor" type="xs:string"/>
      <xs:element name="titulo" type="xs:string"/>
    <xs:sequence>
  </xs:element>
</xs:schema>
```

Neste exemplo, todas as regras de validação para o elemento **`documento`**, ficarão armazenadas na *namespace* "**http://padrao.com.br/Regras1**".

➢Usando *Namespaces* de destino

Para usarmos a *namespace* que criamos no slide anterior, podemos usar o elemento **`xs:import`**.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:import schemaLocation="padrao1.xsd" namespace="http://padrao.com.br/Regras1"/>
  <xs:element name="gabinete">
    <xs:sequence>
      <xs:element name="autor" type="xs:string"/>
      <xs:element name="titulo" type="xs:string"/>
    <xs:sequence>
  </xs:element>
</xs:schema>
```

No elemento **`xs:import`** usamos o atributo **`schemaLocation`** para apontar para um arquivo onde os elementos do esquema importado se encontram.

E além disso, usamos o atributo **`namespace`** no elemento **`xs:import`** para justamente qualificar o *namespace* que queremos usar.

➢Qualificação do Elemento/Atributo

Podemos controlar associação de elementos e atributos a *namespaces* através dos atributos **`elementFormDefault`** e **`attributeFormDefault`**. Eles podem assumir dois possíveis valores: `qualified` e `unqualified`.

Quando **`elementFormDefault`** for igual a `qualified`, significa que todos os elementos locais e globais PRECISAM ser qualificados, ou seja, pertencerem a um *namespace*.

ATENÇÃO: Isso quer dizer que se você NÃO estiver usando uma *namespace* padrão, então todos os elementos PRECISAM usar prefixos!

Quando **`elementFormDefault`** for igual a `unqualified`, significa que os elementos globais PRECISAM estar associados a *namespaces*, mas não os locais.

Agora, no caso dos atributos, usamos **`attributeFormDefault`**. de forma bem mais simples:

Quando **`attributeFormDefault`** for igual a `qualified`, significa que todos os atributos, sejam eles locais ou globais, PRECISAM estar associados a *namespaces*.

Quando **`attributeFormDefault`** for igual a `unqualified`, significa que todos os atributos, sejam eles locais ou globais, não PRECISAM estar associados a *namespaces*.

➢Resolução de Questões

Questão 1 (CESPE - 2009 - INMETRO - Analista Executivo - Desenvolvimento de Sistemas)
A figura abaixo apresenta o conteúdo parcial de um documento XML. Acerca das informações apresentadas, dos conceitos de GED, XML e das tecnologias relacionadas, julgue os seguintes itens.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/dc/elements/1.1/">

  <rdf:Description rdf:about="http://www.inmetro.gov.br">
    <dc:title>Inmetro - Instituto Nacional de Metrologia, Nor
    <dc:creator>Inmetro</dc:creator>
    <dc:subject>
      acessibilidade; acreditação; alimentos; análise; arranj
      certificadas; científica; conformidade; consumo; creden
      instrumentos; ISO; laboratórios; marcas; metrologia; no
    </dc:subject>
    <dc:description>
      O site do Instituto Nacional de Metrologia, Normalizaçã
      tem por objetivo disseminar as informações sobre o perf
      e os programas de trabalho produzidos por suas diversas
    </dc:description>
    <dc:date>2001-01-20</dc:date>
    <dc:type>InteractiveResource</dc:type>
    <dc:format>text/html</dc:format>
  </rdf:Description>
</rdf:RDF>
```

Os valores **http://www.w3.org/1999/02/22-rdf-syntax-ns#** e **http://purl.org/dc/elements/1.1/**, empregados nas segunda e terceira linhas do documento, são mais precisamente chamados de URLs, não de URIs.

( ) CERTO ( ) ERRADO

➢Resolução de Questões

Questão 2 (CESPE - 2009 - INMETRO - Analista Executivo - Desenvolvimento de Sistemas)
A figura abaixo apresenta o conteúdo parcial de um documento XML. Acerca das informações apresentadas, dos conceitos de GED, XML e das tecnologias relacionadas, julgue os seguintes itens.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/dc/elements/1.1/">

  <rdf:Description rdf:about="http://www.inmetro.gov.br">
    <dc:title>Inmetro - Instituto Nacional de Metrologia, Nor
    <dc:creator>Inmetro</dc:creator>
    <dc:subject>
      acessibilidade; acreditação; alimentos; análise; arranj
      certificadas; científica; conformidade; consumo; creden
      instrumentos; ISO; laboratórios; marcas; metrologia; nc
    </dc:subject>
    <dc:description>
      O site do Instituto Nacional de Metrologia, Normalizaçã
      tem por objetivo disseminar as informações sobre o perf
      e os programas de trabalho produzidos por suas diversas
    </dc:description>
    <dc:date>2001-01-20</dc:date>
    <dc:type>InteractiveResource</dc:type>
    <dc:format>text/html</dc:format>
  </rdf:Description>
</rdf:RDF>
```

O documento é um registro de metadado.

( ) CERTO ( ) ERRADO

➢Resolução de Questões

Questão 3 (CESPE - 2009 - INMETRO - Analista Executivo - Desenvolvimento de Sistemas)
A figura abaixo apresenta o conteúdo parcial de um documento XML. Acerca das informações apresentadas, dos conceitos de GED, XML e das tecnologias relacionadas, julgue os seguintes itens.

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:dc="http://purl.org/dc/elements/1.1/">

  <rdf:Description rdf:about="http://www.inmetro.gov.br">
    <dc:title>Inmetro - Instituto Nacional de Metrologia, Nor
    <dc:creator>Inmetro</dc:creator>
    <dc:subject>
      acessibilidade; acreditação; alimentos; análise; arranj
      certificadas; científica; conformidade; consumo; creden
      instrumentos; ISO; laboratórios; marcas; metrologia; nc
    </dc:subject>
    <dc:description>
      O site do Instituto Nacional de Metrologia, Normalizaçã
      tem por objetivo disseminar as informações sobre o perf
      e os programas de trabalho produzidos por suas diversas
    </dc:description>
    <dc:date>2001-01-20</dc:date>
    <dc:type>InteractiveResource</dc:type>
    <dc:format>text/html</dc:format>
  </rdf:Description>
</rdf:RDF>
```

Três *namespaces* são declarados no documento.

( ) CERTO ( ) ERRADO

➢Resolução de Questões

Questão 4 (FGV - 2008 - Senado Federal - Analista de Sistemas)

Considere as seguintes afirmativas sobre um documento XML bem formado:

I. Deve estar sintaticamente correto, seguindo as regras de marcação prescritas para o padrão XML.
II. Deve conter um elemento raiz e pelo menos algum outro elemento.
III. Deve conter uma associação com um documento *XMLSchema* ou uma DTD.
IV. Deve fazer uso de pelo menos um *namespace*.

Estão incorretas as afirmativas:
a) I e II, apenas.
b) III e IV, apenas.
c) II, III e IV, apenas.
d) I, II e IV, apenas.
e) I, II , III e IV.

➢Final da Aula 9

GABARITO:
1 – ERRADO
2 – CERTO
3 – ERRADO
4 – Letra C

Obrigado! Nos vemos em outra videoaula!