**MAC0426 - Sistemas de Bancos de Dados**
**IBI5013 - Bancos de Dados para Bioinformática**

# Aula 26
# Introdução a
# Dados Semiestruturados

Profa. Kelly Rosa Braghetto

17 de junho de 2016

# Dados não estruturados

- Podem ser de qualquer tipo
- Não seguem necessariamente um formato ou sequência
- Não seguem regras
- Não são previsíveis
- **Exemplos:** texto, vídeo, som, imagens ...

# Dados estruturados

- São organizados em blocos semânticos (entidades)
- Entidades similares são mantidas de forma agrupada (relações ou classes)
- Entidades de um mesmo grupo possuem as mesmas descrições (atributos)
- As descrições para todas as entidades de um grupo (esquema) possuem o mesmo formato, o mesmo tamanho, estão todas presentes, e seguem a mesma ordem

# Dados semiestruturados

- Os dados são organizados como entidades semânticas
- Entidades similares são mantidas de forma agrupada
- Entidades em um mesmo grupo podem não ter os mesmos atributos
- A ordem dos atributos não é necessariamente importante
- Nem todos os atributos são obrigatórios
- O tamanho e o tipo de um atributo pode variar dentro de um mesmo grupo

# Dados para acesso eletrônico

- Dados mantidos em um **SGBDR** são **dados estruturados** – possuem uma estrutura de representação (= esquema) rígida, previamente projetada
- Boa parte dos dados disponíveis para acesso eletrônico **não estão mantidos em BDs em SGBDRs**
- Principal exemplo: **dados da web**, que geralmente possuem uma organização bastante heterogênea
  - mistura textos sem nenhuma informação com conjuntos de registros bem formatados
  - grande volume
  - muitos relacionamentos

# Dados heterogêneos

- A heterogeneidade dificulta as operações de consulta aos dados
- Não há um esquema uniforme, a partir do qual as consultas possam ser formuladas
- Isso implica na necessidade de se realizar buscas de "alto custo"
  - buscas exaustivas nos dados
  - buscas por palavras-chaves (por meio de técnicas de recuperação de informação)
- Dados como esses da web são dados **semiestruturados**

# Modelos de dados semiestruturados

- São capazes de representar tanto dados bastante estruturados quanto dados sem estruturação alguma
- São capazes de representar dados irregulares; algumas ocorrências de dados podem possuir informações incompletas ou complementares com relação a outras
- Modelo de dados relacional – há distinção clara entre o tipo dos dados (= esquema do BD) e os dados propriamente ditos (= instâncias).
- Modelo de dados semiestruturado – separação não é tão clara; **os dados são autodescritivos** – a informação do esquema está misturada com valores dos dados

# Dados semiestruturados e a Web

- O modelo de dados semiestruturado é um padrão para a representação e troca de dados na web
- Ele trouxe melhorias importantes para a publicação e reúso de dados eletrônicos, por prover uma **sintaxe simples para os dados** que é, ao mesmo, **facilmente processável por máquinas** e **legível aos usuários**
- Além disso, a flexibilidade da tipagem em dados semiestruturados se tornou essencial para a integração de dados, especialmente na integração de dados heterogêneos por meio de sistemas mediadores

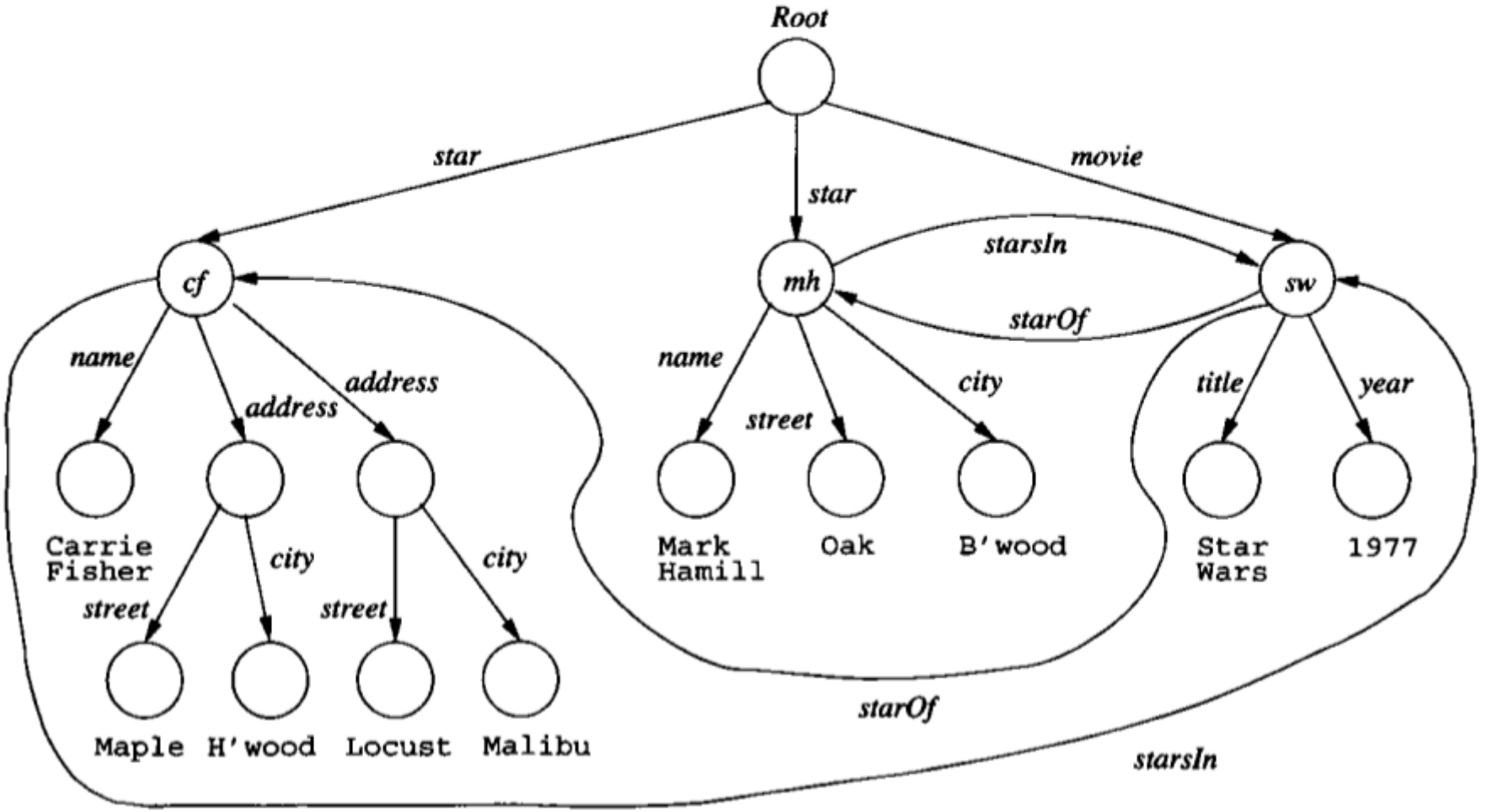
# Resumo das diferenças entre dados estruturados e dados semiestruturados

| Dados estruturados | Dados semi-estruturados |
|---|---|
| Esquema pré-definido | Nem sempre há um esquema pré-definido |
| Estrutura regular | Estrutura irregular (em nível tanto de atributos quanto de tipos) |
| Estrutura independente dos dados | Estrutura embutida nos dados |
| Estrutura reduzida | Estrutura extensa (para refletir as particularidades de cada dado, já que cada dado pode ter uma organização própria) |
| Estrutura fracamente evolutiva | Estrutura fortemente evolutiva (a estrutura dos dados modifica-se tão frequentemente quanto os seus valores) |
| Estrutura prescritiva (que permite definir esquemas fechados e restrições de integridades com relação à semântica dos atributos) | Estrutura descritiva |
| Distinção entre estrutura e dado é clara | Distinção entre estrutura e dado não é clara |

# Modelagem de dados semiestruturados

- Modelos de dados para BDs tradicionais não são os mais apropriados para a representação de dados semiestruturados – neles, as ocorrências de dados devem apresentar uma mesma estrutura
- Modelos para dados semiestruturados são mais flexíveis – suportam representações heterogêneas de dados semanticamente iguais
- Geralmente, os modelos propostos para dados semiestruturados representam os dados como algum tipo de **grafo direcionado rotulado**

# Modelo de grafo de dados semiestruturados




**Exemplo de representação de dados semiestruturados por meio de grafo (dados de um filme e duas de suas estrelas)**
**Extraído de [3].**

# Modelo de grafo de dados semiestruturados

- **Nós do grafo**
  - **Internos:** objetos compostos
  - **Folhas:** têm algum valor atômico associado (como números ou strings)
- **Arcos –** relacionamentos objeto → subobjeto ou objeto → valor
- **Rótulo** em um arco – indica como o objeto de origem do arco se relaciona com o objeto destino
- Não há restrições com relação ao número de arcos que partem de um objeto origem – cada ocorrência de dado pode ter uma estrutura diferente
- Todo BD semiestruturado tem um **objeto raiz**, que é ponto de partida para a investigação da sua estrutura

# Modelo de grafo de dados semiestruturados

- Os rótulos dos arcos desempenham dois papéis. Suponha que temos um arco rotulado com *L* do nó *N* para o nó *M*:
  - É possível pensar que N representa um objeto ou estrutura, e que M é um atributo do objeto ou um campo da estrutura. Assim, L representa o nome de um atributo ou campo, respectivamente.
    - Ex: *name* e *address* para *star*
  - Também é possível pensar que N e M são objetos, e que L é o nome de um relacionamento de N para M.
    - Ex: *starIn* e *startOf* entre *star* e *movie*

# XML

# *(Extensible Markup Language)*

# XML (*Extensible Markup Language*)

- É o modelo de dados semiestruturados mais bem sucedido
- É um padrão para a publicação, combinação e intercâmbio de documentos multimídia, desenvolvido pelo consórcio W3C, com base em linguagens mais antigas, como a SGML e a HTML
- Por ser uma linguagem de marcação, XML lida com instruções embutidas no corpo do documentos chamadas *tags* (marcas), que permitem a descrição de dados
- A diferença principal entre a HTML e a XML é que na HTML o conjunto de *tags* é fixo (body, table, p, etc.), enquanto que na XML pode-se usar quaisquer marcas que se queira
  - Isso dá à linguagem uma enorme flexibilidade de representação, o que permite que ela seja usada no desenvolvimento de aplicações em diversos contextos.

# XML (*Extensible Markup Language*)

- Um documento XML bem formado é constituído basicamente por uma **sequência de elementos** que englobam **valores de texto** e **outros elementos**
- Elementos são identificados num documento XML por meio de *tags*
- Um elemento possui uma marca de início (**< *[nome do elemento]* >**) e uma marca de fim ( **</[nome do elemento]>** ).

  Tudo o que aparece entre essas duas marcas é o conteúdo do elemento
- Exemplo: **<aluno> Ana Clara Machado </aluno>**

# XML (*Extensible Markup Language*)

- Um **elemento complexo** é construído hierarquicamente a partir de outros elementos
- Um **elemento atômico** contém um valor de dado
- Um elemento pode possuir **atributos**, que são especificados dentro de sua marca inicial e possuem um valor informado entre aspas
- **Ex: <aluno nusp=”123456”>**
- Os atributos de XML geralmente são usados para descrever propriedades e características dos elementos dentro dos quais eles aparecem

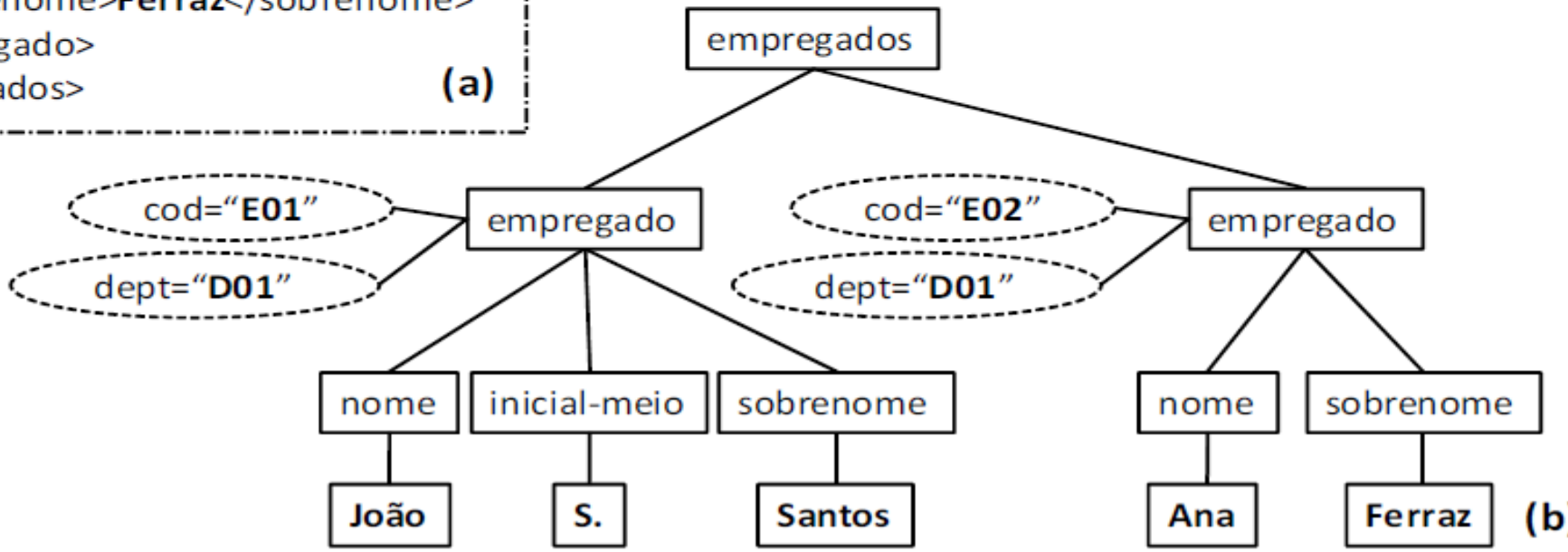
# XML (*Extensible Markup Language*)

- Os nomes das *tags* no XML são usados para descrever o significado dos elementos de dados
- Um **documento XML** é representado por uma estrutura em **árvore com rotulação nos nós**
- Nós – elementos, atributos ou valores texto
- Arcos – relações de elemento/subelemento, ou elemento/valor
- Esse modelo de representação é chamado de **modelo de árvore** ou **modelo hierárquico**
- Um BD XML é geralmente modelado como uma **floresta de árvores** (uma para cada documento)

# Documento XML e seu modelo de árvore

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

(a)




(b)

# Documentos XML bem formados

**Um documento XML  bem formado (= sintaticamente correto) respeita as seguintes regras:**

- Começa como uma declaração XML para indicar a versão de XML utilizada e outros atributos pertinentes
- Segue as diretrizes sintáticas do modelo de árvore que são:
  - possuir um único elemento raiz
  - cada elemento precisa incluir um par correspondente de *tags* de início e fim entre os *tags* de início e fim do elemento paterno (para assegurar um aninhamento de elementos correto, que especifica uma estrutura de árvore bem formada)
- Não conter atributos repetidos num mesmo elemento

# Documentos XML válidos

- Um documento XML pode estar associado a um esquema
- Um documento XML é dito **válido** se ele segue as regras definidas no esquema associado a ele
- Existem duas alternativas para a representação de esquemas para documentos XML:
  - ***Data Type Definition*** **(DTD)**
  - ***XML Schema***

# *Data Type Definition* (DTD)

- Por meio da DTD, é possível definir regras de formação de elementos
- Ela nos permite definir quais elementos podem ou devem aparecer no documento, suas cardinalidades e a ordem em que devem aparecer
- Na DTD, **não é possível** especificar os **tipos dos elementos atômicos** (como inteiros, datas, etc)
  - Geralmente, os elementos atômicos são tratados como strings (o tipo #PCDATA – *parsed character data*)

# *Data Type Definition* (DTD)

- Em DTD,  as seguintes cardinalidades são possíveis:
  - “+” (para 1 ou mais elementos)
  - “*” (para 0 ou mais elementos)
  - “?” (para 0 ou 1 elemento)
- Se um elemento não tem símbolo de cardinalidade associado, então ele é **obrigatório**
- **Deficiência:** não é possível definir de forma explícita um número mínimo e máximo de elementos

# *Data Type Definition* (DTD) – Exemplo

- Arquivo “emps.dtd”

```
<!ELEMENT empregados (empregado+)>
<!ELEMENT empregado (nome, inicial-meio?, sobrenome)>
<!ATTLIST empregado
    cod CDATA #required
    dept CDATA #required
>
<!ELEMENT nome (#PCDATA)>
<!ELEMENT inicial-meio (#PCDATA)>
<!ELEMENT sobrenome (#PCDATA)>
```

# *Data Type Definition* (DTD) – Exemplo

- Arquivo “empregados.xml”, válido sob o esquema “emps.dtd”

```
<?xml version='1.0'?>
<!DOCTYPE empregados SYSTEM 'emps.dtd'>
<empregados>
    <empregado cod='E01' dept='D01'>
        <nome>João</nome>
        <inicial-meio>S.</inicial-meio>
        <sobrenome>Santos</sobrenome>
    </empregado>
    <empregado cod='E02' dept='D01'>
        <nome>Ana</nome>
        <sobrenome>Ferraz</sobrenome>
    </empregado>
</empregados>
```

# *Data Type Definition* (DTD) – Deficiências

- Os tipos de dados em DTD não são muito gerais
- Um documento em DTD tem uma sintaxe especial (diferente da XML) e requer processadores (*parsers*) especializados
- Todos os elementos DTD são forçados a seguir a ordenação especificada no documento (portanto, elementos não ordenados não são permitidos)

# *XML Schema*

- É a linguagem padrão para especificar a estrutura de documentos XML
- A XML Schema é baseada na própria XML, ou seja, um esquema em XML Schema é também um documento XML
- Ela é bem mais expressiva que a DTD, permitindo definir esquemas mais elaborados
- Em XML Schema, cada elemento é associado a um tipo, que pode ser **simples** ou **complexo**
- Existem **tipos simples predefinidos** (como *integer*, *boolean*, *date*, etc.), mas outros tipos podem ser definidos por meio de **restrições** sobre tipos existentes

# XML Schema – Exemplo (parte 1)

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="empregados" type="tEmpregados"/>

    <xs:complexType name="tEmpregados">
        <xs:sequence>
            <xs:element name="empregado"
type="tEmpregado"
                minOccurs="1" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
```

# XML Schema – Exemplo (parte 2)

```
    <xs:complexType name="tEmpregado">
        <xs:sequence>
            <xs:element name="nome" type="xs:string"/>
            <xs:element name="inicial-meio" type="xs:string"
                minOccurs="0"/>
            <xs:element name="sobrenome" type="xs:string"/>
        </xs:sequence>
        <xs:attribute name="cod" type="xs:string" use="required"/>
        <xs:attribute name="dept" type="xs:string" use="optional"/>
    </xs:complexType>
</xs:schema>
```

# XML Schema

- Na definição de um tipo complexo, os subelementos são declarados dentro do tipo complexo
- É preciso escolher um dos seguintes tipos de restrições sobre o conjunto fixo de subelementos de um tipo complexo:
  - ***Sequence*** – estabelece que todos os elementos pertencentes ao grupo devem aparecer na ordem em que foram definidos e nenhum pode ser omitido
  - ***Choice*** – estabelece que apenas um dos elementos pertencentes ao grupo deve aparecer em uma instância XML
  - ***All*** – diz que os elementos podem aparecer em qualquer ordem e podem ser repetidos ou omitidos

# *XML Schema*

- A cardinalidade de um elemento pode ser definida de forma explícita, por meio dos atributos ***maxOccurs*** e ***minOccurs***
- Quando omitida, a cardinalidade de um objeto é min=1 e max=1
- XML Schema permite também definir **unicidade**, **chaves** e **referências a chaves**

# APIs para a Manipulação de Documentos XML

- As bibliotecas que são capazes de manipular documentos XML possuem um processador (= *parser*) XML, que é responsável por disponibilizar o conteúdo do documento para a aplicação
- Esses processadores também são capazes de detectar problemas nos documentos (como má-formação ou documentos inválidos)
- Existem basicamente dois tipos de processadores XML
  - os que fornecem à aplicação a árvore do documento XML
  - os que disparam eventos para a aplicação
- Em ambos os casos, a aplicação deve se comunicar com o processador por meio de uma API

# APIs para a Manipulação de Documentos XML

- As duas principais para manipulação de XML são:
  - **DOM** (*Document Object Model*)
  - **SAX** (*Simple API for XML*)
- **DOM** – disponibiliza métodos para manipular a árvore XML em memória e manipula um documento como um todo
- **SAX** – funciona baseada em eventos; manipula cada parte do documento sequencialmente

# Linguagens de consulta XML

Dentre as várias propostas de linguagens de consulta para a XML existentes, dois padrões (recomendados pelo W3C) se destacaram:

- **XPath** (*XML Path Language*) – possui construções para a especificação de expressões de caminho (como as empregadas em sistemas de arquivos), de modo a possibilitar a “navegação” pelos elementos e atributos de um documento XML
- **XQuery** – é uma linguagem de consulta mais geral (está para XML assim como SQL está para um BD relacional)

# XPath

- Uma expressão XPath geralmente retorna uma sequência de itens que satisfazem o padrão especificado na expressão
- Os itens podem ser valores (nós folha na árvore), elementos ou atributos
- O nomes em uma expressão XPath são nomes de elementos ou de atributos do documento XML
- As expressões podem conter também condições qualificadoras (= filtros), que restringem ainda mais os nós que satisfazem o padrão

# XPath

- Os principais operadores da XPath são "***/***" e "***//***"
- "***/***" serve para "dar um passo" na árvore XML (percorrer um relacionamento pai-filho)
- "***//***" serve para pular vários níveis de uma só vez (relacionamento ascendente-descendente)
- O resultado de cada expressão XPath é um conjunto de itens especificados pelo caminho
- Os itens em um documento XML são ordenados; os itens do resultado de uma expressão XPath são devolvidos de acordo com a sua ordenação no documento XML

# XML de Exemplo

- Vamos considerar o seguinte esquema DTD “emps.dtd”:

```
<!ELEMENT empregados (empregado+)>
<!ELEMENT empregado (nome, inicial-meio?, sobrenome)>
<!ATTLIST empregado
    cod CDATA #required
    dept CDATA #required
>
<!ELEMENT nome (#PCDATA)>
<!ELEMENT inicial-meio (#PCDATA)>
<!ELEMENT sobrenome (#PCDATA)>
```

# XML de Exemplo

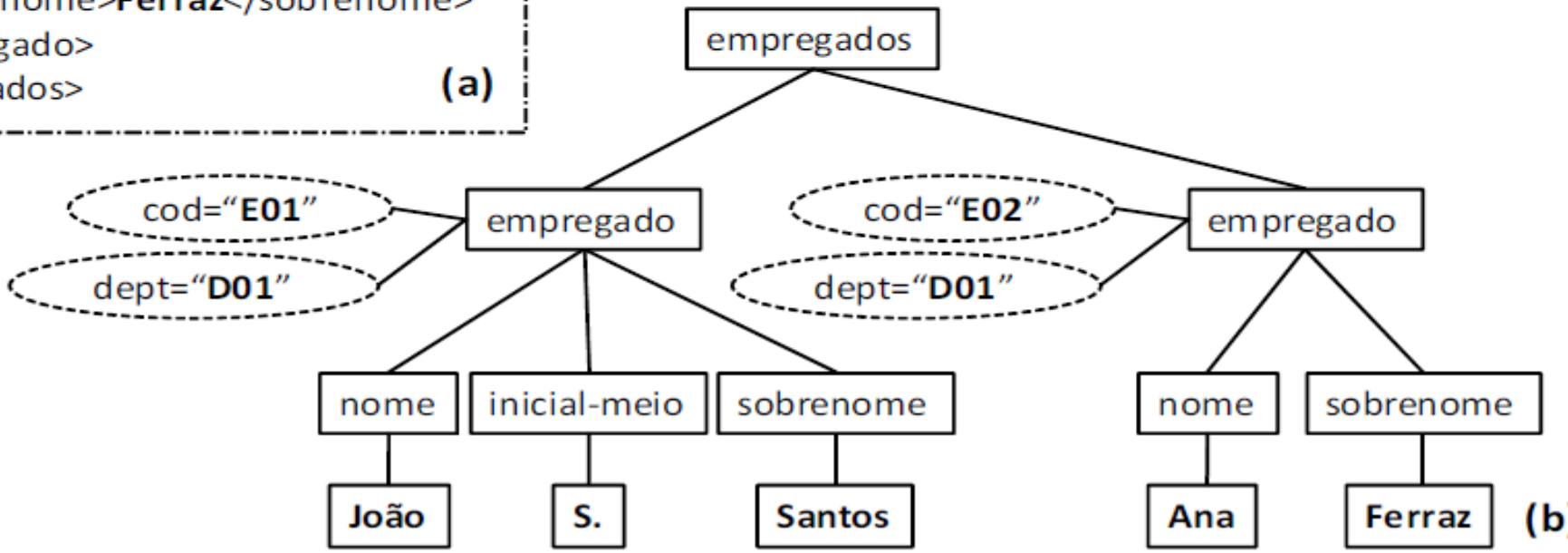
- Vamos considerar o documento “emps.xml”, válido sob o esquema “emps.dtd”

```
<?xml version='1.0'?>
<!DOCTYPE empregados SYSTEM 'emps.dtd'>
<empregados>
    <empregado cod='E01' dept='D01'>
        <nome>João</nome>
        <inicial-meio>S.</inicial-meio>
        <sobrenome>Santos</sobrenome>
    </empregado>
    <empregado cod='E02' dept='D01'>
        <nome>Ana</nome>
        <sobrenome>Ferraz</sobrenome>
    </empregado>
</empregados>
```

# XML de Exemplo

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

(a)




(b)

# XPath – Exemplo de consultas sobre “emps.xml”

**/empregados/empregado**

- Essa expressão seleciona os dois elementos **empregado** do documento
- Cada "/" muda o contexto atual da consulta
- O primeiro "/" coloca o contexto na raiz do documento e caminha para ela (**empregados**)
- O segundo "/" caminha para os filhos **empregado** do contexto anterior (**empregados**)

**//empregado**

- Também seleciona os dois elementos **empregado** do documento
- Semântica: “retorne os elementos **empregado** que estejam em qualquer profundidade no documento XML”

# XPath – Exemplo de consultas sobre “emps.xml”

**/empregados/.**

- Equivale a **/empregados** ; retorna o elemento **empregados**
- “**.**” referencia o elemento corrente

**//empregado/..**

- Seleciona o pai do elemento **empregado**, que é o elemento **empregados**
- “**..**” seleciona o pai do elemento do contexto atual

**//empregado/***

- Seleciona os “filhos” dos elementos **empregado** (**nome**, **inicial-meio** e **sobrenome**)
- “*****” é o símbolo curinga, substitui o nome de um elemento em uma expressão de caminho

# XPath – Exemplo de consultas sobre “emps.xml”

**//empregado/@cod**

- Retorna o atributo **cod** dos empregados armazenados no documento XML
- Para acessar um atributo, usa-se "@" na frente de seu nome (para diferenciá-lo do nome de um elemento)

**//empregado[@cod='E01']**

- Retorna o elemento **empregado** cujo código é 'E01'
- Expressões booleanas colocadas entre colchetes funcionam como **filtros**

**//empregado[@cod='E01']/nome**

- Retorna o **nome** do empregado cujo código é 'E01'
- Um filtro não altera o contexto atual, ou seja, depois dele a expressão pode continuar do ponto onde havia parado antes do filtro (como no último exemplo)

# XPath – Exemplo de consultas sobre “emps.xml”

**//empregado[position() = 1]**

- Seleciona o primeiro empregado
- Esse tipo de filtro é chamado de **filtro de posição**
- Forma abreviada: ***//empregado[1]***

**//empregado[@dept='D01' AND nome='João']**

- Seleciona os empregados que trabalham no departamento 'D01' e que possuem nome 'João'
- É possível usar operadores lógicos AND, OR e NOT dentro dos filtros

# XPath – Operadores e funções

- XPath possui vários operadores e funções que podem ser usados nos filtros:
  - Operadores de comparação (=, !=, <, <=, >, >=)
  - Operadores aritméticos (+, -, *, div, mod, etc.)
  - Funções de manipulação de *strings* (*starts-with*, etc.)
  - ...

  Exemplo: ***//empregado[starts-with(nome,'J')]***
- Retorna os elementos **empregado** cujo elemento **nome** começa com 'J'

# XQuery

- A XQuery é uma linguagem poderosa, capaz de gerar:
  - respostas com uma estrutura diferente da do documento consultado
  - texto puro
  - fragmentos de documentos XML
- Expressões XPath podem ser usadas dentro de consultas XQuery
- XQuery possui construtores de elementos, além de operadores mais complexos como os do tipo “FLWOR”

# XQuery – Construtores de elementos

- Construtores de elementos permitem estruturar a resposta a uma consulta em elementos não contidos no documento original

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

- Na consulta acima, há dois construtores de elementos: `<emp-dept>` e `$e/nome`

# XQuery – Construtores de elementos

- Sobre os construtores do exemplo anterior:
  - O primeiro cria no resultado uma marca **<emp-dept>** que não existe no documento original
  - O **for** usa a variável **$e** para iterar sobre todos os empregados do documento. Para cada empregado, a expressão **return** é executada e **$e/nome** constrói no resultado um elemento com o nome e o conteúdo do elemento **nome** do documento consultado
- A chave (“{”) indica o início de um trecho de consulta que precisa ser processado

# XQuery – Construtores de elementos

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

- O resultado da consulta é:

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

# XQuery – Predicados do iterador **for**

- Cláusulas **for** podem ter predicados de seleção (**where**) e ordenação (**order by**)

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

# XQuery – Consultas aninhadas correlacionadas

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

# XQuery – Consultas aninhadas correlacionadas

- Na consulta do slide anterior temos alguns novos conceitos:
  - A função ***distinct-values***, que permite iterar apenas sobre valores distintos (ou seja, ignora as repetições)
  - Dentro da primeira cláusula ***return***, existe uma consulta aninhada, com uma outra cláusula ***for***
    - Essa consulta seleciona empregados relacionados ao departamento da primeira consulta (portanto, trata-se de uma consulta **aninhada correlacionada**)

# Xquery – Consulta aninhada (resultado)

- O resultado da consulta aninhada do slide 27 é:

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

# XQuery – Exemplo

- Considere o seguinte arquivo “depts.xml”:

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

# Xquery – Consultas com junção

- O exemplo a seguir mostra uma consulta com junção:

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

# Xquery – Consultas com junção

- O resultado da consulta com junção do slide anterior é:

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

# XQuery – Operações de agregação

- XQuery também é capaz de realizar as seguintes operações de agregação: **count, sum, avg, min** e **max**
- Exemplo:

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

- Observe que o exemplo não usa o iterador **for**, mas sim a expressão **let**, que atribui à uma variável um conjunto de elementos

# XQuery – Expressões FLWOR

- As expressões **let** e **for** são parte das expressões **FLWOR** (que podem ser usadas em conjunto):
  - **for**, **let**, **where**, **order by**, **return**

- XQuery também possui:
  - expressões condicionais (**if-then-else**)
  - quantificadores existencial e universal (**some** e **every**),
  - *cast* de tipos

# Referências Bibliográficas


- [1] “Desmistificando XML: da Pesquisa à Prática Industrial”, Mirella M. Moro, Vanessa Braganholo. Em: André C. P. L. F. de Carvalho; Tomasz Kowaltowski (Editores) Atualizações em Informática 2009.
- [2] “Dados Semi-Estruturados”, Ronaldo dos Santos, Carina Friedrich Dorneles, Adrovane Kade, Carlos Alberto Heuser. [Material de um tutorial para o SBBD 2000].
- [3] “Sistemas de Bancos de Dados” (6ª edição), Elmasri e Navathe, Capítulo 12 - “XML – Extensible Markup Language”
- [4] “Databases Systems – The Complete Book”, Garcia-Molina, Ullman, Widom, Seções 4.6 e 4.7