

MAC0439

Laboratório de Bancos de Dados

Dados Semiestruturados

# Introdução a XML (Extensible Markup Language)

Profa. Kelly Rosa Braghetto  
DCC-IME-USP

02 de março de 2018

# XML (*Extensible Markup Language*)

- Desenvolvido pelo consórcio W3C, com base em linguagens mais antigas, como a SGML e a HTML
- É o modelo de dados semiestruturados mais bem sucedido
- É um padrão para a publicação, combinação e intercâmbio de dados na web
- Por ser uma linguagem de marcação, XML lida com instruções embutidas no corpo dos documentos, chamadas de *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
  - XML tem 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 de um documento XML simples

```
<empregados>
  <empregado>
    <nome>João</nome>
    <sobrenome>Santos</sobrenome>
  </empregado>
  <empregado cpf='111.111.111-11'>
    <nome>Ana</nome>
    <sobrenome>Ferraz</sobrenome>
  </empregado>
</empregados>
```

# 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:** `<empregado cpf='111.111.111-11'> ...`
- Os atributos de XML geralmente são usados para descrever propriedades e características dos elementos dentro dos quais eles aparecem

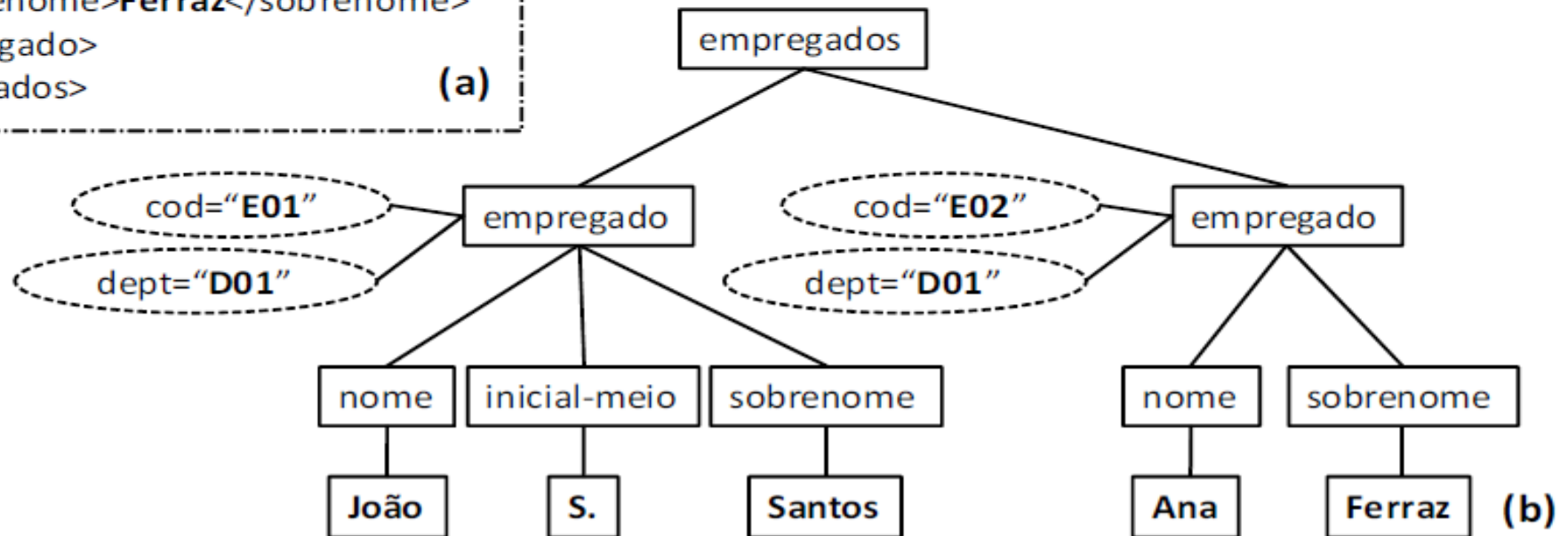
# XML (*Extensible Markup Language*)

- 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 do tipo 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)



# Um parênteses: Atributos vs Subelementos

- XML centrado em atributos:
  - Mais compacto (ocupa menos espaço)
  - Para alguns tipos de dados, não há como atribuir valores nulos (ex.: int)
  - Não possibilita expressar tipos complexos
- XML centrado em elementos:
  - Mais verboso (ocupa mais espaço)
  - Expressa tipos complexos
  - Expressa melhor dados nulos (ex.: atributo `xsi:nil="true"`)
  - Mais fácil de ser estendido (com a inclusão de subelementos)
  - Mais rápido de processar (*parser* apenas dos elementos)



# 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) – 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)

- 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 “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

- **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 *string*, *integer*, *float*, *double*, *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 restrições de **unicidade, chaves e referências a chaves**

# XML Schema

- XML Schema possui um mecanismo de derivação de tipos, permitindo a criação de novos tipos a partir de outros já existentes
- Isto pode ser feito de duas maneiras: por **restrição** ou por **extensão (herança)**
- Tipos simples só podem ser derivados por restrição, aplicando-se “facetar” a um tipo básico ou utilizando uma linguagem de expressões regulares

# *XML Schema*

- Derivação de um tipo simples por restrição

```
<simpleType name="MeuInteiro" base="integer">  
  <minInclusive value="1"/>  
  <maxInclusive value="20"/>  
</simpleType>
```

# XML Schema

- Derivação de um tipo complexo por extensão

```
<complexType name="tEmpregadoEstendido"  
    base="tEmpregado" derivedBy="extension">  
    <element name="endereço" type="string"  
        minOccurs="0" maxOccurs="unbounded"/>  
</complexType>
```

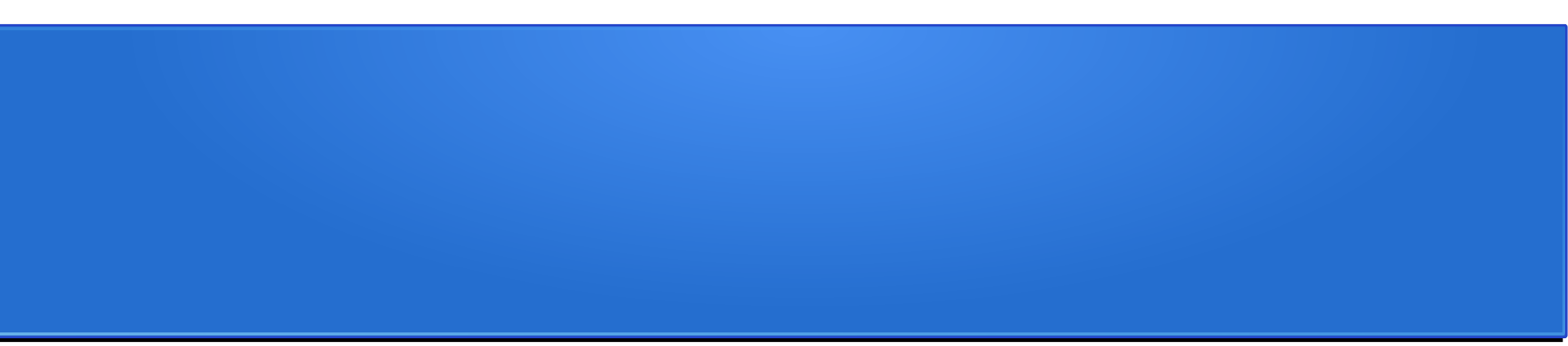
# APIs para a Manipulação de Documentos XML

- As bibliotecas que são capazes de manipular documentos XML possuem um processador (= *parser*) XML, que é responsável por disponibilizar o conteúdo do documento para uma aplicação (programa)
- 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 APIs 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 o documento como um todo
- **SAX** – funciona baseada em eventos; manipula cada parte do documento sequencialmente



# Linguagens de Consulta para XML

# 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 folhas na árvore), elementos ou atributos
- Os 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

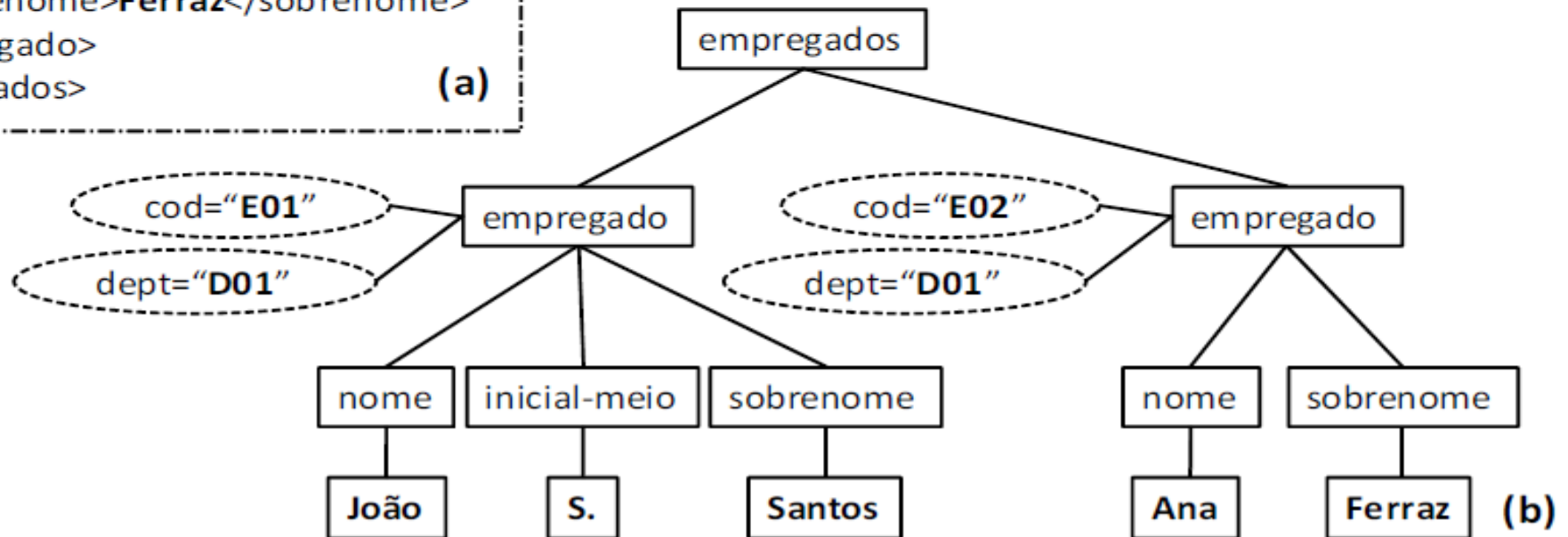
- Vamos considerar o documento “emps.xml”, válido sob o esquema “emps.dtd”

```
<?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 45 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 a 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



# Extração de documentos XML de BDs Relacionais

# Caracterização de documentos XML

- É possível dividir os documentos XML em três categorias:
  - Centrados em dados
  - Centrados em documento
  - Híbridos

# Caracterização de documentos XML

- **Documentos XML centrados em dados**
  - Possuem muitos itens de dados pequenos, que seguem um esquema fixo
  - Podem ser extraídos de um BD estruturado
  - São formatados como documentos XML para que possam ser usados (trocados ou exibidos) na Web

# Caracterização de documentos XML

- **Documentos XML centrados em documento**
  - Documentos com grande quantidade de texto
  - Exemplos: artigos de notícias, livros
  - Possuem nenhum ou poucos elementos de dados estruturados

# Caracterização de documentos XML

- **Documentos XML híbridos**
  - Possuem partes que contêm dados estruturados e outras que são predominantemente textuais ou não estruturadas
  - Podem ou não ter um esquema predefinido

# Armazenando e extraindo documentos XML de BDs

**Técnicas mais comuns de organização de conteúdo de documentos XML (para facilitar consulta e recuperação):**

- Usar um SGBD Relacional para armazenar os documentos como um texto
- Usar um SGBD Relacional para armazenar conteúdos de documento como elementos de dados
- Usar um sistema especializado para armazenar dados XML nativos
- Criar ou publicar documentos XML personalizados de BDs relacionais preexistentes



# Uso de um SGBD para armazenar documentos XML como um texto

- Um SGBD relacional ou de objeto pode ser usado para armazenar documentos XML inteiros como campo de texto
- Para isso, o SGBD precisa ter um módulo especial para o processamento dos documentos XML
- Essa abordagem é mais apropriada para documentos XML sem esquema fixo e centrados em documento
- Exemplos de SGBDs com essa funcionalidade: IBM DB2, Oracle, MS SQL Server, PostgreSQL

# Uso de um SGBD para armazenar conteúdos de documento como elementos de dados

- Técnica apropriada para armazenar uma coleção de documentos que possuem um esquema definido
- Projeta-se um BD relacional ou de objeto para armazenar os elementos de dados em nível de folha dos documentos XML
- Técnica requer algoritmos de mapeamento de um esquema XML para um esquema de BD compatível
- O SGBD usa os algoritmos para recriar os documentos XML a partir dos dados armazenados no BD
- Esses algoritmos podem ficar integrados ao SGBD (como um módulo interno) ou podem ser implementados como um *middleware*

# Uso de um sistema especializado para armazenar dados XML nativos

- Existem SGBDs “XML nativos” (também chamados de “SGBDs XML puros”)
- Esses SGBDs se baseiam no modelo de dados hierárquico (de árvore)
- Incluem técnicas especializadas de indexação, consulta e compactação de documentos XML
- Funcionam para qualquer tipo de documento XML
- Exemplos de sistemas desse tipo: **eXist-db** e **BaseX**
  - Relacionados a sistemas NoSQL do tipo “orientado a documentos” (*document stores*)

# Criação ou publicação de documentos XML personalizados de BDs relacionais preexistentes





- Existe uma enorme quantidade de dados em BDs relacionais
- Parte desses dados pode ter que ser formatada como documentos para a troca ou exibição na Web
- Pode-se usar *middlewares* para tratar das conversões necessárias entre o BD relacional e os documentos XML

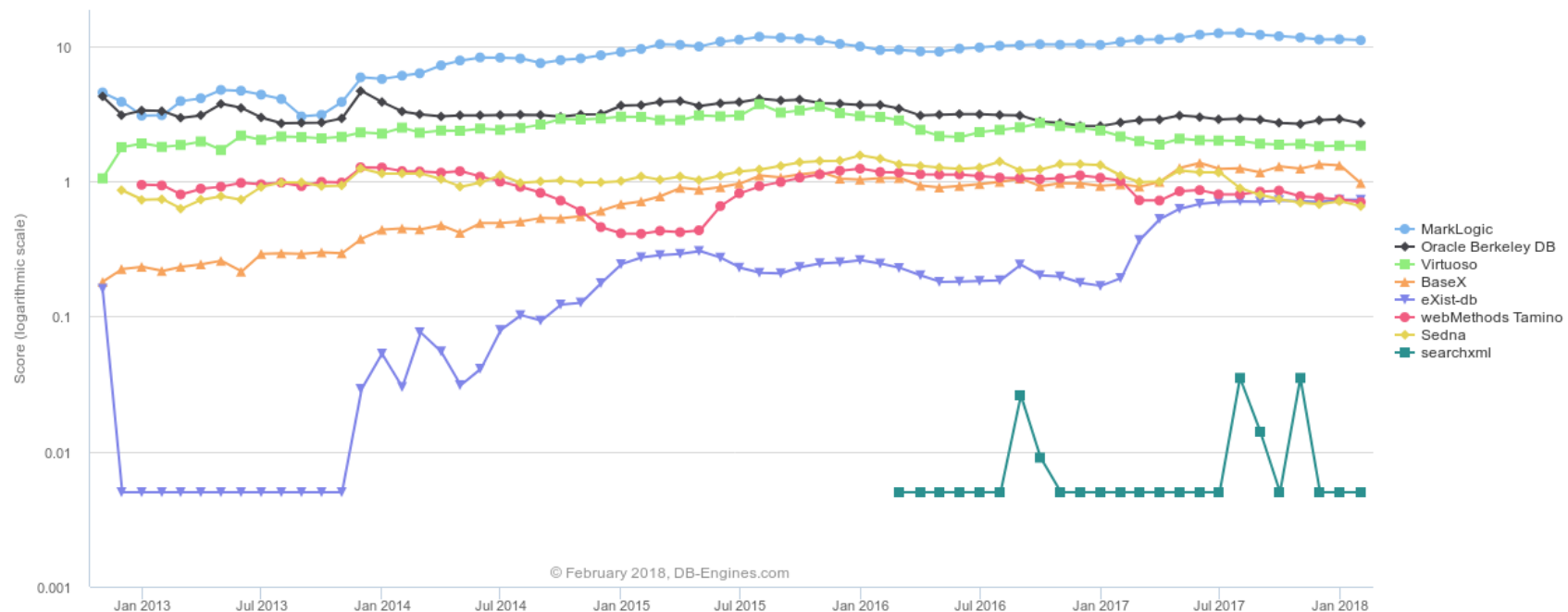
# Ranking de popularidade dos SGBDs XML Nativos

8 systems in ranking, February 2018

Fonte:

<https://db-engines.com/en/ranking/native+xml+dbms>

Rank	Rank			DBMS	Database Model	Score		
	Feb 2018	Jan 2018	Feb 2017			Feb 2018	Jan 2018	Feb 2017
1.	1.	1.	1.	MarkLogic	Multi-model 	11.03	-0.18	+0.29
2.	2.	2.	2.	Oracle Berkeley DB	Multi-model 	2.68	-0.20	-0.04
3.	3.	3.	3.	Virtuoso	Multi-model 	1.83	-0.00	-0.31
4.	4.	4.	6.	BaseX	Native XML DBMS	0.97	-0.34	+0.02
5.	6.	7.	7.	eXist-db	Native XML DBMS	0.73	+0.00	+0.54
6.	5.	5.	5.	webMethods Tamino	Native XML DBMS	0.70	-0.03	-0.30
7.	7.	4.	4.	Sedna	Native XML DBMS	0.65	-0.06	-0.44
8.	8.	8.	8.	searchxml	Multi-model 	0.00	±0.00	±0.00



# Criação ou publicação de documentos XML personalizados de BDs relacionais preexistentes

- Existe uma enorme quantidade de dados em BDs relacionais
- Parte desses dados pode ter que ser formatada como documentos para a troca ou exibição na Web
- Pode-se usar *middlewares* para tratar das conversões necessárias entre o BD relacional e os documentos XML

# Extração de documentos XML de BDs Relacionais

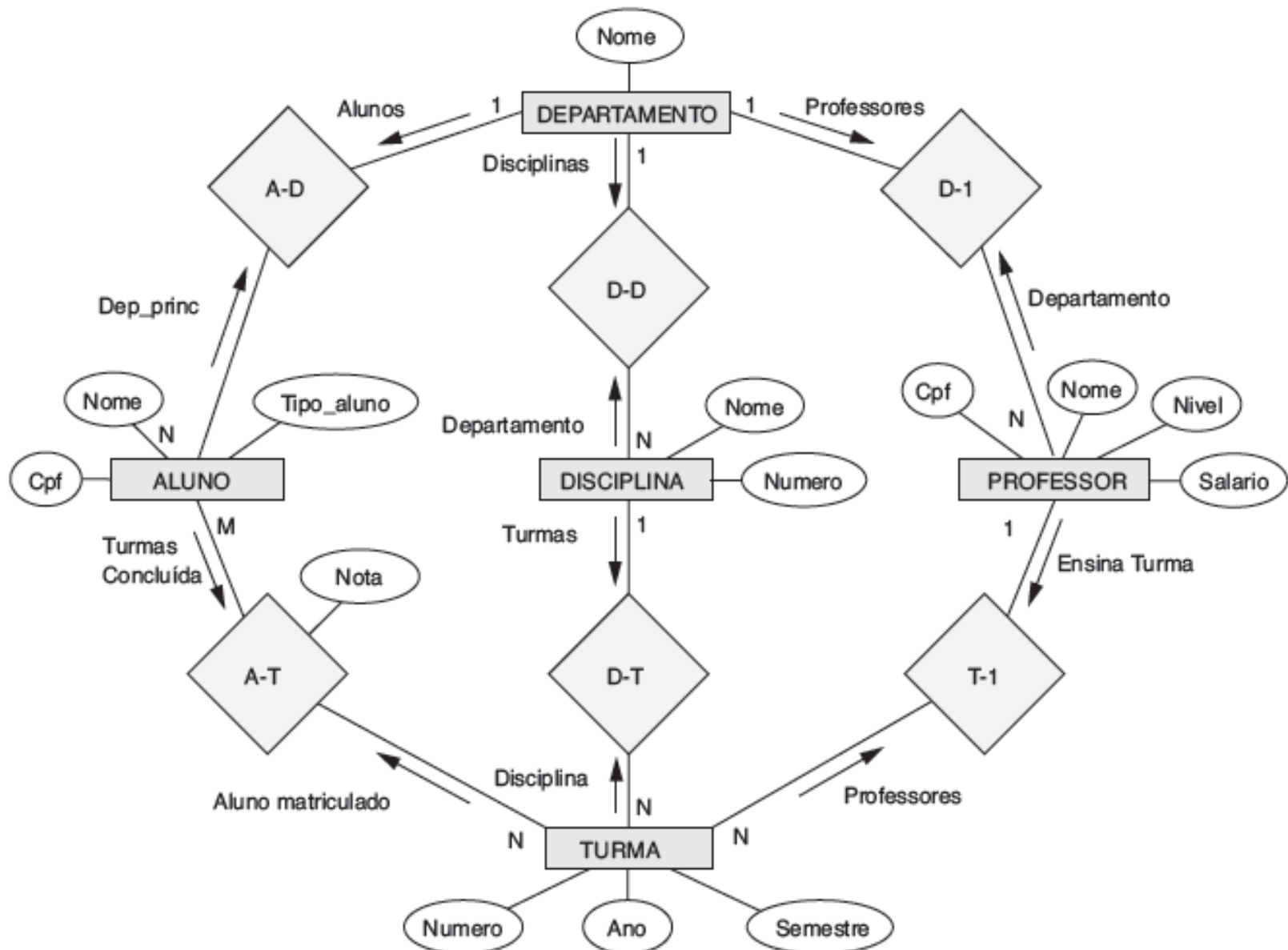
- Dois casos possíveis:
  - Criação de visões XML hierárquicas sobre modelos de dados planos ou baseados em grafos
  - Criação de visões XML hierárquicas sobre modelos de dados contendo ciclos

# Criação de visões XML hierárquicas sobre modelos de dados planos

- O modelo relacional “puro” é plano
- Quando acrescenta-se restrições de integridade referencial, ele pode ser visto como um grafo
- É possível representar conceitualmente um esquema de BD relacional usando um esquema ER correspondente (que é um modelo gráfico)
- De forma análoga, podemos representar um esquema relacional em XML



# Exemplo – Esquema simplificado do BD Universidade

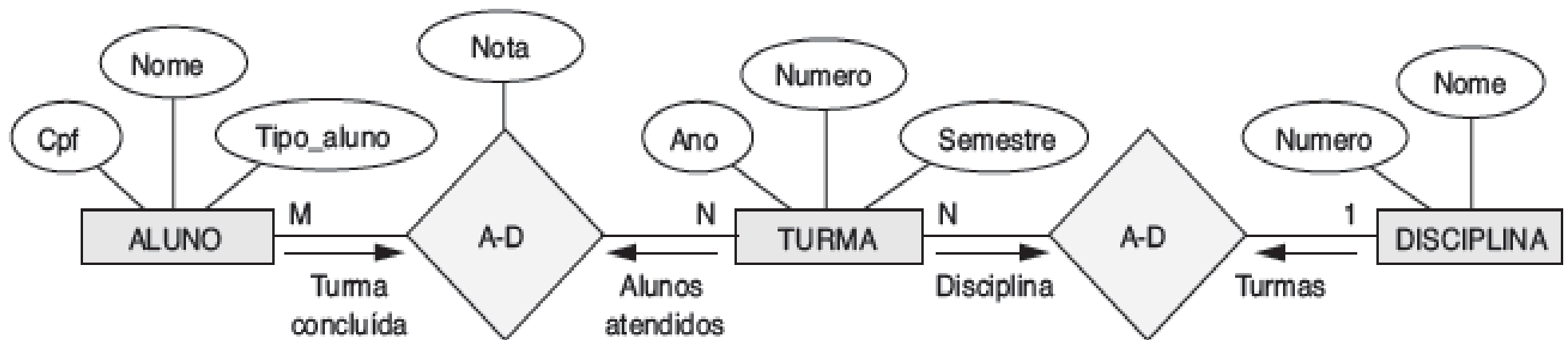


# Criação de visões XML hierárquicas sobre modelos de dados planos

- Requisito: documentos XML contendo dados sobre alunos, disciplinas e notas
- Esses dados são mantidos nos tipos de entidade DISCIPLINA, TURMA e ALUNO, e nos tipos de relacionamento T-A e D-T do BD Universidade
  - A maioria dos documentos extraídos a partir de um BD relacional se refere somente a um subconjunto dos elementos mantidos no BD
- Três diferentes hierarquias XML podem ser extraídas do subconjunto definido acima

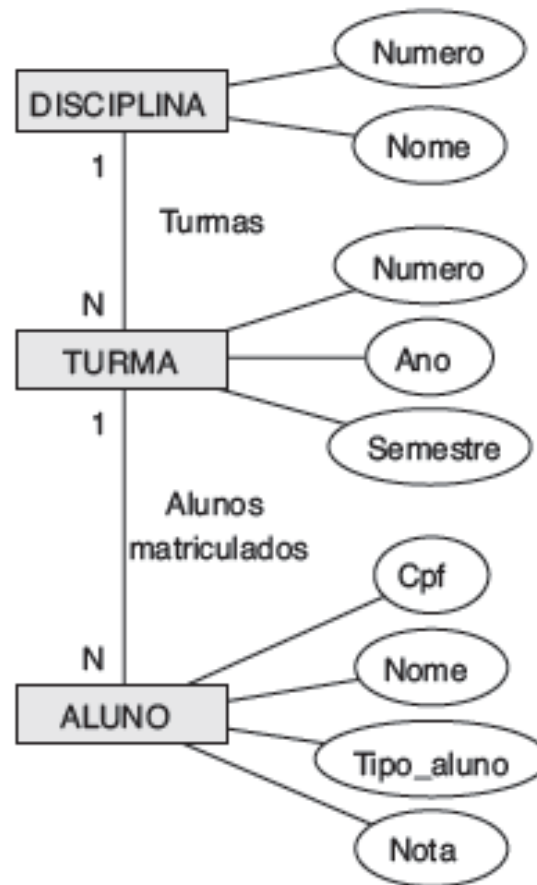
# Exemplo – Esquema simplificado do BD Universidade

- Subconjunto do BD da Universidade necessário para a extração do documento XML



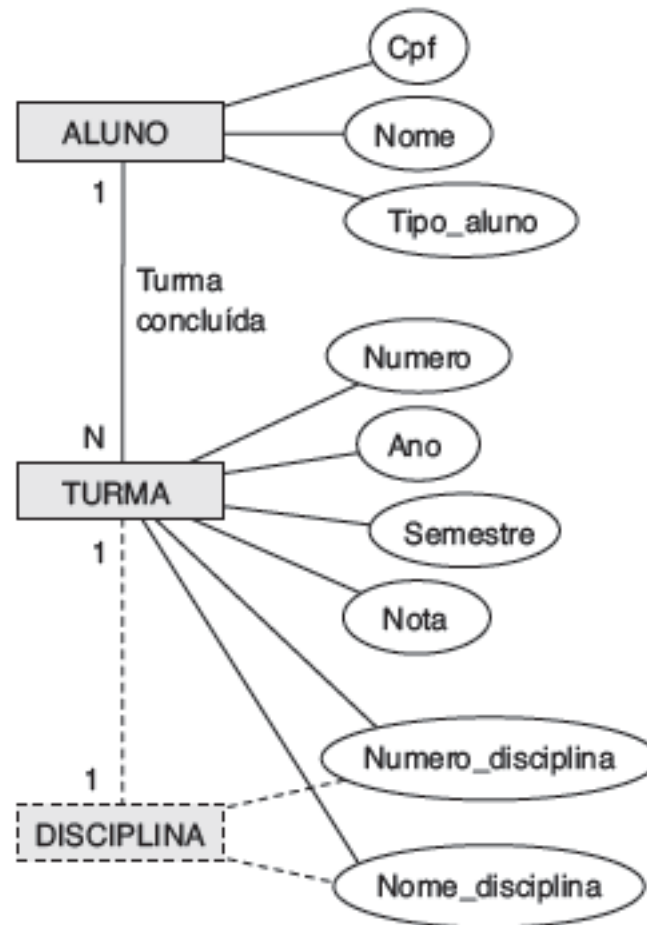
# Criação de visões XML hierárquicas sobre modelos de dados planos

- Visão hierárquica XML com DISCIPLINA como raiz



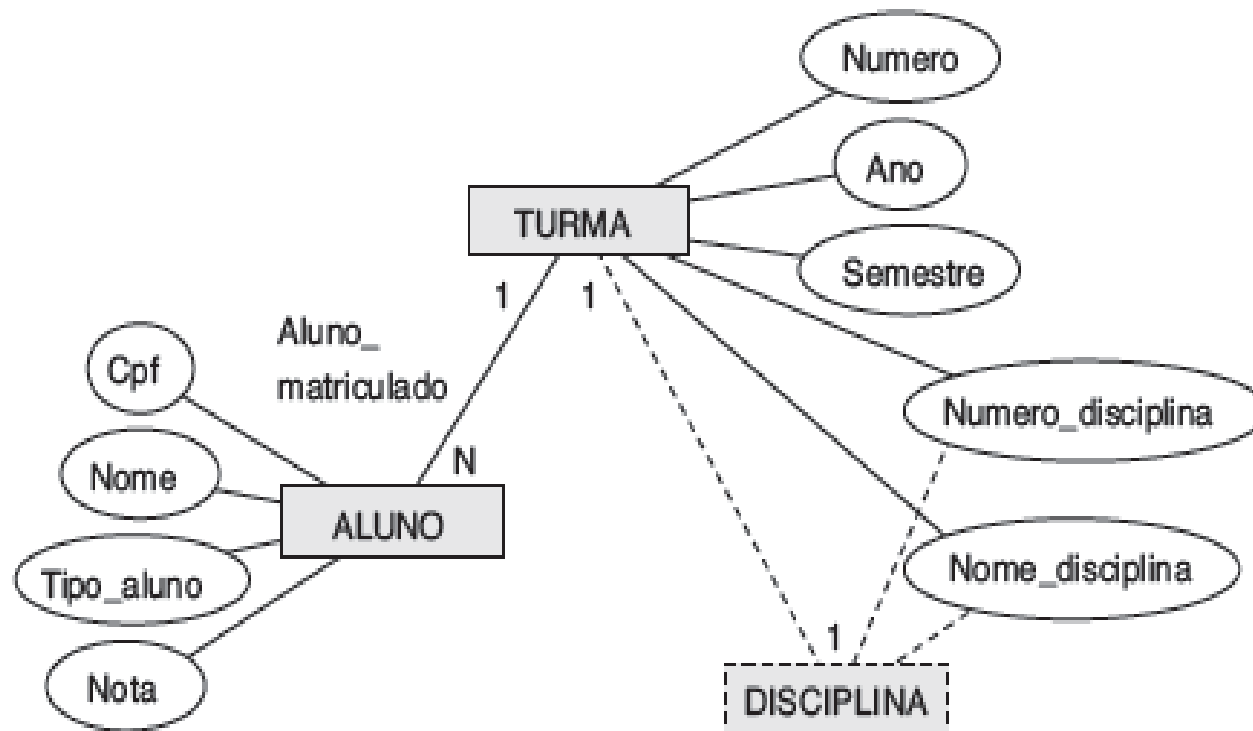
# Criação de visões XML hierárquicas sobre modelos de dados planos

- Visão hierárquica XML com ALUNO como raiz



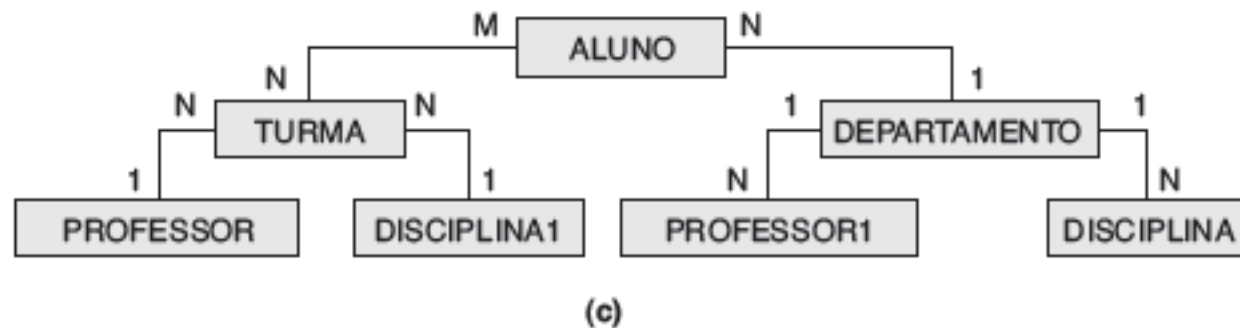
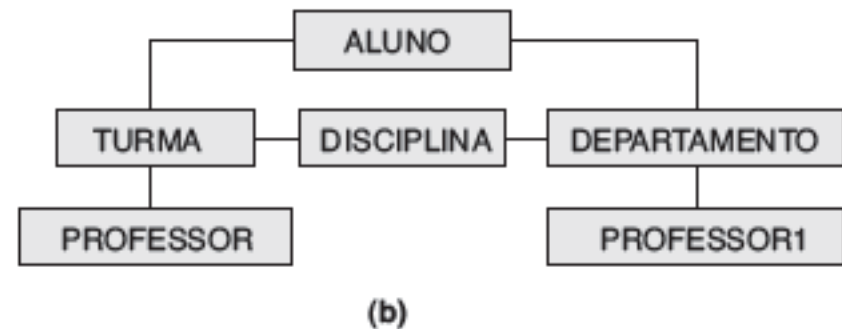
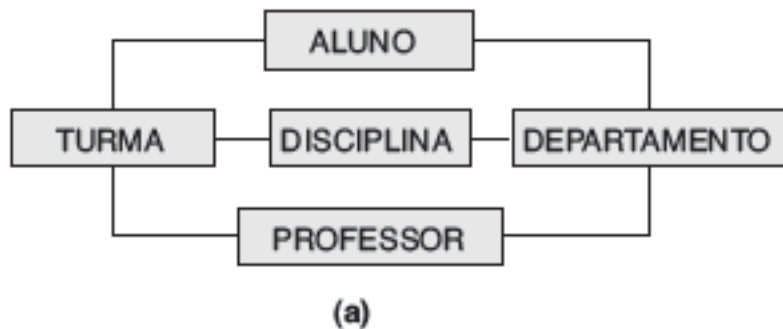
# Criação de visões XML hierárquicas sobre modelos de dados planos

- Visão hierárquica XML com TURMA como raiz



# Quebra de ciclos para a conversão de grafos em árvores

- (a) Grafo com ciclos
- (b) Replicação de PROFESSOR para a quebra do primeiro ciclo
- (c) Replicação de DISCIPLINA para a quebra do segundo ciclo, resultando em uma estrutura hierárquica (em árvore)



# Ferramentas online para XML

- Validadores de Esquema (DTD e XML Schema):
  - <http://xmlvalidator.new-studio.org/>
  - <https://www.freeformatter.com/xml-validator-xsd.html>
  - <http://www.corefiling.com/opensource/schemaValidate.html>
  - <http://www.xmlforasp.net/SchemaValidator.aspx>
- Processadores de consultas XPath e XQuery:
  - <https://www.freeformatter.com/xpath-tester.html>
  - <https://codebeautify.org/Xpath-Tester>
  - <http://www.webtoolkitonline.com/xml-xpath-tester.html>
  - <http://xmlgrid.net/xpath.html>
  - <http://brettz9.github.io/xqueryeditor/>



# Nas próximas aulas

- Mais sobre dados semiestruturados:
  - JSON
- Sistemas NoSQL

# Referências bibliográficas

- “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.
- “Dados Semi-Estruturados”, Ronaldo dos Santos, Carina Friedrich Dorneles, Adrovane Kade, Carlos Alberto Heuser. [Material de um tutorial para o SBBD 2000].
- “Sistemas de Bancos de Dados” (6ª edição), Elmasri e Navathe, Capítulo 12 - “XML – Extensible Markup Language”
- “Querying XML - XQuery, XPath, and SQL/XML in Context”. Melton e Buxton, 2011.

<http://www.sciencedirect.com/science/book/9781558607118>

- XPath – <http://www.w3schools.com/xpath>
- XQuery – <http://www.w3schools.com/xquery/>