

Capítulo 9: XML

- XML para transferência de dados
- Estrutura hierárquica do XML
- DTDs e XML Schema
- Consultas de documentos XML: Xpath
- Consultas de documentos XML: XQuery
- Transformação de documentos XML: XSLT
- Mapeamento entre documentos XML e Bases de Dados relacionais

Documento XML de exemplo

```
<?xml version = "1.0" standalone = "no"?>
```

```
<!DOCTYPE banco SYSTEM "banco.dtd">
```

```
<banco>
```

```
  <cliente>
```

```
    <nome-cliente>Luis</nome-cliente>
```

```
    <rua-cliente>5 de Outubro</rua-cliente>
```

```
    <local-cliente>Lisboa</local-cliente>
```

```
  </cliente>
```

```
  <cliente>
```

```
    <nome-cliente>Maria</nome-cliente>
```

```
    <rua-cliente>1 de Maio</rua-cliente>
```

```
    <local-cliente>Caparica</local-cliente>
```

```
  </cliente>
```

```
<conta>
```

```
  <num-conta>A-102</num-conta>
```

```
  <agencia>Caparica</agencia>
```

```
  <saldo>400</saldo>
```

```
</conta>
```

```
<conta>
```

```
  <num-conta>A-101</num-conta>
```

```
  <agencia>Lisboa</agencia>
```

```
  <saldo>100</saldo>
```

```
</conta>
```

```
<depositante>
```

```
  <nome-cliente>Luis</nome-cliente>
```

```
  <num-conta>A-102</num-conta>
```

```
</depositante>
```

```
<depositante>
```

```
  <nome-cliente>Maria</nome-cliente>
```

```
  <num-conta>A-101</num-conta>
```

```
</depositante>
```

```
</banco>
```

Exemplo de DTD

```
<!DOCTYPE banco [  
  <!ELEMENT banco ( ( conta | cliente | depositante)+)>  
  <!ELEMENT conta (num-conta, agencia, saldo)>  
  <!ELEMENT cliente (nome-cliente, rua-cliente, local-cliente)>  
  <!ELEMENT depositante (nome-cliente, num-conta)>  
  <!ELEMENT num-conta (#PCDATA)>  
  <!ELEMENT agencia (#PCDATA)>  
  <!ELEMENT saldo (#PCDATA)>  
  <!ELEMENT nome-cliente (#PCDATA)>  
  <!ELEMENT rua-cliente (#PCDATA)>  
  <!ELEMENT local-cliente (#PCDATA)>  
>
```

Exemplo de DTD com atributos

```
<!DOCTYPE banco-2[
  <!ELEMENT conta (agencia, saldo)>
  <!ATTLIST conta
    num-conta      ID      #REQUIRED
    clientes       IDREFS #REQUIRED>
  <!ELEMENT cliente (nome-cliente, rua-cliente,
                    local-cliente)>
  <!ATTLIST cliente
    id-cliente     ID      #REQUIRED
    contas        IDREFS #REQUIRED>
  ... ..
]>
```

Documento XML com atributos ID e IDREF

```
<?xml version = "1.0" standalone = "no"?>
<!DOCTYPE banco-2 SYSTEM "http://centria.fct.unl.pt/~jleite/banco2.dtd">
<banco-2>
  <conta num-conta="A-401" clientes="C100 C102">
    <agencia> Caparica </agencia>
    <saldo>500 </saldo>
  </conta>
  <cliente id-cliente="C100" contas="A-401">
    <nome-cliente> Luís </nome-cliente>
    <rua-cliente> R. República </rua-cliente>
    <local-cliente> Lx </local-cliente>
  </cliente>
  <cliente id-cliente="C102" contas="A-401 A-402">
    <nome-cliente> Maria </nome-cliente>
    <rua-cliente> R. 5 de Outubro </rua-cliente>
    <local-cliente> Porto </local-cliente>
  </cliente>
</banco-2>
```

Versão XML Schema do banco

```
<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema>
<xsd:element name="banco" type="TipoBanco"/>
<xsd:element name="conta">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="num-conta" type="xsd:string"/>
      <xsd:element name="agencia" type="xsd:string"/>
      <xsd:element name="saldo" type="xsd:decimal"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

..... Definições para cliente e depositante....

```
<xsd:complexType name="TipoBanco">
  <xsd:sequence>
    <xsd:element ref="conta" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="cliente" minOccurs="0" maxOccurs="unbounded"/>
    <xsd:element ref="depositante" minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

Transformação e Consulta de dados XML

■ Linguagens para transformação/pesquisa em documentos XML

★ XPath

- ❖ Linguagem simples, que consiste em path expressions

★ XQuery

- ❖ Linguagem mais complexa de pesquisa de informação em documentos XML

★ XSLT

- ❖ Linguagem desenhada para tradução de documentos XML para XML e XML para HTML

Exemplos de XPath

- `/banco-2/cliente`
- `/banco-2/cliente/nome-cliente`
- `/banco-2/cliente/nome-cliente/text( )`
- `/banco-2/conta[saldo > 400]/@num-conta`
- `/banco-2/conta[count(cliente//text()) > 2]`
- `/banco-2/conta[@num-conta="A-401"]/id(@clientes)`
- `/banco-2//nome`

XQuery

- Linguagem de mais alto nível para perguntas genéricas a documentos XML.
- Usa a sintaxe: **for ... let ... where .. order by ... return ...**
 - for** ⇔ SQL from
 - where** ⇔ SQL where
 - order by** ⇔ SQL order by
 - return** ⇔ SQL select
 - let** não tem equivalente em SQL (para variáveis temporárias)
- A parte do **for** tem expressões XPath e variáveis que vão tomando os vários valores retornados pela path expression
- A parte do **where** impõe condições sobre essas variáveis
- A parte **order by** permite especificar a ordenação
- A parte do **return** especifica o que deve aparecer no output, para cada valor da variável

Sintaxe FLWOR em XQuery

■ Uma expressão FLWOR em XQuery

- ★ Encontrar todas as contas com saldo > 400, onde cada elemento do resultado deve ser apresentado entre um *tag*

<num-conta> .. </num-conta>

for \$x in /banco-2/conta

let \$acctno := \$x/@num-conta

where \$x/saldo > 400

return <num-conta> { \$acctno } </num-conta>

- ★ Os itens na cláusula **return** são texto XML, a não ser que estejam dentro de {}, caso em que eles são avaliados

- ## ■ A cláusula **let** não é absolutamente necessária nesta expressão, e a cláusula **where** poderia ser incorporada na expressão XPath. A consulta pode ser expressa como:

for \$x in /banco-2/conta[saldo > 400]

return <num-conta> { \$x/@num-conta } </num-conta>

Junções

- As junções são especificadas de uma forma semelhante à que se encontra no SQL:

```
for      $a in /banco/conta,  
          $c in /banco/cliente,  
          $d in /banco/depositante  
  
where    $a/num-conta = $d/num-conta  
          and $c/nome-cliente = $d/nome-cliente  
  
return   <cliente_conta> { $c $a } </cliente_conta>
```

- A mesma consulta pode ser expressa com a selecção especificada como selecções XPath:

```
for      $a in /banco/conta,  
          $c in /banco/cliente,  
          $d in /banco/depositante[  
              num-conta = $a/num-conta and  
              nome-cliente = $c/nome-cliente]  
  
return   <cliente_conta> { $c $a } </cliente_conta>
```

Consultas Imbricadas

- A consulta que se segue converte os dados da estrutura flat da informação **banco** na estrutura imbricada usada em **banco-1**

```
<banco-1> {  
  for $c in /banco/cliente  
  return  
    <cliente>  
    { $c/* }  
    { for $d in /banco/depositante[nome-cliente= $c/nome-cliente],  
      $a in /banco/conta[num-conta=$d/num-conta]  
      return { $a } }  
    </cliente>  
} </banco-1>
```

- **\$c/*** denota todos os filhos do nó ao qual **\$c** está associado, excluindo o *tag* de mais alto nível.
- **\$c/text()** devolve o conteúdo textual de um elemento sem quaisquer sub-elementos / *tags*.

Ordenação em XQuery

- A cláusula **order by** pode ser usada no fim de qualquer expressão. E.g. para devolver os clientes ordenados pelo nome:

for	\$c in /banco/cliente
order by	\$c/nome-cliente
return	<cliente> { \$c/* } </cliente>

- Usa-se **order by** \$c/customer_name **descending** para ordenar de forma descendente:

for	\$c in /banco/cliente
order by	\$c/nome-cliente descending
return	<cliente> { \$c/* } </cliente>

Ordenação em XQuery (cont.)

- Podemos usar vários níveis de ordenação (por exemplo: ordenação por nome de cliente, seguida de ordenação por número de conta dentro de cada cliente)

```
<banco-1> {  
  for $c in /banco/cliente  
  order by $c/nome-cliente  
  return  
    <cliente>  
      { $c/* }  
      { for $d in /banco/depositante[nome-cliente= $c/nome-cliente],  
        $a in /banco/conta[num-conta=$d/num-conta]  
        order by $a/num-conta  
        return <conta> { $a/* } </conta> }  
    </cliente>  
} </banco-1>
```

Funções e outras características da XQuery

- Funções definidas pelo utilizador com o sistema de tipos do XMLSchema
function saldos(xs:string \$c) **returns** list(xs:decimal*) {
 for \$d **in** /banco/depositante[nome-cliente = \$c],
 \$a **in** /banco/conta[num-conta = \$d/num-conta]
 return \$a/saldo/text()
}
- A especificação dos tipos dos parâmetros e dos valores de retorno é opcional.
- O * (por ex: decimal*) indica uma sequência de valores desse tipo.
- Permite a utilização de quantificadores universal e existencial nas cláusulas dos predicados **where**
 - ★ **some** \$e **in** *path* **satisfies** *P*
 - ★ **every** \$e **in** *path* **satisfies** *P*
- XQuery também suporta cláusulas if-then-else.

Mais informação...

- <http://www.w3.org>
 - ✦ Consórcio para a World Wide Web
- <http://www.w3schools.com>
 - ✦ Tutoriais de XML, DTD, XML Schema, XPath, XQuery, ...
- <http://www.w3.org/TR/xpath20/>
 - ✦ Recomendação XPath 2.0 (23 de Janeiro de 2007)
- <http://www.w3.org/TR/xquery/>
 - ✦ Recomendação XML Query 1.0 (23 de Janeiro de 2007)

Visualização de documentos XML

- Para visualizar um documento XML é preciso associar formatação a cada uma das tags.
- A forma mais simples de fazer isso é via folhas de estilo CSSs (Cascading Style Sheets)

```
<?xml version = "1.0" standalone = "no"?>  
<!DOCTYPE banco SYSTEM http://centria.fct.unl.pt/~jleite/banco.dtd>  
<?xml-stylesheet type="text/css" href="banco.css"?>  
<banco>  
...  
</banco>
```

- Separam-se os dados da sua formatação.

Exemplo de CSS

- Ficheiro banco.css

conta {display: block; font-size: 25; font-weight: bold; text-align: center}

num-conta {display: block; color:red; font-size: 25; font-weight: bold}

agencia {display: block; font-size: 15; text-align: left}

saldo {display: block; color:green; font-size: 55; text-align: right}

...

- Muito limitado!

- Outra abordagem: transformação de documentos XML em documentos HTML.

[Link para exemplo](#)

[Ficheiro CSS](#)

XSLT

- Uma **folha de estilos** guarda as opções de formatação do documento, separadamente do documento propriamente dito:
 - ✦ Por ex. uma folha de estilos para html pode especificar tipos de fonte, cores, alinhamentos, etc.
- O **XSL (XML Stylesheet Language)** foi concebido para gerar HTML a partir de XML
- O XSLT é uma linguagem geral de transformação de documentos XML
 - ✦ Pode transformar XML em XML, e XML em HTML
- Em XSLT as transformações são definidas à custa de **templates** (modelos/padrões)
 - ✦ Os templates combinam a selecção usando XPath com a construção dos resultados da transformação

Templates XSLT

- Exemplo de template XSLT com **match** e **select**

```
<xsl:template match="/banco-2/cliente">  
  <xsl:value-of select="nome-cliente"/>  
</xsl:template>
```

- O atributo match da tag xsl:template especifica um padrão em XPath
- Os elementos do documento XML que estão de acordo com esse padrão, são processados de acordo com o especificado no elemento **xsl:template**
 - ★ **xsl:value-of** selecciona, para output, valor específicos (no caso, **nome-cliente**)
- Para nós que não estejam de acordo com nenhum padrão existem padrões implícitos que :
 - ★ Nós de Texto são escritos no output sem processamento
 - ★ Templates são aplicados recursivamente ao filho da raiz e de elementos
- Se um elemento está de acordo com vários templates, apenas um deles é usado. A escolha é feita por um esquema complexo de prioridades. As regras implícitas são as de menor prioridade.

Documento XML de exemplo

```
<?xml version = "1.0" standalone = "no"?>  
<!DOCTYPE banco-2 SYSTEM "banco.dtd">
```

```
<banco>
```

```
  <cliente>
```

```
    <nome-cliente>Luis</nome-cliente>
```

```
    <rua-cliente>5 de Outubro</rua-cliente>
```

```
    <local-cliente>Lisboa</local-cliente>
```

```
  </cliente>
```

```
  <cliente>
```

```
    <nome-cliente>Maria</nome-cliente>
```

```
    <rua-cliente>1 de Maio</rua-cliente>
```

```
    <local-cliente>Caparica</local-cliente>
```

```
  </cliente>
```

```
  <conta>
```

```
    <num-conta>A-102</num-conta>
```

```
    <agencia>Caparica</agencia>
```

```
    <saldo>400</saldo>
```

```
  </conta>
```

```
  <conta>
```

```
    <num-conta>A-101</num-conta>
```

```
    <agencia>Lisboa</agencia>
```

```
    <saldo>100</saldo>
```

```
  </conta>
```

```
  <depositante>
```

```
    <nome-cliente>Luis</nome-cliente>
```

```
    <num-conta>A-102</num-conta>
```

```
  </depositante>
```

```
  <depositante>
```

```
    <nome-cliente>Maria</nome-cliente>
```

```
    <num-conta>A-101</num-conta>
```

```
  </depositante>
```

```
</banco>
```

Exemplo de output XML

■ Ficheiro XSL:

```
<xsl:template match="/banco/cliente">  
  <cliente>  
    <xsl:value-of select="nome-cliente"/>  
  </cliente>  
</xsl:template>  
<xsl:template match="contaldepositante"/>
```

■ Exemplo de output:

```
<cliente> Luís </cliente>  
<cliente> Maria </cliente>
```

Criação de atributos em XSLT

- O XSLT não permite uma tag `xsl:value-of` dentro de outra tag
 - ★ E.g. não se pode criar um atributo para `<cliente>` usando directamente um `xsl:value-of`

- ★ Para esse efeito o XSLT tem o `xsl:attribute`

- ❖ `xsl:attribute` adiciona atributos a um elemento

- ❖ Exemplo XSL:

```
<xsl:template match="/banco/cliente">
  <cliente>
    <xsl:attribute name="id">
      <xsl:value-of select="num-cliente"/>
    </xsl:attribute>
    <xsl:value-of select="nome-cliente"/>
  </cliente>
</xsl:template>
<xsl:template match="contaldepositante"/>
```

- ❖ Exemplo de output:

```
<cliente id="C100"> Luís </cliente>
<cliente id="C102"> Maria </cliente>
```

Recursão estrutural

- As acções dos templates podem ser simplesmente a de aplicar recursivamente os templates ao conteúdo do padrão reconhecido

- ```
<xsl:template match="/banco">
 <clientes>
 <xsl:apply-templates />
 </clientes>
</xsl:template>
<xsl:template match="/banco/cliente">
 <cliente>
 <xsl:value-of select="nome-cliente"/>
 </cliente>
</xsl:template>
<xsl:template match="*" />
```

- O `<xsl:template match="*" />` é usado aqui para garantir que para todos os outros elementos, não é produzido nenhum output

- Exemplo de output:

```
<clientes>
 <cliente> Luís </cliente>
 <cliente> Maria </cliente>
</clientes>
```

# Regras implícitas

- As regras implícitas de tratamento dos nós são as seguintes:

```
<!-- Raiz e elementos aplicam templates aos filhos -->
<xsl:template match="*|/">
 <xsl:apply-templates/>
</xsl:template>
```

```
<!-- Nós texto e atributos são escritos -->
<xsl:template match="text()|@*">
 <xsl:value-of select="."/>
</xsl:template>
```

```
<!-- Ignorar comentários e instruções de processamento -->
<xsl:template match="processing-instruction()
 | comment()" />
```

# Ordenação em XSLT

- Dentro de um template, um `xsl:sort` ordena todos os elementos de acordo com o padrão do template.

★ A ordenação é feita antes de se aplicarem outros templates

```
<xsl:template match="/banco">
 <xsl:apply-templates select="cliente">
 <xsl:sort select="nome-cliente"/>
 </xsl:apply-templates>
</xsl:template>
<xsl:template match="cliente">
 <cliente>
 <xsl:value-of select="nome-cliente"/>
 <xsl:value-of select="rua-cliente"/>
 <xsl:value-of select="local-cliente"/>
 </cliente>
</xsl:template>
<xsl:template match="*" />
```

# Exemplo de ficheiro XSLT (XML -> HTML)

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match = "/">
 <html>
 <xsl:apply-templates/>
 </html>
</xsl:template>
<xsl:template match = "/banco">
 <body>
 <p> Contas: </p>
 <xsl:apply-templates/>
 </body>
</xsl:template>
<xsl:template match = "/banco/conta">
 <p> A conta <xsl:value-of select="num-conta" />
 da agencia <xsl:value-of select="agencia" />
 tem saldo <xsl:value-of select="saldo" />.
 </p>
</xsl:template>
</xsl:stylesheet>
```

[Link para exemplo](#)

[Link para exemplo](#)

# Junções em XSLT

- As **keys** do XSLT permitem a indexação de elementos por valores de sub-elementos ou atributos
  - As keys têm que ser declaradas (com um nome)
  - A função key() é usado para ir buscar os valores indexados. E.g.  

```
<xsl:key name="numct" match="conta" use="num-conta"/>
<xsl:value-of select=key("numct", "A-101")/>
```
- Isto permite exprimir (alguns tipos de) junções com XSLT  

```
<xsl:key name="numct" match="conta" use="num-conta"/>
<xsl:key name="nomcl" match="cliente" use="nome-cliente"/>
<xsl:template match="depositante">
 <cliente-conta>
 <xsl:value-of select="key('nomcl', nome-cliente)"/>
 <xsl:value-of select="key('numct', num-conta)"/>
 </cliente-conta>
</xsl:template>
```

[Link para exemplo](#)

# API (Application Program Interface)

- Há dois APIs standard para dados XML:

- ★ **SAX (Simple API for XML)**

- ❖ Baseado no modelo de parsers, o utilizador fornece handlers para tratar eventos.
  - E.g. início de elemento; fim de elemento
  - Não apropriado para aplicações de Bases de Dados

- ★ **DOM (Document Object Model)**

- ❖ Dados **XML** são transformados para uma representação em árvore
- ❖ Fornece um conjunto de funções para percorrer a árvore DOM
- ❖ E.g.: Java DOM API fornece classe Node com métodos:
  - `getParentNode( )`, `getFirstChild( )`, `getNextSibling( )`
  - `getAttribute( )`, `getData( )` (for text node)
  - `getElementsByTagName( )`, ...
- ❖ Também fornece funções para actualizar a árvore DOM.

# Armazenamento de Dados XML

■ Os dados XML podem ser armazenados em

★ Formas não relacionais:

❖ Ficheiros *flat*

- Natural para guardar XML
- Tem os problemas discutidos no início do semestre

❖ Base de dados XML

- Base de dados desenvolvida especificamente para guardar dados XML, fornecendo o modelo DOM e consultas declarativas.
- De momento não existem sistemas de nível comercial

★ Bases de Dados Relacionais

- ❖ Dados têm que ser traduzidos para a forma relacional
- ❖ Vantagem: SGBDs maduros
- ❖ Desvantagem: overhead da tradução de dados e consultas

# Armazenamento de Dados XML em BDs Relacionais

## ■ Alternativas:

- ✦ Representação sob a forma de cadeias de caracteres (strings)
- ✦ Representação sob a forma de árvores
- ✦ Mapeamento em relações

# Representação em cadeias de caracteres

- Guardar cada elemento de nível superior como um atributo do tipo string de um tuplo numa base de dados relacional.
  - ★ Usando uma relação para guardar todos os elementos, ou
  - ★ Usando uma relação diferente para cada elemento de tipo superior
    - ❖ E.g. Relações conta, cliente, etc...
      - Cada uma com um atributo do tipo string para guardar o elemento
- Indexação:
  - ★ Guardar valores de atributos/sub-elementos a serem indexados como atributos extra da relação, e criar índices sobre esses atributos
    - ❖ E.g. nome\_cliente, número\_conta...
  - ★ Algumas bases de dados suportam índices funcionais, usando o resultado de uma função como valor a indexar.
    - ❖ A função deverá devolver o valor do atributo/sub-elemento

# Representação em cadeias de caracteres (Cont.)

## ■ Benefícios:

- ✦ Pode guardar XML mesmo sem DTD
- ✦ Desde que existam vários elementos de topo, as cadeias de caracteres são pequenas quando comparadas com o documento.
  - ❖ Permite acesso rápido a elementos individuais.

## ■ Defeitos:

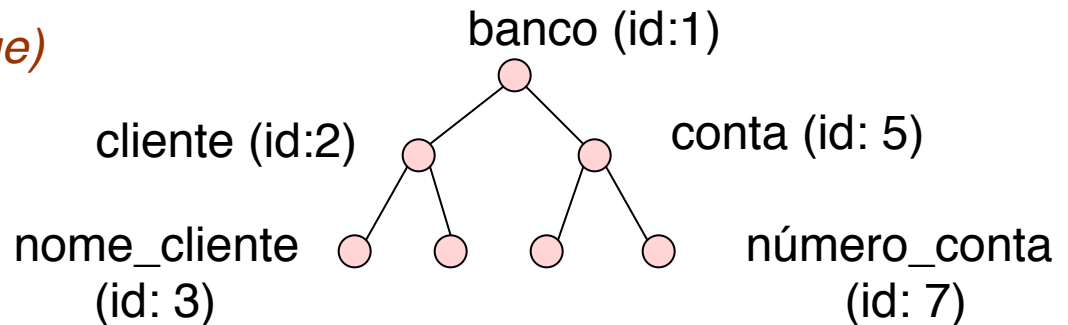
- ✦ É necessário o processamento de strings para aceder aos valores dentro dos elementos.
- ✦ O processamento de strings é lento.

# Representação em árvores

- **Representação em árvores:** modelar o XML em árvore e guardar usando relações:

*nodes(id, type, label, value)*

*child (child\_id, parent\_id)*



- A cada elemento/atributo é dado um identificador único, *id*
- *type* indica elemento/atributo
- *label* especifica o nome do tag do elemento/ nome do atributo
- *value* é o valor textual do elemento/atributo
- A relação *child* guarda as relações pai-filho da árvore.
- Pode ser adicionado um atributo extra à relação filho para manter a ordem dos filhos.

# Representação em árvores (Cont.)

## ■ Vantagens:

- ✦ Pode guardar XML mesmo sem DTD

## ■ Desvantagens:

- ✦ Os dados são divididos em muitas partes, aumentando o overhead de espaço.
- ✦ Mesmo as consultas mais simples requerem um número elevado de junções, o que torna o processo muito lento.

# Conversão de XML para Relações

- Relação criada para cada elemento quando se conhece o seu esquema:
  - ✱ Um atributo id para guardar o identificador único de cada elemento
  - ✱ Um atributo na relação correspondente a cada atributo do elemento
  - ✱ Um atributo id\_pai para manter informação sobre o elemento pai
    - ❖ Como na representação em árvore
    - ❖ Info sobre a posição tb pode ser guardada
- Todos os sub-elementos que ocorrem apenas uma vez podem tornar-se atributos da relação.
  - ✱ Para elementos com valor textual, guardar o texto como valor de um atributo.
  - ✱ Para sub-elementos complexos, guardar id do sub-elemento.
- Sub-elementos que podem ocorrer várias vezes são representados noutra relação.
  - ✱ Semelhante ao tratamento dado a atributos multi-valor na passagem do DER para o modelo relacional.

# Mapeamento de dados de/para XML

- Hoje em dia os sistemas de bases de dados já têm mecanismos que facilitam a importação de dados vindos de ficheiros XML (*shredding*), bem como a exportação de dados para formato XML (*publishing*).
- A transformação de dados para XML pode ser útil, não só para transferência, como também para interface com o utilizador.
- Alguns SGBDs já estão preparados para converter de/para XML.
- Alguns sistemas (e.g. Oracle) oferecem armazenamento nativo em XML usando o tipo de dados xml. Estruturas de dados internas e índices especiais são usados para aumentar a eficiência.

# SQL/XML

- Existe um standard de extensão ao SQL que permite a criação de XML imbricado.

- ✦ Cada tuplo é mapeado para um elemento XML do tipo *row*

<bank>

<account>

<row>

<account\_number> A-101 </account\_number>

<branch\_name> Downtown </branch\_name>

<balance> 500 </balance>

</row>

.... *mais rows se existirem mais tuplos no resultado ...*

</account>

</bank>

# Extensões SQL

- **xmlelement** cria elementos XML
- **xmlattributes** cria atributos

```
select xmlelement (name "account",
 xmlattributes (account_number as account_number),
 xmlelement (name "branch_name", branch_name),
 xmlelement (name "balance", balance))
from account
```

- *Resulta em:*

```
<account>
 <row account_number = "A-101">
 <branch_name> Downtown </branch_name>
 <balance> 500 </balance>
 </row>
.... mais rows se existirem mais contas ...
</account>
```