XML - eXtensible Markup Language

- Tópicos:
  - XML para transferência de dados
  - Estrutura hierárquica do XML
  - DTDs e XML Schema
  - Consultas de documentos XML:
    - Xpath
    - XQuery
  - Transformação de documentos XML: XSLT
  - Mapeamento entre documentos XML e Bases de Dados relacionais
- Bibliografia:
  - Capítulo 23 do livro recomendado

XPath

- O XPath serve para selecionar partes de documento usando para tal **path expressions**
- Uma *path expression* é uma sequência de passos, separados por "/"
  - Semelhante a nome de ficheiros numa hierarquia de diretorias
- Resultado duma *path expression*:
  - Conjunto de nós, e correspondentes subelementos, e atributos quando for caso disso, que correspondem ao caminho (*path)* dado

Exemplo de Xpath

```
<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">
                <nome-cliente> Maria </nome-cliente>
                <rua-cliente> R. 5 de Outubro </rua-cliente>
                <local-cliente> Porto </local-cliente>
        </cliente>
</banco-2>
```

- A path expression /banco-2/cliente/nome-cliente devolve (os nome dos clientes):

  <nome-cliente> Luís </nome-cliente>

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

Exemplo de Xpath

```
<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">
                <nome-cliente> Maria </nome-cliente>
                <rua-cliente> R. 5 de Outubro </rua-cliente>
                <local-cliente> Porto </local-cliente>
        </cliente>
</banco-2>
```

- A path expression /banco-2/cliente/nome-cliente/text( ) devolve (o texto dos nomes dos clientes):

  Luís

  Maria

Exemplo de Xpath

- A path expression /banco-2/cliente devolve (os clientes):

```
<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">
        <nome-cliente> Maria </nome-cliente>
        <rua-cliente> R. 5 de Outubro </rua-cliente>
        <local-cliente> Porto </local-cliente>
</cliente>
```

XPath (Cont.)

- O "/" inicial denota a raiz do documento
- As *path expressions* são avaliadas da esquerda para a direita
  - Cada passo é aplicado ao **conjunto** de nós resultantes da aplicação do passo anterior
- Podem-se usar predicados de seleção (entre [ ]) em qualquer dos passos do path.
- E.g. /banco-2/conta[saldo > 400]
  - Devolve os elementos de todas as contas com saldo superior a 400
  - /banco-2/conta[saldo] devolve os elementos de todas as contas que contêm um subelemento saldo
- Pode-se aceder aos atributos, usando "@"
  - E.g. /banco-2/conta[saldo > 400]/@num-conta
    - Devolve os números das contas cujo saldo é maior que 400
  - Os atributos IDREF não são desreferenciados automaticamente (mais sobre este assunto mais à frente)

Funções em XPath

- O XPath fornece várias funções:
  - E.g. função count() aplicada a uma expressão, conta o número de elementos do conjunto gerado pelo path
    - E.g. /banco-2/conta[count(cliente) > 2]
      - Devolve conjunto de nós conta com mais de 2 subelementos cliente (conjunto vazio para o exemplo dado)
  - Também há funções para testar a posição de um nó relativamente aos seus irmãos, somar valores, operadores sobre strings, inteiros, etc. Exemplos:
    - sum(), contains(st1,st2), concat(st1,st2,st), position(), last(), round(num), ...
- Nos predicados podem-se usar os conectivos Booleanos and e or e a função not()

Funções em XPath (cont.)

- As IDREFs podem-se desreferenciar usando para tal a função id()
  - id() pode também ser aplicado a conjuntos de referências (IDREFS e strings de IDREFs separadas por espaços)
  - E.g. /banco-2/conta/id(@clientes)
    - Devolve todos os clientes referenciados por contas, no seu atributo clientes.
  - E.g. /banco-2/conta[@num-conta="A-401"]/id(@clientes)
    - Devolve todos os clientes da conta A-401
  - Semelhante às *path expression* das bases de dados objeto-relacional

Mais características do XPath

- Operador “|” para uniões
  - E.g. /banco-2/conta/id(@clientes) | /banco-2/emprestimo/id(@clientes)
    - Devolve os clientes com contas ou empréstimos
    - NOTA: O “|” não pode estar imbricado noutros operadores.
- Operador “//” para saltar vários níveis de uma árvore de uma só vez
  - E.g. /banco-2//nome
    - Devolve qualquer subelemento com nome nome que esteja dentro do elemento /banco-2, independentemente do número de níveis entre os dois.

Mais características do XPath

- Um passo no caminho (path) pode ir para o pai, irmãos, antecessores, descendentes (e não apenas para os filhos, como vimos até aqui). E.g.:
  - O "//", acima, denota todos os descendentes
  - ".." denota o pai.
  - "." denota o próprio nó.
- doc(nome) retorna a raiz do documento com nome "*nome*"
  - *E.g.* se o exemplo do banco estivesse contido num ficheiro banco.xml, então, a *path expression* doc('banco.xml')/banco-1/conta devolveria todos os elementos conta.
  - Permite a especificação de *path expressions* sobre outros documentos.

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

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 <num-conta> e </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 {}; nesse caso são avaliados
- A cláusula **let** não é absolutamente necessária nesta expressão, e a clausula **where** poderia ser incorporada na expressão XPath. A consulta acima 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 do SQL:

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

- A mesma consulta pode ser expressa com a seleção especificada como seleções 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>
```

Estrutura flat (banco)

```
<banco>
        <cliente>
              <nome-cliente>Luís</nome-cliente>
              <rua-cliente>5 de Outubro </rua-cliente>
              <local-cliente>Lisboa</ local-cliente>
        </cliente>
        <conta>
            <num-conta> A-102</num-conta>
            <balcao>Caparica</balcao>
            <saldo>400</saldo>
        </conta>
        <depositante>
            <num-conta>A-102</num-conta>
            <nome-cliente>Luís</nome-cliente>
        </depositante>
        ...
</banco>
```

Estrutura imbricada (banco-1)

```
<banco-1>
    <cliente>
      <nome-cliente> Luís              </nome-cliente>
      <rua-cliente>   5 de Outubro </rua-cliente>
      <local-cliente>  Lisboa          </ local-cliente>
      <conta>
                <num-conta> A-102     </num-conta>
                <agencia>    Caparica </agencia>
                <saldo>        400         </saldo>
      </conta>
      <conta>
                <num-conta> A-103     </num-conta>
                <agencia>    Lisboa </agencia>
                <saldo>        600         </saldo>
      </conta>
      <conta>
                …
      </conta>
    </cliente>
      …
</banco-1>
```

Documento XML de exemplo

```
<?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”>
        <nome-cliente> Maria </nome-cliente>
        <rua-cliente> R. 5 de Outubro </rua-cliente>
        <local-cliente> Porto </local-cliente>
    </cliente>
</banco-2>
```

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 subelementos / *tags*.

Ordenação em XQuery

- A cláusula **order by** pode ser usada em 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 **descending** para ordenar de forma descendente. E.g.:

```
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 XML Schema

```
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 * (e.g. em 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/xpath-3/
  - Recomendação XPath 3.1 (17 de Dezembro de 2015)
- http://www.w3.org/TR/xquery-3/
  - Recomendação XQuery 3.1 (17 de Dezembro de 2015)

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.di.fct.unl.pt/~jja/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

```
cliente {display: block; font-size: 25; font-weight: bold;}
nome-cliente {display: block; color:red; font-size: 25; font-weight: bold}
rua-cliente {display: block; color: green; font-size: 15; text-align: left}
local-cliente {display: block; font-size: 55}
…
```

- 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 seleção 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 seleciona, para output, valores específicos (no caso, nome-cliente)
- Para nós que não estejam de acordo com nenhum padrão existem padrões implícitos:
  - 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 diretamente 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="contaIdepositante"/>
```

    - Exemplo de output:

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

Recursão estrutural

- As ações 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>
```

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><b> Contas: </b></p>
        <xsl:apply-templates/>
    </body>
</xsl:template>
<xsl:template match = "/banco/conta">
    <p> A conta <b> <xsl:value-of select="num-conta" /> </b>
        da agencia <xsl:value-of select="agencia" />
        tem saldo <xsl:value-of select="saldo" />.
    </p>
</xsl:template>
<xsl:template match=“*”/>
</xsl:stylesheet>
```

Junções em XSLT

- As **keys** do XSLT permitem a indexação de elementos por valores de subelementos 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>
```

API (Application Program Interface)

- Há dois APIs standard para dados XML:
  - **SAX** (**S**imple **A**PI for **X**ML)
    - 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** (**D**ocument **O**bject **M**odel)
    - 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 atualizar a árvore DOM.

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

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.
  - Bases de Dados Relacionais
    - Dados têm que ser traduzidos para a forma relacional
    - Vantagem: SGBDs consistentes
    - 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/subelementos 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/subelemento

Representação em cadeias de caracteres (Cont.)

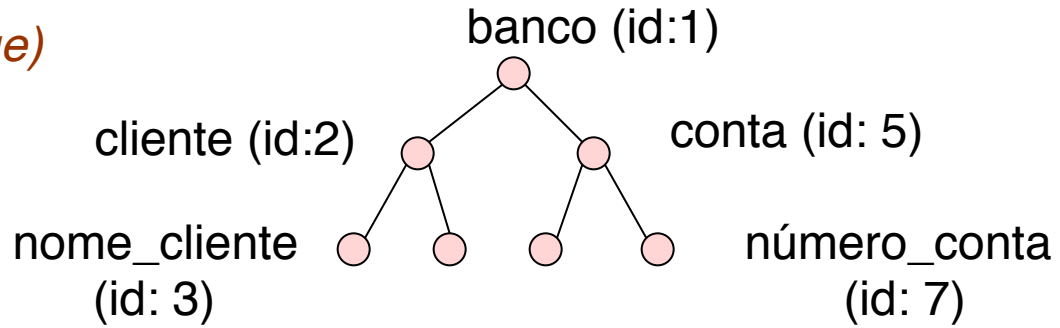
- Benefícios:
  - Pode guardar XML mesmo sem DTD
  - Desde que existam vários elementos de topo, as cadeias de caracteres são pequenas quando comparadas com o documento.
    - Permite acesso rápido a elementos individuais.
- Defeitos:
  - É necessário o processamento de strings para aceder aos valores 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 também pode ser guardada
- Todos os subelementos 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 subelementos complexos, guardar id do subelemento.
- Subelementos 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 SGBD já estão preparados para converter de/para XML.
- Alguns sistemas (e.g. Oracle) oferecem armazenamento nativo em XML usando o tipo de dados xml. São usadas estruturas de dados internas e índices especiais para aumentar a eficiência.