XML - eXtensible Markup Language

- Tópicos:
  - XML para transferência de dados
  - Estrutura hierárquica do XML
  - DTDs e XML Schema
  - Consultas de documentos XML:
    - Xpath
    - XQuery
- Bibliografia:
  - Capítulo 30 do livro recomendado (7ª edição – Cap. 23 na 6ª edição)

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:

  <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:

  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

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 ou o próprio nó, o mesmo que /descendant-or-self::node()/
  - ".." denota o pai, o mesmo que parent::node()
  - "." denota o próprio nó, o mesmo que self::node()
    - /banco-2/conta/saldo/.. vs. /banco-2/conta/saldo vs. /banco-2/conta[saldo]
    - /banco-2/conta[saldo > 400]/@num-conta[. = 'A-401']
- 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.

Eixos e Testes XPath

- A linguagem XPath permite muito mais formas de navegação e seleção de nós recorrendo a eixos e testes:

3.3.2.1 Axes

```
[41]  ForwardAxis  ::=  ("child" "::")
                      | ("descendant" "::")
                      | ("attribute" "::")
                      | ("self" "::")
                      | ("descendant-or-self" "::")
                      | ("following-sibling" "::")
                      | ("following" "::")
                      | ("namespace" "::")
[44]  ReverseAxis  ::=  ("parent" "::")
                      | ("ancestor" "::")
                      | ("preceding-sibling" "::")
                      | ("preceding" "::")
                      | ("ancestor-or-self" "::")
```

3.3.2.2 Node Tests

[Definition: A **node test** is a condition on the name, kind (element, attribute, text, document, comment, or processing instruction), and/or type annotation of a node. A node test determines which nodes contained by an axis are selected by a step.]

```
[46]  NodeTest   ::=  KindTest | NameTest
[47]  NameTest   ::=  EQName | Wildcard
[48]  Wildcard   ::=  "*"                          /* ws: explicit */
                      | (NCName ":*")
                      | ("*:" NCName)
                      | (BracedURILiteral "*")
[112] EQName     ::=  QName | URIQualifiedName
```

- Muito cuidado com os predicados [] com posições:
  - //b/*[2] = /descendant-or-self::node()/child::b/child::*[position()=2]
  - (//b/*)[2] = (/descendant-or-self::node()/child::b/child::*)[position()=2]

Casting e Comparações de Nós

- Casting de conjunto de nós para Booleano verdadeiro se for não vazio

  /banco-2/conta[saldo]

- Comparação de escalar com conjunto de nós verdadeiro se existir um valor textual de um dos nós que torne a comparação verdadeira

  /banco-2/cliente[* = 'Maria']

  (clientes que têm um elemento filho com valor textual 'Maria')

- Comparação entre dois conjuntos de nós verdadeira se existir um nó N1 do primeiro conjunto e um nó N2 do segundo conjunto para qual a condição é verdadeira para os valores textuais dos nós

  /banco-2/conta[@* = //cliente/*]

  Qual o significado?

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

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>
```

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 (21 de Março de 2017)
- http://www.w3.org/TR/xquery-3/
  - Recomendação XQuery 3.1 (21 de Março de 2017)

Dados Semi-estruturados

- Muitas aplicações necessitam de armazenar dados complexos cujos esquemas sofrem alterações frequentemente
- O requisito de ter toda a informação na primeira forma normal (tipos de dados atómicos) pode ser contraproducente
  - E.g., guardar os interesses de um perfil de um utilizador como um atributo multivalor pode ser mais simples e eficiente do que normalizá-lo
- O intercâmbio de dados pode beneficiar grandemente de dados em formatos semi-estruturados, na
  - troca de dados entre aplicações ou entre o back-end e o front-end de uma aplicação
  - Nos serviços Web utilizados atualmente, com dados complexos obtidos pelo front-end e apresentados recorrendo a uma aplicação móvel ou JavaScript
- JSON e XML são modelos de dados semi-estruturados amplamente utilizados hoje em dia que permitem adicionalmente representar dados hierarquicamente

Características dos Modelos de Dados Semi-estruturados

- Esquemas flexíveis
  - Representação **Wide Column**: permite que cada tuplo tenha um conjunto diferente de atributos, podendo-se adicionar novos atributos a qualquer momento
  - Representação **Sparse Column:** esquema tem um conjunto grande mas fixo de atributos, podendo cada tuplo armazenar um subconjunto deles
- Tipos de dados multivalor
  - **Conjuntos**, **multiconjuntos**
    - E.g.,: conjunto de interesses {‘basketball, ‘La Liga’, ‘cooking’, ‘anime’ }
  - **Mapas Chave-valor** (ou só **mapa** para abreviar)
    - Guardam um conjunto de pares chave-valor (key-value)
    - E.g., {(brand, Apple), (ID, MacBook Air), (size, 13), (color, silver)}
    - Operações em mapas: *put*(key, value), *get*(key), *delete*(key)
  - **Arrays**
    - Utilizados para aplicações científicas e de monitoração

Características dos Modelos de Dados Semi-estruturados

- Arrays
  - Utilizados para aplicações científicas e de monitoração
  - E.g., leituras obtidas a intervalos regulares podem ser representadas como um array de valores values em vez de pares (tempo, valor)
    - [5, 8, 9, 11] em vez de{(1,5), (2, 8), (3, 9), (4, 11)}
  - Time series database: disponibilizam suporte especializado para lidar com pares (tempo,valor), sendo exemplos conhecidos a InfluxDB e Prometheus
- Tipos de atributos multi-valor
  - Modelados usando um modelo que não obedece à primeira forma normal (*non first-normal-form* (*NFNF*)
  - Suportado pela maioria dos sistemas de bases de dados atuais

JSON

- Representação textual em uso generalizado
- Exemplo de dados JSON

```
{
        "ID": "22222",
        "name": {
                "firstname: "Albert",
                "lastname: "Einstein"
        },
        "deptname": "Physics",
        "children": [
                {"firstname": "Hans", "lastname": "Einstein" },
                {"firstname": "Eduard", "lastname": "Einstein" }
        ]
}
```

- Tipos: integer, real, string, e
  - *Objectos:* são mapas chave-valor, i.e. conjuntos de pares (nome atributo, valor)
  - Arrays também são mapas chave-valor (da posição para o valor)

JSON

- JSON é omnipresente nas aplicações atuais que trocam dados
  - Amplamente utilizado em web services
  - A maioria das aplicações modernas são construídas sobre/com web services
- Existem extensões SQL para
  - Suportar tipos JSON para armazenar dados JSON
  - Extrair dados de objetos JSON utilização caminhos
    - E.g. *V-> ID*, or *v.ID*
  - Gerar dados JSON a partir de dados relacionais
    - E.g. json.build_object('ID', 12345, 'name', 'Einstein')
  - Criação de coleções JSON usando agregação
    - E.g. a função de agregação json_agg do PostgreSQL
  - Sintaxe varia muito dependendo do SGBD
- JSON é extenso (menos que o XML, contudo)
  - Existem representações comprimidas como o BSON (Binary JSON)

Conclusões

- Análise à posteriori do programa de Bases de Dados
- Análise e discussão dos objetivos da disciplina
- Apresentação das disciplinas opcionais, oferecidas no MEI, da área de Bases de Dados

Objetivos de BD

- *Pretende-se dotar os alunos das bases necessárias à conceção, construção e análise de bases de dados relacionais*.
  - Serem capazes de conceber uma base de dados relacional
    - Fazer o seu design
    - Compreender o que devem ser as restrições
  - Estarem à vontade com a manipulação e interrogação de bases de dados com SQL
  - Conseguirem fazer um pequeno projeto de bases de dados, do princípio ao fim (trabalho)

Objectivos de BD (Cont.)

- Estarem cientes de que há mais coisas em bases de dados e, pelo menos, saberem o que são algumas delas
  - Linguagens de interface com bases de dados
    - Linguagens Embedded-SQL e seus mecanismos de base
    - Linguagens de interface ODBC e JDBC
  - XML
    - Para transferência de dados
    - Para interface de visualização de dados
    - Como modelo hierárquico de dados
  - Noções breves de: sessões, utilizadores e transações em bases de dados

Programa

- Introdução (1 aula – esta)
- Modelos de dados
  - Modelo de Entidades e Relações (2 aulas)
  - Modelo Relacional (1 aula)
- Normalização de Bases de Dados
  - Dependências funcionais e multi valor (1½ aula)
  - Formas normais: 3ª e de Boyce-Cood (1½ aula)
- Linguagens de interrogação e manipulação de bases de dados
  - Álgebra relacional (2 aulas)
  - SQL (3 aulas)
  - Outras linguagens (1 aula)
- Integridade de Bases de Dados
  - Integridade de referência (½ aula)
  - Asserções e triggers (½ aula)
- Interação com bases de dados (2 aulas)
  - Embedded SQL, ODBC, JDBC
  - Segurança e autorizações
  - Transações
- Discussão de outros modelos de bases de dados
  - XML (2 aulas)

E depois de BD?

- UC de Consolidação:
  - Sistemas de Bases de Dados
    - Funcionamento de SGBD (armazenamento, processamento de perguntas, protocolos de transações, BD distribuídas)
  - Representação do Conhecimento e Sistemas de Raciocínio
    - Linguagens formais para representação de modelos de dados, e dedução e imposição de restrições de integridade sobre os dados
- UC de Especialização:
  - Modelação de Dados
    - Linguagens para modelação de dados na Web (Linked Open Data)
    - Processamento Analítico de Dados (OLAP e Datewarehouses)
  - Processamento de Streams
    - Conceção de sistemas e manipulação de dados de eventos em tempo real
  - Prospeção de Dados
    - Dedução automática de padrões e relações entre dados
  - Tecnologias de Informação Geográfica
    - Armazenamento de dados georeferenciados
  - Visualização Interativa de Dados
    - Visualização de grandes quantidades de dados