Database Tutorial 6: XML, DTDs, XPath, XQuery

2015-01-27

1 Repetition

1.1 XML

- Tree structure
- Opening and closing Tags
- Case-sensitive
- Elements, Attributes and Text elements

Example:

```
<?xml version="1.0" standalone="yes" ?>

<!-- put the DTD here -->

<Hogwarts>
 <Rooms>
  <Room name="The_Dungeon" nrSeats="34" />
  <Room name="The_Cabin" nrSeats="163" />
 </Rooms>
 <Teachers>
  <Teacher name="Snape" room="The_Dungeon" >
   <Title>Professor</Title>
  </Teacher>
  <Teacher name="Hagrid" room="The_Cabin" />
 </Teachers>
 <Courses>
  <Course name="Potioncraft" teacher="Snape" nrStudents="28">
   <Class day="Monday" hour="10" />
  </Course>
  <Course name="Handling_of_Wild_Creatures" teacher="Hagrid">
   <Class day="Saturday" hour="13" />
   <Class day="Thursday" hour="7" />
  </Course>
 </Courses>
</Hogwarts>
```

1.2 DTD

- Rules for Elements, Attributes (and Entities).

- Keys and references (ID and IDREF)
- Choice (—) and Cardinalities (? at most one, + one ore more, * zero or more)

```
element ::= '<!ELEMENT' Name content '>'
content ::= 'EMPTY' | 'ANY' | #PCDATA | children
children::= children '+' | children '*' | children '?' | children '|' children | ...

attlist ::= '<!ATTLIST' Name (Name type default)* '>'
type    ::= 'CDATA' | 'ID' | 'IDREF' | ...
default ::= '#REQUIRED' | '#IMPLIED' | ('#FIXED')? value
```

Example:

```
<!DOCTYPE Hogwarts [

<!ELEMENT Hogwarts (Rooms, Teachers, Courses) >

<!ELEMENT Rooms (Room*) >
 <!ELEMENT Room EMPTY >
  <!ATTLIST Room
    name    ID    #REQUIRED
    nrSeats CDATA #IMPLIED >

<!ELEMENT Teachers (Teacher*) >
 <!ELEMENT Teacher (Title*) >
  <!ELEMENT Title (#PCDATA) >
  <!ATTLIST Teacher
    name ID    #REQUIRED
    room IDREF #REQUIRED >

<!ELEMENT Courses (Course*) >
 <!ELEMENT Course (Class*) >
  <!ATTLIST Course
    name       ID    #REQUIRED
    teacher    IDREF #REQUIRED
    nrStudents CDATA #IMPLIED >
  <!ELEMENT Class EMPTY >
   <!ATTLIST Class
     day    CDATA #REQUIRED
     hour   CDATA #REQUIRED >
]>
```

1.3 XPath

Symbol	Meaning
/	Root
.	Current Element
..	Parent Element
//*	All elements anywhere
elem1/elem2	Path
[*test*]	Condition (to filter)
@Att	Attribute

Example:

```
Find all courses that have at least 20 students:
//Course[@nrStudents >= 20]

List all professors at the school:
//Teacher[Title = "Professor"]

Find all rooms that are used on Mondays:
//Room[@name = //Teacher[@name = //Course[Class/@day = "Monday"]/@teacher]/@room]
```

1.4 XQuery

- Basic structure of an XQuery expression is:
  - FOR-LET-WHERE-ORDER BY-RETURN.
  - Called FLWOR expressions (pronounce as flower).
- A FLWOR expression can have any number of FOR (iterate) and LET (assign) clauses, possibly mixed, followed by possibly a WHERE clause and possibly an ORDER BY clause.
- Only required part is RETURN.

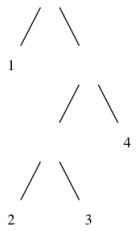
Example:

```
for $class in //Class[@day="Monday"]
for $teacher in $class/../@teacher
for $room in data(//Teacher[@name=$teacher]/@room)
return <room>{$room}</room>
```

2 Exercises

2.1 Exam HT2016

1. (8 points)
   A binary tree is a tree whose every node either branches to two binary trees or is a leaf, i.e. contains a value. Here is an example of a binary tree:



   (a) (3 points) Design a DTD for representing binary trees and nothing but binary trees. The branching nodes should not carry any information, whereas every leaf should carry a value that can be any string (#PCDATA).

   Solution:

```
<!DOCTYPE BT [
  <!ELEMENT BT ((BT,BT) | Leaf)>
  <!ELEMENT Leaf (#PCDATA)>
  ]>
```

(b) (3 points) Show an XML element representing the above example tree, and which is valid according to your DTD.

Solution:

```
<BT>
  <BT><Leaf>1</Leaf></BT>
  <BT>
    <BT>
      <BT><Leaf>2</Leaf></BT>
      <BT><Leaf>3</Leaf></BT>
    </BT>
    <BT><Leaf>4</Leaf></BT>
  </BT>
</BT>
```

(c) (2 points) Write an XPath query that returns all leaf elements of a binary tree. For the above example, it should return 1,2,3,4 (without any XML tags).

Solution:

```
//Leaf/*
```

2.2 Exam HT2014

1. (8 points)

```
<Question7>
  <Applicants>
    <Applicant name="Andersson" appNum="a1" />
    <Applicant name="Jonsson" appNum="a2" />
    <Applicant name="Larsson" appNum="a3" />
  </Applicants>
  <Choices>
    <Choice applicant="a1" code="MPSOF" choiceNum="1" meritScore="750" />
    <Choice applicant="a1" code="MPALG" choiceNum="2" meritScore="750" />
    <Choice applicant="a1" code="MPCSN" choiceNum="3" meritScore="800" />
    <Choice applicant="a2" code="MPALG" choiceNum="1" meritScore="700" />
    <Choice applicant="a3" code="MPCSN" choiceNum="1" meritScore="850" />
    <Choice applicant="a3" code="MPALG" choiceNum="2" meritScore="850" />
  </Choices>
</Question7>
```

(a) (2 points) Write a Document Type Definition (DTD) for the XML that is given above

Solution:

```
<!DOCTYPE Question7 [
  <!ELEMENT Question7 (Applicants, Choices)>
    <!ELEMENT Applicants (Applicant*)>
      <!ELEMENT Applicant EMPTY>
      <!ATTLIST Applicant
                name CDATA #REQUIRED
                appNum ID #REQUIRED >
    <!ELEMENT Choices (Choice*)>
      <!ELEMENT Choice EMPTY>
      <!ATTLIST Choice
                applicant IDREF #REQUIRED
                code CDATA #REQUIRED
                choiceNum CDATA #REQUIRED
                meritScore CDATA #REQUIRED>
]>
```

(b) (1 point) Write an XPath expression that finds Choice elements where the choice number is 1 and the merit score is greater than 800.

Solution:

```
//Choice[@choiceNum="1" and @meritScore>800]
```

(c) (2 points) The flexibility of XML enables us to nest elements in a more natural way than in the example shown at the top of this question. Write a piece of XML that contains the same information as in the example shown above, but which uses nesting, and avoids duplication of applicant identifiers.

Solution:

```
<Question7>
  <Applicant appNum="a1" name="Andersson">
    <Choice meritScore="750" choiceNum="1" code="MPSOF"/>
    <Choice meritScore="750" choiceNum="2" code="MPALG"/>
    <Choice meritScore="800" choiceNum="3" code="MPCSN"/>
  </Applicant>
  <Applicant appNum="a2" name="Jonsson">
    <Choice meritScore="700" choiceNum="1" code="MPALG"/>
  </Applicant>
  <Applicant appNum="a3" name="Larsson">
    <Choice meritScore="850" choiceNum="1" code="MPCSN"/>
    <Choice meritScore="850" choiceNum="2" code="MPALG"/>
  </Applicant>
</Question7>
```

(d) (3 points) Assuming that the XML shown above is in file exam.xml, write an XQuery expression that constructs your solution to part (c).

Solution:

```
<Question7>
  {
    let $d := doc("exam.xml")
```

```
    for $a in $d//Applicant
    let $choices := (
      for $c in $d//Choices/Choice[@applicant = $a/@appNum]
      return <Choice code="{$c/@code}"
                     choiceNum="{$c/@choiceNum}"
                     meritScore="{$c/@meritScore}" /> )
    return <Applicant name="{$a/@name}" appNum="{$a/@appNum}" >
              {$choices}
           </Applicant>
  }
</Question7>
```