CIND-110
Data Organization for Data Analysts

Lab Manual Module 8
XML, XPath and XQuery

Lead Instructor: Dr. Tamer ABDOU

Contents

Objectives

To practice the following aspects of XML:

- XML Schema.
- XPath Expressions.
- XQuery Language.

1. Installation

Install **JRE and XML Editor** by running the following commands.

Installation of JRE

```
sudo apt-get update
sudo apt-get install default-jre
```

Download XML Editor and Install

Installation of XML Editor

```
cd ~/Downloads
wget http://editix.com/download/editix2017.zip
unzip -d editix2017 editix2017.zip
```

Run the editor

```
sh ~/Downloads/editix2017/bin/editix.sh
```

From the menu bar, select $File->OpenDocument$ and choose $personal.xml$ from the "$editix$2017/$samples/xml$" folder.
Review the content of the XML file $personal.xml$ and its XML Schema document $personal.xsd$.
Notice how the schema (.XSD) defines how elements can be validly constructed.
The schema defines:

1. Which elements and attributes are permitted and in which order. Otherwise, an XML file is a relatively free set of elements and attributes.
2. The data types, default, and fixed values for elements and attributes.

2. XPath Expressions

XPath expressions allow us to select fragments of an XML document that meet some search criteria. The expressions consist of a path to the elements or attributes of interest and, optionally, conditions to be met in filtering the results.

- From the course shell download CompanyDB XML files: *Company.xml* and *company.xsd*
- Open *Company.xml* from editix2017 application.
- Open the XPath Panel by selecting from the menu $XML->XPathview$
- Write the following commands and observe the results

Practice XPath expressions

```
1  /companyDB/departments/department
2  /companyDB/employees/employee/@supervisor
3  /companyDB/employees/employee/lname/text()
4  /companyDB/employees/employee/*
5  /companyDB/employees/employee/@*
6  //dob
7  /companyDB/employees/employee[7]
8  /companyDB/employees/employee[1]/dependents/
     dependent[2]
9  /companyDB/employees/employee[1]/dependents/
     dependent[last()]
10 /companyDB/employees/employee[position()<=3]
11 /companyDB/employees/employee[minit="E"]
12 /companyDB/projects/project[
     @controllingDepartment>6]
13 /companyDB/employees/employee[starts-with(lname,"
     S")]
14 /companyDB/employees/employee[contains(address,"
     Philadelphia")]
15 //employee[@worksFor=7 and sex="M" and dependents
     /dependent[sex="M"]]
16 /companyDB/departments/department/dname/text() |
     /companyDB/projects/project/pname/text()
17 /companyDB/employees/employee[@worksFor=
     ancestor::companyDB/departments/department[
     dname="Administration"]/@dno]
```

3. XQuery Statements

XQuery statements allow us to construct a query program, similar to SQL, to yield elements and attributes of interest from an XML document. These statements are composed of *For*, *Let*, *Where*, *Order By*, and *Return* clauses called *FLWOR* queries. (A letter in the acronym stands for the first letter of each type of clause).

- Open the XQuery Panel by selecting from the menu XML -> XQuery builder.
- Write the following cpmmands and observe the results.

Practice XQuery statements

```
let $d:=doc("company.xml")
return $d//companyDB/employees/employee[2]

(2*3)-(8*7)

concat("Hello"," World")

matches("Monday","^.*da.*$")

current-time()

let $list:=(1,5,10,12,15)
return count($list)

let $d:=doc("company.xml")
return $d//companyDB/employees/employee[2]

let \$d:=doc("company.xml")//employee[@worksFor=6
   ]
return
<dept6Salary>{\$d/salary}
</dept6Salary>
```

3.1 FLWOR Queries

Get all projects.

```
1 let $d:=doc("company.xml") for $p in
2 $d/companyDB/projects/project
3 return $p
```

Get distinct project numbers of projects in which employees work.

```
1 <projects>
2 {
3 let $d:=doc("company.xml")
4 for $p in distinct-values(
5 $d/companyDB/employees/employee/projects/worksOn/
    @pno) return
6 <project>{$p}</project>
7 }
8 </projects>
```

using order by clause

```
1 <projects>
2 {
3 let $d:=doc("company.xml")
4 for $p in distinct-values(
5 $d/companyDB/employees/employee/projects/worksOn/
    @pno) order
6 by number($p) return <project>{$p}
7 </project>
8 }
9 </projects>
```

Get social security numbers of employees whose last name starts with "S".

```
let $d:=doc("company.xml") for $e in
$d/companyDB/employees/employee
where starts-with($e/lname,"S")
return <sssn>{$e/@ssn}</sssn>
```

Get last names and first names of employees in the "Research" department.

```
let $d:=doc("company.xml")
let
$r:=$d/companyDB/departments/department[dname="
   Research"]
for $e in $d/companyDB/employees/employee where
$e/@worksFor=$r/@dno
return
<ResearchEmp>{$e/lname}{$e/fname}</ResearchEmp>
```

Get employees who work more than 40 hours.

```
let $d:=doc("company.xml") for $e in
$d/companyDB/employees/employee where
sum($e/projects/worksOn/@hours)>40.0 return
<OverWorkedEmp>{$e/lname}
{$e/fname}<TotalHours>{sum($e/projects/worksOn/
   @hours)}
</TotalHours>
</OverWorkedEmp>
```

Get last names of employees without dependents

```
let $d:=doc("company.xml")
let $empsWithDeps :=
$d/companyDB/employees/employee[dependents] for $
   e in
$d/companyDB/employees/employee where
empty(index-of($empsWithDeps,$e))
return $e/lname
```

Get department names and the total number of employees working in the department

```
let $d:=doc("company.xml") for $r in
$d/companyDB/departments/department return
<deptNumEmps>{$r/dname}
<numEmps>{count(tokenize($r/employees/@essns,"\s+
   "))}
</numEmps>
</deptNumEmps>
```

Get last names of employees who work for a project located in “Houston”.

```
<empsWorkingOnHoustonProjects>
{ distinct-values( let
$d:=doc("company.xml")
for $r in $d/companyDB/projects/project[plocation
   ="Houston"]
return
for $e in $d/companyDB/employees/employee where
exists(index-of($r/workers/worker/@essn,$e/@ssn))
    return
$e/lname
)
}
</empsWorkingOnHoustonProjects>
```

Get last names of employees with dependents.

```
let $d:=doc("company.xml")
for $e in $d/companyDB/employees/employee[
   dependents] return
$e/lname
```

Having that: Low Income Group (earning to 40000), Middle-Income-Group (From 40000 to 60000) and High Income Group (earning more than 80000).

Get last names of employees from Milwaukee along with their income group:

```
{}
<IncomeGroup>
{
let $d:=doc("company.xml") for
$e in
$d/companyDB/employees/employee[contains(address,
   "Milwaukee")]
return <emp>{$e/lname}
<income>
{if ($e/salary >= 80000) then "High Income"
else if ($e/salary >=60000) then "Middle Income"
else "Low Income"
}
</income>
</emp>
}
</IncomeGroup>
```

Get employee names of employees who work on all projects located in “Houston”.

```
let $d:=doc("company.xml")
let $houstonProjs :=
$d/companyDB/projects/project[plocation="Houston"
   ]
for $e in $d/companyDB/employees/employee where
every $p in $houstonProjs satisfies (some $q in
$e/projects/worksOn satisfies
$p/@pnumber = $q/@pno)
return concat($e/fname,", ",$e/lname)
```