

Database Systems

- Chapter 12
 - Querying Semistructure Data

DATABASE
SYSTEMS

THE
COM CourseSmart

A First Course in Database Systems, Third Edition

Exit Reader



Search



Page 517

Chapter 12

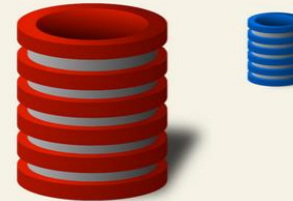
Programming Languages

Hector G
Jeffrey
Jennif

XPath and XQuery

YOU ARE REGISTERED FOR THIS COURSE

VIEW COURSEWARE



overview

[Return to main page for all Database mini-courses](#)

ABOUT THIS MINI-COURSE



Course Number

DB6

Price

Free



DB6

Courseware

Course Info

Discussion

Wiki

Progress

Readings

Software Guide

Getting Started

Querying XML

XPath Introduction

XPath Demo

XQuery Introduction

XQuery Demo

XML Course-Catalog XPath and XQuery Exercises

Exercise

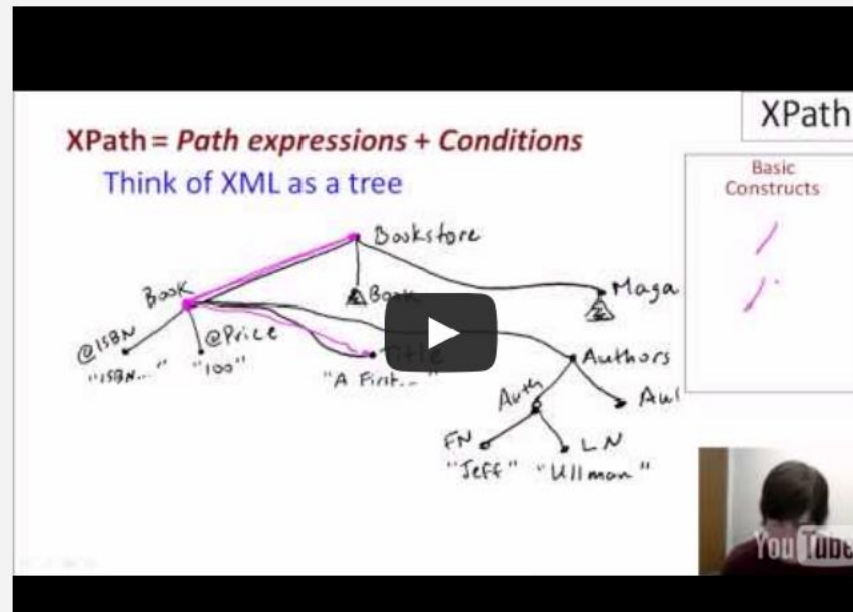
XML Course-Catalog XPath and XQuery Exercises Extras

XML World-Countries XPath and XQuery Exercises

Exercise

XML World-Countries XPath and XQuery

VIDEO



and I've introduced the concept of navigation axes.

But the real way to

learn and understand XPath is to run some queries.

So I urge you to watch the next video which is

a demo of XPath queries over

our bookstore data and then try some queries yourself.

XPath, the XML Path Language

- XPath was designed as a way to write down and interchange pointers into document trees from outside.
- XPath is a *domain specific language* (DSL) for XML.
- XPath 1 had its own data model, but many implementations use the DOM. XPath 2 uses the XDM.
- XPath is widely available: in Web browsers, in most programming languages, even in ISO SQL.
- XPath is compact, easy to learn and use, and powerful.

```

- <Course_Catalog>
  - <Department Code="CS">
    <Title>Computer Science</Title>
    + <Chair></Chair>
    + <Course Number="CS106A" Enrollment="1070"></Course>
    + <Course Number="CS106B" Enrollment="620"></Course>
    - <Course Number="CS107" Enrollment="500">
      <Title>Computer Organization and Systems</Title>
      + <Description></Description>
      - <Instructors>
        - <Lecturer>
          <First_Name>Julie</First_Name>
          <Last_Name>Zelenski</Last_Name>
        </Lecturer>
      </Instructors>
      + <Prerequisites></Prerequisites>
    </Course>
    + <Course Number="CS109" Enrollment="280"></Course>
    + <Course Number="CS124" Enrollment="60"></Course>
    + <Course Number="CS143" Enrollment="90"></Course>
    + <Course Number="CS145" Enrollment="130"></Course>
    + <Course Number="CS221" Enrollment="180"></Course>
    - <Course Number="CS228" Enrollment="110">
      + <Title></Title>
      + <Description></Description>
      - <Instructors>
        - <Professor>
          <First_Name>Daphne</First_Name>
          <Last_Name>Koller</Last_Name>
        </Professor>
      </Instructors>
    </Course>
  </Department Code="CS">
</Course_Catalog>

```

XPath Expressions

- XPath “expressions” are designed to be like Unix file paths:
`/Course_Catalog/Department/Title`
- Unlike file paths, an XML element can have more than one child with the same name.
- The above example returns a *list* of all `<Title>` elements directly contained in `<Department>` elements that are directly inside `<Course_Catalog>` elements.
- `/Course_Catalog/Department[1]/Title` returns Title elements inside `<Department>` elements that are the first child of `<Course_Catlog>` elements.

/Course_Catalog/Department/Title

```
-<Course_Catalog>
  -<Department Code="CS">
    <Title>Computer Science</Title>
    +<Chair></Chair>
    +<Course Number="CS106A" Enrollment="1070"></Course>
    +<Course Number="CS106B" Enrollment="620"></Course>
    -<Course Number="CS107" Enrollment="500">
      <Title>Computer Organization and Systems</Title>
      +<Description></Description>
      -<Instructors>
        -<Lecturer>
          <First_Name>Julie</First_Name>
          <Last_Name>Zelenski</Last_Name>
        </Lecturer>
      </Instructors>
      +<Prerequisites></Prerequisites>
    </Course>
    +<Course Number="CS109" Enrollment="280"></Course>
    +<Course Number="CS124" Enrollment="60"></Course>
    +<Course Number="CS143" Enrollment="90"></Course>
    +<Course Number="CS145" Enrollment="130"></Course>
    +<Course Number="CS221" Enrollment="180"></Course>
    -<Course Number="CS228" Enrollment="110">
      +<Title></Title>
      +<Description></Description>
      -<Instructors>
        -<Professor>
          <First_Name>Daphne</First_Name>
          <Last_Name>Koller</Last_Name>
        </Professor>
      </Instructors>
    </Course>
  </Department>
</Course_Catalog>
```

Your Query Result:

```
<Title>Computer Science</Title>
<Title>Electrical Engineering</Title>
<Title>Linguistics</Title>
```

doc("courses.xml")/Course_Catalog/Department/Title

```
-<Course_Catalog>
-  <Department Code="CS">
    <Title>Computer Science</Title>
    +<Chair></Chair>
    +<Course Number="CS106A" Enrollment="1070"></Course>
    +<Course Number="CS106B" Enrollment="620"></Course>
    -<Course Number="CS107" Enrollment="500">
        <Title>Computer Organization and Systems</Title>
        +<Description></Description>
        -<Instructors>
            -<Lecturer>
                <First_Name>Julie</First_Name>
                <Last_Name>Zelenski</Last_Name>
            </Lecturer>
        </Instructors>
        +<Prerequisites></Prerequisites>
    </Course>
    +<Course Number="CS109" Enrollment="280"></Course>
    +<Course Number="CS124" Enrollment="60"></Course>
    +<Course Number="CS143" Enrollment="90"></Course>
    +<Course Number="CS145" Enrollment="130"></Course>
    +<Course Number="CS221" Enrollment="180"></Course>
    -<Course Number="CS228" Enrollment="110">
        +<Title></Title>
        +<Description></Description>
        -<Instructors>
            -<Professor>
                <First_Name>Daphne</First_Name>
                <Last_Name>Koller</Last_Name>
            </Professor>
        </Instructors>
    </Course>
</Department>
</Course_Catalog>
```

Your Query Result:

```
<Title>Computer Science</Title>
<Title>Electrical Engineering</Title>
<Title>Linguistics</Title>
```


XPath Node Types and their Tests

- Here's how to test for each XPath node type:
 - *name* An element of that name
 - *@attr* An **attribute** called *attr* on the current node
 - `data(@attr)`: used to output the `@attr` value.
 - *** Any element

XPath Contexts

- XPath expressions are evaluated in a *context*; this includes the idea of a *current node*.
- XPath isn't a full stand-alone language: it is always used from a “host” language such as PHP.
- The host language defines the context when XPath is invoked;
 - the `current()` function returns the initial context node.
- The context is also changed by starting an expression with `/` (the root node) or `//` (root or any descendent) and in predicates.

XPath Predicates

- Any XPath sequence can be *filtered* by a *predicate*. A predicate is a test expression in [square brackets] evaluated in the context of some given node: if the predicate returns true then that node is kept.
 - A numeric predicate like [3] is short for [position() = 3]
 - Otherwise, the *effective boolean value* of the expression is determined, and the node is kept if the value is true.
- Example: Course entries for Department with title equal to Computer Science:

```
/Course_Catalog/Department[Title='Computer Science']/Course/Title
```

XPath Predicates

- Example: Course entries for Department with title equal to Computer Science:

```
Doc("courses.xml")/
```

```
Course_Catalog/Department[Title='Computer Science']/Course/Title
```

Your Query Result:

```
<Title>Programming Methodology</Title>
<Title>Programming Abstractions</Title>
<Title>Computer Organization and Systems</Title>
<Title>Introduction to Probability for Computer Scientists</Title>
<Title>From Languages to Information</Title>
<Title>Compilers</Title>
<Title>Introduction to Databases</Title>
<Title>Artificial Intelligence: Principles and Techniques</Title>
<Title>Structured Probabilistic Models: Principles and Techniques</Title>
<Title>Machine Learning</Title>
```

Effective Boolean Value

- In an expression:
 - An empty sequence is false;
 - Any sequence whose first item is a node is true;
 - A boolean value - `true()`, `false()`, 0 or 1, returns its value;
 - A string (or URI) is false if it has zero length, otherwise true;
 - Numeric values are false if they are 0 (zero) or NaN (Not a Number), and true otherwise.
- Note, `true()` and `false()` are boolean constants; `true` and `false` are node tests for `<true>` and `<false>` element names!

XPath Axes

- An XPath *axis* is a direction, like *up* or *forward*.
- Each / or // represents a “step” in the current direction, or axis; the default direction is “child” so that head/title refers to a title element that’s a child of a head element in the current context.
- You can change direction by naming an axis:
- /html/body/h1/following-sibling::h2
- This matches h2 elements in the “following-sibling” direction after the h1 element - *i.e.* at the same level in the tree.

XPath Axes

- `child::` The default, `/`, “directly contained within”
- `descendant::` Also written `//`, “anywhere underneath”
- `attribute::` Also written `@`, e.g. `attribute::*` or `@*`
- `self::` this node, e.g. `self::p` is true if this is a `<p>`
- `following-sibling::` After the context node but at the same level in the tree and with the same parent;
- `ancestor::` Goes upwards, e.g. `ancestor::div` returns a list of all the `<div>` elements anywhere directly above this node.

Relative paths

If the context node is *entry*

- .. gets you to the document root;

body/p[1] is the first paragraph;

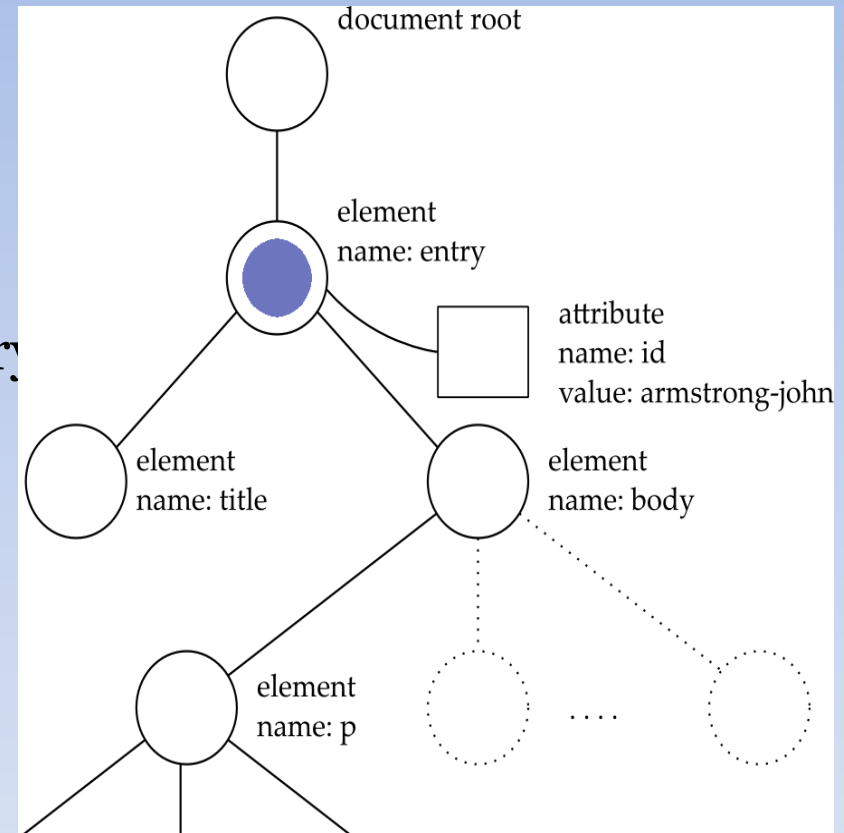
title/ancestor::entry is the entry

entry//p is all the p elements;

entry/@id is *armstrong-john*;

`/entry` is the entry node;

//body/p is the list of all p elements under <body>.



Example

- `doc("courses.xml")//Course/Title`

Your Query Result:

```
<Title>Programming Methodology</Title>
<Title>Programming Abstractions</Title>
<Title>Computer Organization and Systems</Title>
<Title>Introduction to Probability for Computer Scientists</Title>
<Title>From Languages to Information</Title>
<Title>Compilers</Title>
<Title>Introduction to Databases</Title>
<Title>Artificial Intelligence: Principles and Techniques</Title>
<Title>Structured Probabilistic Models: Principles and Techniques</Title>
<Title>Machine Learning</Title>
<Title>Digital Systems I</Title>
<Title>Digital Systems II</Title>
<Title>From Languages to Information</Title>
```

Example

`doc("courses.xml")/Course_Catalog//Course/ancestor::Department/Title`

Your Query Result:

```
<Title>Computer Science</Title>  
<Title>Electrical Engineering</Title>  
<Title>Linguistics</Title>
```

XPath Operators

- Operators take arbitrary XPath expressions. They are mostly the same as other languages: $3 + 1$, $3 * (2 + 5)$
 - Since $/$ is “step”, divide is `div` or `idiv` for integer division;
 - `mod` is remainder after division (modulo): $16 \text{ mod } 7$ gives 2.
- XPath 2 and later also have:
 - `to` gives a sequence of integers - e.g. $1 \text{ to } 10$
 - $a \ll b$, $a \gg b$ is true if a occurs before (after) b in the document.
 - (a, b, c) builds a sequence.

Operators Example

- Calculate Population Density

```
- <country name="Afghanistan" population="22664136" area="647500">
  <language percentage="11">Turkic</language>
  <language percentage="35">Pashtu</language>
  <language percentage="50">Afghan Persian</language>
</country>
<country name="Albania" population="3249136" area="28750"/>
- <country name="Algeria" population="29183032" area="2381740">
  - <city>
    <name>Algiers</name>
    <population>1507241</population>
  </city>
</country>
<country name="American Samoa" population="59566" area="199"/>
<country name="Andorra" population="72766" area="450"/>
<country name="Angola" population="10342899" area="1246700"/>
- <country name="Anguilla" population="10424" area="91">
  <language percentage="100">English</language>
</country>
- <country name="Antigua and Barbuda" population="65647" area="440">
  <language percentage="100">English</language>
</country>
- <country name="Argentina" population="34672996" area="2766890">
  - <city>
    <name>La Matanza</name>
    <population>1111811</population>
  </city>
  - <city>
    <name>Cordoba</name>
    <population>1208713</population>
  </city>
- <city>
```

Density

```
doc("countries.xml")//country  
  [(@population div @area)> 100]  
  /data(@name)
```

Your Query Result:

```
Bermuda Gibraltar Holy See Macau Malta Monaco Singapore
```

test

Firefox

file:///D:/Documents/...xml.json/products.xml

file:///D:/Document

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
-<Products>
  -<Maker name="A">
    -<PC model="1001" price="2114">
      <Speed>2.66</Speed>
      <RAM>1024</RAM>
      <HardDisk>250 </HardDisk>
    </PC>
    -<PC model="1002" price="995">
      <Speed>2.10</Speed>
      <RAM>512 </RAM>
      <HardDisk>250 </HardDisk>
    </PC>
    -<Laptop model="2004" price="1150">
      <Speed>2.00</Speed>
      <RAM>512 </RAM>
      <HardDisk>60</HardDisk>
      <Screen>13.3</Screen>
    </Laptop>
    -<Laptop model="2005" price="2500">
      <Speed>2.16</Speed>
      <RAM>1024</RAM>
      <HardDisk>120</HardDisk>
      <Screen>17.0</Screen>
    </Laptop>
  </Maker>
  -<Maker name="E">
    -<PC model="1011" price="959">
      <Speed>1.86</Speed>
```

Firefox

file:///D:/Documents/...xml.json/products.xml

file:///D:/Document

```
-<Maker name="E">
  -<PC model="1011" price="959">
    <Speed>1.86</Speed>
    <RAM>2048</RAM>
    <HardDisk>160</HardDisk>
  </PC>
  -<PC model="1012" price="649">
    <Speed>2.80</Speed>
    <RAM>1024</RAM>
    <HardDisk>160</HardDisk>
  </PC>
  -<Laptop model="2001" price="3673">
    <Speed>2.00</Speed>
    <RAM>2048</RAM>
    <HardDisk>240</HardDisk>
    <Screen>20.1</Screen>
  </Laptop>
  -<Printer model="3002" price="239">
    <Color>>false</Color>
    <Type>laser</Type>
  </Printer>
</Maker>
-<Maker name="H">
  -<Printer model="3006" price="100">
    <Color>>true</Color>
    <Type>ink-jet</Type>
  </Printer>
  -<Printer model="3007" price="200">
    <Color>>true</Color>
    <Type>laser</Type>
  </Printer>
</Maker>
</Products>
```

Questions 12.1.1

- a. Find the amount of RAM on each PC.
- b. Find the price of each product of any kind.
- c. Find all the printer elements.
- d. Find the makers of laser printers.
- e. Find the makers of PC ' s and / or laptops.
- f. Find the model numbers of PC ' s with a hard disk of at least 2 0 0 gigabytes.
- g. Find the makers of at least two PC's.

Questions 12.1.1

a) Find the amount of RAM on each PC.

a) //PC/RAM

<RAM>1024</RAM>

<RAM>512</RAM>

<RAM>2048</RAM>

<RAM>1024</RAM>

Questions 12.1.1

b) Find the price of each product of any kind.

b) //@price

2114

995

1150

2500

959

649

3673

239

100

200

Questions 12.1.1

c) Find all the printer elements.

c) //Printer

```
<Printer model="3002" price="239">...</Printer>
```

```
<Printer model="3006" price="100">...</Printer>
```

```
<Printer model="3007" price="200">...</Printer>
```

Questions 12.1.1

d) Find the makers of laser printers.

d) /Products/Maker[Printer/Type = "laser"]/@name

E

H

e) Find the makers of PC ' s and / or laptops.

e) /Products/Maker[PC or Laptop]/@name

A

E

Questions 12.1.1

f) Find the model numbers of PC ' s with a hard disk of at least 2 0 0 gigabytes.

f) //PC[HardDisk >= 200]/@model

1001

1002

g) Find the makers of at least two PC's.

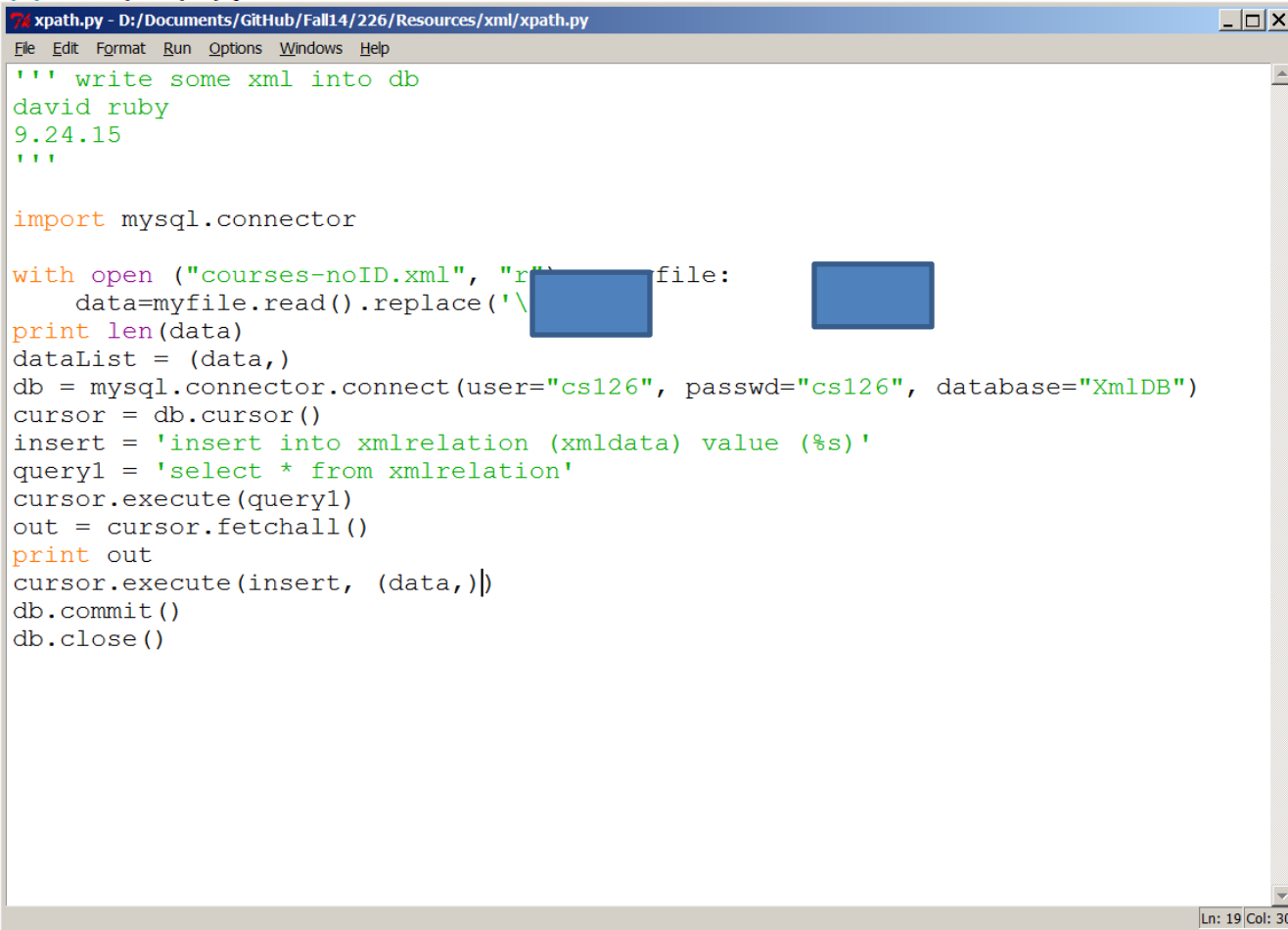
g) /Products/Maker[count(PC) >= 2]/@name

A

E

Xpath in MySql

```
1 # xml
2 • create database if not exists XmlDB;
3 • use XmlDB;
4
5 • create table IF NOT EXISTS xmlrelation (xmldata varchar(10000));
```



```
xpath.py - D:/Documents/GitHub/Fall14/226/Resources/xml/xpath.py
File Edit Format Run Options Windows Help

''' write some xml into db
david ruby
9.24.15
'''

import mysql.connector

with open ("courses-noID.xml", "r") as myfile:
    data=myfile.read().replace('\n', '\t')
print len(data)
dataList = (data,)
db = mysql.connector.connect(user="cs126", passwd="cs126", database="XmlDB")
cursor = db.cursor()
insert = 'insert into xmlrelation (xmldata) value (%s)'
query1 = 'select * from xmlrelation'
cursor.execute(query1)
out = cursor.fetchall()
print out
cursor.execute(insert, (data,))
db.commit()
db.close()
```

In these exercises, you will be working with a small XML data set drawn from the Stanford course catalog. There are multiple departments, each with a department chair, some courses, and professors and/or lecturers who teach courses. The XML data is [here](#).

Instructions: Each problem asks you to write a query in XPath or XQuery. When you click "Check Answer" our back-end runs your query against the sample database using Saxon. It displays the result and compares your answer against the correct one. When you're satisfied with your solution for a given problem, click the "Submit" button to check your answer.

You may perform these exercises as many times as you like, so we strongly encourage you to keep working with them until you complete the exercises with full credit.

Q1 (1/1 point)

Return all Title elements (of both departments and courses).

Note: Your solution will need to reference ***doc("courses.xml")*** to access the data.


```

1  <?xml version="1.0" ?>
2
3  <Course_Catalog>
4
5      <Department Code="CS">
6
7          <Title>Computer Science</Title>
8
9          <Chair>
10             <Professor>
11                 <First_Name>Jennifer</First_Name>
12                 <Last_Name>Widom</Last_Name>
13             </Professor>
14         </Chair>
15
16         <Course Number="CS106A" Enrollment="1070">
17             <Title>Programming Methodology</Title>
18             <Description>Introduction to the engineering of computer applications emphasizing modern software engineering principles.</D
19             <Instructors>
20                 <Lecturer>
21                     <First_Name>Jerry</First_Name>
22                     <Middle_Initial>R.</Middle_Initial>
23                     <Last_Name>Cain</Last_Name>
24                 </Lecturer>
25                 <Professor>
26                     <First_Name>Eric</First_Name>
27                     <Last_Name>Roberts</Last_Name>
28                 </Professor>
29                 <Professor>
30                     <First_Name>Mehran</First_Name>

```

ExtractValue()

MySQL 5.7 Reference Manual :: 12 Functions and Operators :: 12.11 XML Functions

12.11 XML Functions

Table 12.15 XML Functions

Name	Description
<u>ExtractValue()</u>	Extracts a value from an XML string using XPath notation
<u>UpdateXML()</u>	Return replaced XML fragment

- select ExtractValue(xmldata, "<my query>")
from relation xmlrelation;

```





1  <?xml version="1.0" ?>
2
3  <Course_Catalog>
4
5      <Department Code="CS">
6
7          <Title>Computer Science</Title>
8
9          <Chair>
10             <Professor>
11                 <First_Name>Jennifer</First_Name>
12                 <Last_Name>Widom</Last_Name>
13             </Professor>
14         </Chair>
15
16         <Course Number="CS106A" Enrollment="1070">
17             <Title>Programming Methodology</Title>
18             <Description>Introduction to the engineering of computer applications emphasizing modern software engineering principles.</D
19             <Instructors>
20                 <Lecturer>
21                     <First_Name>Jerry</First_Name>
22                     <Middle_Initial>R.</Middle_Initial>
23                     <Last_Name>Cain</Last_Name>
24                 </Lecturer>
25                 <Professor>

```

```

10 • select ExtractValue(xmldata, "//Course[@Enrollment>800]/Title") from xmlrelation;
11

```

sult Grid |   Filter Rows: | Export:  | Wrap Cell Content: 

```

ExtractValue(xmldata,
"//Course[@Enrollment>800]/Title")

```

Programming Methodology

Database Systems

- Chapter 12
 - XQuery & XSLT

DATABASE SYSTEMS THE COMPLETE BOOK SECOND EDITION

Hector Garcia-Molina
Jeffrey D. Ullman
Jennifer Widom

CourseSmart

A First Course in Database Systems, Third Edition

Exit Reader



Search



Page 517

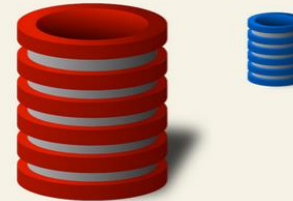
Chapter 12

Programming Languages for XML

XPath and XQuery

YOU ARE REGISTERED FOR THIS COURSE

[VIEW COURSEWARE](#)



[overview](#)

[Return to main page for all Database mini-courses](#)

[ABOUT THIS MINI-COURSE](#)



[Course Number](#)

DB6

[Price](#)

Free



Hello, XQuery!

- XQuery is a *domain specific language* for processing querying XML-native databases
 - and data structures that look like type-annotated sequences and trees.
- XQuery expressions operate over instances of the XQuery and XPath Data Model (XDM).
- XQuery implementations can be very fast (petabytes of data, response times in a few milliseconds), rather like ISO SQL used by relational databases.

XQuery, XPath and XSLT

- XQuery *extends* XPath.
 - Every valid XPath expression is also an XQuery expression, in the right context.
- XSLT *uses* XPath:
 - you embed XPath expressions in XSLT elements.
- Today, XSLT, XPath and XQuery all use the same underlying data model, the XDM.

XQuery and XPath Differences

- XQuery does not have a default context item: you have to identify the document or collection of documents explicitly.

- The following XPath expression:

`/dictionary/entry[6]`

might appear like this in XQuery:

`doc("dictionary.xml")/dictionary/entry[6]`

- XQuery is a *bigger* language, and, unlike XPath, can modify existing documents and construct new ones.

How XQuery Is Used In Practice

- Standalone:
 - as a program that reads an XQuery expression and one or more XML documents and prints a result.
- As part of another language, such as SQL.
- As a library or API from Java, C++, PHP or other languages.
- As the query language for an XML-native database.
- Embedded inside an application or product, such as a customer relationship manager or a stock quote program, or even in mobile devices.

First XQuery Example

- Pretty much any XQuery implementation can run this:

```
doc("courses.xml")//title
```

- This is a complete XQuery expression and returns all `<title>` elements from the sample document.

FLWOR, Modules, Functions

- For XQuery the central part is the FLWOR expression (pronounced *flower expression*).
- User-defined functions, and function libraries (modules) are also major parts of XQuery in practice.

FLWOR Expressions

- **FLWOR** is made up of the following parts:
- **F**or: loop through items
- **L**et, to create bindings;
- **W**here, to filter results;
- **O**der by, to sort results;
- **R**eturn, to return results.
- You must have one or more *for* and/or *let*, and one *return*.

FLWOR Example

```
for $country in doc("countries.xml")//country
where $country/@population<10000
order by $country/@population
return data($country/@name)
```

- This query returns the name attribute from each `<country>` element in the long sample document,
 - where the entry's *population* attribute is less than 10000.
 - The results are sorted by the *population* attribute as a string.

Your Query Result:

```
Niue Norfolk Island Falkland Islands Svalbard Pitcairn Islands Cocos Islands Saint Helena
Saint Pierre and Miquelon Christmas Island Holy See
```

FLWOR Example

```
for $country in doc("countries.xml")//country
where $country/@population<10000
order by xs:integer($country/@population)
return data($country/@name)
```

- This query returns the name attribute from each `<country>` element in the long sample document,
 - where the entry's *population* attribute is less than 10000.
 - The results are sorted by the *population* attribute as an integer.

Your Query Result:

```
Pitcairn Islands Cocos Islands Christmas Island Holy See Niue Norfolk Island Falkland
Islands Svalbard Saint Helena Saint Pierre and Miquelon
```

FLWOR Example With Let

- The previous example did not use a *let*; here's a new version with a let clause:

```
for $country in doc("countries.xml")//country
let $pop := data($country/@population)
where $pop<10000
order by xs:integer($country/@population)
return data($country/@name)
```

- This version introduces a *binding*, a name for the expression `xs:integer($dude/@died)`
- Reusing the variable for the order by clause means the list is now sorted properly. Variables can help reduce bugs.

FLWOR works on “tuples”

- The *for* expression generates a list of “tuples” -

```
for $a in 1 to 5, $b in ('a', 'b', 'c')  
return <e a="{ $a}" b="{ $b}" />
```

- The {curly braces} are used to introduce nested XQuery expressions inside constructed elements and attributes.
- The query processor evaluates the *return* clause for every combination of \$a and \$b. It can do the evaluation in any order as long as the result is in the right order, like this:

```
<e a="1" b="a"/>  <e a="1" b="b"/>  <e a="1" b="c"/>  
<e a="2" b="a"/>  <e a="2" b="b"/>  <e a="2" b="c"/>  
<e a="3" b="a"/>  . . . .
```


Illegal Output

- The examples so far have produced sequences of elements, but XML requires a single outermost element in every document.
- You can use a wrapper *element constructor* and curly braces inside it, to make a legal document, like this:

```
<myNiceResults>{  
    for $a in 1 to 5, $b in ('a', 'b', 'c')  
    return <e a="{ $a}" b="{ $b}" />  
}</myNiceResults>
```

Example

```
for $country in  
doc("countries.xml")//country  
where $country/@population<10000  
order by $country/@population  
return
```

```
<country>  
<name>{$country/@name}</name>  
<population>{$country/@population}</population>  
</country>
```

Your Query Result:

```
<country>
  <name name='Niue' />
  <population population='2174' />
</country>
<country>
  <name name='Norfolk Island' />
  <population population='2209' />
</country>
<country>
  <name name='Falkland Islands' />
  <population population='2374' />
</country>
<country>
  <name name='Svalbard' />
  <population population='2715' />
</country>
<country>
  <name name='Pitcairn Islands' />
  <population population='56' />
</country>
<country>
  <name name='Cocos Islands' />
  <population population='609' />
</country>
<country>
  <name name='Saint Helena' />
  <population population='6782' />
</country>
```

Example

```
for $country in  
doc("countries.xml")//country  
where $country/@population<10000  
order by  
xs:integer($country/@population)  
return
```

```
<country>  
  <name>{$country/@name}</name>  
  <population>{$country/@population}</population>  
</country>
```

Your Query Result:

```
<country>
  <name name='Pitcairn Islands' />
  <population population='56' />
</country>
<country>
  <name name='Cocos Islands' />
  <population population='609' />
</country>
<country>
  <name name='Christmas Island' />
  <population population='813' />
</country>
<country>
  <name name='Holy See' />
  <population population='840' />
</country>
<country>
  <name name='Niue' />
  <population population='2174' />
</country>
<country>
  <name name='Norfolk Island' />
  <population population='2209' />
</country>
<country>
  <name name='Falkland Islands' />
  <population population='2374' />
</country>
<country>
```

Your Query Result:

```
<country>
  <name name='Niue' />
  <population population='2174' />
</country>
<country>
  <name name='Norfolk Island' />
  <population population='2209' />
</country>
<country>
  <name name='Falkland Islands' />
  <population population='2374' />
</country>
<country>
  <name name='Svalbard' />
  <population population='2715' />
</country>
<country>
  <name name='Pitcairn Islands' />
  <population population='56' />
</country>
<country>
  <name name='Cocos Islands' />
  <population population='609' />
</country>
<country>
  <name name='Saint Helena' />
  <population population='6782' />
</country>
```

Some Actual XQuery Engines

- XQuery engines that parse the XML each time:
 - Saxon implements XSLT and XQuery in Java, .Net and JavaScript.
 - Qizx/open is a Java engine that does not use a database (in the free version).
- XML-Native databases:
 - BaseX, Sedna, dbxml, eXist are open source.
 - MarkLogic has a commercial native-XML database; EMC X-Hive is another.
- XQuery in Relational Databases:
 - Oracle, Microsoft SQL Server and IBM DB2 support XQuery directly.

Element Constructors

- Element constructors in XQuery look like fragments of XML,
 - They make new nodes that are not in any document tree.
- XPath alone cannot do this:
 - XPath only returns pointers into an existing document tree, not new nodes.
 - This is why XPath can't return an element without its children.
- Inside element constructors, use {braces} to enclose XQuery expressions.
 - These expressions have no default context item.

Element Constructor Example

- At its simplest, an element constructor just looks like XML.
- The following is a complete XML Query (with no prolog):

```
<entry id="jim">  
  <title>Armstrong, James</title>  
</entry>
```

- If you add braces, the constructor can contain expressions:

```
<entry id="jim" on="{ fn:current-date() }">  
  <title>{  
    doc("dates.xml")//entry[@id eq "jim"]/title  
  }</title>  
</entry>
```


Element Constructor as a Value

- You can use an element constructor anywhere you can use any other value in XQuery:

```
let $jim := <entry id="jim">
  <title>Armstrong, James</title>
</entry>
return $jim//title
```

- You can include sub-expressions, too:

```
let $nums := <numbers>{
  for $i in 1 to 10 return <n>{$i * $i}</n>
}</numbers>
return $nums/n[position() gt 3]
```

Computed Element Constructors

- Alternate syntax:

```
let $nums := element numbers {  
    for $i in 1 to 10  
    return element n {$i * $i}  
}  
return $nums/n[position() gt 3]
```

- You can mix the styles, but this can be confusing:

```
let $nums := element numbers {  
    for $i in 1 to 10 return <n>{$i * $i}</n>  
}  
return $nums/n[position() gt 3]
```

Computed Constructors

- The computed element constructors get their name because you can have an expression instead of a name:

```
let $name := "sock"  
return element { $name } { "argyle" }
```

- The braces show that it's an expression. You can also have computed attribute constructors:

```
element sock {  
  attribute clean { "yes" },  
  "argyle"  
}
```

Result: <sock clean="yes">argyle</sock>

Where and Predicates

- If you have only one bound variable in your FLWOR expression, you could use a predicate:

```
for $a in (1 to 11)[. mod 2 eq 0]  
return $a (: a sequence of even numbers :)
```

- Predicates affect a single sequence, but the where clause operates on tuples, so you can't easily use predicates if you have more than one bound variable.
- Some implementations optimize predicates and where clauses differently, so there may also be performance differences.

The *order by* Clause 1

- Use the *order by* clause to sort the tuples. The syntax is:
[stable] order by *expr* [ascending] [empty least]
- Use *ascending* or *descending* for smallest first or largest first.
- Use *empty least* or *empty greatest* to say where null values go in the result: all at the start or all at the end.
- The value of *expr* is used as a sort key for each tuple.
- If the expression has the same value for two tuples, the *stable* keyword forces those tuples to remain in their original order.

The *return* Clause

- The *return* clause is evaluated once for each tuple that has not been filtered out by a false *where* clause.
- The result of FLWOR expression is a sequence of items, each item generated by one evaluation of *return*.
- XQuery sequences do not nest:
 - nested sequences are flattened into a single sequence containing all the items:

(1, 2, 3, (4, (5), 6)) is (1, 2, 3, 4, 5, 6)

Grouping 1

- Let's sort a dictionary so everyone born in the same year is together:
- We'll group by the *born* attribute on each entry, and also sort the groups by the same value:

```
for $e in /dictionary/letter/entry[@born]
let $born := xs:integer($e/@born)
group by $born stable order by $born ascending
return <g born="{ $born }">{$e}</g>
```

- Now each time the *return* expression is evaluated, *\$e* will be a sequence of all entry elements with the same *born*.

Grouping 2

- The results look like this (showing just entry titles here):

<g born="100">Palafox, John De</g>

<g born="102">Theophrastus</g>

. . .

<g born="1614">

Annesley, Arthur

Carrenno De Miranda, Don Juan

Dod, John

. . .

</g>

. . .

Try and Catch (XQuery 3)

- The *try* and *catch* expressions evaluate an expression and allow the *catch* expression to handle the error.
- You can only catch dynamic errors, not (for example) syntax errors or obvious static errors.

```
for $i in (1, 2, 3, 0, 12)
return try { 12 div $i }
catch * { 42 }
```

- result: 12 6 4 42 1

Switch expressions (XQuery 3)

- Use *switch* to choose between multiple choices.
for \$word in ("apple", "boy", "girl", "orange")
return concat(
 switch (substring(\$word, 1, 1))
 case "a" case "e" case "i" case "u" case "o"
 return "an"
 default return "a",
 " ", \$word, "
")
- Each *case* is followed by a single expression.
- The *default* value is used if the switch expression is not `deep-equal()` to any of the case expressions.
- Result: an apple a boy a girl an orange

Equality in XPath

- The expression `fred = george`, where *fred* and *george* are nodelists or sequences, is true if-and-only-if some item in *fred* is equal to some item in *george*.
- This is called *implicit existential quantification* in logic.
 - `(1, 2, 3) = (2, 4, 6, 8)` is true.
 - `(1, 2, 3) != (2, 4, 6, 8)` is true because `1 != 2` (1 not equal to 2).
 - `not((1, 2, 3) = (2, 4, 6, 8))` is false - `not()` negates its argument.
 - `() = ()` is false (empty sequences, no items in common).
- `/play/scene[character = "Caliban"]` finds all the scenes in which one or more of the characters is called Caliban (probably in *The Tempest*).

Variables

```
for $born in /dictionary/entry/@born  
  return xs:integer(@born) idiv 100
```

```
let $np := count(//p) return //entry[@died = $np]
```

Expressions

- “for”, “let” (already mentioned).
- if (e1) then e2 else e3; the else part is required but you can use the empty sequence, (), in many cases.
- “some...satisfies” and “every...satisfies” (boolean), e.g.

`some entry in //entry satisfies @born = @died`

- Define functions inline and also refer to functions.
- `matches()` and `replace()` use *regular expressions*.
- Many more functions added, e.g. `sqrt()`. See Book Appendix.

Q6 (1/1 point)

Return the countries with the highest and lowest population densities. Note that because the "/" operator has its own meaning in XPath and XQuery, the division operator is infix "div". To compute population density use "(@population div @area)". You can assume density values are unique. The result should take the form:

```
<result>
  <highest density="value">country-name</highest>
  <lowest density="value">country-name</lowest>
</result>
```

Note: Your solution will need to reference ***doc("countries.xml")*** to access the data.

(This problem is quite challenging; congratulations if you get it right.)

```

1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <countries>
4   <country name="Afghanistan" population="22664136" area="647500">
5     <language percentage="11">Turkic</language>
6     <language percentage="35">Pashtu</language>
7     <language percentage="50">Afghan Persian</language>
8   </country>
9   <country name="Albania" population="3249136" area="28750"/>
10  <country name="Algeria" population="29183032" area="2381740">
11    <city>
12      <name>Algiers</name>
13      <population>1507241</population>
14    </city>
15  </country>
16  <country name="American Samoa" population="59566" area="199"/>
17  <country name="Andorra" population="72766" area="450"/>
18  <country name="Angola" population="10342899" area="1246700"/>
19  <country name="Anguilla" population="10424" area="91">
20    <language percentage="100">English</language>
21  </country>
22  <country name="Antigua and Barbuda" population="65647" area="440">
23    <language percentage="100">English</language>
24  </country>
25  <country name="Argentina" population="34672996" area="2766890">
26    <city>
27      <name>La Matanza</name>
28      <population>1111811</population>
29    </city>
30    <city>
31      <name>Cordoba</name>
32      <population>1208713</population>
33    </city>

```

Q6 (1/1 point)

Return the countries with the highest and lowest population densities. Note that because the "/" operator has its own meaning in XPath and XQuery, the division operator is infix "div". To compute population density use "(@population div @area)". You can assume density values are unique. The result should take the form:

```
<result>
  <highest density="value">country-name</highest>
  <lowest density="value">country-name</lowest>
</result>
```

Note: Your solution will need to reference ***doc("countries.xml")*** to access the data.

(This problem is quite challenging; congratulations if you get it right.)

- Calculate density
- Take max
- Take min

Q6 (1/1 point)

Return the countries with the highest and lowest population densities. Note that because the "/" operator has its own meaning in XPath and XQuery, the division operator is infix "div". To compute population density use "(@population div @area)". You can assume density values are unique. The result should take the form:

```
<result>
  <highest density="value">country-name</highest>
  <lowest density="value">country-name</lowest>
</result>
```

Note: Your solution will need to reference **`doc("countries.xml")`** to access the data.

(This problem is quite challenging; congratulations if you get it right.)

- Calculate density for all countries:
`doc("countries.xml")//country/(@population div @area)`
- Take max
 - use the Xquery function: `max()`
`max(doc("countries.xml")//country/(@population div @area))`
- Assign the value to a XQuery Variable
`let $x=max(doc("countries.xml")//country/(@population div @area))`

Element Constructors

Q6 (1/1 point)

Return the countries with the highest and lowest population densities. Note that because the "/" operator has its own meaning in XPath and XQuery, the division operator is infix "div". To compute population density use "(@population div @area)". You can assume density values are unique. The result should take the form:

```
<result>
  <highest density="value">country-name</highest>
  <lowest density="value">country-name</lowest>
</result>
```

Note: Your solution will need to reference `doc("countries.xml")` to access the data.

(This problem is quite challenging; congratulations if you get it right.)

- We need to construct XML to match the desired output:

```
<highest density="value">country-name</highest>
```

Element Constructors

- We need to construct XML to match the desired output:

```
<highest density="value">country-name</highest>
```

```
<highest density='{ $x } '>
```

```
{ doc("countries.xml")//country[(@population div @area)=$x]/data(@name) }
```

```
</highest>
```

Element Constructors

- We need to construct XML to match the desired output:

<result>

<highest density="value">country-name</highest>

<lowest density="value">country-name</lowest>

</result>

PARTIAL SOLUTION IN XQUERY:

<result>

{

let \$x := max(doc("countries.xml")//country/(@population div @area))

return

<highest density='{ \$x } '>

{ doc("countries.xml")//country[(@population div @area)=\$x]/data(@name) }

</highest>

}

</result>

Partial Solution

```
1 <result>
2
3   {let $x := max(doc("countries.xml")//country/(@population div @area))
4   return
5   <highest density='{ $x } '>
6     {doc("countries.xml")//country[(@population div @area)=$x]/data(@name)}
7   </highest>}
8
9 </result>
10
```

 Incorrect

Incorrect

Your Query Result:

```
<result>
  <highest density='31052.3125'>Macau</highest>
</result>
```