4D2b – Querying XML Documents with XQuery

Owen.Conlan@scss.tcd.ie
What is XQuery?

• Focuses on retrieval of information from XML documents
  – Update is considered separately
• XQuery 1.0 W3C Specifications
  – www.w3c.org/XML/Query
• Used in conjunction with XPath
For-Let-Where-OrderBy-Return: “FLWOR”
expressions
(pronounced “FLOWER”)

1. One or more FOR and/or LET expressions
2. Optional WHERE clause
3. Optional ORDER BY clause
4. RETURN clause
LET Clause

- LET <variable> := <path expression>,...
  - Variable “binds to” the set returned by path expression
  - Does not iterate over set like the FOR clause does
  - It cannot be redefined within the scope of the function
Example LET Clause

<?xml version="1.0"?>
<assessments>
    <student name="Smith">
        <mark thecourse="4BA5"> 99 </mark>
        <mark thecourse="4BA1"> 75 </mark>
    </student>
    <course name="4BA1" takenby="Smith,Jones">
        <mark>60</mark>
    </course>
    <course name="4BA5" takenby="Smith,Bond">
        <mark>70</mark>
    </course>
</assessments>

XQuery

let $c:= doc("data/tcd.xml")/assessments/course/mark
return
    <list_of_avg_course_marks>
    {$c}
    </list_of_avg_course_marks>

Result

<list_of_avg_course_marks>
    <mark>60</mark>
    <mark>70</mark>
</list_of_avg_course_marks>
FOR Clause

FOR <variable> IN <path expression>, ...

- Variable (starting with $) “binds to” (in turn) each object in the set returned by path expression
- For each variable value the rest of FLOWR expression is executed
- More than one variable/path expression binding can be specified by separating with comma (,)

Example FOR Clause

Brackets useful for grouping sequence of operations. Especially with respect to variable scoping.
RETURN Clause

• One limitation of Xpath is that it can only operate on existing elements/attributes within the document

• XQuery allows the generation of new elements/attributes nodes
  - The element's content (if any) is either literally given between start- and end-tag, or provided as an “enclosed expression”, or as a mixture of both.
  - Curly brackets {} are used for enclosed expressions and indicate that the expression enclosed needs to be evaluated by the Xquery processor
Example RETURN Clause

<?xml version="1.0"?>
<assessments>
  <student name="Smith">
    <mark thecourse="4BA5"> 99 </mark>
    <mark thecourse="4BA1"> 75 </mark>
  </student>
  <course name="4BA1" takenby="Smith,Jones">
    <mark>60</mark>
  </course>
  <course name="4BA5" takenby="Smith,Bond">
    <mark>70</mark>
  </course>
</assessments>

Example of Xquery node generation

for $j in doc("data/tcd.xml")/assessments/course/@name
return
  <one_of_courses_is>
  {$j}
  </one_of_courses_is>

Result

<one_of_courses_is name="4BA1"/>
<one_of_courses_is name="4BA5"/>
WHERE Clause

- Filters the binding tuples produced by the FOR and LET clauses
- If the filter expression evaluates to true then the RETURN clause is executed
Example WHERE Clause

XML Source

```xml
<?xml version="1.0"?>
<assessments>
  <student name="Smith">
    <mark thecourse="4BA5"> 99 </mark>
    <mark thecourse="4BA1"> 75 </mark>
  </student>
  <course name="4BA1" takenby="Smith,Jones">
    <mark>60</mark>
  </course>
  <course name="4BA5" takenby="Smith,Bond">
    <mark>70</mark>
  </course>
</assessments>
```

XQuery

```xquery
for $j in doc("data/tcd.xml")/assessments/course
where contains($j/@takenby,"Bond")
return
  <Bond_courses_is>
  {string($j/@name)}
  </Bond_courses_is>
```

Result

```
<Bond_courses_is>4BA5</Bond_courses_is>
```

Notes:
1. Why not just use Xpath statement here?
2. Built in function “contains” used
3. Built in “string” function used
Over to you...

Source
<database>
  <person age='34'>
    <name>
      <title> Mr </title>
      <firstname> John </firstname>
      <firstname> Paul </firstname>
      <surname> Murphy </surname>
    </name>
    <hobby> Football </hobby>
    <hobby> Racing </hobby>
  </person>

  <person>
    <name>
      <firstname> Mary </firstname>
      <surname> Donnelly </surname>
    </name>
  </person>
</database>

- Define a query which will return an element called “paul_hobbys” which contains the hobby elements for each of person elements who have “Paul” as a firstname
Over to you...

**Source**

```xml
<database>
  <person age='34'>
    <name>
      <title> Mr </title>
      <firstname> John </firstname>
      <firstname> Paul </firstname>
      <surname> Murphy </surname>
    </name>
    <hobby> Football </hobby>
    <hobby> Racing </hobby>
  </person>

  <person>
    <name>
      <firstname> Mary </firstname>
      <surname> Donnelly </surname>
    </name>
  </person>
</database>
```

**XQuery**

```xquery
for $p in doc("persondb.xml")/database/person
  where $p/name/firstname=" Paul "
  return
  <paul_hobbys>
    {$p/hobby}
  </paul_hobbys>
```

**Result**

```xml
<paul_hobbys>
  <hobby> Football </hobby>
  <hobby> Racing </hobby>
</paul_hobbys>
```
Querying over several documents

XML Source
Tcd.xml

<?xml version="1.0"?><assessments>
  <student name="Smith">
    <mark thecourse="4BA5"> 99 </mark>
  </student>
  <mark thecourse="4BA1"> 75 </mark>
</student>
<course name="4BA1" takenby="Smith, Jones">
  <mark>60</mark>
</course>
<course name="4BA5" takenby="Smith, Bond">
  <mark>70</mark>
</course>
</assessments>

XQuery

for $w in doc("data/details.xml")/studentdetails/student,
  $x in doc("data/tcd.xml")/assessments/student
where $x/@name = $w/@name
return <studentpercourse>
  {$w/@name}
  {$w/address}
  {$x/mark/@thecourse}
</studentpercourse>

Result

<studentpercourse name="Smith" thecourse="4BA5" thecourse="4BA1">
  <address> 101 Pine </address>
  <enrolled> 2001 </enrolled>
</studentpercourse>

<studentpercourse name="Bond">
  <address> 007 Fleming </address>
  <enrolled> 2002 </enrolled>
</studentpercourse>
The return clause of a FLWOR expression is evaluated once for each tuple in the tuple stream, and the results of these evaluations are concatenated to form the result of the FLWOR expression.

- If no `order by` clause is present, the order of the tuple stream is determined by the orderings of the sequences returned by the expressions in the `for` clauses.
- If an order by clause is present, it determines the order of the tuple stream.
Example ORDER BY clause

```
<?xml version="1.0"?>
<studentdetails>
  <student name="Smith">
    <address> 101 Pine </address>
    <enrolled> 2001 </enrolled>
  </student>
  <student name="Bond">
    <address> 007 Fleming </address>
    <enrolled> 2002 </enrolled>
  </student>
</studentdetails>
```

```
for $x in doc("data/details.xml")/studentdetails/student
order by $x/enrolled descending
return
  <byyear>
    {$x}
  </byyear>
```

```
<byyear>
  <student name="Bond">
    <address> 007 Fleming </address>
    <enrolled> 2002 </enrolled>
  </student>
</byyear>
<byyear>
  <student name="Smith">
    <address> 101 Pine </address>
    <enrolled> 2001 </enrolled>
  </student>
</byyear>
```
Sequence Operations

- A **union** of two node sequences is a sequence containing all the nodes that occur in either of the operands.
- The **intersect** operator produces a sequence containing all the nodes that occur in both of its operands.
- The **except** operator produces a sequence containing all the nodes that occur in its first operand but not in its second operand.
Example UNION clause

```xml
<?xml version='1.0'?>
<assessments>
  <student name="Smith">
    <mark thecourse="4BA5">99</mark>
    <mark thecourse="4BA1">75</mark>
  </student>
  <course name="4BA1" takenby="Smith,Jones">
    <mark>60</mark>
  </course>
  <course name="4BA5" takenby="Smith,Bond">
    <mark>70</mark>
  </course>
</assessments>
```

```xquery
<recent_good_results>
  {doc("data/tcd.xml")/assessments/student[mark > 80] union doc("data/tcd2002.xml")/assessments/student[mark > 80]}
</recent_good_results>
```

```xml
<?xml version='1.0'?>
<assessments>
  <student name="Ledwidge">
    <mark thecourse="4BA5">45</mark>
    <mark thecourse="4BA1">55</mark>
  </student>
  <student name="ONeill">
    <mark thecourse="4BA5">85</mark>
  </student>
</assessments>
```

```xml
<?xml version='1.0'?>
<assessments>
  <student name="Smith">
    <mark thecourse="4BA5">99</mark>
    <mark thecourse="4BA1">75</mark>
  </student>
  <student name="ONeill">
    <mark thecourse="4BA5">85</mark>
  </student>
</assessments>
```
Conditional Clauses

- if (test expression) then expression else expression

- The result of a conditional expression depends on the value of the test expression in the if clause
  - If the value of the test expression is the Boolean value true, or a sequence containing at least one node (serving as an “existence test”), the then clause is executed.
  - If the value of the test expression is the Boolean value false or an empty sequence, the else clause is executed.

- **All three clauses (if, then, and else) are required**

- Nesting of if clauses possible
Example Conditional clause

<xml version="1.0"?>
<studentdetails>
  <student name="Smith">
    <address> 101 Pine </address>
    <enrolled> 2001 </enrolled>
  </student>
  <student name="Bond">
    <address> 007 Fleming </address>
    <enrolled> 2002 </enrolled>
  </student>
</studentdetails>

XQuery

for $x in doc("data/details.xml")/studentdetails/student
return
  <status>
    {$x/@name}
    {if ($x/enrolled = 2002)
      then "new student"
    else if ($x/enrolled < 1999)
      then "should be finished"
    else "student"}
  </status>

Result

<status name="Smith">
  new student
</status>
<status name="Bond">
  new student
</status>
Quantified Expression

- Allows a variable to iterate over the items in a sequence. For each variable binding, a test expression is evaluated.
  - A quantified expression that begins with **some** keyword returns the value true if the test expression is true for some variable binding
  - A quantified expression that begins with **every** keyword, returns the value true if the test expression is true for every variable binding
Example Quantified Expression clauses

<?xml version="1.0"?>
<assessments>
  <student name="Ledwidge">
    <mark thecourse="4BA5"> 45 </mark>
    <mark thecourse="4BA1"> 55 </mark>
  </student>
  <student name="ONeill">
    <mark thecourse="4BA5"> 85 </mark>
    <mark thecourse="4BA1"> 55 </mark>
  </student>
</assessments>

XQuery

<results>
  {if (every $m in doc("data/tcd2002.xml")/assessments/student satisfies $m/mark > 60)
    then "excellent results"
  else if (some $t in doc("data/tcd2002.xml")/assessments/student satisfies $t/mark > 50)
    then "average results"
  else "bad results"
  }
</results>

Result

<results>average results</results>
Built-in Functions

- Over 100 built-in functions
  - Document in IPSI-XQ documentation directory describes
- Functions that operate on Basic Types
  - manipulation of dates, strings, numbers etc
  - E.g. fn:string-join for joining strings together
- Functions that operate on Nodes
  - E.g. fn:name() returns name of node
- Functions that operate on Sequences
  - A sequence is an ordered collection of zero or more items
  - E.g. fn:distinct-values returns sequence with all duplicates removed
- Functions that operate on Context
  - obtain information from the evaluation context
  - E.g. fn:last returns the number of items in the sequence being processed
<table>
<thead>
<tr>
<th>Function</th>
<th>Commentary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Math: ++, --, *, div, idiv, mod, =, !=, &lt;, &gt;, &lt;=, &gt;= floor(), ceiling(), round(), count(), min(), max(), avg(), sum()</td>
<td>Division is done using div rather than a slash because a slash indicates an XPath step expression. idiv is a special operator for integer-only division that returns an integer and ignores any remainder.</td>
</tr>
<tr>
<td>Strings and Regular Expressions: compare(), concat(), starts-with(), ends-with(), contains(), substring(), string-length(), substring-before(), substring-after(), normalize-space(), upper-case(), lower-case(), translate(), matches(), replace(), tokenize()</td>
<td>compare() dictates string ordering. translate() performs a special mapping of characters. matches(), replace(), and tokenize() use regular expressions to find, manipulate, and split string values.</td>
</tr>
<tr>
<td>Date and Time: current-date(), current-time(), current-dateTime() ++, --, div eq, ne, lt, gt, le, gt</td>
<td>XQuery has many special types for date and time values such as duration, dateTime, date, and time. On most you can do arithmetic and comparison operators as if they were numeric. The two-letter abbreviations stand for equal, not equal, less than, greater than, less than or equal, and greater than or equal.</td>
</tr>
<tr>
<td>XML node and QNames: node-kind(), node-name(), base-uri() eq, ne, is, isnot, get-local-name-from-QName(), get-namespace-from-QName() deep-equal() &gt;&gt;, &lt;&lt;</td>
<td>node-kind() returns the type of a node (i.e. &quot;element&quot;). node-name() returns the QName of the node, if it exists. base-uri() returns the URI this node is from. Nodes and QName values can also be compared using eq and ne (for value comparison), or is and isnot (for identity comparison). deep-equal() compares two nodes based on their full recursive content. The &lt;&lt; operator returns true if the left operand precedes the right operand in document order. The &gt;&gt; operator is a following comparison.</td>
</tr>
<tr>
<td>Sequences: item-at(), index-of(), empty(), exists(), distinct-nodes(), distinct-values(), insert(), remove(), subsequence(), unordered().position(), last()</td>
<td>item-at() returns an item at a given position while index-of() attempts to find a position for a given item. empty() returns true if the sequence is empty and exists() returns true if it’s not. distinct-nodes() returns a sequence with exactly identical nodes removed and distinct-values() returns a sequence with any duplicate atomic values removed. unordered() allows the query engine to optimize without preserving order. position() returns the position of the context item currently being processed. last() returns the index of the last item.</td>
</tr>
<tr>
<td>Type Conversion: string(), data(), decimal(), boolean()</td>
<td>These functions return the node as the given type, where possible. data() returns the &quot;typed value&quot; of the node.</td>
</tr>
<tr>
<td>Booleans: true(), false(), not()</td>
<td>There’s no &quot;true&quot; or &quot;false&quot; keywords in XQuery but rather true() and false() functions. not() returns the boolean negation of its argument.</td>
</tr>
<tr>
<td>Input: document(), input(), collection()</td>
<td>document() returns a document of nodes based on a URI parameter. collection() returns a collection based on a string parameter (perhaps multiple documents). input() returns s general engine-provided set of input nodes.</td>
</tr>
</tbody>
</table>
User Functions

• XQuery allows users to define functions of their own
  – A function may take zero or more parameters.
  – A function definition must specify the name of the function and the names of its parameters if they exist
    • It may optionally specify types for the parameters
      – If no type is specified for a function parameter, that parameter accepts values of any type.
    • It may optionally specify types for the result of the function.
      – If no type is specified for the result of the function, the function may return a value of any type.
  – Body of the function is an expression enclosed in curly braces.
Example Function clause

```xml
<?xml version="1.0"?>
<assessments>
  <student name="Smith">
    <mark thecourse="4BA5"> 99 </mark>
    <mark thecourse="4BA1"> 75 </mark>
  </student>
  <course name="4BA1" takenby="Smith,Jones">
    <mark>60</mark>
  </course>
  <course name="4BA5" takenby="Smith,Bond">
    <mark>70</mark>
  </course>
</assessments>

<?xml version="1.0"?>
<assessments>
  <student name="Ledwidge">
    <mark thecourse="4BA5"> 45 </mark>
    <mark thecourse="4BA1"> 55 </mark>
  </student>
  <student name="ONeill">
    <mark thecourse="4BA5"> 85 </mark>
  </student>
</assessments>
```

Declare function all_students()
```
def all_students()
    for $s in doc("data/tcd.xml")/assessments/student
        union doc("data/tcd2002.xml")/assessments/student
    return
    <student>
        {$s/@name}
        {$s/mark/@thecourse}
    </student>
```;

```xml
<all>
{all_students()}
</all>
```

Result
```
<all>
  <student name="Smith" thecourse="4BA5" thecourse="4BA1"/>
  <student name="Ledwidge" thecourse="4BA5" thecourse="4BA1"/>
  <student name="ONeill" thecourse="4BA5"/>
</all>
```
Type References

- Sometimes necessary to refer to a particular type in query
- One way to refer to a type is by its qualified name, or QName.
  - A QName may refer to a built-in type such as `xs:integer` or to a type that is defined in some schema, such as `abc:student`.
  - If the QName has a namespace prefix (the part to the left of the colon), that prefix must be bound to a specific namespace URI using the "declare namespace" clause.
Example Function clause with param

```
<?xml version="1.0"?>
<assessments>
  <student name="Smith">
    <mark thecourse="4BA5"> 99 </mark>
    <mark thecourse="4BA1"> 75 </mark>
  </student>
  <course name="4BA1" takenby="Smith,Jones">
    <mark>60</mark>
  </course>
  <course name="4BA5" takenby="Smith,Bond">
    <mark>70</mark>
  </course>
</assessments>

<?xml version="1.0"?>
<assessments>
  <student name="Ledwidge">
    <mark thecourse="4BA5"> 45 </mark>
    <mark thecourse="4BA1"> 55 </mark>
  </student>
  <student name="ONeill">
    <mark thecourse="4BA5"> 85 </mark>  
  </student>
</assessments>
```

declare function find_students($stuname as xs:string)
{
    for $s in doc("data/tcd.xml")/assessments/student union
doc("data/tcd2002.xml")/assessments/student
    where $stuname = string($s/@name)
return
    <student>
        {$s/@name}
        {$s/mark/@thecourse}
    </student>
}

find_students("Smith")

```
<student name="Smith" thecourse="4BA5" thecourse="4BA1"/>
```
Type References

• Another way to refer to a type is by a generic keyword such as element or attribute.
  – May optionally be followed by a QName that further restricts the name or type of the node.
  – For example,
    • element denotes any element;
    • element student denotes an element whose name is student;
    • element of type abc:student denotes an element whose type is student as declared in the namespace abc.
  – A reference to a type may optionally be followed by one of three occurrence indicators:
    • “*” means “zero or more”;
    • “+” means “one or more,”
    • “?” means “zero or one.”
    • The absence of an occurrence indicator denotes exactly one occurrence of the indicated type.
Example Function clause with output type declared

XML Source
tcd.xml

```xml
<?xml version="1.0"?>
<assessments>
  <student name="Smith">
    <mark thecourse="4BA5"> 99 </mark>
    <mark thecourse="4BA1"> 75 </mark>
  </student>
  <course name="4BA1" takenby="Smith,Jones">
    <mark>60</mark>
  </course>
  <course name="4BA5" takenby="Smith,Bond">
    <mark>70</mark>
  </course>
</assessments>
```

XML Source	
tcd2002.xml

```xml
<?xml version="1.0"?>
<assessments>
  <student name="Ledwidge">
    <mark thecourse="4BA5"> 45 </mark>
    <mark thecourse="4BA1"> 55 </mark>
  </student>
  <student name="ONeill">
    <mark thecourse="4BA5"> 85 </mark>
  </student>
</assessments>
```

XQuery

```xquery
declare function all_students()
  as element()*
  {
    for $s in doc("data/tcd.xml")/assessments/student
        union
    doc("data/tcd2002.xml")/assessments/student
    return
    <student>
      {$s/@name}
      {$s/mark/@thecourse}
    </student>
  };

<all>
  {all_students()}
</all>
```

Result

```xml
<all>
  <student name="Smith" thecourse="4BA5" thecourse="4BA1"/>
  <student name="Ledwidge" thecourse="4BA5" thecourse="4BA1"/>
  <student name="ONeill" thecourse="4BA5"/>
</all>
```
Type References

- Not only occur in function definitions
- Can test for type using `instanceof` operator, e.g.
  - 49 instance of `xs:integer` returns true
  - "Hello" instance of `xs:integer` returns false
- Can convert result of an expression into one of XML schema built-in types using `cast` operator, e.g.
  - `xs:double(11 div 5)` returns 2.0
  - `xs:string(11 div 5)` returns “2”
Computed Constructors

- Used to dynamically create name of an element, attribute etc.
- Takes the form
  - Keyword \{computation expression for name\}
    \{computation expression for value\}
- Example

```xml
attribute
{if $p/sex "M" then "father" else "mother"}
{$p/name}
=><person father="john"> Paul Smith </person>
```
IPSIXQ from Fraunhofer Institute
From Windows: Interactive Query Engine

Type Queries here

Result

Execute
IPSIXQ from Fraunhofer Institute
From Command Line: runConsole

D:\Declan\TCD\Research\Prototype2\source\xquery\source>runConsole

XQuery Interpreter Version 1.3.0
----------------------------------------

Options: []
Usage is:
    java de.xquery.ui.console.ConsoleUI [options] [input]

    options :
        -nodebug      - turns off verbose mode (default)
        -debug        - turns on verbose mode
        -CoreSyntax   - displays Core mapping(default)
        -noCoreSyntax - turns off display of Core mapping
        -infer        - infers the result type(default)
        -noinfer      - turns off result type infer
        -eval         - evaluates query(default)
        -n_eval       - turns of query evaluation

    -output:outputfile - direct output to outputfile
    input : (=file:inputfile | -query:querytext)

    -file:inputfile - uri of inputfile
    -query:querytext - direct input of a querytext

Output filename
Filename of file with Xquery text
IPSIXQ from Fraunhofer Institute
From a Java Program, Example below
Full API described in documentation directory

```
import de.fraunhofer.ipsi.ipsixq.api.*;
import de.fraunhofer.ipsi.xquery.api.*;
import java.util.Properties;
import org.w3c.dom.*;
import java.io.*;

public class callXQapi
{
    public static void main(String[] args)
    {
        Properties props = new Properties();
        props.setProperty("CorePrettyPrint", "false");
        props.setProperty("infer", "true");
        props.setProperty("eval", "true");
        try
        {
            File file;
            String query="";
            XQIDriver driver = new XQIDriver();
            Connection connection = driver.connect(".", props);
            System.out.println("Connection made");
        }
    }
}
```

Import statements; make sure xquery lib in your path and classpath environments

Like the “options” of runConsole

Defining useful variables

Connect to data source

Specification of where to find XML data, a URI
Example continued

```java
file = new File(args[0]);
if(!file.canRead()){
    throw new IOException("Cannot read file!");
}
if(file != null){
    StringBuffer q = new StringBuffer();
    BufferedReader in = new BufferedReader(new FileReader(file));
    for(int c = in.read(); c > -1; c = in.read()){
        q.append((char)c);
    }
    query = q.toString();
}

QueryResult result = connection.executeQuery(query);
System.out.println("Query made");
NodeList nodes = result.getNodeList();
Etc...
}
catch (Exception XQueryException)
{
    System.out.println("XQuery Exception caught");
}
```
eXist Database

- Start up eXist database server
- Start up Client

Or

- Go to Local eXist web page in browser
Review

```
<book year="1994">
  <title>TCP/IP Illustrated</title>
  <author><last>Stevens</last><first>W.</first></author>
  <publisher>Addison-Wesley</publisher>
  <price>65.95</price>
</book>

<book year="1992">
  <title>Advanced Programming in the Unix environment</title>
  <author><last>Stevens</last><first>W.</first></author>
  <publisher>Addison-Wesley</publisher>
  <price>65.95</price>
</book>

<book year="2000">
  <title>Data on the Web</title>
  <author><last>Abiteboul</last><first>Serge</first></author>
  <publisher>Morgan Kaufmann Publishers</publisher>
  <price>39.95</price>
</book>

<book year="1999">
  <title>The Economics of Technology and Content for Digital TV</title>
  <editor><last>Gerbarg</last><first>Darcy</first></editor>
  <publisher>Kluwer Academic Publishers</publisher>
  <price>129.95</price>
</book>
```

XQuery

```
<bib>
    where $b/publisher = "Addison-Wesley" and $b/@year > 1991
    return
      <book year="{ $b/@year }">
        { $b/title }
      </book>
  }
</bib>
```

Result

```
<book year="1994">
  <title>TCP/IP Illustrated</title>
</book>

<book year="1992">
  <title>Advanced Programming in the Unix environment</title>
</book>
```
Review

<bib>
  <book year="1994">
    <title>TCP/IP Illustrated</title>
    <author><last>Stevens</last><first>W.</first></author>
    <publisher>Addison-Wesley</publisher>
    <price>65.95</price>
  </book>
  <book year="1992">
    <title>Advanced Programming in the Unix environment</title>
    <author><last>Stevens</last><first>W.</first></author>
    <publisher>Addison-Wesley</publisher>
    <price>65.95</price>
  </book>
  <book year="2000">
    <title>Data on the Web</title>
    <author><last>Abiteboul</last><first>Serge</first><last>Buneman</last><first>Peter</first><last>Suciu</last><first>Dan</first></author>
    <publisher>Morgan Kaufmann Publishers</publisher>
    <price>39.95</price>
  </book>
  <book year="1999">
    <title>The Economics of Technology and Content for Digital TV</title>
    <editor><last>Gerbarg</last><first>Darcy</first><affiliation>CITI</affiliation></editor>
    <publisher>Kluwer Academic Publishers</publisher>
    <price>129.95</price>
  </book>
</bib>

<reviews>
  <entry>
    <title>Data on the Web</title>
    <price>34.95</price>
    <review>
      A very good discussion of semi-structured database systems and XML.
    </review>
  </entry>
  <entry>
    <title>Advanced Programming in the Unix environment</title>
    <price>65.95</price>
    <review>
      A clear and detailed discussion of UNIX programming.
    </review>
  </entry>
  <entry>
    <title>TCP/IP Illustrated</title>
    <price>65.95</price>
    <review>
      One of the best books on TCP/IP.
    </review>
  </entry>
</reviews>

<book-with-prices>
  <title>TCP/IP Illustrated</title>
  <price-amazon>65.95</price-amazon>
  <price-bn>65.95</price-bn>
</book-with-prices>
<book-with-prices>
  <title>Advanced Programming in the Unix environment</title>
  <price-amazon>65.95</price-amazon>
  <price-bn>65.95</price-bn>
</book-with-prices>
<book-with-prices>
  <title>Data on the Web</title>
  <price-amazon>34.95</price-amazon>
  <price-bn>39.95</price-bn>
</book-with-prices>
XML Source bib.xml

<book year="1994">
  <title>TCP/IP Illustrated</title>
  <author>
    <last>Stevens</last>
    <first>W.</first>
  </author>
  <publisher>Addison-Wesley</publisher>
  <price>65.95</price>
</book>

<book year="1992">
  <title>Advanced Programming in the Unix environment</title>
  <author>
    <last>Stevens</last>
    <first>W.</first>
  </author>
  <publisher>Addison-Wesley</publisher>
  <price>65.95</price>
</book>

<book year="2000">
  <title>Data on the Web</title>
  <author>
    <last>Abiteboul</last>
    <first>Serge</first>
    <last>Buneman</last>
    <first>Peter</first>
    <last>Suciu</last>
    <first>Dan</first>
  </author>
  <publisher>Morgan Kaufmann Publishers</publisher>
  <price>39.95</price>
</book>

<book year="1999">
  <title>The Economics of Technology and Content for Digital TV</title>
  <editor>
    <last>Gerbarg</last>
    <first>Darcy</first>
  </editor>
  <publisher>Kluwer Academic Publishers</publisher>
  <price>129.95</price>
</book>

XML Source reviews.xml

<reviews>
  <entry>
    <title>Data on the Web</title>
    <price>34.95</price>
    <review>
      A very good discussion of semi-structured database systems and XML.
    </review>
  </entry>
  <entry>
    <title>Advanced Programming in the Unix environment</title>
    <price>65.95</price>
    <review>
      A clear and detailed discussion of UNIX programming.
    </review>
  </entry>
  <entry>
    <title>TCP/IP Illustrated</title>
    <price>65.95</price>
    <review>
      One of the best books on TCP/IP.
    </review>
  </entry>
</reviews>

XQuery

for $b in doc("http://www.bn.com/bib.xml")//book,
  $a in doc("http://www.amazon.com/reviews.xml")//entry
where $b/title = $a/title
return
  <book-with-prices>
    { $b/title }
    <price-amazon>{ $a/price/text() }</price-amazon>
    <price-bn>{ $b/price/text() }</price-bn>
  </book-with-prices>

Result

<book-with-prices>
  { "TCP/IP Illustrated" }
  <price-amazon>65.95</price-amazon>
  <price-bn>65.95</price-bn>
</book-with-prices>

<book-with-prices>
  { "Advanced Programming in the Unix environment" }
  <price-amazon>65.95</price-amazon>
  <price-bn>65.95</price-bn>
</book-with-prices>

<book-with-prices>
  { "Data on the Web" }
  <price-amazon>34.95</price-amazon>
  <price-bn>39.95</price-bn>
</book-with-prices>
XQuery and other XML technologies

• XQuery versus XSLT
  – Easier to use
    • Procedurally oriented
  – Less verbose
  – Less document-centric

• XQuery versus XPath
  – Operates over several documents
  – Allows the construction of new nodes
  – Has mechanism for variables
  – Has mechanism for user defined functions
Summary

- Basis of XQuery is the FLOWR expression
  1. One or more FOR and/or LET expressions
  2. Optional WHERE clause
  3. Optional ORDER BY clause
  4. RETURN clause
- XQuery uses XPath in its clauses in order to identify individual parts of a document
References


- “Other pointers” at W3C site http://www.w3.org/XML/Query