COMP207 Lab Exercises
Tutorial 8 (Week 10)

The exercises below provide an opportunity for you to experiment with and become more familiar with XPath and XQuery. They assume that you have access to an XQuery interpreter. During this lab session, you can use the web-based XQuery interpreter introduced in Section 2, but in general any XQuery interpreter is fine. Alternatively, or if you do not have access to an XQuery interpreter, you could evaluate XPath and XQuery expressions manually, using the specifications of XPath 3.1[1] and XQuery 3.1[2] as a reference.

1 Example XML Document

The exercises refer to the following XML document "mydoc.xml" (based on the product data XML file in [1, Chapter 12]), which can be downloaded from the Vital course page:

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<products>

   <maker name="A">
      <pc model="1001" price="1800">
         <speed>3.6</speed>
         <ram>16</ram>
         <harddisk>1024</harddisk>
      </pc>
      <pc model="1002" price="1250">
         <speed>2.3</speed>
         <ram>8</ram>
         <harddisk>256</harddisk>
      </pc>
      <laptop model="2004" price="1250">
         <speed>2.3</speed>
         <ram>8</ram>
         <harddisk>128</harddisk>
         <screen>13</screen>
      </laptop>
      <laptop model="2005" price="2350">
         <speed>2.8</speed>
         <ram>16</ram>
         <harddisk>256</harddisk>
         <screen>15</screen>
      </laptop>
   </maker>
```

[1] https://www.w3.org/TR/xpath-31/
[2] https://www.w3.org/TR/xquery-31/

```
   <maker name="E">
      <pc model="1011" price="2728">
         <speed>4.3</speed>
         <ram>32</ram>
         <harddisk>500</harddisk>
      </pc>
      <pc model="1012" price="878">
         <speed>3.5</speed>
         <ram>8</ram>
         <harddisk>250</harddisk>
      </pc>
      <laptop model="2001" price="1499">
         <speed>3.8</speed>
         <ram>8</ram>
         <harddisk>250</harddisk>
         <screen>15.6</screen>
      </laptop>
      <printer model="3002" price="226">
         <colour>true</colour>
         <type>laser</type>
      </printer>
   </maker>
   <maker name="H">
      <printer model="3006" price="60">
         <colour>true</colour>
         <type>ink-jet</type>
      </printer>
      <printer model="3007" price="150">
         <colour>false</colour>
         <type>laser</type>
      </printer>
   </maker>
</products>
```

2 Interpreter for XPath and XQuery

As pointed out in the lectures, XPath and XQuery interpreters come in various forms, for the command line, as libraries for programming languages, built into database management systems such as MySQL and PostgreSQL, etc. For this lab session, you can use the online live demo provided by the Zorba website:

`http://try.zorba.io`

2.1 Preparations

To prepare Zorba for the exercises, open your web browser and visit `http://try.zorba.io`. You should see the page shown in Figure 1. Open up the bottom area “XML Documents” (click

on the two red arrows), and copy the contents of the file "mydoc.xml" (see Section 1) into the text area "mydoc.xml" (see Figure 2). You are now ready to go.

2.2 Executing XQuery Expressions

You can enter XQuery expressions into the text area "XQuery" at the top and execute these on "mydoc.xml" by clicking the button "Execute". The result will be displayed at the bottom of the page under "Result". Figure 3 shows this for the XQuery expression

```
let $doc := doc("mydoc.xml")
for $m in $doc/products/maker
where $m/pc/@price < 1500
return <result>{data($m/@name)}</result>
```

2.3 Executing XPath Expressions

Zorba is an XQuery processor, but you can evaluate XPath expressions by wrapping these into XQuery expressions. For example, to execute the XPath expression `/products/maker/pc` on "mydoc.xml", we execute the XQuery expression

```
let $doc := doc("mydoc.xml")
return <result>{$doc/products/maker/pc}</result>
```

2.4 Useful Features Not Covered In The Lectures

Here are some useful features of XPath and XQuery that you might want to use when you experiment with XPath and XQuery:

- In XQuery, the function `data()` can be applied to a leaf node or to an attribute node of an XML document to extract the text associated with that node. The XQuery in Section 2.2 illustrates this. The function `data()` is applied to the attribute node `$m/@name` to extract the value of that attribute. More generally, the function can be applied to any node of an XML document, in which case it returns the list of all text elements associated with the leaf nodes below that node.

- If you have an XPath such as `/products/maker/pc/speed` that leads to a leaf of an XML document, then you can append `/text()` to that XPath in order to obtain the text enclosed between the opening and the closing tag of the element selected by `/products/maker/pc/speed`. Thus, the XPath `/products/maker/pc/speed/text()` returns a list containing the speed (as a string value, not the speed element!) of every PC in the XML document.

- Similarly, to extract the value of an attribute node, append `/data()` to the XPath. For example, `/products/maker/@name/data()` will return the names of all makers.

2.5 Common Errors

Here are some errors which you might encounter regularly when typing in XQueries:

- **Syntax errors resulting from not using lower case letters for keywords.** This can be difficult to spot, because Zorba will not tell you directly that you've used upper case letters in some keyword (let, for, where, etc.). Depending on where this error occurs, you might get different error messages such as:

  ```
  static error [err:XPST0003]: invalid expression: syntax error,
  unexpected "$", expecting "end of file" or "," or "}"
  ```

  if you try to execute the following:

  ```
  Let $doc := doc("mydoc.xml")
  for $m in $doc/products/maker/@name
  let $name := data($m)
  return $name
  ```

- **Syntax errors resulting from using = instead of := in a let clause.** If you get an error message such as

  ```
  static error [...]: invalid expression: syntax error, unexpected
  "=", expecting "as" or ":="
  ```

  then this is because you have written a let clause in which a variable is followed by a simple equality symbol = instead of :=. To fix this error, put a colon directly in front of the equality symbol.

- **Attempts to return attribute nodes.** Zorba is not able to deal with XQueries that return attribute nodes. For instance, the following variant of the XQuery in Section 2.2 will return an error message:

  ```
  let $doc := doc("mydoc.xml")
  for $m in $doc/products/maker
  where $m/pc/@price < 1500
  return $m/@name
  ```

  The error message for this example is:

  ```
  serialization error [...]: "A": can not serialize attribute node
  ```

  so if you get an error message which tells you that an attribute node cannot be serialized, this simply means that Zorba cannot output that node. To fix this error, simply wrap the attribute into an XML element, as in the XQuery in Section 2.2. The function `data()` in that XQuery is not strictly necessary, but makes the output more readable.

3 Exercises

The following two exercises cover XPath and XQuery. When you use conditions in XPath or XQuery, keep in mind that equalities have an implicit *existential* semantics. For example, if E_1 and E_2 are XPath or XQuery expressions, c is a constant value (e.g., a string), and $*$ is any of the comparison relationships in XPath or XQuery (=, !=, <, ...), then $E_1 * c$ is true if and only if there exists an item i in the result to E_1 such that $i * c$ (if c is a string constant, then the tags around i will be removed before the comparison). Similarly, $E_1 * E_2$ is true if and only if there exist items i_1 and i_2 in the result to E_1 and E_2, respectively, such that $i_1 * i_2$.

Exercise 1 (Exercise 12.1.1 in [1])**.** Write XPath expressions for the following queries. What do these XPaths return? First try to work this out yourself, then verify your answer using an interpreter for XPath or XQuery (e.g., for the web-based interpreter suggested above, see the instructions in Section 2.3).

(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 PCs and/or laptops.

(f) Find the model numbers of PCs with a hard disk of at least 500GB.

(g) Find the makers of all colour laser printers.

(h) Find the makers of at least two PCs.

Solutions:

The following just gives the queries. The results can be obtained using any interpreter for XPath or XQuery.

(a) `/products/maker/pc/ram`. This will return the `ram` elements for each PC. If we want to return the actual amounts, then we have to use `/products/maker/pc/ram/text()`.

(b) `/products/maker/*/@price`. This will return the `price` attribute nodes for each product. If we want to return the actual prices, then we have to use `/products/maker/*/@price/data()`.

(c) `/products/maker/printer`

(d) `/products/maker[printer/type="laser"]/@name`

(e) `/products/maker[pc or laptop]/@name`

(f) `/products/maker/pc[harddisk >= 500]/@model`

(g) `/products/maker[printer[type="laser" and colour="true"]]/@name`. The obvious first approach `/products/maker[printer/type="laser" and printer/colour="true"]/@name` does *not* work.

(h) `/products/maker[pc != pc]/@name`. This might be confusing at first, but is correct due to the existential semantics of comparison predicates using XPaths: `pc != pc` is true at a node N_0 if and only if there exists a node N_1 that is selected by the XPath `pc` at N_0 and a node N_2 that is selected by the XPath `pc` at N_0, and the two nodes are different. In other words, `pc != pc` is true at a node N_0 if and only if there are two different children at N_0 which have the tag `pc`.

Exercise 2 (Exercise 12.2.1 in [1])**.** Write XQuery expressions for the following queries. What do these XQuery expressions return? First try to work this out yourself, then verify your answer using an interpreter for XQuery.

(a) Find the `printer` elements with a price less than 100.

(b) Find the `printer` elements with a price less than 100, and produce the sequence of these elements surrounded by tags `<cheapprinters>` and `</cheapprinters>`.

(c) Find the names of the makers of both printers and laptops.

(d) Find the names of the makers that produce at least two PCs with a speed of 3.00 or more.

(e) Find the makers such that every PC they produce has a price no more than 2000.

(f) Produce a sequence of elements of the form

```
<laptop><model>x</model><maker>y</maker></laptop>
```

where `x` is the model number and `y` is the name of the maker of the laptop.

Solutions:

(a)
```
let $doc := doc("mydoc.xml")
for $p in $doc/products/maker/printer
where $p/@price < 100
return $p
```

(b)
```
let $doc := doc("mydoc.xml")
for $p in $doc/products/maker/printer
where $p/@price < 100
return <cheapprinter>{$p}</cheapprinter>
```

(c)
```
let $doc := doc("mydoc.xml")
for $m in $doc/products/maker
where $m/printer and $m/laptop
return <result>{data($m/@name)}</result>
```

(d)
```
let $doc := doc("mydoc.xml")
for $m in $doc/products/maker
where $m/pc[speed >= 3] != $m/pc[speed >= 3]
return <result>{data($m/@name)}</result>
```

A better and more readable way is to use the aggregate `count`:

```
let $doc := doc("mydoc.xml")
for $m in $doc/products/maker
let $fast_pcs := $m/pc[speed >= 3]
where count($fast_pcs) >= 2
return <result>{data($m/@name)}</result>
```

(e)

```
let $doc := doc("mydoc.xml")
for $m in $doc/products/maker
where every $p in $m/pc satisfies $p/@price <= 2000
return <result>{data($m/@name)}</result>
```

(f)

```
let $doc := doc("mydoc.xml")
for $m in $doc/products/maker
for $l in $m/laptop
return
    <laptop>
        <model>{$l/@model/data()}</model>
        <maker>{$m/@name/data()}</maker>
    </laptop>
```

References

[1] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. *Database Systems - The Complete Book*. Pearson Education, 2nd edition, 2009.

Figure 1: Zorba Live Demo

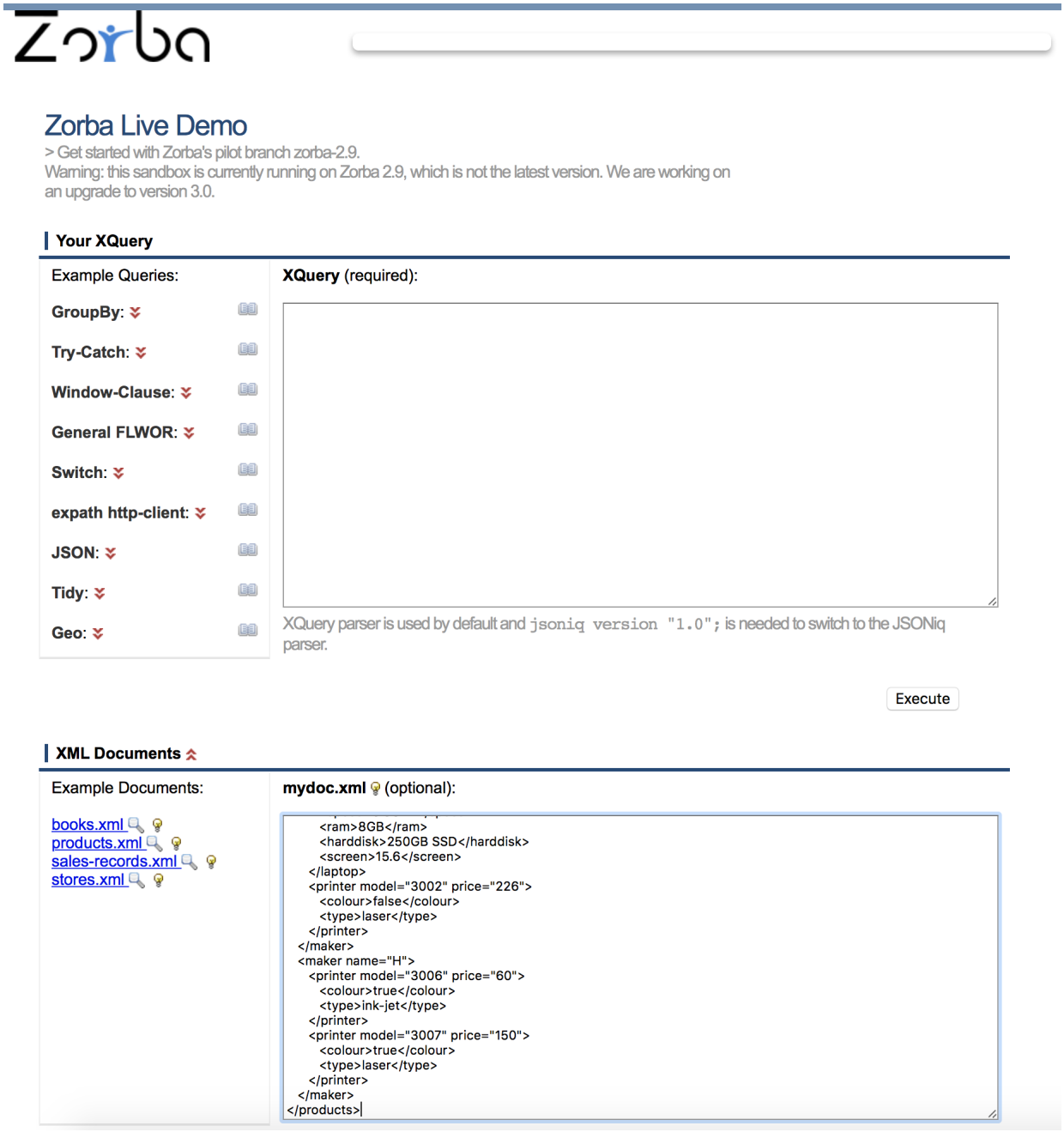

Figure 2: Zorba Live Demo with “mydoc.xml”

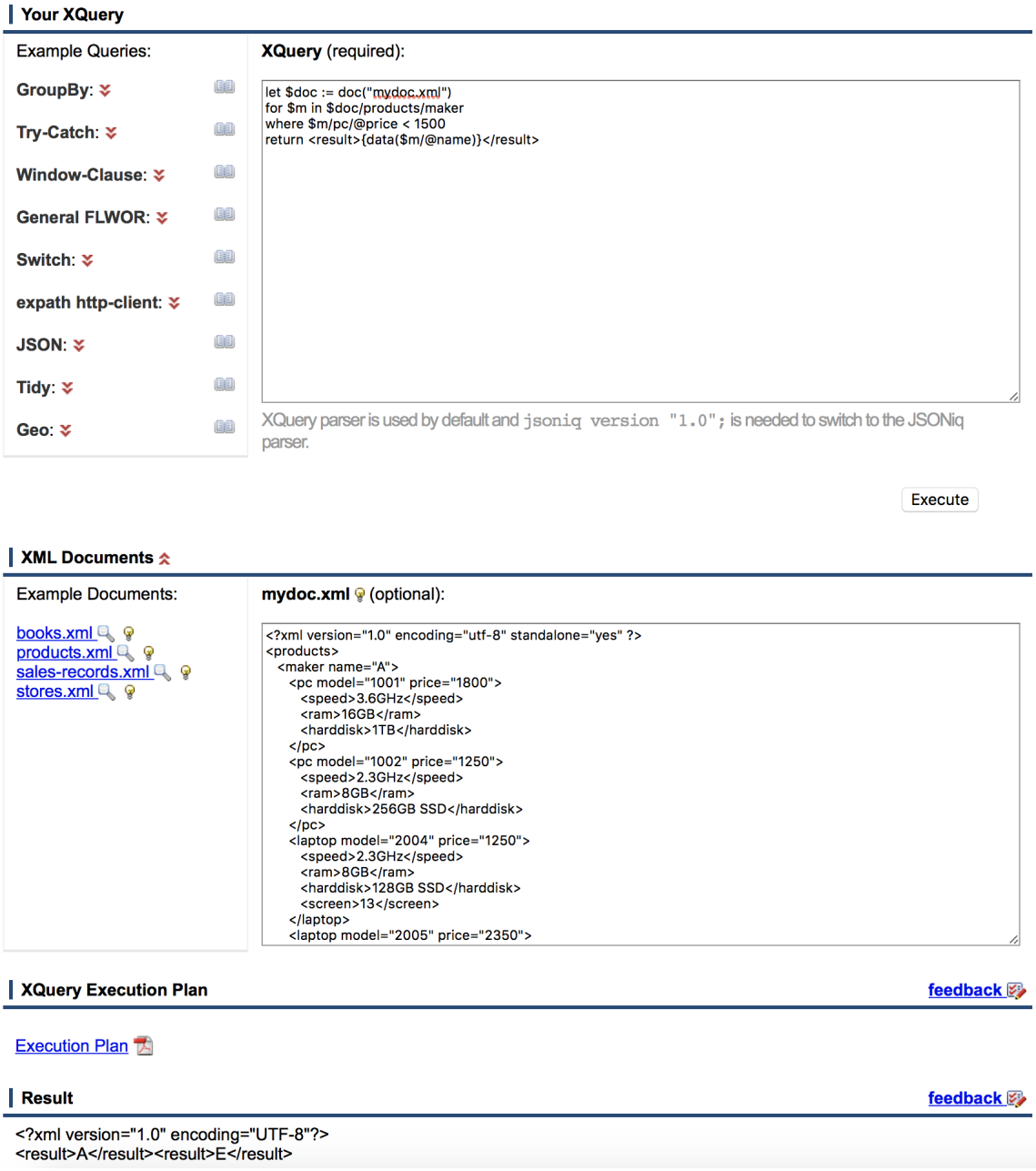

Figure 3: Zorba Live Demo showing the result of an XQuery expression on “mydoc.xml”