XQuery

XML Query Language Requirements

- Query Language Syntax
  - Two bindings
  - One query language syntax for people to read and write.
  - One to be expressed in XML so that it reflects the underlying structure of the query.
- Declarativity
  - Declarative. There is no particular evaluation strategy.

Xquery 3.0 - http://www.w3.org/TR/xquery-30/

XML Query Language Requirements

- Protocol Independence
  - Defined independently of any protocols with which it is used.
- Error Conditions
  - Define standard error conditions that can occur during the execution of a query, such as processing errors within expressions, unavailability of external functions to the query processor, or processing errors generated by external functions.
- Updates
  - Being able to extend later in future versions.

XQuery

- XQuery is a general purpose query language for XML data
- Alpha version of XQuery engine available free from Microsoft
- XQuery is derived from the Quilt query language, which itself borrows from SQL, XQL and XML-QL

- XQuery 1.0 became a W3C Recommendation January 23, 2007
  Xquery 3.0 currently proposed by the World Wide Web Consortium (W3C)

- XQuery uses a
  for … let … where .. return …
  syntax
  for ⇔ SQL from
  where ⇔ SQL where
  return ⇔ SQL select
  let allows temporary variables, and has no equivalent in SQL

An XML File

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<bookstore>
<book category="COOKING">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
     <price>30.00</price>
 </book>
<book category="CHILDREN">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
 </book>
<book category="WEB">
<title lang="en">XQuery Kick Start</title>
<author>James McGovern</author>
 <author>Per Bothner</author>
 <author>Kurt Cagle</author>
<author>James Linn</author>
 <author>Vaidyanathan Nagarajan</author>
 <year>2003</year>
 <price>49.99</price>
 </book><book category="WEB">
 <title lang="en">Learning XML</title>
<author>Erik T. Ray</author>
<year>2003</year>
 <price>39.95</price>
</book>
</bookstore>
```

Simple Expressions

- doc("books.xml")
  - The doc() function is used to open the "books.xml" file
- doc("books.xml")/bookstore/book/title
  - XQuery uses path expressions to navigate through elements in an XML document
- doc("books.xml")/bookstore/book[price<30]
  - XQuery uses predicates to limit the extracted data from XML documents

<title lang="en">Everyday Italian</title>
<title lang="en">Harry Potter</title>
<title lang="en">XQuery Kick Start</title>
<title lang="en">Learning XML</title>

Value Comparison

- A value comparison is used to compare single values only
  - First are operands evaluated
  - If both operands consist of exactly one value, and value types are appropriate, the comparison is true if $o_1 \theta o_2$ evaluates true

- The following value comparison is `true`

```
fn:doc("books.xml")//title/@lang eq "en"
```

  only if the XPath expression returns only one `@lang` node and the node has a value of "`s1`", otherwise XQuery will raise an error (more nodes), or return `false` (not "`en`")

- The following value comparison returns `true`

```
<a>5</a> eq <b>5</b>
```

General Comparison

- A general comparison is used to compare **sequences** of **any** length
- General comparisons are existentially quantified, they return true if at lest one pair of elements from both sequences satisfy the comparison operator

- The following general comparison evaluates `true`

```
fn:doc("books.xml")//title/@lang = "en"
```

  if the XPath expression returns at least one `@lang` node with a value of `"en"`, otherwise XQuery will return `false`

- The following general comparison evaluates `true`

```
(2, 3) = (3, 4, 5)
```

Node Comparison

- Node comparisons are used to compare two nodes, by their identity, or by their document order
- Each operand has to be either a single node, or an empty sequence
  - If an operand is an empty sequence, the result is also an empty sequence
  - A comparison with the `is` operator returns `true`, if both operands have the same identifier
  - A comparison with `<<` operator returns `true`, if the left operand has a smaller identifier value than the right operand with regard to the document order

Comparisons

- $bookstore//book/@q > 10
  - The expression above returns true if any q attributes have values greater than 10.
- $bookstore//book/@q gt 10
  - The expression above returns true if there is only one q attribute returned by the expression, and its value is greater than 10. If more than one q is returned, an error occurs.

Logical Expressions

- A logical expression is either an **and**-expression or an **or**-expression
- If a logical expression does not raise an error, its value is always either `true` or `false`
- The following expression returns true

```
1 eq 1 and 2 le 3
```

- XQuery also provides a function named `fn:not`
  - The function reduces its parameter to a Boolean value
  - It returns `true` if the Boolean value of the parameter is `false` and vice versa

Direct Element Constructor

- A direct element constructor contains:
  - Start and end tags, with a content between them
  - The content of an element constructor may be:
    - Text,
    - Other element constructors, and
    - Expressions, placed between **curly braces**
  - Expressions within curly braces are **evaluated** and replaced by values
  - The result of evaluating an expression can be any sequence of nodes and atomic values
  - If an expression returns a **node**, the node and all its **descendants** will be included in the content of the element constructor

FLWOR

- `FLWOR` stands for
  - `for`, (used to iterate through the result of an XPath expression and to bind a variable intern to each object of the result)
  - `let`, (used to bind a variable to the whole sequence of objects returned by an XPath expression)
  - `where`, (used to select those objects from the result returned by an Xpath expression that satisfy given conditions)
  - `order by`, (used to sort the result of an XPath expression) and
  - `return` (used to construct the query result)
- `FLWOR` is pronounced as flower
- `FLWOR` expressions allow:
  - Iterating through XML documents
  - Binding variables to the iteration results,
  - Restructuring XML documents, and
  - Joining XML documents

FLOWR

```
doc("books.xml")/bookstore/book[price>30]/title
```

```
for $x in doc("books.xml")/bookstore/book
where $x/price>30
return $x/title
```

<title lang="en">XQuery Kick Start</title>

<title lang="en">Learning XML</title>

```
for $x in doc("books.xml")/bookstore/book
where $x/price>30
order by $x/title
return $x/title
```

WHERE

- WHERE Clause
  - Selects values that satisfy some condition expressed as a Boolean expression
  - Universal and Existential Quantification supported

```
for $b in doc(“books.xml”)//book
where some $a in $b/author satisfies $a/forename=“Michael”
return $b/title
```

```
for $b in doc(“books.xml”)//book
where every $a in $b/author satisfies $a/forename=“Michael”
return $b/title
```

`order by` and `return`

- The result of applying for and where clauses is a sequence of tuples bound to variables
- The return clause is executed for each tuple and produces the result of a FLWOR expression
- Each execution of a return clause is an evaluation of expressions in curly braces of the element constructor
- If there is no order by clause before the return clause, the query result is sorted according to the way the for clause iterates through values
- The order by clause is used to sort query result

OrderBy

- for $x in doc("books.xml")/bookstore/book/title order by $x return $x
  - The expression above will select all the title elements under the book elements that are under the bookstore element, and return the title elements in alphabetical order

```
<ul>
{ for $x in doc("books.xml")/bookstore/book/title
        order by $x
        return <li>{$x}</li>
}
</ul>
```

Positional Variables

- The FOR clause supports positional variables which identify the position of a given item in the expression that generated it

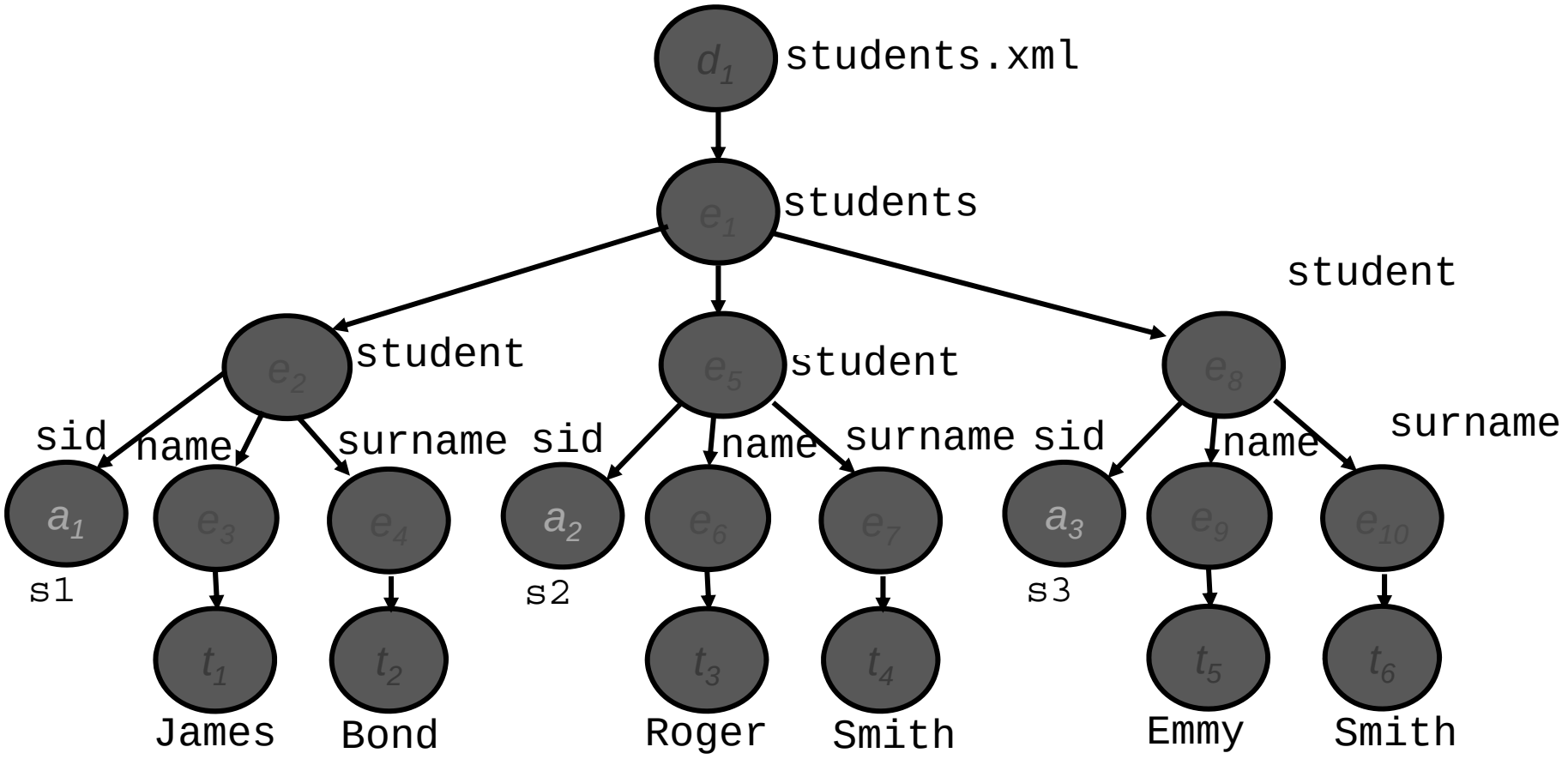
```
for $t at $i in doc("books.xml")//title
return <title pos="{$i}">{string($t)}</title>
```

```
<title pos="1">XQuery: The XML Query Language</title>
<title pos="2">XML in a Nutshell</title>
```

students.xml

```
<?xml version = "1.0" standalone = "yes"?>
  <students>
    <student sid = "s1">
       <name>James</name>
       <surname>Bond</surname>
    </student>
    <student sid = "s2">
       <name>Roger</name>
       <surname>Smith</surname>
    </student>
    <student sid = "s3">
       <name>Emmy</name>
       <surname>Smith</surname>
    </student>
  </students>
```

Tree Representation



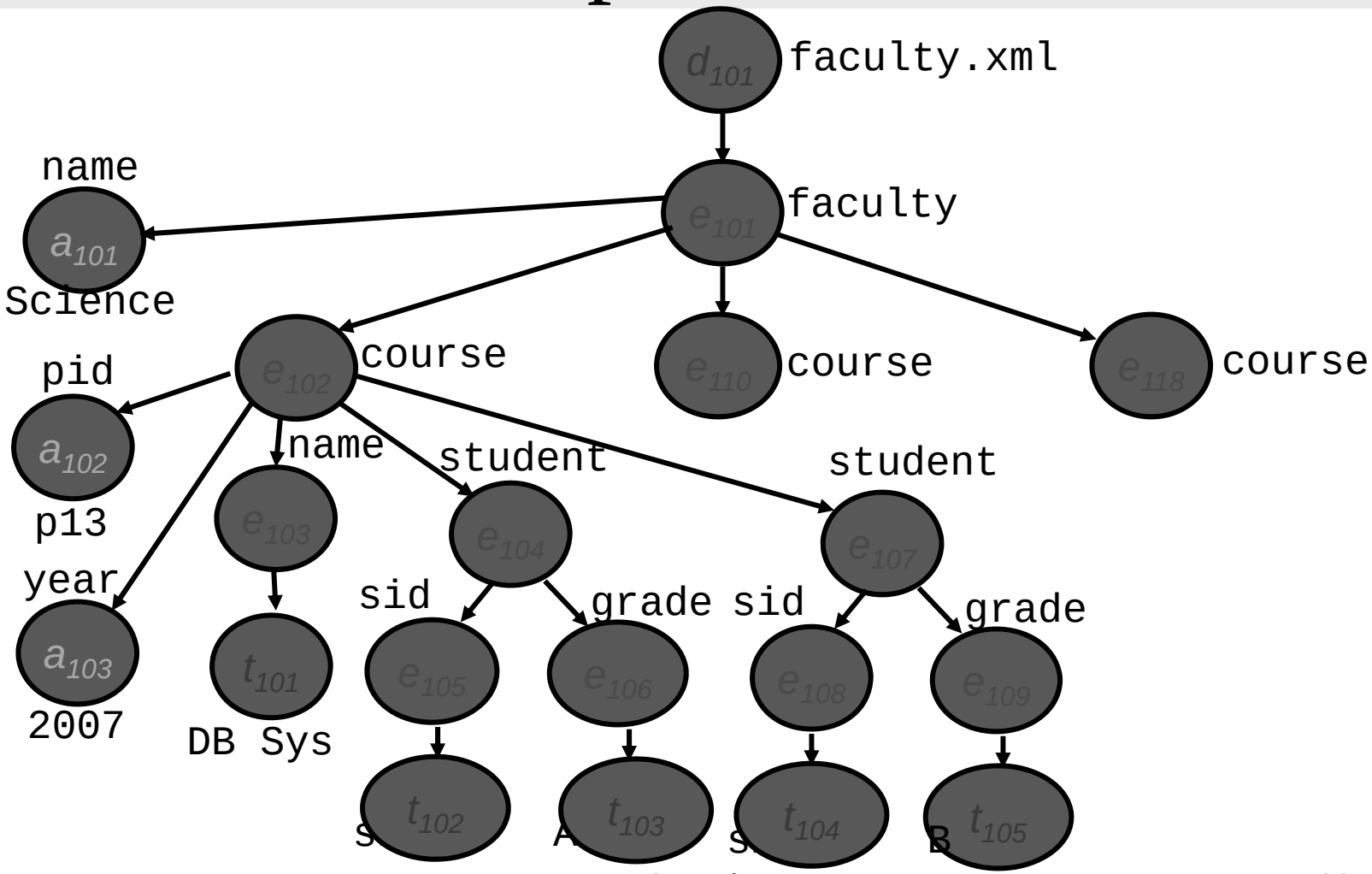
faculty.xml

```
<?xml version = "1.0" standalone = "yes"?>
   <faculty name = "Science">
      <course pid = "p13" year = "2004">
            <name>DB Sys</name>
            <student>
               <sid>s1</sid>
               <grade>A+</grade>
            </student>
            <student>
               <sid>s2</sid>
               <grade>B</grade>
            </student>
        </course>
```

faculty.xml

```
<course pid = "p14" year = "2004">
        <name>Issues in DB&Inf Sys</name>
        <student>
            <sid>s1</sid>
            <grade>A+</grade>
        </student>
        <student>
                <sid>s2</sid>
                <grade>D</grade>
        </student>
</course>
<course pid = "p13" year = "2003">
        <name>DB Sys</name>
        <lecturer>Pavle</lecturer>
        <student>
          <sid>s2</sid>
          <grade>D</grade>
        </student>
</course>
</faculty>
```

Tree Representation



OrderBy - Sorting

```
<student_sort>
   {for $s in //students/student
   order by $s/name
   return
   <student sid="{$s/@sid}">{$s/name}</student>}
</student_sort>
```

```
<student_sort>
        <student sid="s3">
                <name>Emmy</name>
        </student>
        <student sid="s1">
                <name>James</name>
        </student>
        <student sid="s2">
                <name>Roger</name>
        </student>
</student_sort>
```

Inverted Queries

- The natural way of traversing an XML document is from the root towards leaves
- Queries asking for subordinated concepts are simple
- Inverted queries are implemented by query nesting
- The outer query returns a sequence of subordinated concepts, whereas the inner query returns a sequence of corresponding superior concepts
- Next query returns a sequence of course groups passed by each student from the XML document faculty.xml

Inverted Sequences

```
<student_course>
    {
    for $s in distinct-values(//faculty/course/student/sid)
            return
            <student sid="{$s}">
            {
            for $c in distinct-values
                                    (//faculty/course[student/sid=$s]/name)
                    return
                            <course>{$c}</course>
            }
            </student>
    }
</student_course>
```

Analysis Of The Inverted Query

- The outer query binds the variable $s to those sid elements that have different string values using the function fn:distinct-values
- For each $s value, the inner query traverses the whole faculty.xml document looking for those course elements that have student elements with the sid value equal to the current value of the variable $s and binds name elements of these course elements to the variable $c
- The query constructs the result for each value of $s and the sequence of associated $c values

Queries with Aggregates

- Database queries frequently use aggregate functions with grouping
- The following query returns courses having the number of students enrolled grater than a given constant
- In the query, the variable `$c` binds in tern each `course` element, and the variable `$s` binds all `student` elements that are subordinated to the current `course`
- The value of the variable `$s` is a sequence of student elements

Aggregate Query

```
<popular_courses>
{
        for $c in //faculty/course
        let $s:=$c/student
        for $t in $c[count($s) > 1]
        return
                <course year="{$t/@year}">
                        {$t/name}
                        <no_of_std>
                                {count($s)}
                        </no_of_std>
                </course>
}
</popular_courses>
```

Aggregate Query (std Sugg)

```
<popular_courses>
{
        for $c in //faculty/course[student]
        let $s:= count($c/student)
        where $s > 1
        order by $s descending
        return
                <course year="{$c/@year}">
                        {$t/name}
                        <no_of_std>
                                {count($s)}
                        </no_of_std>
                </course>
}
</popular_courses>
```

Joining Documents

- XQuery allows joining of XML documents
- Let d_1 and d_2 be two XML documents, p_1 and p_2 paths through d_1 and d_2, respectively, and let $\$t_1$ binds result returned by p_1, and $\$t_2$ binds result returned by p_2
- Joining of d_1 and d_2 (or parts of them) is accomplished by:
  - Defining a nested query with p_1 in the outer and p_2 in the inner query, and
  - Equating $\$t_1$ and $\$t_2$ in the inner query
- The following query returns students enrolled into the Database Systems course in year 2004

Join Query

```
for $c in fn:doc(faculty.xml)/faculty/course
 return
   <course name="{$c/name/text()}" year="{$c/@year}">
          {
          for $s in  fn:doc(students.xml)/students/
                                              student[@sid=$c/student/sid]
            return
              <student sid ="{$s/@sid}">
                     {
                     $s/name,
                     $s/surname,
                     $c/student[sid=$s/@sid]/grade
                     }
              </student>
          }
    </course>
```

Conditional Expressions

- XQuery provides usual if, then, else clauses for program branching
- Each of these three clauses is associated with an expression
- The expression associated with the if clause is called test expression and it is placed within curly braces
- If the test expression is true, or it returns a non empty sequence, the then expression is executed, otherwise the else expression is executed

XQuery with a Conditional Expression

```
for $c in fn:doc(faculty.xml)/faculty/course
        return
                <course name="{$c/name/text()}“
                                        year="{$c/@year}">
                {
                if ($c/lecturer) then
                        $c/lecturer
                else
                        "Lecturer is still undetermined!"
                }
                </course>
```

Another Join Example

- Books that cost more at Amazon than MyBooks

```
let $amazon :=
    doc("http://www.amazon.co.uk/books.xml"),
    $mb := doc("http://www.mybooks.co.uk/books.xml")

for $a in $amazon/bib/book, $b in $mb/bib/book
where $a/isbn = $b/isbn and $a/price > $b/price
return <book>{ $a/title, $a/price, $b/price }</book>
```

Left Outer Join

- Books at amazon and mybooks with both prices and all books at amazon with price

```
for $a in $amazon/bib/book, $b in $mb/bib/book
where $a/isbn = $b/isbn
return <book>{ $a/title, $a/price, $b/price }</book>

for $a in $amazon/bib/book
where not($a/isbn = $mb/bib/book/isbn)
return <book>{ $a/title, $a/price }</book>
```

Equivalent Expressions

```
doc("books.xml")/bib/book[@year < 2000]/title
=
let $root := doc("books.xml") return
    for $books in $root/bib return
        for $book in $books/book return
        if (
                not(empty(
                        for $year in $book/@year return
                                if $year < 2000 then $year else ( )
                ))
        ) then
                $book/title
        else
                ( )
```

A Running Example

```
<!DOCTYPE BARS [
    <!ELEMENT BARS (BAR*, BEER*)>
    <!ELEMENT BAR (PRICE+)>
        <!ATTLIST BAR name ID #REQUIRED>
    <!ELEMENT PRICE (#PCDATA)>
        <!ATTLIST PRICE theBeer IDREF #REQUIRED>
    <!ELEMENT BEER EMPTY>
        <!ATTLIST BEER name ID #REQUIRED>
        <!ATTLIST BEER soldBy IDREFS #IMPLIED>
]>
```

FOR vs LET

```
for $beer in document(”bars.xml”)/BARS/BEER/@name
return   <BEERNAME> {$beer} </BEERNAME>

let $d := document(”bars.xml”)
let $beers := $d/BARS/BEER/@name
Return <BEERNAMES> {$beers} </BEERNAMES>
```

Bonus question: what are the results of the above queries?

Query Sample

- Let us produce the PRICE elements (from all bars) for all the beers that are sold by Vinc's Bar.
- The output will be BBP elements with the names of the bar and beer as attributes and the price element as a subelement.

Query Sample

1. Create a triple for-loop, with variables ranging over all BEER elements, all BAR elements, and all PRICE elements within those BAR elements.
2. Check that the beer is sold at Vinc's Bar and that the name of the beer and theBeer in the PRICE element match.
3. Construct the output element.

Query Sample

```
let $bars := doc(”bars.xml”)/BARS
for $beer in $bars/BEER
   for $bar in $bars/BAR
     for $price in $bar/PRICE
   where $beer/@soldAt = ”VincBar”
   and $price/@theBeer =
   $beer/@name
return <BBP bar = {$bar/@name} beer
   = {$beer/@name}>{$price}</BBP>
```

Element Comparison

- What is the result of the following?

```
/BARS/BAR[@name=”VincBar”]/
PRICE[@theBeer=”Bud”] eq
/BARS/BAR[@name=”MaryBar”]/
PRICE[@theBeer=”Bud”]
```

is false, even if Vinc and Mary charge the same for Bud.

Element Comparison

- For elements to be equal, they must be the same, physically, in the implied document.
- Subtlety: elements are really pointers to sections of particular documents, not the text strings appearing in the section.
- To extract just the value (e.g., the price itself) from an element *E*, use data(*E*).

Distinct Values

```
return distinct-values(
   let $bars = doc(”bars.xml”)
   return $bars/BARS/BAR/PRICE
)
```

Some

- The bars that sell at least one beer for less than $2.

```
for $bar in doc(”bars.xml”)/BARS/BAR
where some $p in $bar/PRICE
      satisfies $p < 2.00
return $bar/@name
```

Every

- The bars that sell no beer for more than $5.

```
for $bar in doc(”bars.xml”)/BARS/BAR
where every $p in $bar/PRICE
      satisfies $p <= 5.00
return $bar/@name
```

Xquery Functions

- XQuery includes over 100 built-in functions.
  - There are functions for string values, numeric values, date and time comparison, node and QName manipulation, sequence manipulation, Boolean values, and more.
  - The URI of the XQuery function namespace is http://www.w3.org/2005/02/xpath-functions
  - e.g. fn:abs(), fn:string(), fn:substring(string,start,len)
- Can also define your own functions in XQuery.

Xquery Functions

```
declare function local:minPrice($p as xs:decimal?, $d as
    xs:decimal?) AS xs:decimal?
    {
        let $disc := ($p * $d) div 100
        return ($p - $disc)
    }
```

- <minPrice>{local:minPrice($book/price,$book/discount)}
  </minPrice>
-