CS 4604: Introduction to Database Management Systems

B. Aditya Prakash

Lecture #23: Querying XML
The XPath/XQuery Data Model

- Corresponding to the fundamental “relation” of the relational model is: *sequence of items*. 

- An *item* is either:
  1. A primitive value, e.g., integer or string.
  2. A node.
Principal Kinds of Nodes


2. *Elements* are pieces of a document consisting of some opening tag, its matching closing tag (if any), and everything in between.

3. *Attributes* are names that are given values inside opening tags.
Document Nodes

- Formed by doc(URL) or document(URL) (or doc(filename) or document(filename))

- **Example**: doc("/usr/class/cs4604/bars.xml")

- All XPath (and XQuery) queries refer to a doc node, either explicitly or implicitly.
Example DTD

<!DOCTYPE Bars [
  <!ELEMENT BARS (BAR*, BEER*)>
  <!ELEMENT BAR (PRICE+)>
  <!ATTLIST BAR name = ID>
  <!ELEMENT PRICE (#PCDATA)>
  <!ATTLIST PRICE theBeer = IDREF>
  <!ELEMENT BEER ()>
  <!ATTLIST BEER name = ID, soldBy = IDREFS>
]>
Example Document

<BARS>
  
  <BAR name = ”JoesBar”>
  <PRICE theBeer = ”Export”>2.50</PRICE>
  <PRICE theBeer = ”Gr.Is.”>3.00</PRICE>
  </BAR> …

  <BEER name = ”Export” soldBy = ”JoesBar SuesBar … ”/>

  …

</BARS>

An element node

An attribute node

Document node is all of this, plus the header ( <? xml version… ).
Nodes as Semistructured Data

Blue = document
Green = element
Orange = attribute
Purple = primitive value
XPATH and XQUERY

- XPATH is a language for describing paths in XML documents.
  - Really think of the semi-structured data graph and its paths.
  - The result of the described path is a sequence of items.
  - Compare with SQL:
    - SQL is a language for describing relations in terms of other relations.
    - The result of a query is a relation (bag) made up of tuples

- XQUERY is a full query language for XML documents with power similar to SQL.
Path Descriptors

- Simple path descriptors are sequences of tags separated by slashes (/).
  - The format used is strongly reminiscent of UNIX naming conventions.
  - Construct the result by starting with just the doc node and processing each tag from the left.
- If the descriptor begins with /, then the path starts at the root and has those tags, in order.
- If the descriptor begins with //, then the path can start anywhere.
Example: /BARS/BAR/PRICE

<BARS>
  <BAR name = “JoesBar”>
    <PRICE theBeer = “Bud”>2.50</PRICE>
    <PRICE theBeer = “Miller”>3.00</PRICE>
  </BAR> ...
  <BEER name = “Bud”, soldBy = “JoesBar, SuesBar,...”>
  </BEER> ...
</BARS>

/BARS/BAR/PRICE describes the set with these two PRICE objects as well as the PRICE objects for any other bars.
Example: //PRICE

<BARS>
  <BAR name = "JoesBar">
    <PRICE theBeer = "Bud">2.50</PRICE>
    <PRICE theBeer = "Miller">3.00</PRICE>
  </BAR>
  ...
  <BEER name = "Bud", soldBy = "JoesBar, SuesBar,...">
    //PRICE describes the same PRICE objects, but only because the DTD forces every PRICE to appear within a BARS and a BAR.
  </BEER>
  ...
</BARS>
Wild-Card *

- A star (*) in place of a tag represents any one tag.

- Example: /*/*/PRICE represents all price objects at the third level of nesting.
Example: /BARS/*

<BARS>
  <BAR name = "JoesBar">
    <PRICE theBeer = "Bud">2.50</PRICE>
    <PRICE theBeer = "Miller">3.00</PRICE>
  </BAR>
  ...
  <BEER name = "Bud", soldBy = "JoesBar, SuesBar,...">
    ...
  </BEER>
  ...
</BARS>

/BARS/* captures all BAR and BEER objects, such as these.
Attributes

- In XPATH, we refer to attributes by prepending @ to their name.

- Attributes of a tag may appear in paths as if they were nested within that tag.
Example: /BARS/*/@name

/BARS/*/@name selects all name attributes of immediate subobjects of the BARS object.
Selection Conditions

- A condition inside [...] may follow a tag.

- If so, then only paths that have that tag and also satisfy the condition are included in the result of a path expression.
Example: Selection Condition

- `/BARS/BAR/PRICE[PRICE < 2.75]`

```xml
<BARS>
  <BAR name = "JoesBar">
    <PRICE theBeer = "Bud">2.50</PRICE>
    <PRICE theBeer = "Miller">3.00</PRICE>
  </BAR>
  ...
</BARS>
```

The condition that the PRICE be < $2.75 makes this price, but not the Miller price
Example: Attribute in Selection

- `/BARS/BAR/PRICE[@theBeer = “Miller”]`

```xml
<BARS>
  <BAR name = “JoesBar”>
    <PRICE theBeer = “Bud”>2.50</PRICE>
    <PRICE theBeer = “Miller”>3.00</PRICE>
  </BAR>
  ...  
</BARS>
```

Now, this PRICE object is selected, along with any other prices for Miller.
In general, path expressions allow us to start at the root and execute a sequence of steps to find a set of nodes at each step.

At each step, we may follow any one of several axes.

The default axis is child:: --- go to any child of the current set of nodes.
Example: Axes

- /BARS/BEER is really shorthand for /BARS/child::BEER.
- @ is really shorthand for the attribute:: axis.
  - Thus, /BARS/BEER[@name = “Bud”] is shorthand for
    /BARS/BEER[attribute::name = “Bud”]
More Axes

- Some other useful axes are:
  - parent:: = parent(s) of the current node(s).
  - descendant-or-self:: = the current node(s) and all descendants.
    - Note: // is really a shorthand for this axis.
  - ancestor::, ancestor-or-self, etc.
XQuery

- XQuery extends XPath to a query language that has power similar to SQL.
- Uses the same sequence-of-items data model as XPath.
- XQuery is an expression language.
  - Like relational algebra --- any XQuery expression can be an argument of any other XQuery expression.
FLWR Expressions

- The most important form of XQuery expressions involves for-, let-, where-, return- (FLWR) clauses.
- A query begins with one or more for and/or let clauses.
  - The for’s and let’s can be interspersed.
- Then an optional where clause.
- A single return clause.

Form:
- for variable in expression
- let variable := expression
- where condition
- return expression
Semantics of FLWR Expressions

- Each `for` creates a loop.
  - `let` produces only a local variable assignment.

- At each iteration of the nested loops, if any, evaluate the `where` clause.

- If the `where` clause returns TRUE, invoke the `return` clause, and append its value to the output.
  - So `return` can be thought of as “add to result”
FOR Clauses

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

- Variables begin with $.
- A FOR variable takes on each object in the set denoted by the path expression, in turn.
- Whatever follows this FOR is executed once for each value of the variable.
Example: FOR

FOR $beer IN /BARS/BEER/@name
RETURN <BEERNAME>$beer</BEERNAME>

- $beer ranges over the name attributes of all beers in our example document.
- Result is a list of tagged names, like <BEERNAME>Bud</BEERNAME> <BEERNAME>Miller</BEERNAME>...
LET Clauses

LET <variable> := <path expression>,...

- Value of the variable becomes the set of objects defined by the path expression.
- Note LET does not cause iteration; FOR does.
Example: LET

```sql
LET $beers := /BARS/BEER/@name
RETURN <BEER NAMES>$beers</BEER NAMES>
```

- Returns one object with all the names of the beers, like:

  ```xml
  <BEER NAMES>Bud, Miller,…</BEER NAMES>
  ```
Order-By Clauses

- FLWR is really FLWOR: an order-by clause can precede the return.
- Form: order by <expression>
  - With optional ascending or descending.
- The expression is evaluated for each assignment to variables.
- Determines placement in output sequence.
Example: Order-By

- List all prices for Export, lowest price first.

Let $d := \text{document("bars.xml")} \quad \text{for } p \text{ in } \quad \text{generate bindings for } p \text{ to } \text{price elements.}

Order those bindings by the values inside the elements.

Each binding is evaluated for the output. The result is a sequence of PRICE elements.
XQUERY (but not XPATH) allows us to use paths that follow attributes that are IDREF’s.

If $x$ denotes a set of IDREF’s, then $x \Rightarrow y$ denotes all the objects with tag $y$ whose ID’s are one of these IDREF’s.
Example

- Find all the beer objects where the beer is sold by Joe’s Bar for less than 3.00.

- Strategy:
  1. $beer will for-loop over all beer objects.
  2. For each $beer, let $joe be either the Joe’s-Bar object, if Joe sells the beer, or the empty set of bar objects.
  3. Test whether $joe sells the beer for < 3.00.
Example: The Query

FOR $beer IN /BARS/BEER
LET $joe := $beer/@soldBy=>BAR[@name="JoesBar"]
LET $joePrice := $joe/PRICE[@theBeer=$beer/@name]
WHERE $joePrice < 3.00
RETURN <CHEAPBEER>$beer</CHEAPBEER>

Attribute soldBy is of type IDREFS. Follow each ref to a BAR and check if its name is Joe’s Bar.

Find that PRICE subobject of the Joe’s Bar object that represents whatever beer is currently $beer.

Only pass the values of $beer, $joe, $joePrice to the RETURN clause if the string inside the PRICE object $joePrice is < 3.00.
Aggregations

- XQuery allows the usual aggregations, such as sum, count, max, min.
- They take any sequence as argument.
- E.g. find bars where all beers are under $5.

```xml
let $bars = doc("bars.xml")/BARS
for $price in $bars/BAR/PRICE
where max($price) < 5
return $bar/BAR/@name
```
Plan till the end of classes

- PHP/HTML, a quick tutorial
- Practice Problems (from previous exams)
- **NO CLASS on Friday May 3**
- Final Review on Wed May 8 (last class)
  - Project demos in-person on Mon May 6 and Wed May 8 during my office hours in McBryde 122C: Sign-up link posted (sign-up by May 3, 5pm)
- Finals on (check University Schedule too)
  - Saturday May 11, 1:05-3:05pm