XML Query Languages

XPATH

XQUERY

XPATH and XQUERY

- XPATH is a language for describing paths in XML documents.
- Really think of the semistructured data graph and its paths.
- XQUERY is a full query language for XML documents with power similar to OQL.

Example DTD

```xml
<!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

```xml
<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>
```

Path Descriptors

- Simple path descriptors are sequences of tags separated by slashes (/).
- 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

```xml
<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>
```
Example: //PRICE

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

```xml
<BARS>
  <PRICE theBeer = "Bud">2.50</PRICE>
  <PRICE theBeer = "Miller">3.00</PRICE>
  ...  
  <BARS>* captures all BAR and BEER objects, such as these.
</BARS>
```

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

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

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]
  <BARS>
  <BAR name = "JoesBar">
  <PRICE theBeer = "Miller">3.00</PRICE>
  </BAR> ...  The condition that the PRICE be < 2.75 makes this price but not the Miller price satisfy the path descriptor.
  </BARS>

Example: Attribute in Selection

◆ /BARS/BAR/PRICE[@theBeer = "Miller"]
  <BARS>
  <BAR name = "JoesBar">
  <PRICE theBeer = "Bud">2.50</PRICE>
  </BAR> ...  Now, this PRICE object is selected, along with any other prices for Miller.
  </BARS>

Axes

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:
1. parent:: = parent(s) of the current node(s).
2. descendant-or-self:: = the current node(s) and all descendants.
   • Note: // is really a shorthand for this axis.
3. ancestor::, ancestor-or-self, etc.

XQUERY

XQUERY allows us to query XML documents, using path expressions from XPATH to describe important sets.
Corresponding to SQL’s `select-from-where` is the XQUERY FLWR expression, standing for “for-let-where-return.”
**FLWR Expressions**

1. One or more FOR and/or LET clauses.
2. Then an optional WHERE clause.
3. A RETURN clause.

**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**

LET $beers := /BARS/BEER/@name
RETURN
<BEERNAME>$beers</BEERNAME>

- Returns one object with all the names of the beers, like:
  <BEERNAME>Bud, Miller,...</BEERNAME>

**Following IDREF’s**

- 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 $x => y$ 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 :=
LET $joePrice :=
RETURN <CHEAPBEER>$beer</CHEAPBEER>

Attribute soldBy is of type IDREPS. 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 whichever 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.