

Query Languages for XML

XPath

XQuery

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* (defined next).

Principal Kinds of Nodes

1. *Document nodes* represent entire documents.
2. *Elements* are pieces of a document consisting of some opening tag, its matching closing tag (if any), and everything in between.
3. *Attributes* names that are given values inside opening tags.

Document Nodes

- Formed by `doc(URL)`.
- **Example:** `doc(/usr/class/cs145/bars.xml)`
- All XPath (and XQuery) queries refer to a doc node, either explicitly or implicitly.
 - **Example:** key definitions in XML Schema have Xpath expressions that refer to the document described by the schema.

DTD for 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>  
>
```

Example Document

Document node is all of this, plus the header (<? xml version...).

```
<BARS>
```

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

An element node



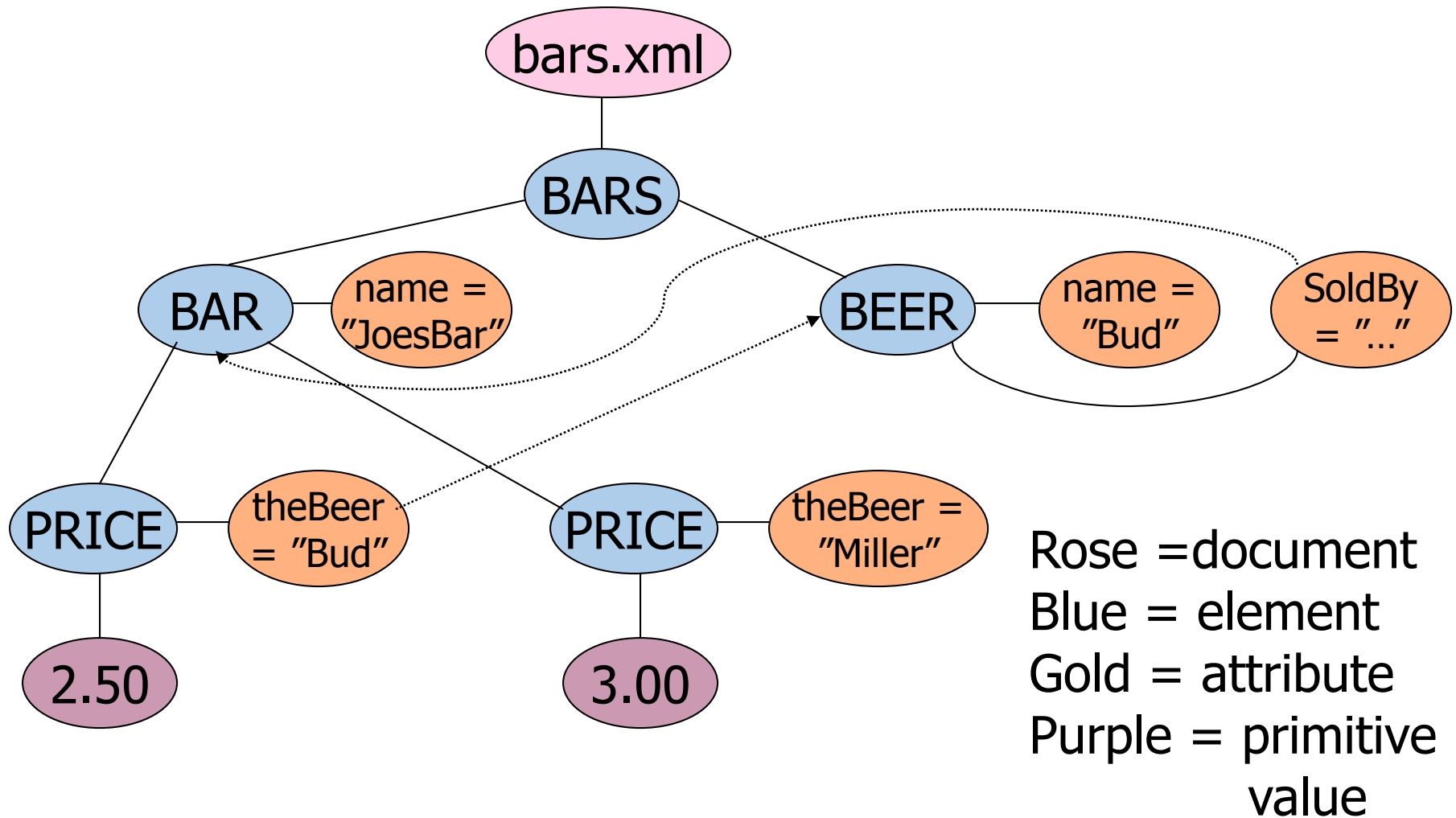
```
<BEER name = "Bud" soldBy = "JoesBar  
SuesBar ... "/> ...
```

An attribute node



```
</BARS>
```

Nodes as Semistructured Data



Paths in XML Documents

- XPath is a language for describing paths in XML documents.
- The result of the described path is a sequence of items.

Path Expressions

- Simple path expressions are sequences of slashes (/) and tags, starting with /.
 - **Example:** /BARS/BAR/PRICE
- Construct the result by starting with just the doc node and processing each tag from the left.

Evaluating a Path Expression

- Assume the first tag is the root.
 - Processing the doc node by this tag results in a sequence consisting of only the root element.
- Suppose we have a sequence of items, and the next tag is X .
 - For each item that is an element node, replace the element by the subelements with tag X .

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 ... "/> ...
</BARS>
```

One item, the
BARS element

Example: /BARS/BAR

<BARS>

<BAR name = "JoesBar">

<PRICE theBeer = "Bud">2.50</PRICE>

<PRICE theBeer = "Miller">3.00</PRICE>

</BAR> ...

<BEER name = "Bud" soldBy = "JoesBar

SuesBar ..."/> ...

</BARS>

This BAR element followed by
all the other BAR elements

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 ..."/> ...

</BARS>

These PRICE elements followed
by the PRICE elements
of all the other bars.

Attributes in Paths

- Instead of going to subelements with a given tag, you can go to an attribute of the elements you already have.
- An attribute is indicated by putting @ in front of its name.

Example: /BARS/BAR/PRICE/@theBeer

<BARS>

<BAR name = "JoesBar">

<PRICE theBeer = "Bud">2.50</PRICE>

<PRICE theBeer = "Miller">3.00</PRICE>

</BAR> ...

<BEER name = "Bud" soldBy = "JoesBar

SuesBar ..."/> ...

</BARS>

These attributes contribute "Bud" "Miller" to the result, followed by other theBeer values.

Remember: Item Sequences

- Until now, all item sequences have been sequences of elements.
- When a path expression ends in an attribute, the result is typically a sequence of values of primitive type, such as strings in the previous example.

Paths that Begin Anywhere

- If the path starts from the document node and begins with `//X`, then the first step can begin at the root or any subelement of the root, as long as the tag is `X`.

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 ..."/> ...

</BARS>

These PRICE elements and
any other PRICE elements
in the entire document

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/*

This BAR element, all other BAR elements, the BEER element, all other BEER elements

<BARS>

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

```
<BEER name = "Bud" soldBy = "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[. < 2.75]

<BARS>

<BAR name = "JoesBar">

<PRICE theBeer = "Bud">2.50</PRICE>

<PRICE theBeer = "Miller">3.00</PRICE>

</BAR> ...

The current element.

The condition that the PRICE be < \$2.75 makes this price but not the Miller price part of the result.

Example: Attribute in Selection

- /BARS/BAR/PRICE[@theBeer = "Miller"]

<BARS>

<BAR name = "JoesBar">

<PRICE theBeer = "Bud">2.50</PRICE>

<PRICE theBeer = "Miller">3.00</PRICE>

</BAR> ...

↑
Now, this PRICE element
is selected, along with
any other prices for Miller.

Axes

- In general, path expressions allow us to start at the root and execute steps to find a sequence of nodes at each step.
- At each step, we may follow any one of several *axes*.
- The default axis is *child::* --- go to all the children 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"]

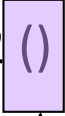
More Axes

- 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 shorthand for this axis.
 3. **ancestor::**, **ancestor-or-self**, etc.
 4. **self** (the dot).

XQuery

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

More About Item Sequences

- XQuery will sometimes form sequences of sequences.
- All sequences are flattened.
- **Example:** (1 2  (3 4)) = (1 2 3 4).

↑
Empty
sequence

FLWR Expressions

1. One or more **for** and/or **let** clauses.
2. Then an optional **where** clause.
3. A **return** clause.

Semantics of FLWR Expressions

- Each **for** creates a loop.
 - **let** produces only a local definition.
- 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.

FOR Clauses

for <variable> in <expression>, . . .

- Variables begin with \$.
- A **for**-variable takes on each item in the sequence denoted by the expression, in turn.
- Whatever follows this **for** is executed once for each value of the variable.

Example: FOR

Our example
BARS document

“Expand the enclosed string by replacing variables and path exps. by their values.”

for \$beer in document("bars.xml")/BARS/BEER/@name
return

<BEERNAME>{\$beer}</BEERNAME>

- \$beer ranges over the name attributes of all beers in our example document.
- Result is a sequence of BEERNAME elements:
<BEERNAME>Bud</BEERNAME>
<BEERNAME>Miller</BEERNAME> . . .

Use of Braces

- When a variable name like `$x`, or an expression, could be text, we need to surround it by braces to avoid having it interpreted literally.
 - **Example:** `<A>$x` is an A-element with value "`$x`", just like `<A>foo` is an A-element with "`foo`" as value.

LET Clauses

let <variable> := <expression>, . . .

- Value of the variable becomes the *sequence* of items defined by the expression.
- Note **let** does not cause iteration; **for** does.

Example: LET

```
let $d := document("bars.xml")
```

```
let $beers := $d/BARS/BEER/@name
```

```
return
```

```
<BEERNAMES> {$beers} </BEERNAMES>
```

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

```
<BEERNAMES>Bud Miller ...</BEERNAMES>
```

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 Bud, lowest first.

```
let $d := document("bars.xml")
```

```
for $p in $d/BARS/BAR/PRICE[@theBeer="Bud"]
```

```
order by $p
```

```
return $p
```

Order those bindings by the values inside the elements .

Generates bindings for \$p to PRICE elements.

Each binding is evaluated for the output. The result is a sequence of PRICE elements.

Aside: SQL ORDER BY

- SQL works the same way; it's the result of the FROM and WHERE that get ordered, not the output.

- **Example:** Using R(a,b),

```
SELECT b FROM R
```

```
WHERE b > 10
```

```
ORDER BY a;
```

Then, the b-values are extracted from these tuples and printed in the same order.

R tuples with $b > 10$ are ordered by their a-values.

Predicates

- Normally, conditions imply existential quantification.
- **Example:** /BARS/BAR[@name] means “all the bars that have a name.”
- **Example:** /BARS/BEER[@soldAt = "JoesBar"] gives the set of beers that are sold at Joe's Bar.

Example: Comparisons

- Let us produce the PRICE elements (from all bars) for all the beers that are sold by Joe'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.

Strategy

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 Joe's Bar and that the name of the beer and **theBeer** in the PRICE element match.
3. Construct the output element.

The Query

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

True if "JoesBar"
appears anywhere
in the sequence

Actions

- Read Chapters 12.1 (XPath) and 12.2 (XQuery)
- Review slides!