Theme 7.2

XQuery

Querying in XML

- A query language can be seen as **specialized language** used for requesting information from a **database**.
- While there are several different query languages for database (QL languages)
- XML uses XPath and XQuery

XQuery

- XQuery was design to **select content** from an XML data source…
- …**transform** the content as directed…
- …then **return** the new content as either xml or some other **format**.

- Query utilises XPath to navigate and **manipulate** the content.

- **Query other data** including databases whose structure is similar to XML

Use of XQuery

- Finding **textual documents** in a native XML database and presenting **styled results**
- Generating reports on data stored in a database for presentation on the Web as **XHTML**
- Extracting information from a relational database for use in a **web service** (SOAP, WSDL)

Use of XQuery

- Pulling data from databases or packaged software and transforming it for **application integration**
- Combining content from traditionally non-XML sources to implement content **management** and **delivery**
- Ad hoc querying of standalone XML documents for the **purposes** of **testing** or **research**

XQuery Path Expressions

- The most straight forward kind of query simply selects elements or attributes from an input document.

```
doc("catalog.xml")/catalog/product
```

- The basic structure of many (but not all) queries is the **FLWOR** expression.

FLWOR Expressions

- XQuery introduces FLWOR (pronounced “flower”) expressions.
- Their structure is based on SQL Select.
- **FLWOR**= **F**or, **L**et, **W**here, **O**rder by, **R**eturn.
- You must have at least one **for** or **let**; you can have more.
- The **where** and **order by** clauses are optional.

FLWOR Expressions

- **For** - This clause sets up an **iteration** through the an elements…
- …and the rest of the FLWOR is evaluated once for each of the element selected.
- **Let** - Clause serves as a programmatic convenience that avoids repeating the same expression multiple times.
- **Where** - This clause selects only results where the condition is met.
- **Order by** -This clause sorts the results by a condition.
- **Return –** This clause indicates the final product that should be returned

FLWOR Expressions

```
for $employee in doc("source.xml")/employees/employee
let $emplID:= $employee/@emplID
where $employee/name="Gabe"
order by $employee/surname
return($employee/surname, $emplID)
```

FLWOR Expressions

Interpretation:

Use $employee to represent **each employee** in the node set...

Use $emplID to represent each employee's ID...

Test each employee to see if their name is "Gabe"...

Order the matching results by employee surnames...

And finally return each matching employee's surname and ID.

Path expression – Axis Step

- Tells the query engine which way to **navigate from the context node**…
- ..which **test** to perform when it encounters nodes along the way

- An axis step has three parts:
  - an optional **axis specifier**
  - **node test**
  - zero or more **predicates**

```
…path/axis-specifier::nodetest(nodes)
```

Path expression – Axis Specifiers

Axis Specifer	Refers to…	Shorthand form
`self::`	the context node itself	.
`attribute::`	all attribute nodes of the context node	@
`ancestor::`	all ancestors of the context node (parent, grandparent, etc)	..
`descendant::`	all descendants of the context node (children, grandchildren, etc)	/

Path expression – Node Tests

Node test	Refers to…
`node()`	nodes of any kind
`text()`	text nodes
`element()`	element nodes (same as star: *)
`attribute()`	attribute nodes

Putting things to perspective

- Longhand form:

```
doc('cookbook.xml')/
descendant-or-self::element(recipe)/
child::element(title)
```

- Short form:

```
doc('cookbook.xml')//recipe/title
```

Combination of XPath and XQuery

Method calling:

```
doc('animal.xml')//snake[string-length(legs) = 0]
```

Positional Predicates:

```
doc('queue.xml')/bank/counter[position() = 2]
```

Boolean Predicates:

```
doc('mine.xml')/gold[output[empty(gold)]]
```

Constructing XML Elements and Attributes

- Sometimes you want to **reorganize** or **transform** the elements from…
- …the input document(s)…
- …into differently named or structured elements.

- XML constructors can be used to create elements and attributes that appear in the query results.

Adding Elements

```
<ul>{
for $chicken in doc("chicken.xml")/chickens/chicken
          where $chicken/@alive='true'
          order by $chicken/name
          return $chicken/name
}</ul>
```

```
<ul>
   <name> Dark One-Leg Zoe </name>
   <name> Its Ma'am </name>
</ul>
```

Adding Attribute

```
<ul type="square">{
for $toiletpaper in doc("house.xml")/toilet/toiletpaper
return <li status="{$toiletpaper/@status}">
         {data($toiletpaper/brand)}
       </li>
}</ul>
```

Data() function extracts the contents of an element

Adding Attribute

```
<ul type="square">
  <li status = "safe">Twinsaver</li>
  <li status = "unsafe">Very dodge brand</li>
</ul>
```

Joining Data Sources

- One benefits of FLWORs is joining data from multiple sources.
- For example:
  - You want to join information from your product catalog (catalog.xml) and your order (order.xml).
  - You want a list of all the items in the order, along with their number, name, and quantity.

Joining Data Sources

```
for $item in doc("order.xml")//item
let $name := doc("catalog.xml")//product[id = $item/@id]/name
return <item id="{$item/@id}“
             name="{$name}“
             quan="{$item/@quantity}"/>
```

```
<item id="557" name="Fleece Pullover" quan="1"/>
<item id="563" name="Floppy Sun Hat" quan="1"/>
```

Aggregating and Grouping Values

- One common use for XQuery is to **summarize** and **group** XML data.
- It is sometimes useful to find the sum, average, or maximum of a sequence of values, grouped by a particular value.

Aggregating and Grouping Values

```
for $dep in doc("studies.xml")//module/@depart
let $mod := doc("studies.xml")//module[@depart = $dep]
order by $dep
return <department name="{$dep}">{
        for $m in $mod
        order by $m/@code
        return $m
}
</department>
```

Aggregating and Grouping Values

```
<department name="cs">
      <module depart="cs" code="110"/>
</department>
<department name="mm">
      <module depart="mm" code="210"/>
      <module depart="mm" code="211"/>
</department>
```

End of Theme 7