Lecture 12: XQuery in SQL Server

Monday, October 23, 2006
Announcements

• Homework 2 due on Wednesday

• Midterm on Friday. To study:
  – SQL
  – E/R diagrams
  – Functional dependencies and BCNF

• Project phase 2 due next Wednesday
Sorting in XQuery

```xml
<publisher_list>
{ FOR $b IN document("bib.xml")//book[year = "97"]
  ORDER BY $b/price/text()
  RETURN <book>
    { $b/title , $b/price }
  </book>
}
</publisher_list>
```
If-Then-Else

```xml
FOR $h$ IN //holding
RETURN <holding>
{  $h/title,
    IF $h/@type = "Journal"
        THEN $h/editor
       ELSE $h/author
    }
</holding>
```
Existential Quantifiers

FOR $b$ IN //book

WHERE SOME $p$ IN $b$//para SATISFIES

contains($p$, "sailing")

AND contains($p$, "windsurfing")

RETURN { $b/title$ }
Universal Quantifiers

FOR $b$ IN //book

WHERE EVERY $p$ IN $b$//para SATISFIES

contains($p$, "sailing")

RETURN { $b/title$ }
Duplicate Elimination

- **distinct-values** (list-of-text-values)
- How do we eliminate duplicate “tuples”? 

<table>
<thead>
<tr>
<th>row</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>500</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>500</td>
</tr>
</tbody>
</table>
FOR v.s. LET

FOR

• Binds *node variables* $\rightarrow$ iteration

LET

• Binds *collection variables* $\rightarrow$ one value
FOR v.s. LET

FOR $x$ IN /bib/book
RETURN <result> { $x$ } </result>

LET $x := /bib/book$
RETURN <result> { $x$ } </result>

Returns:

<result> <book>...</book></result>
<result> <book>...</book></result>
<result> <book>...</book></result>
...

Returns:

<result> <book>...</book></result>
<book>...</book></result>
<book>...</book></result>
...

</result>
XQuery

Summary:

• FOR-LET-WHERE-RETURN = FLWR

FOR/LET Clauses

WHERE Clause

RETURN Clause

List of tuples

List of tuples

Instance of Xquery data model
Collections in XQuery

• Ordered and unordered collections
  – `/bib/book/author/text()` = an ordered collection: result is in document order
  – `distinct-values(/bib/book/author/text())` = an unordered collection: the output order is implementation dependent

• `LET $a := /bib/book` → $a is a collection

• `$b/author` → a collection (several authors...)

```
RETURN <result> { $b/author } </result>
```

Returns:
```
<result> <author>... </author> <author>... </author> <author>... </author> ...
</result>
```
Collections in XQuery

What about collections in expressions?

- $b/price \rightarrow \text{list of n prices}$
- $b/price \times 0.7 \rightarrow \text{list of n numbers}$
- $b/price \times b/quantity \rightarrow \text{list of n x m numbers} ??$
- $b/price \times (b/quant1 + b/quant2) \neq b/price \times b/quant1 + b/price \times b/quant2 !!!
Other XML Topics

- Name spaces
- XML API:
  - DOM = “Document Object Model”
- XML languages:
  - XSLT
- XML Schema
- Xlink, XPointer
- SOAP

Available from www.w3.org
(but don’t spend rest of your life reading those standards !)
XML in SQL Server 2005

• Create tables with attributes of type XML

• Use Xquery in SQL queries

• Rest of the slides are from:
  Shankar Pal et al., *Indexing XML data stored in a relational database*, VLDB’2004
CREATE TABLE DOCS (  
  ID int primary key,  
  XDOC xml)

SELECT ID, XDOC.query('  
  for $s in /BOOK[@ISBN= “1-55860-438-3”]//SECTION  
  return <topic>{data($s/TITLE)} </topic>')  
FROM DOCS
XML Methods in SQL

- Query() = returns XML data type
- Value() = extracts scalar values
- Exist() = checks conditions on XML nodes
- Nodes() = returns a rowset of XML nodes that the Xquery expression evaluates to
Examples

• From here:
XML Type

CREATE TABLE docs (  
   pk INT PRIMARY KEY,  
   xCol XML not null  
  )
Inserting an XML Value

```
INSERT INTO docs VALUES (2,  
'<doc id="123">  
  <sections>  
  <section num="1"><title>XML Schema</title></section>  
  <section num="3"><title>Benefits</title></section>  
  <section num="4"><title>Features</title></section>  
  </sections>  
</doc>')
```
Query( )

```
SELECT pk, xCol.query('/doc[@id = 123]//section')
FROM   docs
```
Exists( )

SELECT xCol.query('(/doc[@id = 123]/section')
FROM docs
WHERE xCol.exist ('(/doc[@id = 123])') = 1
SELECT xCol.value(
    'data((/doc//section[@num = 3]/title)[1])', 'nvarchar(max)'
) FROM docs
Nodes( )

```
SELECT nref.value('first-name[1]', 'nvarchar(50)')
    AS FirstName,
    nref.value('last-name[1]', 'nvarchar(50)')
    AS LastName
FROM   @xVar.nodes('//author') AS R(nref)
WHERE nref.exist('.[first-name != "David"]') = 1
```
Nodes( )

```
SELECT nref.value('@genre', 'varchar(max)') LastName
FROM docs CROSS APPLY xCol.nodes('///book') AS R(nref)
```
Internal Storage

• XML is “shredded” as a table
• A few important ideas:
  – Dewey decimal numbering of nodes; store in clustered B-tree index
  – Use only odd numbers to allow insertions
  – Reverse PATH-ID encoding, for efficient processing of postfix expressions like //a/b/c
  – Add more indexes, e.g. on data values
<BOOK ISBN="1-55860-438-3">
  <SECTION>
    <TITLE>Bad Bugs</TITLE>
    Nobody loves bad bugs.
    <FIGURE CAPTION="Sample bug"/>
  </SECTION>
  <SECTION>
    <TITLE>Tree Frogs</TITLE>
    All right-thinking people <BOLD>love</BOLD> tree frogs.
  </SECTION>
</BOOK>
<table>
<thead>
<tr>
<th>ORDPATH</th>
<th>TAG</th>
<th>NODE_TYPE</th>
<th>VALUE</th>
<th>PATH_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 (BOOK)</td>
<td>1 (Element)</td>
<td>Null</td>
<td>#1</td>
</tr>
<tr>
<td>1.1</td>
<td>2 (ISBN)</td>
<td>2 (Attribute)</td>
<td>'1-55860-438-3'</td>
<td>#2#1</td>
</tr>
<tr>
<td>1.3</td>
<td>3 (SECTION)</td>
<td>1 (Element)</td>
<td>Null</td>
<td>#3#1</td>
</tr>
<tr>
<td>1.3.1</td>
<td>4 (TITLE)</td>
<td>1 (Element)</td>
<td>'Bad Bugs'</td>
<td>#4#3#1</td>
</tr>
<tr>
<td>1.3.3</td>
<td>10 (TEXT)</td>
<td>4 (Value)</td>
<td>'Nobody loves Bad bugs.'</td>
<td>#10#3#1</td>
</tr>
<tr>
<td>1.3.5</td>
<td>5 (FIGURE)</td>
<td>1 (Element)</td>
<td>Null</td>
<td>#5#3#1</td>
</tr>
<tr>
<td>1.3.5.1</td>
<td>6 (CAPTION)</td>
<td>2 (Attribute)</td>
<td>'Sample bug'</td>
<td>#6#3#1</td>
</tr>
<tr>
<td>1.5</td>
<td>3 (SECTION)</td>
<td>1 (Element)</td>
<td>Null</td>
<td>#3#1</td>
</tr>
<tr>
<td>1.5.1</td>
<td>4 (TITLE)</td>
<td>1 (Element)</td>
<td>'Tree frogs'</td>
<td>#4#3#1</td>
</tr>
<tr>
<td>1.5.3</td>
<td>10 (TEXT)</td>
<td>4 (Value)</td>
<td>'All right-thinking people'</td>
<td>#10#3#1</td>
</tr>
<tr>
<td>1.5.5</td>
<td>7 (BOLD)</td>
<td>1 (Element)</td>
<td>'love'</td>
<td>#7#3#1</td>
</tr>
<tr>
<td>1.5.7</td>
<td>10 (TEXT)</td>
<td>4 (Value)</td>
<td>'tree frogs'</td>
<td>#10#3#1</td>
</tr>
</tbody>
</table>
SELECT SerializeXML (N2.ID, N2.ORDPATH) 
FROM infosettab N1 JOIN infosettab N2 ON (N1.ID = N2.ID) 
WHERE N1.PATH_ID = PATH_ID(/BOOK/@ISBN) 
  AND N1.VALUE = '1-55860-438-3' 
  AND N2.PATH_ID = PATH_ID(BOOK/SECTION) 
  AND Parent (N1.ORDPATH) = Parent (N2.ORDPATH)