ECS 165B: Database System Implementation
Lecture 18

UC Davis
May 7, 2010

Some slide content due to Zack Ives
Class Agenda

• Last time:
  – Overview of DavisDB Part 3: System Manager
  – Mid-course evaluation

• Today:
  – Results of mid-course evaluation
  – Querying XML

• Reading:
  – none
Mid-Course Evaluation Results

• First, thanks for your thoughtful and constructive answers!

• Things that seem to be working well:
  – Lectures seem well-received (only one of you dislikes powerpoint?!?)
  – Availability of extra help
  – Most find class relevant and useful
  – High-level ideas seem clear

• Main complaints:
  – Project is way, way, way too time-consuming
  – Not enough guidance on project; how to turn high-level ideas into code
  – My other classes are suffering
What Could Be Done Differently Next Year

• Gentler ramp-up and slower pacing for the project
• More time on preliminary fundamentals
  – General C++ refresher
  – C++ memory management
  – Serialization/deserialization techniques
• Preparatory written assignments before starting the project?
• More cookbook code for early parts of project?
• 5 credits?
What Could be Done Differently this Quarter

• Ease up on the project?
  – No, seriously. EASE UP DUDE. We know where you live.

• Some down time between assignments?

• A little more credit for those of us who worked hard but still came up short?

• Extra credit opportunities?

• Make Quiz #2 worth a little more?
Adjustments for the Rest of the Quarter

• Grading for Part 2 will weight effort (% implementation complete) higher than was done in Part 1

• More time for Part 3
  – **Now due Sunday, 5/16 @11:59pm**
  – Part 4 due date pushed back to Friday, 6/4

• Part 4 will be significantly easier than planned:
  – Bar for full credit will be relatively low
  – More starting code will be provided
  – There will be opportunities for significant extra credit
  – Details TBD

• Remember: the class will be graded on a curve
The Plan for the Rest of the Quarter (2)

Vote: should Quiz #2 be worth more (say, 20% of grade)?

Result of vote: NO
Clarifying Some Misconceptions

• "This course won't be useful because I won't be building a database in industry"

• "I don't anticipate passing this class"
Querying XML
How to Query a Directed Graph? A Tree?

General approach used by many XML, semistructured, and object-oriented query languages:

– Define some sort of a template describing traversals from the root of the directed graph

– In XML, the basis of this template is called an XPath
On the Aesthetics of XML [Wadler|Buneman]

The Evolution of Language

\[ 2x \quad \text{(Descartes)} \]

\[ \lambda x. 2x \quad \text{(Church)} \]

\[(\text{LAMBDA} \ (X) \ (* \ 2 \ X)) \quad \text{(McCarthy)}\]
XPaths

• In its simplest form, an XPath is like a path in a file system:

/mypath/subpath/*/morepath

• The XPath returns a node set representing the XML nodes (and their subtrees) at the end of the path

  – XPaths can have node tests at the end, returning only particular node types, e.g., text(), processing-instruction(), comment(), element(), attribute()

  – XPath is fundamentally an ordered language: it contains order-based predicates, and it returns nodes in document order
Sample XML

<?xml version="1.0" encoding="ISO-8859-1" ?>
<dblp>
  <mastersthesis mdate="2002-01-03" key="ms/Brown92">
    <author>Kurt P. Brown</author>
    <title>PRPL: A Database Workload Specification Language</title>
    <year>1992</year>
    <school>Univ. of Wisconsin-Madison</school>
  </mastersthesis>
  <article mdate="2002-01-03" key="tr/dec/SRC1997-018">
    <editor>Paul R. McJones</editor>
    <title>The 1995 SQL Reunion</title>
    <journal>Digital System Research Center Report</journal>
    <volume>SRC1997-018</volume>
    <year>1997</year>
    <ee>db/labs/dec/SRC1997-018.html</ee>
    <ee>http://www.mcjones.org/System_R/SQL_Reunion_95/</ee>
  </article>
</dblp>
XML Data Model Visualized

Root

?xml
dblp

mastersthesis

mdate	key
2002...

author
title
year
school

ms/Brown92
PRPL...
Kurt P....
Univ....

 article

date	key
2002...

editor	title	journal	volume	year	ene

tr/dec/...
The...
Digital...
 SRC...
http://www.

root
attribute
p-i
element
text
Some Example XPath Queries

- /dblp/mastersthesis/title
- /dblp/*/editor
- //title
- //title/text()
Context Nodes and Relative Paths

XPath has a notion of a *context* node: it’s analogous to a current directory

- “.” represents this context node
- “..” represents the parent node
- We can express relative paths:

  subpath/sub-subpath/..../..

  gets us back to the context node

➢ By default, the document root is the context node
Predicates – Selection Operations

A *predicate* allows us to filter the node set based on selection-like conditions over sub-XPaths:

```
/dblp/article[title = “Paper1”]
```

which is equivalent to:

```
/dblp/article[./title/text() = “Paper1”]
```
Axes: More Complex Traversals

• Thus far, we’ve seen XPath exps. that go down the tree (or up one step)
• But we might want to go up, left, right, etc.
  – These are expressed with so-called axes:

    self::path-step
    child::path-step                       parent::path-step
    descendant::path-step                  ancestor::path-step
    descendant-or-self::path-step          ancestor-or-self::path-step
    preceding-sibling::path-step           following-sibling::path-step
    preceding::path-step                   following::path-step

  – The previous XPaths we saw were in “abbreviated form”
Querying Order

• We saw in the previous slide that we could query for preceding or following siblings or nodes

• We can also query a node for its position according to some index:
  – \texttt{fn::first()}, \texttt{fn::last()} return index of first or last element matching the last step
  – \texttt{fn::position()} gives the relative count of the current node
  – e.g.,

\[
\text{child::article[fn::position() = fn::last()]}\]
Users of XPath

• XML Schema uses simple XPaths in defining keys and uniqueness constraints

• XQuery

• XSLT

• XLink and XPointer, hyperlinks for XML
XQuery

• A strongly-typed, Turing-complete XML manipulation language
  – Attempts to do static typechecking against XML Schema
  – Based on an object model derived from Schema

• Unlike SQL, fully compositional, highly orthogonal:
  – Inputs & outputs collections (sequences or bags) of XML nodes
  – Anywhere a particular type of object may be used, may use the results of a query of the same type
  – Influenced by ideas from functional programming

• Tension: attempts to satisfy the needs of data management and document management
  – The database-style core is mostly complete (even has support for NULLs in XML!!)
  – The document keyword querying features are still in the works – shows in the order-preserving default model
XQuery’s Basic Form

• Has an analogous form to SQL’s
  
  `select .. from .. where .. group by .. order by` blocks

• Semantics: bind nodes (or node sets) to variables; operate over each legal combination of bindings; produce a set of nodes

• “FLWOR” statement:
  
  `for {iterators that bind variables}`
  `let {collections}`
  `where {conditions}`
  `order by {order-conditions}`
  `return {output constructor}`
Iteration ("for-loops") in XQuery

A series of (possibly nested) for statements binding the results of XPaths to variables

```xml
for $root in document("http://my.org/my.xml"),
  $sub in $root/rootElement,
  $sub2 in $sub/subElement, ...
```

• Essentially, a pattern to be matched, producing a "binding tuple"

• For each binding, evaluate the where clause and possibly output results constructed using return template

• `document()` or `doc()` function specifies an input file as a URI
  
  – Old version was “document”; now “doc” but it depends on your XQuery implementation
Two XQuery Examples

```xml
<root-tag> {
    for $p in document("dblp.xml")/dblp/proceedings,
        $yr in $p/yr
    where $yr = "1999"
    return <proc> {$p} </proc>
} </root-tag>

for $i in document("dblp.xml")/dblp/inproceedings[
    author/text() = "John Smith"
return <smith-paper>
    <title>{ $i/title/text() }</title>
    <key>{ $i/@key }</key>
    { $i/crossref }
</smith-paper>
```
Nesting in XQuery

• Nesting XML trees is a very common operation
• In XQuery, it’s easy – put a subquery in the return clause where you want things to repeat!

    for $u in document("dblp.xml")/universities
    where $u/country = "USA"
    return <ms-theses-99>
      { $u/title } 
      { 
        for $mt in $u/../mastersthesis
        where $mt/year/text() = "1999" and ____________
        return $mt/title 
      }
    </ms-theses-99>
Collections and Aggregation in XQuery

- In XQuery, many operations return *collections*
  - XPaths, sub-XQueries, functions over these, ...
  - The `let` clause assigns the collection to a variable

- Aggregation simply applies a function from collections to values (very elegant compared to SQL!)

```xquery
let $allpapers := document("dblp.xml")/dblp/article in
return <article-authors>
  <count>{
    fn:count(fn:distinct-values($allpapers/authors))
  }</count>{
    for $paper in doc("dblp.xml")/dblp/article
    let $pauth := $paper/author
    return <paper> {$paper/title}
    <count>{
      fn:count($pauth)
    }</count>
  }
</article-authors>
```
Collections and Aggregation (2)

Unlike in SQL, we can compose aggregations and create new collections from old:

```xml
<result> {
  let $avgItemsSold := fn:avg(
    for $order in document("my.xml")/orders/order
    let $totalSold = fn:sum($order/item/quantity)
    return $totalSold)
  return $avgItemsSold
} </result>
```
Sorting in XQuery

• SQL allows you to sort query output, with a special `order by` clause

• XQuery borrows this idea

• In XQuery, what we order is the sequence of “result tuples” output by the `return` clause:

```xml
for $x in document("dblp.xml")/proceedings
order by $x/title/text()
return $x
```
What If Order Doesn’t Matter?

• By default:
  – Relations in SQL are unordered
  – Collections in XQuery are ordered

• But, unordered queries are much faster to answer!

• XQuery allows the user to say "don't worry about preserving order here":

  unordered {
    for $x$ in (mypath)...
  }

Removing Duplicates

• In XQuery, duplicate elimination is performed by a function over a collection (returning another collection)

• But since we have nodes with nested structure in XML, can do duplicate removal according to (1) value or (2) node identifier
  – Intuition from C++: (1) "strcmp(s1,s2) == 0" versus (2) "a == b"
  – fn:distinct-values(collection): (1) remove by value
  – fn:distinct-nodes(collection): (2) remove by node identifier
Another Difference wrt SQL: Metadata is Data in XQuery

Can get a node’s name by using `node-name()`:
```xml
for $x in document("dblp.xml")/dblp/*
return node-name($x)
```

Can construct elements and attributes using `computed names`:
```xml
for $x in document("dblp.xml")/dblp/*,
$year in $x/year,
$title in $x/title/text(),
element node-name($x) {
    attribute {"year-" + $year} {$title}
}
```
XQuery Summary

Very flexible and powerful language for XML

– Clean and orthogonal: can always replace a collection with an expression that creates collections

– DB and document-oriented (we hope)

– The core is relatively clean and easy to understand

Turing Complete – we’ll talk more about XQuery functions soon