XQuery Advanced Topics

Alin Deutsch
Roadmap

- Use of XQuery for Web Data Integration
- XQuery Evaluation Models
- Optimization
- Flavor of Standardization Issues
  - Equality in XQuery
- More on Optimization
The Web as Database Queried in XQuery

XML Publishing
(IBM DB2, Oracle 9i, MS Access)

The internet

The internet

Q, X, X1, ..., Xn are XQueries
A Simple Publishing Scenario

- **Usage**
  - John: 2/day aspirin
  - Jane: 3/day cortisone

- **Patient Name**
  - John: migraine
  - Jane: allergy

**Published Data**

**Virtual Data**

**User Query** (XQuery)

**Reformulation** (SQL)

**How to express the view?**

**How to “compose” the user query with the view, obtaining the reformulation?**
Encoding relational data as XML

Want to specify view from proprietary → published data as XML → XML view expressed in XQuery

<table>
<thead>
<tr>
<th>usage</th>
<th>drug</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>2/day</td>
<td>aspirin</td>
<td>John</td>
</tr>
<tr>
<td>3/day</td>
<td>cortisone</td>
<td>Jane</td>
</tr>
</tbody>
</table>

```xml
<prescription>
  <tuple>
    <usage>2/day</usage>
    <drug>aspirin</drug>
    <name>John</name>
  </tuple>
  <tuple>
    <usage>3/day</usage>
    <drug>cortisone</drug>
    <name>Jane</name>
  </tuple>
</prescription>
```

```xml
<patient>
  <tuple>
    <name>John</name>
    <diag>migraine</diag>
  </tuple>
  <tuple>
    <name>Jane</name>
    <diag>allergy</diag>
  </tuple>
</patient>
```
Proprietary → Published View: XML → XML

```
<study>
  <case><diag>migraine</diag><drug>aspirin</drug><usage>2/day</usage></case>
  <case><diag>allergy</diag><drug>cortisone</drug><usage>3/day</usage></case>
</study>
```

```
<prescription>
  <tuple><usage>2/day</usage><drug>aspirin</drug><name>John</name></tuple>
  <tuple><usage>3/day</usage><drug>cortisone</drug><name>Jane</name></tuple>
</prescription>
```
The View

<study>
  for $t1 in document("encoding.xml")//patient/tuple,
      $n1 in $t1/name/text(),
      $di in $t1/diagnosis/text(),
      $t2 in document("encoding.xml")//prescription/tuple,
      $n2 in $t2/name/text(),
      $dr in $t2/drug/text(),
      $u in $t2/usage/text(),
    where $n1=$n2
  return
    <case>
      <diag>$di</diag>
      <drug>$dr</drug>
      <usage>$u</usage>
    <case>
</study>
A Client Query

Find high-maintenance illnesses (require drug usage thrice a day):

```xml
<results>
  for $c in document("public.xml")//case,
    $d in $c/diag/text(),
    $u in $c/usage/text(),
  where $u="3/day"
  return <drug>$d</drug>
</results>
```

Not directly executable, public.xml does not exist
The Reformulated Query

Directly executable, expressed in SQL against the proprietary database:

Select   pr.drug

From     patient pa, prescription pr

Where    pa.name = pr.name and
          pr.usage = “3/day”
Roadmap

• Use of XQuery for Web Data Integration
• XQuery Evaluation Models
• Optimization
• Flavor of Standardization Issues
  - Equality in XQuery
• More on Optimization
XQuery Semantics: Navigation & Tagging

XML data model is a tagged tree

XQueries compute in two stages:

- **Navigation in XML tree:** binds variables to nodes, text, tags, etc.
- **Tagging:** Output of a new XML element, for every tuple of variable bindings
let $d = $d = \text{document(“drugs.xml”)}$
<result>
  for $x$ in $d$/drug, $n$ in $x$/name/text(),
    $p$ in $x$/price/text()
  where $p$ = “$4”
  return <found>$n</found>
</result>
XQuery Semantics: Tagging

```
let $d = document("drugs.xml")
$result>
  for $x in $d//drug, $n in $x//name/text(),
      $p in $x//price/text() 
  where $p = "$4"
  return
    <found>$n</found>
</result>
```

<table>
<thead>
<tr>
<th>d1</th>
<th>“aspirin”</th>
<th>“$4”</th>
</tr>
</thead>
<tbody>
<tr>
<td>d2</td>
<td>“tylenol”</td>
<td>“$4”</td>
</tr>
</tbody>
</table>
Descendant Navigation

Direct implementation of descendant navigation is wasteful:

\[
\text{for } x \text{ in } d/\text{drug}
\]

Go to all descendants of the root (all elements), keep <drug>-tagged ones

To find the 3 <drug> elements, a direct implementation visits all elements in the document (e.g. <notes>). The full query does so repeatedly. In general, a query with n descendant steps may visit $|\text{doc size}|^n$ elements!
Roadmap

• Use of XQuery for Web Data Integration
• XQuery Evaluation Models
  – Index-based
  – Stream-based
• Optimization
• Flavor of Standardization Issues
  – Equality in XQuery
• More on Optimization
Index-based Evaluation

Idea 1: keep an index (associative array, hash table) associating tags with lists of node ids. Allows random access into XML tree.

idx:  | tag     | node ids     | lookup operation: idx[price] = [p1,p2,p3]
drug  | d1,d2,d3 | name        | n1,n2,n3
price | p1,p2,p3 | price        | p1,p2,p3
Index-based Evaluation (2)

idx:  tag        node ids
      drug      d1,d2,d3
      name      n1,n2,n3
      price     p1,p2,p3

lookup operation:  idx[price] = [p1,p2,p3]

foreach $p in idx[price]                                             // p1, p2, p3
  if $p/text() = "$4"                                                // p1, p2
    foreach $x in idx[drug]                                           // d1, d2, d3
      if $p descendant_of  $x                                          // p1 of d1, p2 of d2
        foreach $n in idx[name]                                       // n1, n2, n3
          if $n descendant_of $x                                        // n1 of d1, n2 of d2
            return <found>$n</found>

Only 9 elements visited, regardless of size of irrelevant XML
subtrees.

But doesn’t the implementation of descendant_of require more visiting?
Ancestor-Descendant Testing in $O(1)$

Idea 2: identify each node $n$ by a pair of integers $\text{pre}(n), \text{post}(n)$, with

$\text{pre}(n) = \text{the rank of } n \text{ in the preorder traversal of the tree}$

$\text{post}(n) = \text{the rank of } n \text{ in the postorder traversal}$

Then

$d$ is descendant of $a$ if

$\iff$

$\text{pre}(d) \geq \text{pre}(a) \text{ and } \text{post}(d) \leq \text{post}(a)$
Example post-preorder node ids

Additional advantage: node identity independent of particular in-memory representation of DOM objects.
Roadmap

- Use of XQuery for Web Data Integration
- XQuery Evaluation Models
  - Index-based
  - Stream-based
- Optimization
- Flavor of Standardization Issues
  - Equality in XQuery
- More on Optimization
Stream-based XQuery Execution

- So far, we assumed construction of DOM tree in memory.

- XML documents can be XML representations of databases. The DOM approach does not scale to typical database sizes.

- We want an execution model that minimizes the memory footprint of the XQuery engine.
Applications of Stream-based Execution

- Besides scaling to database sizes. There are applications where the data is inherently received in streamed form:
  - Sensor networks (attend faculty candidate Sam Madden’s talk)
  - Network monitoring/XML packet routing
  - XML document publish/subscribe systems
Stream-based XML Parsing

- A parser generates a stream of predefined events (according to the standard SAX API)
- Applications consume these events.
- Each event triggers a handler. The application is coded by providing the code for the handlers.

XML input to parser

```
<a>
<b>
<c>
  someText
</c>
</b>
<d>
  moreText
</d>
</a>
```

stream of events output by parser

```
open("a")
open("b")
open("c")
text("someText")
close("c")
close("b")
open("d")
text("moreText")
close("d")
close("a")
```

- A free SAX parser: http://xml.apache.org/xerces-j/
Stream-Based XQuery Navigation

Idea: turn path expressions into Finite Automata over alphabet containing the set of element tags

E.g. for $x$ in //b//c, $y$ in $x$/d

compiles to

Only one automaton active at any moment. Automaton of $y$ is active only as long as that of $x$ is in final state
Matching XPaths Against Streams

for $x$ in //b//c, $y$ in $x$/d

$y$: 

$x$: 

$y$:
Matching XPaths Against Streams

for $x$ in //b//c, $y$ in $x$/d

$x$: 

$y$: 

Matching XPaths Against Streams

for $x$ in //b//c, $y$ in $x$/d

$x$: 

$y$: 

o(a),
o(b),
o(c), o(d), c(d), o(d), c(d), c(c),
o(c), o(d), c(d), c(c),
c(b),
c(a)
Matching XPaths Against Streams

for $x$ in //b//c, $y$ in $x$/d
Matching XPaths Against Streams

for $x$ in //b//c, $y$ in $x/d$

$x$:  

$y$:  

$\text{o}(a), \text{o}(b), \text{o}(c), \text{o}(d), \text{c}(d), \text{o}(d), \text{c}(d), \text{c}(c), \text{o}(c), \text{o}(d), \text{c}(d), \text{c}(c), \text{c}(b), \text{c}(a)$
Matching XPaths Against Streams

for $x$ in //b//c, $y$ in $x$/d

Need to reset automaton for $y$
Matching XPaths Against Streams

for $x$ in //b//c, $y$ in $x$/d
Matching XPaths Against Streams

for $x$ in //b//c, $y$ in $x$/d

$o(a)$, $o(b)$,
  $o(c)$, $o(d)$, $c(d)$, $o(d)$, $c(d)$, $c(c)$,
  $o(c)$, $o(d)$, $c(d)$, $c(c)$,
  $c(b)$, $c(a)$
Matching XPaths Against Streams

for $x$ in //b//c, $y$ in $x/d$

Need to reset automaton for $x$ to state prior to reading black c element
Matching XPaths Against Streams

for $x$ in //b//c, $y$ in $x$/d

\[
\begin{align*}
\text{o(a),} \\
\text{o(b),} \\
\text{o(c), o(d), c(d), o(d), c(d), c(c),} \\
\text{o(c), o(d), c(d), c(c),} \\
\text{c(b),} \\
\text{c(a)}
\end{align*}
\]
Automaton Extended with Stack

Let $d$ be the transition function of automaton $A$. The corresponding extension of $A$ with a stack is defined as follows:

<table>
<thead>
<tr>
<th>current state</th>
<th>current event in stream</th>
<th>stack action</th>
<th>next state</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>open(tag)</td>
<td>push(Q)</td>
<td>$d(Q)$</td>
</tr>
<tr>
<td>Q</td>
<td>close(tag)</td>
<td>$Q' = \text{pop}()$</td>
<td>$Q'$</td>
</tr>
</tbody>
</table>

Convince yourselves that the run of this automaton on the stream in the example corresponds to the intended sequence of states.

An additional use of PDAs, aside from parsing.
Roadmap

• Use of XQuery for Web Data Integration
• XQuery Evaluation Models
• Optimization
• Flavor of Standardization Issues
  - Equality in XQuery
• More on Optimization
Semantic Optimization

- Sometimes, we can translate away descendant computation.

- Consider the following DTD describing the structure of drug.xml

  \[
  \text{<!ELEMENT pharmacy (drug*)>}
  \]
  \[
  \text{<!ELEMENT drug (name,price,notes?)>}
  \]

- Then for all documents satisfying DTD:

  \[
  \text{for } x \text{ in } d//drug, n \text{ in } x//name/text() \text{ is equivalent to}
  \]
  \[
  \text{for } x \text{ in } d/drug, n \text{ in } x/name/text()}
  \]
Semantic Optimization As Typechecking

For all XML documents conforming to the DTD

```
<!ELEMENT pharmacy (drug*)>
<!ELEMENT drug (name, price, notes?)>
```

we can determine statically that

```
for $x \text{ in } d//drug, $m \text{ in } d/maker
```

returns the empty answer.
Roadmap

• Use of XQuery for Web Data Integration
• XQuery Evaluation Models
• Optimization
• Flavor of Standardization Issues
  - Equality in XQuery
• More on Optimization
Element Equality in XQuery

• Two kinds of equality:
  – "==" id-based (an element node is equal only to itself)
  – "=" value-based

• Value-based equality underwent several drafts,

• Initially (about one year into standardization process):
  text-centric point of view. XML elements are value-equal iff their text values are equal after stripping away the XML annotations.

  E.g. <a><b>f</b><c>oo</c></a> = <m>foo</m>

• Currently:
  XML elements are equal iff their corresponding trees are isomorphic
Id-based Element Equality

Let $x$ be bound to an XML tree. Then

$$<a>$x</a>$$

does not indicate the same tree (fresh node ids) as $x$ does when it is used in this context.

creates a new XML tree (fresh node ids) and it is short for

$$<a>recursive copy of $x</a>$$

Always true:

$$(<a>$x</a>)/a/* = $x$$

(value-based equality)

Always false:

$$(<a>$x</a>)/a/* == $x$$

(id-based equality)
Roadmap

• Use of XQuery for Web Data Integration
• XQuery Evaluation Models
• Optimization
• Flavor of Standardization Issues
  - Equality in XQuery
• More on Optimization
More on XQuery Optimization

• There are many ways to write the same query (i.e., there are many distinct XQuery expressions with identical semantics).

• Some of these expressions lead to cheaper execution than their counterparts.

• Goal of query optimization:
  given a query Q, find the optimal query Q’ with identical semantics (we say that Q and Q’ are equivalent).

• Basic test in query optimization: checking query equivalence.

• The more expressive a language, the harder it is to test equivalence.

• Various classes of XQueries have distinct complexity:

  PTIME (1), NP-complete (1), $\Pi^p_2$-complete (4), PSPACE-complete (1), EXPTIME-complete, undecidable.
The UCSD Database Lab

- Main Focus: XML Query Optimization
- Check out the weekly DB Research Meeting
- Faculty
  - Victor Vianu
  - Yannis Papakonstantinou
  - Alin Deutsch

- San Diego SuperComputer Researchers
  - Ilkay Altintas
  - Amarnath Gupta

www.db.ucsd.edu