The Benefits of Utilizing Closeness in XML

Curtis Dyreson – Utah State University
Shuohao Zhang – Microsoft, Marvell
Hao Jin – Expedia.com
Outline

• Motivation

• Tools
  ■ Polymorphic restructuring
  ■ Closest axis
  ■ Compaction

• Current and Future Tools
1970s Database Controversy

- Hierarchical model vs. relational model

- “A Relational Model of Data for Large Shared Data Banks,” in CACM – 1970
  - Turing award – 1981
  - SIGMOD Innovations Award renamed in his honor – 1994

E. F. Codd
Paper Coverage

- Topics: relations, query language (relational algebra), embedded query languages, keys, foreign keys, referential integrity, views, data redundancy, normal forms and normalization, indexes, query optimization, versioning, recovery

- Covers 75% of topics

- 11 pages vs. 1009 pages
1970’s Database Controversy

- Codd’s two main criticisms of hierarchical model
  1. Logical/physical data independence
  2. *symmetric exploitation* of data

- *part/project* works on some, but not all

- Path expressions are *asymmetric*
Query Language Goal

- *Same* query works on *different* structures
  - *Symmetric* exploitation

- Simplifies querying
  - Lack of schema knowledge
  - Heterogeneous data
  - Irregular data
  - Schema evolution
XML (Hierarchical) Data

- Parsed to a tree data model

```xml
...<author>
  <name>E. F. Codd</name>
  <book>
    <title>The Relational Model for Database Management</title>
    <publisher>Addison Wesley</publisher>
    <price>$46.95</price>
  </book>
  <book>
    <title>Cellular Automata</title>
    <publisher>Academic Press</publisher>
    <price>$46.95</price>
  </book>
</author>
...```
Same Data, Different Structure

...<author>
  <name>E. F. Codd</name>
<book>
  <title>The Relational Model for Database Management</title>
  <publisher>Addison Wesley</publisher>
  <price>$46.95</price>
</book>
<br>cellular Automata</title>
  <publisher>Academic Press</publisher>
  <price>$46.95</price>
</book>
</author>
...
Same Data, Different Trees
Asymmetric Path Expressions

• **Task**
  ■ Find books by E. F. Codd

• **XQuery**
  ■ \( \text{return } \text{doc("author.xml")//author[name= 'E. F. Codd']/book} \)
Same Data, Different Structure

- **Same task**
  - Find books by E. F. Codd
- **Need different XQuery**
  - \( \text{return} \ \text{doc}("book.xml")//\text{book}[\text{author}/\text{name}='E. F. Codd'] \)
Problem: How do we know it is the “same” data?

- Same text values
  - Approximate value matching problem is separate issue
  - Ignore white space between nodes

- Same types
  - Assume type exists
    - Label, e.g., “author”
    - Labels along path from root to node, e.g., “bib.book.author”
    - Includes descendents
  - Approximate type matching problem is separate issue

- Structure can vary
  - Ignore order
  - Ignore duplicates
Determining Sameness

- Data transformation

  ![Diagram of data transformation]

- Important not to lose data! Must be *reversible*.

  ![Diagram of data transformation]
Translative Semantics
Common to Both Structures

• The *Closest* Relation
  \[ \text{Closest}(u, v) \text{ if no two other nodes with the same types as } u \text{ and } v \text{ are closer than } u \text{ and } v \text{ are.} \]

• Nodes closest to the first *title* node
When the First Book Lacks a Price

- Node selection restricted by \textit{minimal type distance}
  - The minimal distance between a title and a price is 2
Using Closeness

• Group nodes that *represent the same entity*
  ■ Assume node *identifiers*
  
  - book ← title;
  - author ← name;
  - name ← ; name

• Connect nodes that are *related*
  ■ *Relatedness* <--> *Closeness*
Same Data?
Group Identical Subtrees
After Grouping
Associate Closest Nodes
After Association
Graphs are Isomorphic
Outline

• Motivation

• Tools
  ■ Polymorphic restructuring
  ■ Closest axis
  ■ Compaction

• Current and Future
Restructuring with XQuery

A simple-minded query

```
for $b in //book
return
  <publisher> {$b/publisher/text()} </publisher>
  <book>
    {$b/title}
    <author> {$b/../name} </author>
    {$b/price}
  </book>
</publisher>
```

A *more correct* query

```
let $x := //book
for $p in distinct-values($x/publisher)
return
  <publisher> {$p} </publisher>
  { for $b in //book
      where $b/publisher = $p
      return <book>
        {$b/title}
        <author> {$b/../name} </author>
        {$b/price}
      </book>
  }
</publisher>
```
Problems with using XQuery

- Restructuring is a complicated query, prone to errors
- Not *polymorphic*
Polymorphic Restructuring

poly-transform

output specification
Output Specification: Signature

publisher#book#((title,author#name,price)

publisher
  Addison Wesley
  book
    title
      DB
    author
      Codd
    price
      46.95

publisher
  Academic Press
  book
    title
      Automata
    author
      Codd
    price
      9.99
Polymorphic Restructuring

- Takes a signature and input data (in different structure)
- Produces output data correctly

Restructuring Process

publisher#book#(title,author#name,price)
The Poly-Transform Algorithm

• Input is *signature* (data guide) of the desired output

• Builds the new forest top-down

• Adds *Closest* children to each node
Example

Target signature:
publisher#book#(title,author#name,price)
Example

Target signature:

```
publisher #book#(title,author#name,price)
```

Diagram:
- **author**
  - **name**
    - Codd
  - **book**
    - **title**
      - DB
    - **publisher**
      - Addison Wesley
    - **price**
      - 46.95
  - **book**
    - **title**
      - Automata
    - **publisher**
      - Academic Press
    - **price**
      - 9.99
Example

Target signature:

\texttt{publisher#book#(title,author#name,price)}
Example

Target signature:

publisher#book#(title,author#name,price)
Example

Target signature:
```
publisher#book#(title, author#name, price)
```
Example

Target signature:
publisher#book#(title, author#name, price)
Example

Target signature:

publisher#book#(title, author#name, price)
Example

Target signature:
`publisher#book#(title,author#name,price)`
Reversibility

Restructure the target using the signature of the source

\[ X = \text{Trans}(D, S_X) \quad D' = \text{Trans}(X, S_D) \]

- Three outcomes of \( \text{Trans}(\text{Trans}(D, S_X), S_D) \equiv D' \)
  1. \( D' \) has at least the same data as \( D \) (inclusive)
  2. \( D' \) has no more data than \( D \) (non-additive)
  3. \( D' \) has the same data as \( D \) (reversible)

- Theorems on reversibility
  - The poly-transform is always inclusive
  - Can analyze signature to decide additive (hence reversibility)
Experiment – Java Implementation

- Does Output Shape Matter? No
  - *Deep* signature: last#publisher#title#first#author
  - *Flat* signature: publisher#(last,title,first,author)
  - *Mixed* signature: publisher#title#author#(last,first)
- Duplicate elimination doubles cost
Cost of Initialization?

- Initialization approx 40%
  - Parse XML
  - Construct data structures (lists of elements)

- One initialization, many poly-transforms
XQuery vs. Poly-Transform XQuery

- More efficient than XQuery
- Especially when large dataset, heavy duplicate elimination
Poly-Transform Enabled XQuery

• User-defined/external function

for $root in //bibliography
return polyTRANSFORM ( $root,
    "bibliography#publisher#book(title,author,editor)"
)
Benefits of Polymorphic Restructuring

- Simple
- Polymorphic – only need to specify structure of output
- (Potentially) efficient
- Good *(reversible or inclusive)*
- Java implementation
  - www.cs.usu.edu/~cdyreson/pub/polytransform
Outline

• Motivation
• Tools
  ■ Polymorphic restructuring
  ■ Closest axis
  ■ Compaction
• Current and Future
Asymmetric Path Expressions

• Task
  ■ Find books by E. F. Codd

• XQuery
  ■ \( \text{return } \text{doc("author.xml")//author[name='E. F. Codd']/book} \)
Same Data, Different Structure

- **Same** task
  - Find books by E. F. Codd
- **Need different** XQuery
  - `return doc("book.xml")//book[author/name='E. F. Codd']`
Existing Axes are Directional
The *Closest Axis* is Non-directional

- **Syntax**
  - `closest::`
  - `->name` is abbreviation for `closest::name`

- **Semantics**
  - A function that takes a context node and returns a sequence of `closest` nodes
Querying with Directional Axes

- **Query#1** -- return doc("author.xml")//author[name= 'E. F. Codd']/book
- **Query#2** -- …
- **Query#3** -- return doc("book.xml")//book[author/name='E. F. Codd']
Querying with the Closest Axes

Same query --
\[ \text{return } \text{doc("any.xml")} \rightarrow \text{author[}\rightarrow \text{name='E. F. Codd']\rightarrow \text{book} \]

Closest axis-enabled XQuery evaluation engine

Result#1

Result#2

Result#3
In-memory Implementation

- Naïve approach
  - Compute *Closest* for every node
  - Time complexity is \(O(sn^2)\)
    - \(s\): number of labels in the signature
    - \(n\): number of nodes

- Converting to a path expression

\[
\text{Find the closest price for a title}
\]

Non-directional expression

\[\text{closest::price}\]

Directional (path) expression

\[\text{parent::*/child::price}\]
Experiment

- Compare directional vs. nondirectional
  for $b$ in doc("bib.xml")//title/closest::publisher
  return $b

  for $b$ in doc("bib.xml")//title/..//publisher
  return $b

- Implemented closest in eXist (an XML DBMS)
Persistent Implementation

- Take advantage of type indexes
- LCA-join
  - Every *Closest* pair related via an LCA
  - Idea is to merge lists of types

- $O(sn)$
Benefits of Closest Axis

- Current XQuery depends on path expressions

- A path expression is directional (asymmetric)
  - May break down if structure changes

- The closest axis is non-directional (symmetric)
  - Simple in syntax
    - Can be easily integrated in XQuery
  - Can be implemented efficiently
    - In-memory
    - Persistent
Outline

• Motivation
• Tools
  ■ Polymorphic restructuring
  ■ Closest axis
  ■ Compaction
• Current and Future
Other Applications: Compact XML

Original XML tree with 17 nodes:

```
<book pub="Doubleday">
  <title>Da Vinci</title>
  <author>Brown</author>
</book>

<book pub="Pocket">
  <title>Angels</title>
  <author>Brown</author>
</book>
```

Compact XML tree with 14 nodes:

```
<author>Brown</author>

<title>Da Vinci</title>
<title>Angels</title>
```

The compact XML tree removes redundant information and merges nodes to reduce the number of nodes from 17 to 14.
Other Applications

- Compression – make smaller
- Compaction – make smaller, output is XML
Choosing a Signature for Restructuring

- Many different signatures possible, $S_1 \ldots S_n$
- Different signatures produce different-sized documents
- Choosing \textit{smallest} is NP-hard
Use Heuristic

- **How many parents to each child?**
  - 1-1 – either can be parent
  - 1-many – make 1 side the parent
  - Many-many – make smallest side the parent

- **Example:**
  - author-name is 1-1, keep as is
  - publisher-book is 1-many so move it above book
  - book-author is many-many (1000-2000), move book up
Experiment

- DBLP article data, 309KB, 7312 elements
  
  `<article>…<year>2005</year>…<journal>TODS</journal>…</article>`

- Compacted, 252 KB, 5441 elements
  
  `<year>2005`<journal>TODS…`<article>…</article>…`<journal>…`<year>`

- Compaction reduced file size by 18%, # of elements by 25%
- Amount of compaction achieved depends on the data
Outline

• Motivation
• Tools
  ■ Polymorphic restructuring
  ■ Closest axis
  ■ Compaction
• Current and Future
Data Matching

- Compute intersection of closest axes
  - If “same” data in a different schema, you’ll get an exact match

- Approximate matching based on “best” intersection of closest axes
Data Matching

Is there a match for this book?
Data Matching

Compute closest axis
Data Matching

Match is largest intersection of (schema, value) pairs

(name, Codd), (title, DB), (publisher, AW), (price, 46.95), (book, ), (author, )
XML Search

- Keyword search chooses SLCA
- Search “A*” and “*9*”

```
<author>
  <name>Edmund Codd</name>
  <book>
    <title>DB</title>
    <publisher>Addison Wesley</publisher>
    <price>46.95</price>
  </book>
  <book>
    <title>Automata</title>
    <publisher>Academic Press</publisher>
    <price>9.99</price>
  </book>
</author>
```
XML Search

- Keyword search chooses SLCA
- Search “A*” and “*9*”
XML Search

- Keyword search chooses SLCA
- Search “A*” and “*9*”
XML Search

- Keyword search chooses SLCA
- Search “A*” and “*9*”
XML Search

- Keyword search chooses SLCA
- Search "A*" and "*9*"
Using Closeness

• Instead of SLCA, compute intersection of closest

• Distance within $k$ threshold
Using Closeness

- Difficult to compare results in heterogeneous data collection
- Restructure using search terms

<table>
<thead>
<tr>
<th>name</th>
<th>book</th>
<th>title</th>
<th>publisher</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Codd</td>
<td></td>
<td></td>
<td>Addison Wesley</td>
<td>46.95</td>
</tr>
</tbody>
</table>
PathFree

- **Idea: Specify only output**
  
  List titles by E. F. Codd
  
  ```
  transform
  author
  → name where text = 'E. F. Codd'
  → title;
  ```

  List book titles by E.F. Codd
  
  ```
  transform
  author
  → name where text = 'E. F. Codd'
  { → book } → title;
  ```

- **Implementation based on merging type lists**
Metadata Restructuring

Source data

Target data
XML Schema Constraints

- XML Schema constraints structure, syntax
- Validate using closeness, non-directional constraint
- A person has one SSN
  - Person closest to exactly 1 SSN
    ```xml
    <person><ssn>
    <person ssn="…
    <ssn><person ...
    <person><ids><ssn> ...
    ```
Polymorphic Directories

- File system is hierarchy
  - ls research->author.xml

- Modified in the last five minutes.
  - ls {file modification (now – 5)}->author.xml
Aspect-Oriented Programming

- Add cross-cutting concern to a program
- Does not modify program directly

Program.java

point cuts

aspects

advice

Persist.java

Log.java

aspect weaver

modified program

Persist.java

Log.java

Program.java
AO Data

- Data has many cross-cutting concerns
  - Privacy, security, time, quality, lineage, provenance

```
SELECT ... FROM ... WHERE ...
```

```
data.rel
```

```
data.cuts
```

```
data.aspects
```

```
data.aspect.weaver
```

```
temporal.rel
```

```
data.rel
```

```
data.aspect.weaver
```

```
privacy.rel
```

```
query.sql
```

```
SELECT ... FROM ... WHERE ...
```

```
aspect-oriented relations and queries
```
## Example

<table>
<thead>
<tr>
<th>Name</th>
<th>Dept.</th>
<th>Sal.</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe</td>
<td>Shoes</td>
<td>40K</td>
<td>1</td>
</tr>
<tr>
<td>Joe</td>
<td>Admin</td>
<td>100K</td>
<td>2</td>
</tr>
<tr>
<td>Sue</td>
<td>Shoes</td>
<td>50K</td>
<td>3</td>
</tr>
<tr>
<td>Fred</td>
<td>Admin</td>
<td>90K</td>
<td>4</td>
</tr>
</tbody>
</table>
Adding Data Cuts

**Privacy Advice**
- admin
- coder
- test suite 20

**Test Advice**

**Temporal Advice**
- 2003 - 2003
- 2005 - 2006
- 2006 - now
- 2007 - now

- (Joe, Shoes, 40K)
- (Joe, Admin, 100K)
- (Sue, Shoes, 50K)
- (Fred, Admin, 90K)
Represent in RM

<table>
<thead>
<tr>
<th>Employees</th>
<th>Data Cuts</th>
<th>Temporal Advice</th>
<th>Test Advice</th>
<th>Privacy Advice</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Dept.</td>
<td>ID</td>
<td>RF</td>
<td>RF</td>
</tr>
<tr>
<td>Joe</td>
<td>Shoes</td>
<td>1</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>Joe</td>
<td>Admin</td>
<td>2</td>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td>Sue</td>
<td>Shoes</td>
<td>3</td>
<td>C</td>
<td>B</td>
</tr>
<tr>
<td>Fred</td>
<td>Admin</td>
<td>4</td>
<td>D</td>
<td>D</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Departments</th>
<th>Data Cuts</th>
<th>Temporal Advice</th>
<th>Test Advice</th>
<th>Privacy Advice</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loc.</td>
<td>Dept.</td>
<td>ID</td>
<td>RF</td>
<td>RF</td>
</tr>
<tr>
<td>E104</td>
<td>Shoes</td>
<td>5</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>A2</td>
<td>Admin</td>
<td>6</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>F77</td>
<td>Shoes</td>
<td>7</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>D</td>
<td>D</td>
</tr>
</tbody>
</table>
[Joins and Cartesian Product] Let $\otimes$ be $\bowtie$, $\bowtie_{\theta}$, $\bowtie$, or $\times$. When tuples are composed, their advice must be as well. For every combination of data, manufacture a “combined” perspective.

$[RD, RC, RA] \otimes [SD, SC, SA] =$

\[
\text{synch}(\text{regulate}([RD \otimes SD, (RC \times SC) \bowtie DA, DA]))
\]

and

$DA = \text{ao-intersect}(RA, SA)$

The \text{regulate} operation ensures that each data tuple has advice.

[Regulate] This operation removes from the data those tuples that do not have any advice.

\text{regulate}([RD, RC, RA]) = [RD \bowtie RC, RC, RA]

The joins use aspect-oriented intersection to figure out if the advice intersects.
To Do

• Defined AORA

• Implement

• Experiment
Thank You!