SilkRoute: Trading between Relations and XML

By: Mary Fernandez, Wang-chiew Tan, Dan Suciu
Presented by: Wenguo Liu
11/11/2003

Outline
- Introduction
- SilkRoute’s Architecture
- Query Composition
- Discussion

Introduction: Motivation
- Need a tool for converting relational data into XML
  - XML is the standard format for data exchange on the Internet
  - Most data is stored in RDBMS or ORDBMS
- Properties of such tool
  - General – No public DTD matches exactly a proprietary relational schema
  - Dynamic – Only materialize when needed
  - Get data from execution, not from definition
  - Efficient – Utilize the mature RDBMS query processors

Introduction: Motivating Example
- Schema of supplier’s relational database
  - Clothing(pid, item, category, description, price, cost)
  - SalePrice(pid, price)
  - Problems(pid, code, comments)

SilkRoute’s Architecture
- Middleware between RDBMS and an application accessing the data over the Web
SilkRoute’s Architecture – Five Parts

- View Query (RXL)
  - RXL query written by database administrator
  - Defines the XML virtual view of the DB

- Application
  - Formulates an user query in XML-QL over the virtual view and sends to SilkRoute

- SilkRoute Query Composer
  - Input: RXL view query and the XML-QL user query
  - Output: A new RXL query (Executable Query)

- SilkRoute Translator
  - Input
    - Executable Query (E) from query composer
    - Description of the relational schema
  - Partition E
    - Data-extraction part (SQL queries)
    - XML-construction part (XML template)
  - Output
    - One tuple stream per SQL query

SilkRoute XML Generator
- Merge tuple streams from Translator

RXL features: Skolem functions
- from Clothing $c
  - construct <category ID=Cat($c.category)
    - name=$c.category
  - <product ID=Prod($c.item)
    - name=$c.item
    - <retail ID=Retail($c.price)

RXL features: Nested Queries
- construct <view
  - from Clothing $c
    - construct <product
      - name=$c.item
      - <category ID=Cat($c.category)
      - <retail ID=Retail($c.price)

RXL View Query (V : RDB -> XML)
1. construct
2. <results>
3. where <supplier>
4. <company>$company</company>
5. <product>
6. <name>$name</name>
7. <retail>$retail</retail>
8. <sale>$sale</sale>
9. </product>
11. $sale < 0.5 * $retail

RXL features: Block Structure
- construct <view ID=View()
  - { from Clothing $c
    - construct <product
      - ID=Prod($c.item)
      - name=$c.item
      - <category ID=Cat($c.category)
      - <description ID=Desc($c.description)
      - <retail ID=Retail($c.price)
  - { from Clearance $d
    - where $d.disc > 50
    - construct <product ID=Prod($d.prodname)
      - name=$d.prodname
      - <discount ID=Discount($d.disc)

Architecture: XML-QL User Query (U : XML -> XML)
1. construct
2. <results>
3. where <supplier ID=Supp()
4. <company ID=Comp()>
5. <product ID=Prod($c.item)
6. <name ID=Name($c.item)
7. <retail ID=Retail($c.price)
8. <sale ID=Sale($c.price)
9. <supplier ID=Supp()
10. <company ID=Comp()>
11. <product ID=Prod($c.item)
12. <name ID=Name($c.item)
13. <retail ID=Retail($c.price)

RXL features: Skolem functions
- from Clothing $c
  - construct <category ID=Cat($c.category)
    - name=$c.category
  - <product ID=Prod($c.item)

If Without Skolem Term
- <category ID=Cat($c.category)
  - <product ID=Prod($c.item)

Architecture: XML-QL User Query (U : XML -> XML)
1. construct
2. <results>
3. where <supplier ID=Supp()
4. <company ID=Comp()>
5. <product ID=Prod($c.item)
6. <name ID=Name($c.item)
7. <retail ID=Retail($c.price)
8. <sale ID=Sale($c.price)
9. <supplier ID=Supp()
10. <company ID=Comp()>
11. <product ID=Prod($c.item)
12. <name ID=Name($c.item)
13. <retail ID=Retail($c.price)

Architecture: XML-QL User Query (U : XML -> XML)
1. construct
2. <results>
3. where <supplier ID=Supp()
4. <company ID=Comp()>
5. <product ID=Prod($c.item)
6. <name ID=Name($c.item)
7. <retail ID=Retail($c.price)
8. <sale ID=Sale($c.price)
9. <supplier ID=Supp()
10. <company ID=Comp()>
11. <product ID=Prod($c.item)
12. <name ID=Name($c.item)
13. <retail ID=Retail($c.price)

Pattern Filter Template
Architecture: The Query Composer (C=U(V(DB)))

Recall: V (RDB -> XML)

1. construct
2. <results>
3. { from Clothing $c, SalePrice $s
where $c.category = "outerwear",
$c.pid = $s.pid,
$s.price < 0.5 * $c.retail
construct
$resultID=Result("Acme Clothing")>
 <supplier>"Acme Clothing"</supplier>
 <name ID=Name($c.pid, $c.item)>$c.item</name>
</result>
} </results>

Architecture: Translator and XML Generator

Translator: C => SQL Query

select c.pid as pid, c.item as item
from Clothing c, SalePrices s
where c.category = "outerwear",
c.pid = s.pid,
s.price < 0.5 * c.retail
sort by c.pid

Translator: C => XML template

<results>
<result ID=Result("Acme Clothing")>
<supplier>"Acme Clothing"</supplier>
</result>
</results>

Recall: U (XML -> XML)

1. construct
2. <elm> { where P, W
3. construct
4. T }
5. </elm>
6. <supplier ID=$supp1>
7. <company ID=Comp()>Acme Clothing</company>
8. Comp():-true
9. <product ID=Prod($pid)>Prod($pid) :-Clothing($cpid, $category, $name, $retail, $sale),
10. $category = "outerwear"
11. Filter (W)
12. $sale < 0.5 * $retail
13. V (RDL)
14. <company ID=Comp()>"Acme Clothing"</company>
15. <product ID=Prod($pid)>
16. <name ID=Name($cpid, $item)>$item</name>
17. </product>

Step1: Pattern Matching

Construction of the view tree

- For each Skolem function F in V
  - Rule F(x, y, ...) :- body

Examples

- <supplier ID=Supp()>Supp() :-true
- <company ID=Comp()>Acme Clothing</company>
  Comp():-true
- <product ID=Prod($pid)>
  Prod($pid) :-Clothing($cpid, $category, $name, $retail, $sale),
  $category = "outerwear"

Pattern Composition: Steps

- Pattern Matching
  - Represents V by a view tree
  - View tree
    - Global template
    - A set of datalog rules
  - Evaluate U on the view tree
- Query Rewriting

Step1: Pattern Matching

Evaluate U on the view tree

U=construct <elm> { where P, W
construct T }
</elm>

Add new variables to P

- <supplier ID=$supp1>
- <company ID=Comp()>"Acme Clothing"</company>
  <product ID=$pid1>
  <name ID=$name1/>$name1</name>
  <retail ID=$ret1>$ret1</retail>
  <sale ID=$sale1>$sale1</sale>
</product>
</supplier>
Step 1: Pattern Matching (Table R)
- Evaluate U on the view tree

<table>
<thead>
<tr>
<th>$t1$</th>
<th>$t2$</th>
<th>$t3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supp()</td>
<td>Comp()</td>
<td>Acme Clothing</td>
</tr>
</tbody>
</table>

Variables in U's pattern P

One matching of U's pattern P with V's template T

Step 2: Query Rewriting
- Use the table R to construct the composed query C
- $C = \text{construct} \{ \text{<elm>} \{ B_1 \} \ldots \{ B_k \} \text{</elm>} \}
- For each block $\{ B \}$
  - Variable substitutions
  - Construct rule $Q(S_0(x), S_0(y), \ldots) :- \ldots$
  - Minimize $Q$
  - Convert $Q$ to \textit{from} and \textit{where} clauses
  - Replace column variables by tuple variables

Discussion
- Recursive XML schema
  - Linear recursion of SQL sufficient?
- Composition technique
  - Evaluates the patterns on the view definition at \textit{compile-time}
  - Filters and constructors are evaluated at \textit{run-time}
- General techniques for RXL to efficient SQL
- Minimization of composed RXL views
- RXL or XML-QL Ü Comprehension

Recall: U (XML -> XML)

```xml
1. construct
2. <results> {
3.  where <supplier>
4.    <company>$company</company>
5.  </supplier>
6.  <product>
7.    <name>$name</name>
8.    <retail>$retail</retail>
9.    <sale>$sale</sale>
10.  </product>
12.  $sale < 0.5 * $retail
13. } }
```

Pattern (P)