University of Washington
Department of Computer Science & Engineering

Managing XML and Semistructured Data

Lecture 18: Publishing XML Data From Relations

Prof. Dan Suciu

Spring 2001

In this lecture

- Virtual XML Publishing
- Materialized XML Publishing

Resources

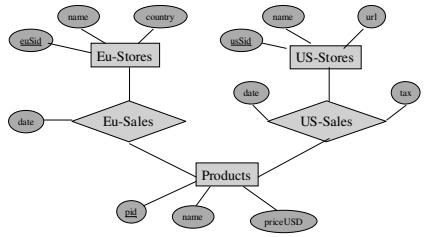
- Efficiently Publishing Relational Data as XML Ducments by Shanmugasundaram, Shekita, Barr, Carey, Lindsay, Pirahesh, Reinwald in VLDB'2000

XML Publishing

- XML view defined declaratively
  - SQL extensions [Exodus]
  - RXL [SilkRoute]
- ***Virtual*** XML publishing
  - Accept XML queries (e.g. XML-QL), translate to SQL
  - Main issue: compose queries
- ***Materialized*** XML publishing
  - Compute entire XML view – large !
  - Main issue: compute a large query efficiently

Virtual XML Publishing

Legacy data in E/R:



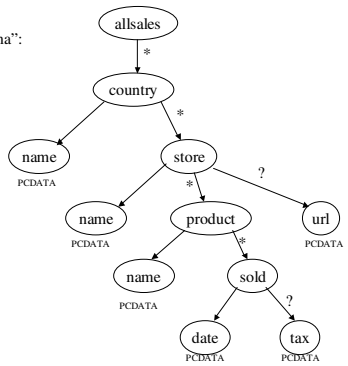
Virtual XML Publishing

- XML view

```
<allsales>
  <country>  <name> France </name>
             <store> <name>  Nicolas </name>
                     <product> <name> Blanc de Blanc </name>
                               <sold> 10/10/2000 </sold>
                               <sold> 12/10/2000 </sold>
                               …
                     </product>
                     <product>…</product>…
             </store>….
  </country> …
</allsales>
```

- In summary: group by country → store → product

Output "schema":



Virtual XML Publishing

In SilkRoute

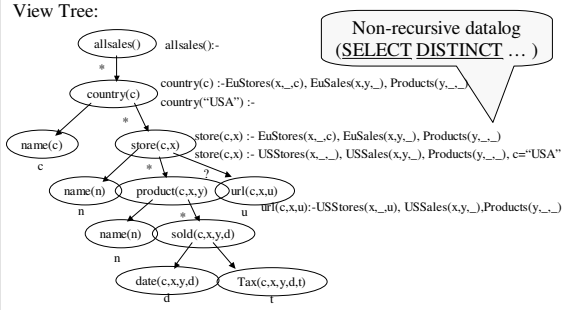
```
{ FROM    EuStores $S, EuSales $L, Products $P
  WHERE  $S.euSid = $L.euSid  AND $L.pid = $P.pid
  CONSTRUCT
    <allsales()>
      <country($S.country)>
        <name> $S.country </name>
        <store($S.euSid)>
              <name> $S.name </name>
              <product($P.pid)>
                    <name> $P.name </name>
                    <price>  $P.priceUSD </price>
              </product>
        </store>
      </country>
    <allsales>
}                                  /* union..... */
```

Virtual XML Publishing

```
  .... /* union */
{ FROM    USStores $S, EuSales $L, Products $P
  WHERE  $S.usSid = $L.euSid  AND $L.pid = $P.pid
  CONSTRUCT
    <allsales()>
      <country("USA")>
        <name> USA </name>
        <store($S.euSid)>
              <name> $S.name </name>
              <url> $S.url </url>
              <product($P.pid)>
                    <name> $P.name </name>
                    <price>  $P.priceUSD </price>
                    <tax> $L.tax </tax>
              </product>
        </store>
      </country>
    </allsales>
}
```

Internal Representation

View Tree:



Virtual XML Publishing

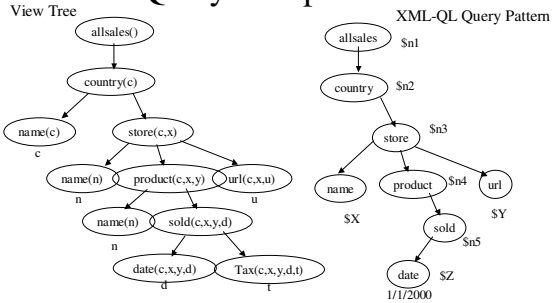
- Don't compute the XML data yet
- Users ask XML queries
- System composes with the view, sends to the RDBMS
- Main issue: compose queries

XML Publishing: Virtual View in SilkRoute

- find names, urls of all stores who sold on 1/1/2000 (in XML-QL / XQuery melange):

```
WHERE <allsales/country/store>
              <product/sold/date> 1/1/2000 </>
              <name> $X </>
              <url> $Y </>
        </>
RETURN $X , $Y
```

Query Composition



"Evaluate" the XML pattern(s) on the view tree, combine all datalog rules

Query Composition

Result (in theory…):

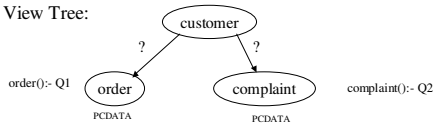
```
( SELECT  DISTINCT S.name, S.url
  FROM USStores S, USSales L, Products P
  WHERE S.usSid=L.usSid AND L.pid=P.pid AND  L.date='1/1/2000')
      UNION
( SELECT DISTINCT S2.name, S2.url
  FROM EUStores S1, EUSales L1, Products P1
          USStores S2, USSales L2, Products P2,
WHERE S1.usSid=L1.usSid AND L1.pid=P1.pid AND  L1.date='1/1/2000'
    AND S2.usSid=L2.usSid AND L2.pid=P1.pid
    AND S1.country="USA" AND S1.euSid = S2.usSid)
```

Complexity of XML Publishing

- But in practice: 5-7 times more joins !
  - Need query minimization
- Could this be avoided ?
  - No: it is NP-hard

XML Publishing Is NP-Hard

View Tree:



XML query:

```
WHERE <customer> <order> $x </>
                 <complaint> $y </>
      </>
RETURN ( )
```

The composed SQL query is : Q1 JOIN Q2
Minimizing it is NP hard ! (can be shown…)

Materialized XML Publishing

Efficiently Publishing Relational Data as XML Documents, Shanmugasundaram et al., VLDB'2001

- Considers several alternatives, both inside and outside the engine

Materialized XML Publishing

- Create the structure (i.e. nesting):
  - Early
  - Late
- Add tags:
  - Early
  - Late
- Do this:
  - Inside relational engine
  - Outside relational engine

Note: may add tags only after structuring has completed

Example

```
CONSTRUCT
 <allsales>
 FROM    EuStores $S
 CONSTRUCT
      <name> $S.name </name>
      FROM Owners $O
      WHERE $S.oID = $O.oID
      CONSTRUCT <owner> $O.name </owner>
      <store>
            FROM    EuSales $L, Products $P
            WHERE  $S.euSid = $L.euSid  AND $L.pid = $P.pid
            CONSTRUCT
               <product>
                    <name> $P.name </name>
                    <price>  $P.priceUSD </price>
               </product>
      </store>
</allsales>
```

Early Structuring, Early Tagging

The Stored Procedure Approach

```
XMLObject result = "<allsales>"
SQLCursor C1 = "Select S.sid, S.name From EuStore S"
FOR x IN C1 DO
    result = result + "<name>" + C1.name + "</name>"
    SQLCursor C2 = "Select O.name From Owners O Where O.oid=%C1.oid
    FOR y IN C2 DO result = result + "<owner>" + C2.name + "</owner>"
    SQLCursor C3 = "Select P.name, P.priceUSD From ... Where ..."
    FOR z IN C3 DO result = result + "<product> <name>" + P.name + ...
result = result + "</allsales>"
```

- Advantage: very simple
- Disadvantage: multiple SQL queries submitted

Early Structuring, Early Tagging

The correlated CLOB approach

```
SELECT XMLAGG(STORE(S.name,
                    XMLAGG(OWNER(SELECT O.oID
                                 FROM Owners O
                                 WHERE S.oID = O.oID)),
                    XMLAGG(PRODUCT(SELECT P.name, P.priceUSD
                                   FROM   EuSales L, Products P
                                   WHERE  S.euSid = L.euSid
                                          AND L.pid = P.pid)))
FROM   EuStores S
```

- Still nested loops...
- Create large CLOBs – problem for the engine

Early Structuring, Early Tagging

The de-correlated CLOB approach

```
GroupBy euSid and XMLAGG (EuStores S1
                          LEFT OUTER JOIN
                          Owners O
                          ON S1.oId = O.oId)
JOIN
GroupBy euSid and XMLAGG(EuStores S2
                         LEFT OUTER JOIN
                         ( SELECT L.euSid, P.name, P.priceUSD
                           FROM   EuSales L, Products P
                           WHERE  L.pid = P.pid)
                         ON S2.euSid = L.euSid
ON S1.euSid = S2.euSid
```

Early Structuring, Early Tagging

The de-correlated CLOB approach

- Modify the engine to do groupBy's and taggings
- Better than nested loops (why ?)
- Still large CLOBs
- Early structuring, early tagging

Late Tagging

- Idea: create a flat table first, then nest and tag
- The flat table consists of outer joins and outer unions:
  - Unsorted → late structuring
  - Sorted → early structuring

Review of Outer Joins and Outer Unions

- Left outer join
  - e.g. R(A,B) ⟕ S(B,C) = T(A,B,C)

A	B
a1	b1
a2	b2
a3	b3

⟕

B	C
b1	c1
b1	c2
b3	c3

=

A	B	C
a1	b1	c1
a1	b1	c2
a2	b2	-
a3	b3	c3

Review of Outer Joins and Outer Unions

- Outer union
  - E.g. R(A,B) outer union S(A,C) = T(A, B, C)

A	B
a1	b1
a2	b2

outer union

A	C
a3	c3
a4	c4
a5	c5

=

Tag	A	B	C
1	a1	b1	-
1	a2	b2	-
2	a3	-	c3
2	a4	-	c4
2	a5	-	c5

Late Tagging, Late Structuring

- Construct the table:

```
(EuStores LEFT OUTER JOIN  Owners)
    OUTER UNION
(EuStores LEFT OUTER JOIN EuSales JOIN Products)
```

- Tagging:
  - Use main memory hash table to group elements on store ID

Late Tagging, Early Structuring

- Same table, but now sort by store ID and tag:

```
(EuStores LEFT OUTER JOIN  Owners)
    OUTER UNION
(EuStores LEFT OUTER JOIN EuSales JOIN Products)
ORDER BY euSid, tag
```

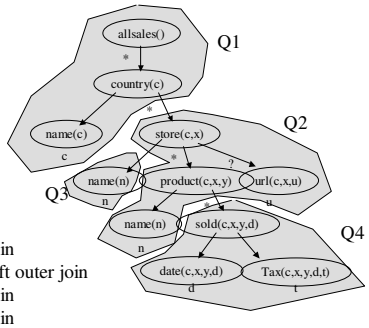
- Constant space tagger

Materialized XML Publishing

SilkRoute, SIGMOD'2001

- The outer union / outer join query is large
- Hard to optimize by some RDBMs
- Split it in smaller queries, then merge sort the tuple streams
- Idea: use the view tree; each partition defines a plan

View Tree



Q1 = ...join
Q2 = ...left outer join
Q3 = ...join
Q4 = ...join

- In general:
  - A "1" edge corresponds to a join
  - A "*" edge corresponds to a left outer join
  - There are 2^n possible plans
- Choose best plan using heuristics