

XML Publishing

CPS 296.1
Topics in Database Systems

XML publishing

- Legacy data: normalized and stored in many flat relations; managed by relational DBMS

Department		Project		Employee		
DeptId	DeptName	ProjId	ProjName	EmpId	EmpName	Salary
10	Purchasing	888	Internet	101	John	50K
		795	Recycling	91	Mary	70K

- XML data: un-normalized, nested; wanted by next-generation applications

```
<department name="Purchasing">
  <emplist><employee> John </employee>
  <employee> Mary </employee>
</emplist>
<projlist><project> Internet </project>
<project> Recycling </project>
</projlist>
</department>
```

2

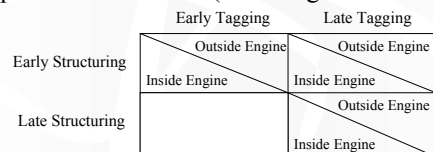
Roadmap

- IBM's XPERANTO project
 - Shanmugasundaram et al. "Efficiently Publishing Relational Data as XML Documents." *VLDB*, 2000
- AT&T's SilkRoute project
 - Fernandez et al. "Efficient Evaluation of XML Middleware Queries." *SIGMOD*, 2001

3

Issues in publishing XML

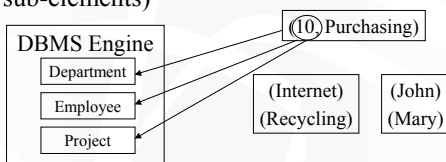
- Structuring and tagging
 - Flat relations → nested XML elements
 - Relational schema → XML tags
- Processing inside versus outside relational engine
 - Space of alternatives (Shanmugasundaram et al.)



4

Stored procedure approach

- Early tagging, early structuring; outside engine
- Application and/or stored procedure explicitly issue queries for each element (and recursively for sub-elements)



- Problem: many small queries

5

Correlated CLOB approach (slide 1)

- Early tagging, early structuring; inside engine
- Use user-defined functions for tagging
 - "XML" is a special type like CLOB

```
define XML Constructor DEPT(dname:VARCHAR(20),
  emplist:XML,
  projlist:XML) as
( <department name={dname}>
  <emplist> {emplist} </emplist>
  <projlist> {projlist} </projlist>
</department>
```

6

Correlated CLOB approach (slide 2)

- Use user-defined aggregate XMLAGG to concatenate XML fragments into one
- Use correlated subqueries for structuring

```
select DEPT(d.name,
           (select XMLAGG(EMP(e.name))
            from Employee e where e.deptno = d.deptno),
           (select XMLAGG(PROJ(p.name))
            from Project p where p.deptno = d.deptno)
          )
from Department d
```

- Problems
 - Correlated execution of sub-queries → nested loop
 - Repeated creation and copying of big CLOB's

7

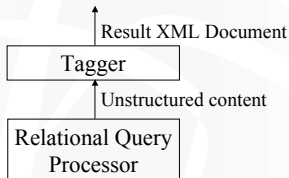
De-correlated CLOB approach

- Early tagging, early structuring; inside engine
- Relational engine may be smart enough to convert correlated subqueries into joins
- Example
 - Compute employee lists associated with all departments
 - Compute project lists associated with all departments
 - Join results on department id
- Still a problem: creation and copying of CLOB's

8

Late tagging, late structuring

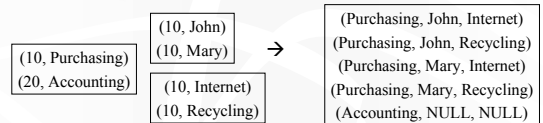
- Contents for XML produced without structure (in arbitrary order)
- Tagger enforces order and inserts tags



9

Redundant relation approach

- Late tagging, late structuring; either inside or outside engine
- Join all source relations together
 - Use outerjoin to preserve childless parents

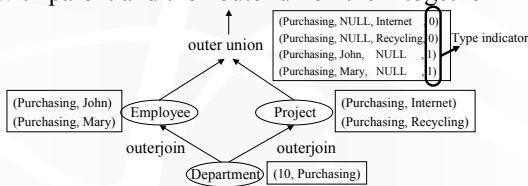


- Problem: large relational output because of redundancy

10

Unsorted outer union approach

- Late tagging, late structuring; either inside or outside engine
- Do not join siblings; outerjoin them separately with parent and then outer union them together



- Problem: wide tuples with many NULL columns

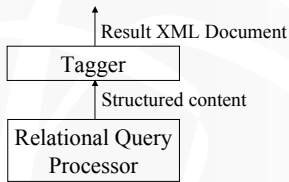
Hash-based tagger

- For late tagging, late structuring
- Use a hash table to group tuples returned by the relational query
 - Example
 - Returned tuple: (Purchasing, NULL, Internet, 0)
 - Hash on "Purchasing" to locate the parent element
 - If not found, create a parent element
 - Group the tuple under the parent
- After all tuples are hashed, go through the hash table to generate XML
- Problem: require memory to be large enough to hold the entire output document

12

Late tagging, early structuring

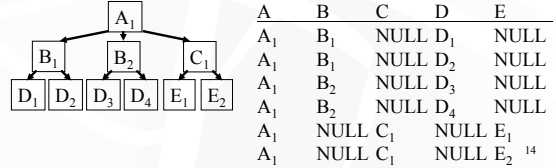
- Contents for XML produced with structure (in sorted order; already grouped)
- Tagger just inserts tags (constant space)



13

Sorted outer union approach

- Late tagging, early structuring; tagging either inside or outside engine
- Same outerjoin/outer union query as the unsorted outer union approach, with the additional ORDER BY clause to sort/group output tuples
 - Example: ORDER BY Aid, Bid, Cid, Did, Eid



Conclusion from experiments

- Inside engine is much faster than outside engine
 - Cost of binding out tuples to host variables is high
- Given sufficient main memory, unsorted outer union with hash-based tagger is the best
- With limited memory, sorted outer union with constant-space tagger is the best

15

Sorted outer union revisited

- One large SQL query, consisting of left outerjoins combined using outer unions followed by sorting
 - Example (in SilkRoute's RXL syntax)

```

from Supplier $s
construct <supplier>
  <name>$s.name</name>
  { from Nation $n
    where $s.nationkey = $n.nationkey
    construct <nation>$n.name</nation> }
  { from PartSupp $sp, Part $p
    where $sp.supkey = $sp.supkey and $sp.parkey = $p.parkey
    construct <part><name>$p.name</name></part> }
  </supplier>
  
```

SORT
((Supplier LEFT OUTERJOIN Nation)
OUTER UNION
(Supplier LEFT OUTERJOIN PartSupp, Part))

- Can relational DBMS really handle them efficiently?
- Do wide tuples with many NULL's affect performance?
- How about letting middleware do some processing? (Fernandez et al.)

16

Other alternatives

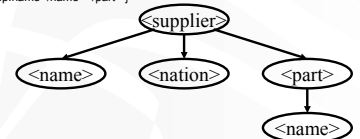
- Multiple SQL queries, each sorted for efficient merging
- Middleware merge-outerjoins sorted results; still constant space requirement
 - Example: fully partitioned
 - SORT(Supplier)
 - SORT(Supplier JOIN Nation)
 - SORT(Supplier JOIN PartSupp, Part)
 - Example: partially partitioned
 - SORT(Supplier LEFT OUTERJOIN Nation)
 - SORT(Supplier LEFT OUTERJOIN PartSupp, Part)
- Questions
 - How to enumerate alternatives?
 - How to pick the best alternative?

17

View tree

```

from Supplier $s
construct <supplier>
  <name>$s.name</name>
  { from Nation $n
    where $s.nationkey = $n.nationkey
    construct <nation>$n.name</nation> }
  { from PartSupp $sp, Part $p
    where $sp.supkey = $sp.supkey and $sp.parkey = $p.parkey
    construct <part><name>$p.name</name></part> }
  </supplier>
  
```

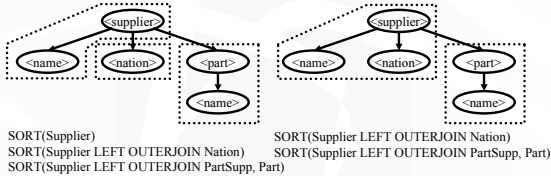


- Captures nesting of output XML elements

18

Plan enumeration using view tree

- Each alternative plan corresponds to a partitioning of the view tree into smaller trees
 - Each edge can be either broken or unbroken in a partitioning
 - Number of alternative plans = 2^E where E is the number of edges in the view tree



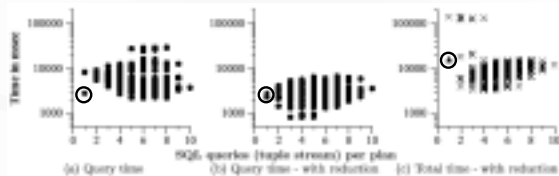
19

View tree reduction

- Edges are labeled by 1, ?, or *
 - 1: Parent has one and only one such sub-element (e.g., <supplier> – <nation>)
 - ?: Parent has zero or one such sub-element
 - *: Parent has zero or more such sub-elements
- Requires the knowledge of database constraints (e.g., Supplier.nationkey references Nation.nationkey)
- Heuristic: after partitioning, nodes connected by 1 edges can be merged together (e.g., <supplier> and <nation>)
 - Use JOIN instead of OUTERJOIN

20

Experiments with the plan space



- Alternatives do matter
- View tree reduction heuristics do help
- Sorted outer union (i.e., no partitioning) is significantly slower than the optimal plan (with partitioning)
- Need a cost-based optimizer to pick the best partitioning

Cost formula

- $cost(q) = a \times evaluation_cost(q) + b \times data_size(q)$
- $data_size(q) = |attrs(q)| \times cardinality(q)$
- Assume estimates of evaluation_cost and cardinality can be obtained from the relational DBMS optimizer
- a and b are parameters that depend on the middleware/database environment
 - May require some benchmarking/tuning to find out

22

Greedy algorithm for plan generation

- Threshold parameters t_1 and t_2
 - But who determines these parameters?
- For each edge, compute the cost of merging two partitions connected by this edge: $cost(q_{combined}) - (cost(q_1) + cost(q_2))$
 - < 0 means beneficial
 - If $< t_1$ (e.g., -60000), mark the edge as mandatory
 - If $< t_2$ (e.g., 600), mark the edge as optional
- Size of search space is reduced from $2^{(\# \text{ of edges})}$ to $2^{(\# \text{ of optional edges})}$
- Experiments show that the greedy algorithm is able to pick near-optimal plans

23

More pointers

- Fernandez et al. "SilkRoute: Trading Between Relations and XML." *WWW*, 1999
- Shanmugasundaram et al. "Querying XML Views of Relational Data." *VLDB*, 2001

24