Lecture 16

Wednesday, May 22, 2002

XML Publishing, Storage

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

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

```
<allsales>
 for    $S in db/EuStores
 return
 <store>
        <name> $S/name </name>
        for $O in db/Owners
        where $S/oID = $O/oID
        return <owner> $O/name </owner>
        for   $L in EuSales, $P in Products
        where  $S/euSid = $L/euSid  AND $L/pid = $P/pid
        return   <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,
                    (select XMLAGG(OWNER(O.oID))
                     from Owners O
                     where S.oID = O.oID),
                    (select XMLAGG(PRODUCT(P.name, P.priceUSD))
                     from    EuSales L, Products P
                     where  S.euSid = L.euSid AND L.pid = P.pid)))
from     EuStores S
```

Early Structuring, Early Tagging

The correlated CLOB approach

```
procedure OWNER(id : varchar(20))
{ return "<owner>" + id + "</owner>" }

procedure PRODUCT(name : varchar(20), price: integer)
{ return "<product> <name>" + name + "</name>" + "<price>" + price + "</price> </product>" }

XMLAGG = builtin aggregate operator; concatenates all strings in a set of strings
```

- 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
```

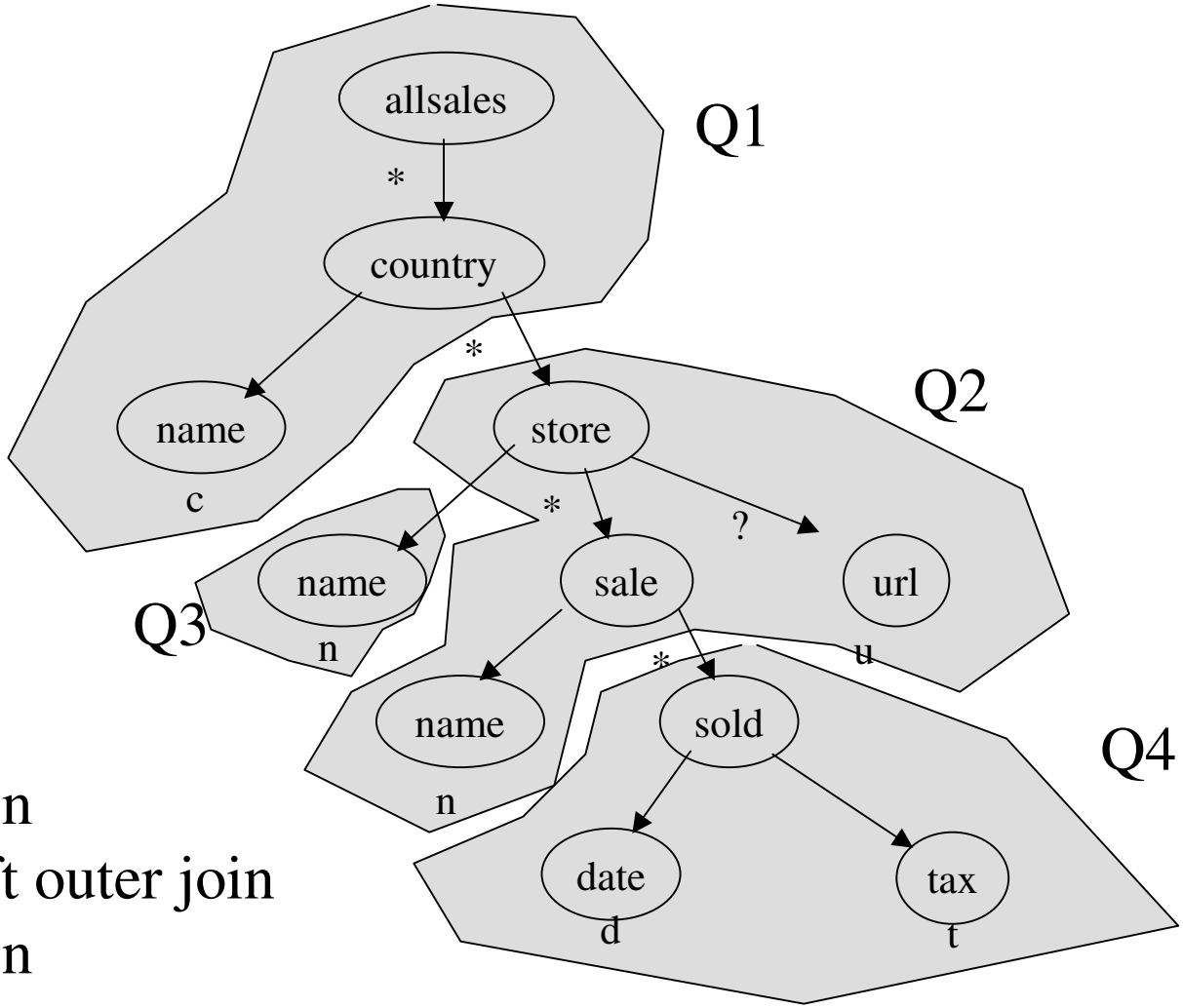
- Constant space tagger

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

XML Storage in a Relational DB

- Use generic schema
  - [Florescu, Kossman 1999]
- Use DTD to derive schema
  - [Shanmugasundaram, et al. 1999]
- Use data mining to derive schema
  - [Deutsch, Fernandez, Suciu 1999]
- Use the Path table
  - [T.Amagasa, T.Shimura, S.Uemura 2001]

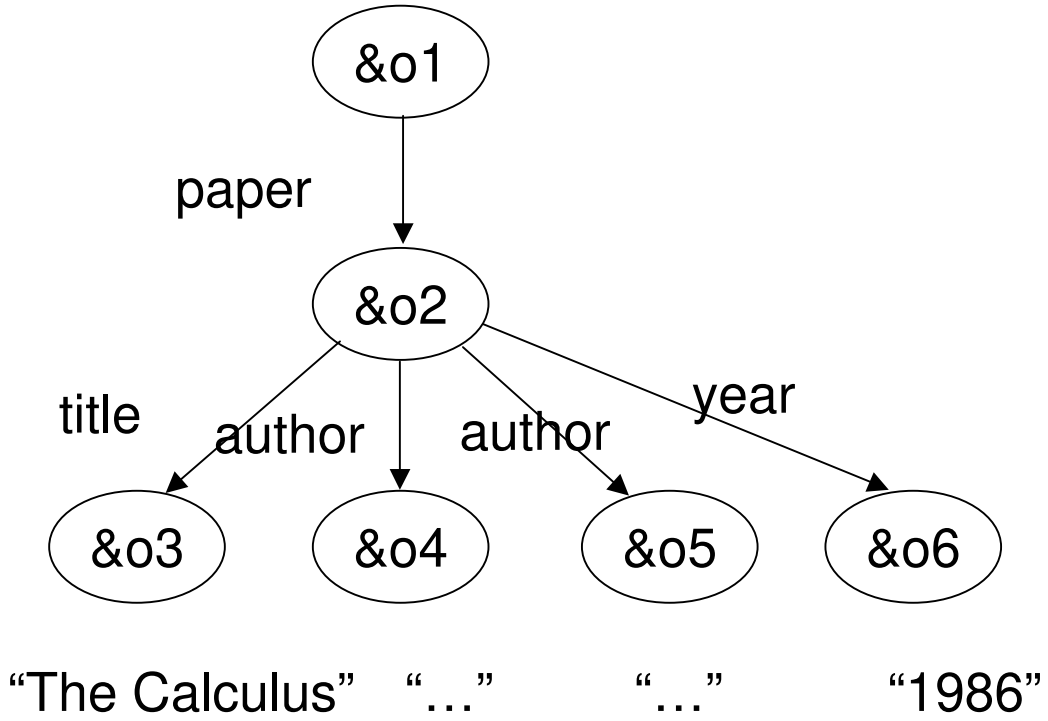
XML Stoarge: Ternary Relation

- [Florescu, Kossman 1999]
- Use generic relational schema (independent on the XML schema):

Ref(source,label,dest)

Val(node,value)

XML Stoarge: Ternary Relation



Ref

Source	Label	Dest
&o1	paper	&o2
&o2	title	&o3
&o2	author	&o4
&o2	author	&o5
&o2	year	&o6

Val

Node	Value
&o3	The Calculus
&o4	…
&o5	…
&o6	1986

[Florescu, Kossman 1999]

XML Stoarge: Ternary Relation

- Xpath to SQL translation:
- Xpath:

```
/paper[year="1986"]/author
```

- SQL:

```
Select . . . . . .

From . . . . . .

Where . . . . . .
```

XML Stoarge: Ternary Relation

- In practice may need more table:

RefTag1(source,dest)

RefTag2(source,dest)

…

IntVal(node,intVal)

RealVal(node,realVal)

…

XML Storage: DTD to Schema

[Christophides, Abiteboul, Cluet, Scholl 1994]

[Shanmugasundaram, Tufte, He, Zhang, DeWitt, Naughton 1999]

- Idea: use the XML schema to derive the relational schema

XML Storage: DTD to Schema

- DTD:

```
<!ELEMENT paper (title, author*, year?)>
<!ELEMENT author    (firstName, lastName)>
```

- Relational schema:

Paper(<u>pid</u>, title, year)
Author(<u>aid</u>, pid, firstName, lastName)

XML Storage: DTD to Schema

- Xpath to SQL translation:

```
/paper[year="1986"]/author
```

- Xpath:
- SQL:

```
Select . . . . . .

From . . . . . .

Where . . . . . .
```

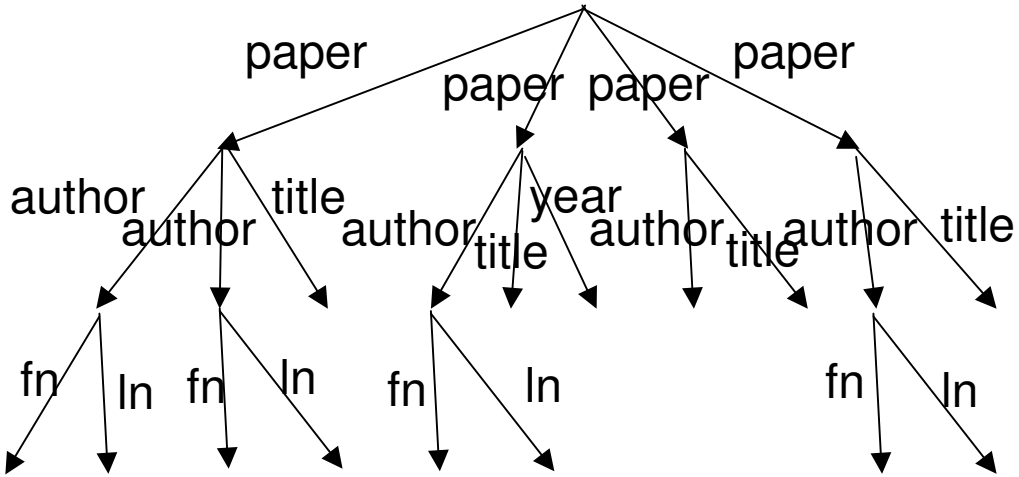
XML Storage: Data Mining to Schema

[Deutsch, Fernandez, Suciu 1999]

- Given:
  - One large XML data instance
  - No schema/DTD
  - Query workload
- Problem: find a “good” relational schema for it

- Notice: even when a DTD is present, it may be imprecise:
  - E.g. when a person may have 1-3 phones: phone*

XML Storage: Data Mining to Schema



Paper1

fn1	ln1	fn2	ln2	title	year
X	X	X	X	X	-
X	X	-	-	X	X
X	X	-	-	X	-

Paper2

author	title
X	X

[Deutsch, Fernandez, Suciu 1999]

XML Storage: Data Mining to Schema

- Xpath to SQL translation:

```
/paper[year="1986"]/author
```

- Xpath:

- SQL:

XML Storage: the Path Relation Method

- [T.Amagasa, T.Shimura, S.Uemura 2001]

- Store paths as strings
- Xpath expressions become the SQL **like** operator
- Additional information for parent/child, ancestor/descendant relationship

XML Storage: the Path Relation Method

Path

pathID	Pathexpr
1	#/bib
2	#/bib#/paper
3	#/bib#/paper#/author
4	#/bib#/paper#/title
5	#/bib#/paper#/year
6	#/bib#/book#/author
7	#/bib#/book#/title
8	#/bib#/book#/publisher

One entry for every *path* in the database
Relatively small

XML Storage: the Path Relation Method

Element

NodeID	pathID	Start	End	ParentID
1	1	0	1000	-
2	2	5	200	1
3	3	8	20	2
4	3	21	30	2
5	3	31	100	2
6	3	101	150	2
7	4	151	180	2
8	2	300	500	1
. . .				

One entry for every *element* in the database
Relatively large

XML Storage: the Path Relation Method

Val

NodeID	Val
3	Smith
4	Vance
5	Tim
6	Wallace
7	The Best Cooking Book Ever
6	3
7	4
8	2
. . .	

One entry for every *leaf* in the database
Relatively large

XML Storage: the Path Relation Method

- Xpath to SQL translation:

```
/bib/paper[year="1986"]//figure
```

- Xpath:
- SQL:

```
Select . . . . . .

From . . . . . .

Where . . . . . .
```