Lecture 15

Monday, May 20, 2002

Size Estimation, XML Processing

Estimating Sizes

- Need size in order to estimate cost
- Example:
  - Cost of partitioned hash-join E1 $\bowtie$ E2 is 3B(E1) + 3B(E2)
  - B(E1) = T(E1)/ block size
  - B(E2) = T(E2)/ block size
  - So, we need to estimate T(E1), T(E2)

Size Estimation

- Crucial for the optimizer's performance
- Idea:
  - Keep statistics for base tables
  - Use heuristics to compute size of operator's results
  - Works well for the first 1-2 levels, then it's way off
  - Hence: don't expect optimizers to return *best plan*, but avoid *worse plan*

Estimating Sizes

Estimating the size of a projection

- Easy: $T(\Pi_L(R)) = T(R)$
- This is because a projection doesn't eliminate duplicates

Estimating Sizes

Estimating the size of a selection

- $S = \sigma_{A=c}(R)$
  - T(S) san be anything from 0 to T(R) – V(R,A) + 1
  - Mean value: T(S) = T(R)/V(R,A)
- $S = \sigma_{A<c}(R)$
  - T(S) can be anything from 0 to T(R)
  - Heuristics: T(S) = T(R)/3

Estimating Sizes

Estimating the size of a natural join, $R \bowtie_A S$

- When the set of A values are disjoint, then $T(R \bowtie_A S) = 0$
- When A is a key in S and a foreign key in R, then $T(R \bowtie_A S) = T(R)$
- When A has a unique value, the same in R and S, then $T(R \bowtie_A S) = T(R)\ T(S)$

Estimating Sizes

Assumptions:

- *Containment of values*: if V(R,A) <= V(S,A), then the set of A values of R is included in the set of A values of S
  - Note: this indeed holds when A is a foreign key in R, and a key in S
- *Preservation of values*: for any other attribute B, $V(R \bowtie_A S, B) = V(R, B)$ (or V(S, B))

Estimating Sizes

Assume V(R,A) <= V(S,A)

- Then each tuple t in R joins *some* tuple(s) in S
  - How many ?
  - On average S/V(S,A)
  - t will contribute S/V(S,A) tuples in R $\bowtie_A$ S
- Hence T(R $\bowtie_A$ S) = T(R) T(S) / V(S,A)

In general: T(R $\bowtie_A$ S) = T(R) T(S) / max(V(R,A),V(S,A))

Estimating Sizes

Example:

- T(R) = 10000, T(S) = 20000
- V(R,A) = 100, V(S,A) = 200
- How large is $R \bowtie_A S$?

Answer: $T(R \bowtie_A S) = 10000\ 20000/200 = 1M$

Estimating Sizes

Joins on more than one attribute:

- $T(R \bowtie_{A,B} S) =$

  $T(R)\ T(S)/\max(V(R,A),V(S,A))\max(V(R,B),V(S,B))$

Histograms

- Statistics on data maintained by the RDBMS
- Makes size estimation much more accurate (hence, cost estimations are more accurate)

Histograms

Employee(ssn, name, salary, phone)

- Maintain a histogram on salary:

Salary:	0..20k	20k..40k	40k..60k	60k..80k	80k..100k	> 100k
Tuples	200	800	5000	12000	6500	500

- T(Employee) = 25000, but now we know the distribution

Histograms

Ranks(rankName, salary)

- Estimate the size of Employee $\bowtie_{\text{Salary}}$ Ranks

Employee	0..20k	20k..40k	40k..60k	60k..80k	80k..100k	> 100k
	200	800	5000	12000	6500	500

Ranks	0..20k	20k..40k	40k..60k	60k..80k	80k..100k	> 100k
	8	20	40	80	100	2

Histograms

- Assume:
  - V(Employee, Salary) = 200
  - V(Ranks, Salary) = 250
- Then T(Employee $\bowtie_{\text{Salary}}$ Ranks) =

  $= \Sigma_{i=1,6} T_i T_i' / 250$

  $= (200x8 + 800x20 + 5000x40 +$
  $12000x80 + 6500x100 + 500x2)/250$

  $= \ldots.$

XML Processing

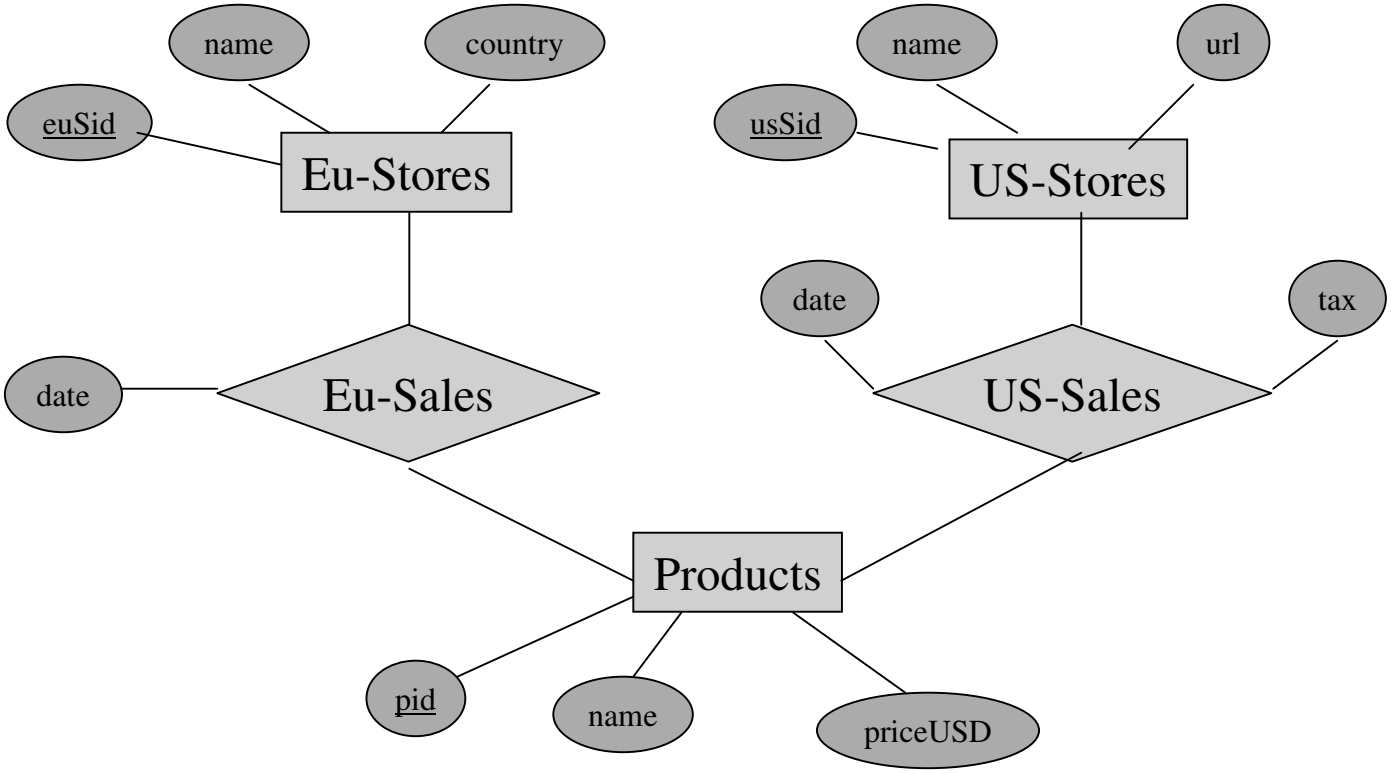
- XML publishing
- XML storage
- XML query rewriting
- XML transport and compression

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:



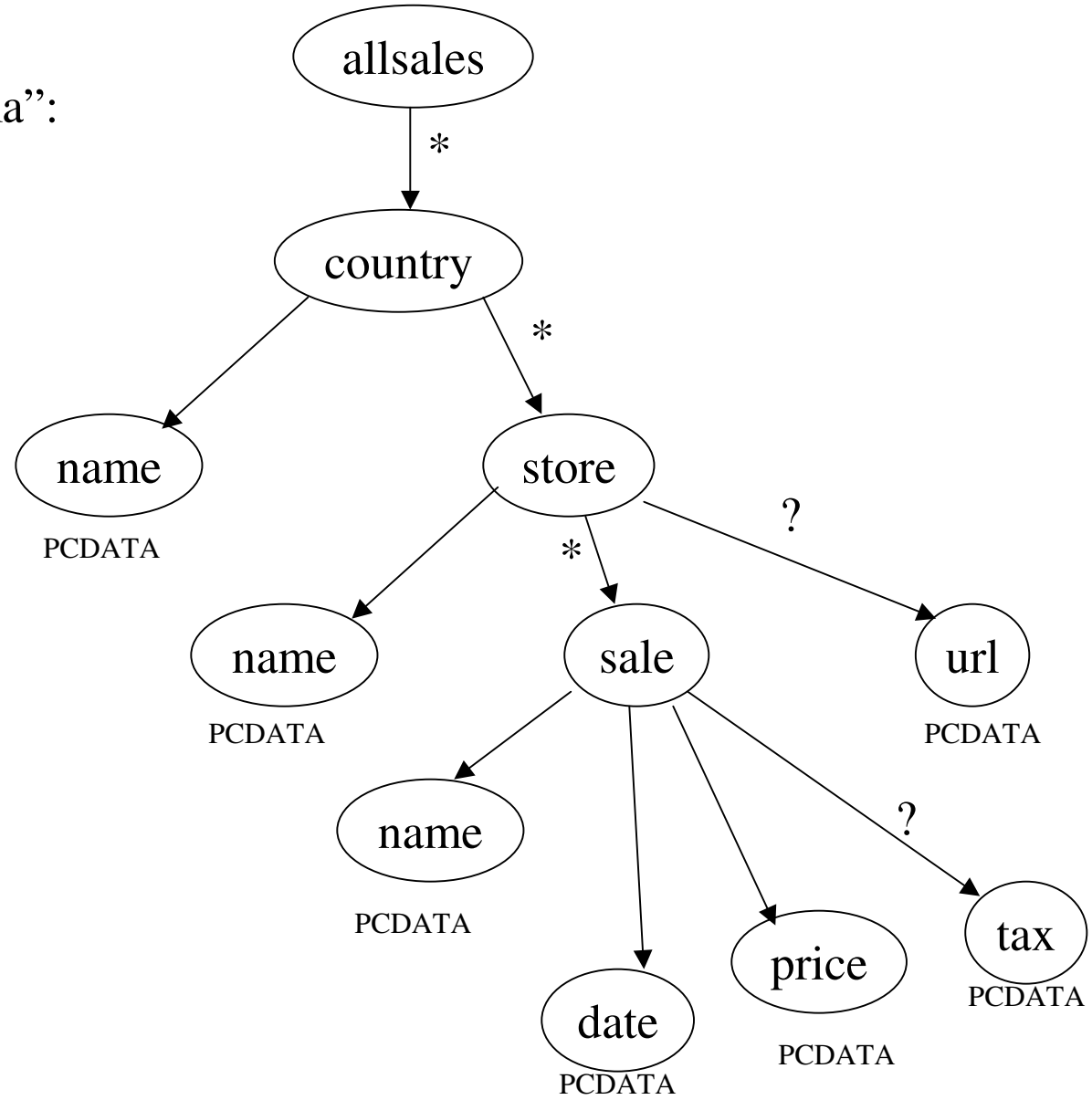
Virtual XML Publishing

- XML view

```
<allsales>
  <country>  <name> France </name>
             <store> <name>  Nicolas </name>
                     <sale> <name> Blanc de Blanc </name>
                            <date> 10/10/2000 </date>
                            <price> 39.99 </price>
                     </sale>
                     <sale>…</sale>…
            </store>….
  </country> …
</allsales>
```

- In summary: group by country → store → product-sale

Output “schema”:



Virtual XML Publishing

In SilkRoute

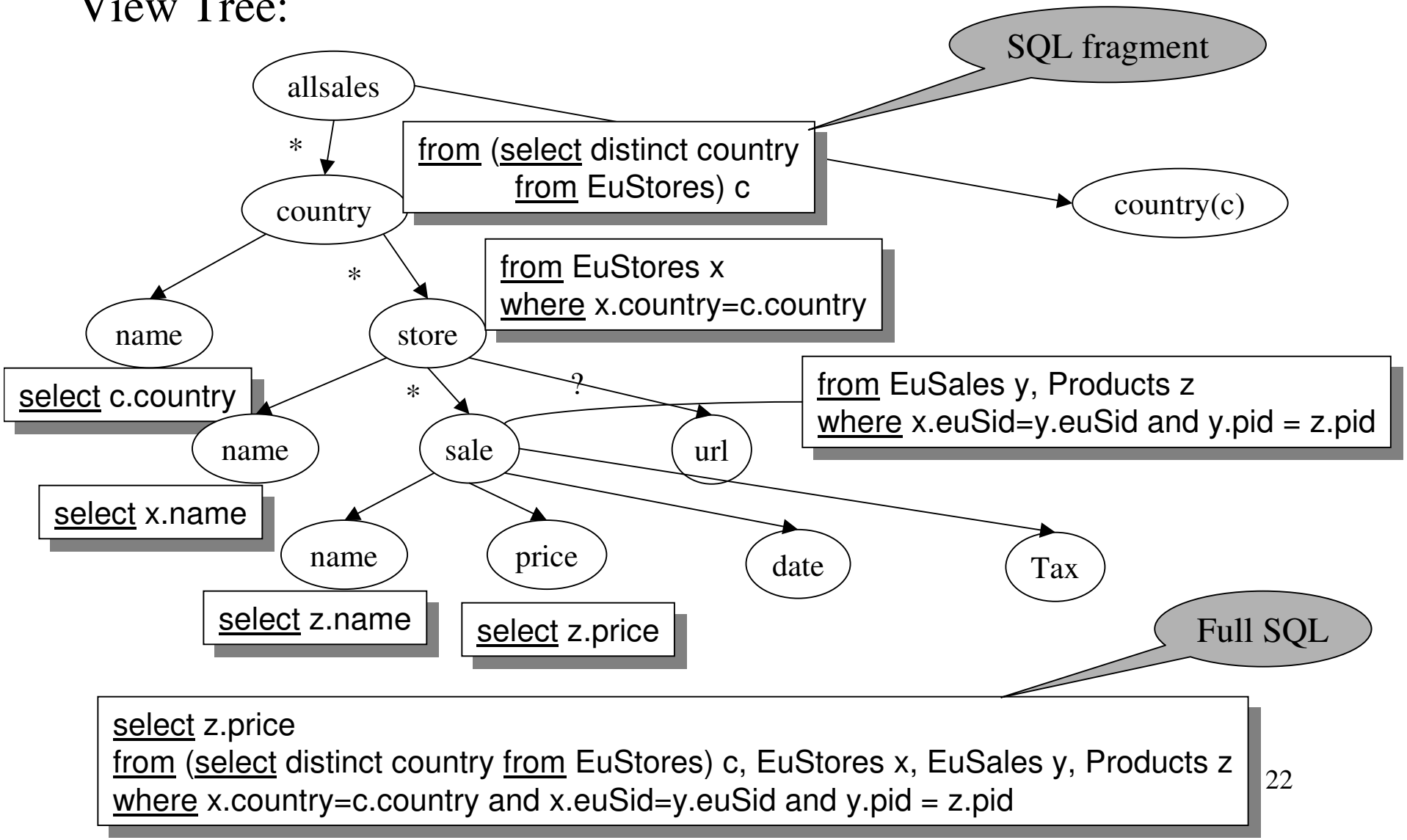
```
<allsales>
  { let $cl = distinct-values(db/EuStores/country/text())
    for $c in $cl
    return
       <country>
          <name> $c/name </name>
           { for $s in db/EuStores[country/text()=$c]
             return
                <store>
                     <name> $s/name/text() </name>
                    { for $l in db/EuSales[euSid=$s/euSid],
                          $p in db/Products[pid=$l/pid]
                      return <sale>
                                    <name> $p/name/text() </name>
                                    <date>  $l/date/text() </date>
                                    <price>  $p/priceUSD/text() </price>
                               </sale>
                </store>
        </country>
   }    union
```

Virtual XML Publishing

```
…. /* union */
   <country>
        <name> USA </name>
        for $s in db/USStores
        return
           <store>
                <name> $s/name/text() </name>
                <url> $s/url/text() </url>
                for $l in db/USSales[usSid=$s/usSid]
                    $p in db/Products[pid=$l/pid]
                return <sale>
                            <name> $p/name/text() </name>
                            <date>  $l/date/text() </date>
                            <price>  $p/priceUSD/text() </price>
                            <tax>  $l/tax/text() </tax>
                       </sale>
            </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):

```
for $s in /allsales/country/store[sale/date/text()="1/1/2000"]
return <result> <name> $s/name/text() </name>
                        <url> $s/url </url>
          </result>
```

Query Composition

Result (in theory…):

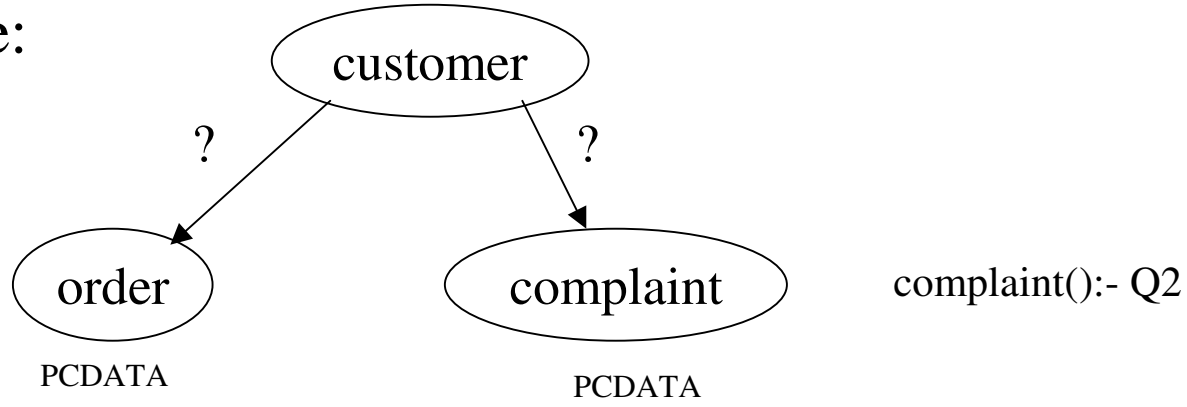
```
( SELECT S.name, S.url
  FROM USStores S, USSales L, Products P
  WHERE S.usSid=L.usSid AND L.pid=P.pid AND  L.date='1/1/2000')
       UNION
( SELECT S2.name, S2.url
   FROM EUStores S1, EUSales L1, Products P1
              USStores S2, USSales L2, Products P2,
WHERE S1.usSid=L1.usSid AND L1.pid=P1.pid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

```
<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,
                    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
```

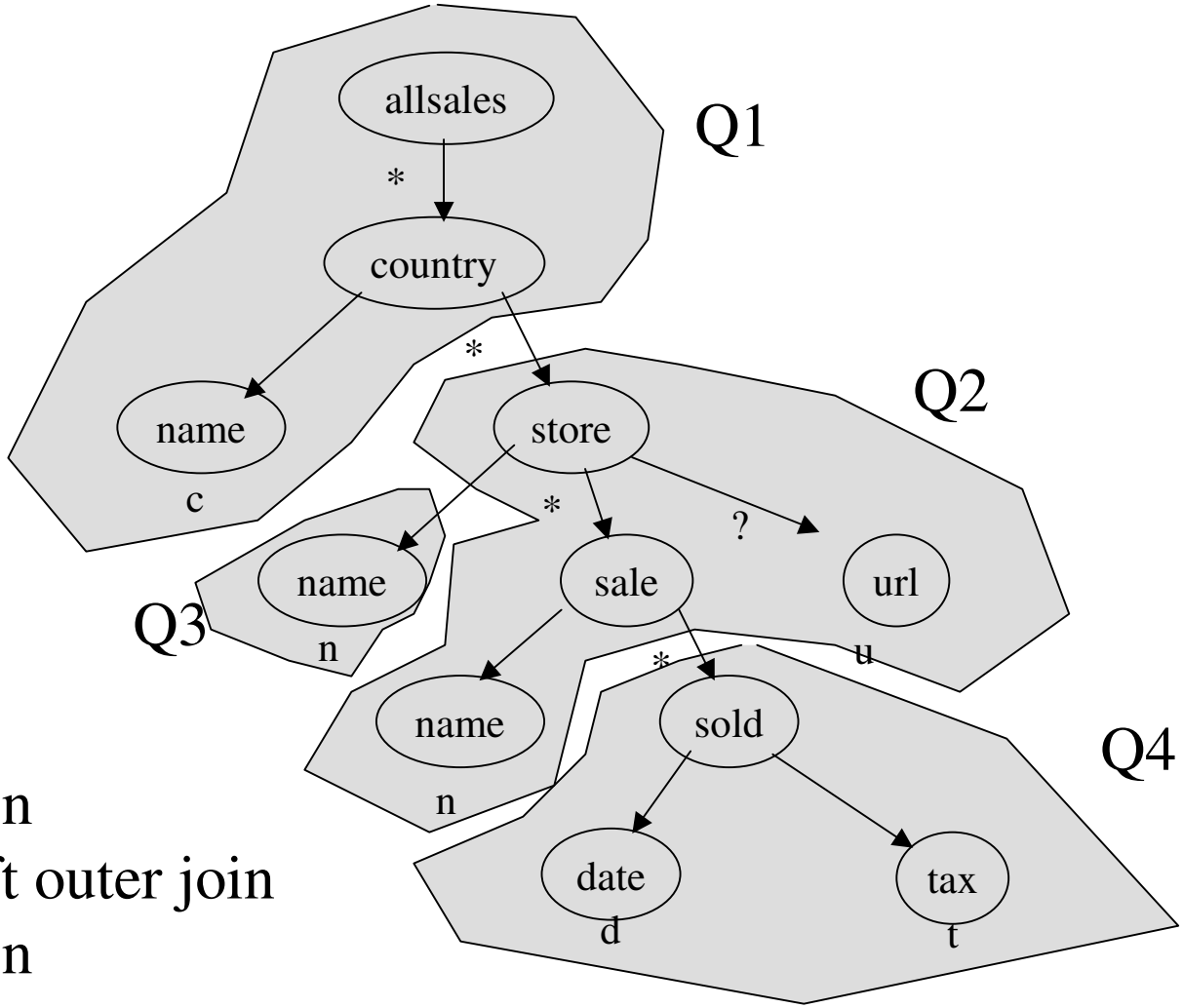
- Constant space tagger

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