A Framework for Using Materialized XPath Views in XML Query Processing

VLDB 2004

Andrey Balmin  Fatma Ozcan  Kevin S. Beyer
Roberta Cochrane  Hamid Pirahesh
Motivation

- Given materialized result of XPath query $V$
- Can it help us to answer XPath query $Q$
- Example:

$$V = //employee//@*$$

$$Q = //employee[@bonus]/employee[@bonus]/@salary$$
XPath Representation

- Limited to Location Paths
- Represented as binary trees with XPS nodes and predicate nodes
- XPS nodes labeled with *axis* and *test*
- First child is predicate
- Second child is next step
- Predicates are subtrees built out of
  - XPS nodes
  - nodes labelled with
    - \( <, \leq, \geq, >, =, \neq, eq, ne, le, gt, ge \)
  - constants
- Last node on top-level path is *extraction point*
XPath Example

//order/lineitem[@price and discount]

XPS^4(root)

null

XPS^5(/order)

null

XPS^6(/lineitem)

AND^7

null

XPS^8(@price)

XPS^9(/discount)
Problem Statement

View utilization comprises two steps

**View Matching** Find those nodes of the query tree whose evaluation can be assisted by view

**Compensation** If view does not contain exact query result, determine algebra expression to produce exact result from view contents
View Definition

- Definition:

  Path XPath expression

  Extract $\in 2^D - \{\emptyset\}$ with $D = \{\text{data}, \text{copy}, \text{path}, \text{ref}\}$

- Views contain for each extracted node any combination of

  data value of the node (typed)
  copy copy of the node’s subtree
  path path from document root to the node
  ref node reference into the full document
View Representation

- Views are represented as relations
- Relation Schema $A \in 2^D$
- Respect typed data
Desired Matching Result

\[ V = \text{//}[@\ast] \quad Q = \text{//order/lineitem} \quad [@\text{price and discount}] \]
View Matching Algorithm

- Rule-based algorithm
- Simultaneously traverse view and query tree
- Record matches during traversal
**View Matching Rules**

\[
\text{matchStep}(v, q_1 \land q_2) \rightarrow \text{matchStep}(v, q_1) \lor \text{matchStep}(v, q_2)
\]

\[
\text{matchStep}(v(\text{descendant}), q) \rightarrow \bigvee \{\text{matchChildren}(v, c)\}, \forall c \in \text{subtree of } q \ldots
\]

\[
\text{matchStep}(v(\text{axis}), q(\text{axis})) \rightarrow \text{matchChildren}(v, q)
\]

\[
\text{matchChildren}(v(\_), q) \rightarrow \text{matchPred}(v_{\text{pred}}, q) \land \text{matchNext}(v_{\text{next}}, q)
\]

\[
\text{matchChildren}(v(\text{test}), q(\text{test})) \rightarrow \text{matchPred}(v_{\text{pred}}, q) \land \text{matchNext}(v_{\text{next}}, q)
\]
Comparisons

- Comparison handling challenging, but included in algorithm
- Value-based comparisons (local filters)
- Node Set-based (intradocument joins)
**Record matches**

- \( \text{matchStep}(v,q) \) only returns true or false, we want map
- repeated calls to \( \text{matchStep}(v,q) \) with same arguments (exponential complexity)
- \( \Rightarrow \) memoize calls in match matrix \( M_{vq} \)
- Invoke \( \text{matchStep}(v\_root,q\_root) \) once
- use resulting match matrix as algorithm result
- Worst-case time consumption: \( O(|V| \cdot |Q|^2) \)
Compensation

- View may not contain exact query result
- e.g. predicates less restrictive
- e.g. view only contains intermediate results
- Construct algebra expression to produce exact query result from view
- Algebra: $\Pi, \sigma, \cap$ (and path matcher)
- Compensation has two steps:
  - **Relax** Drop query constructs answered by view
  - **Optimize** Avoid access of original data
Relax

- Select a query node that maps to extraction point of view
- Rewrite query to start from that node
  - e.g. $Q = //a[b]/c[d]///e[f]/g$
  - $e$ mapped to extraction point
  - rewrite to
    $$
    self :: e[f \land ancestor :: c[d \land /parent :: a[b]]]/g
    $$
- Remove name tests, local predicates, and steps from query that are implied by view
  - e.g. $V = //e[f]$
  - $\Rightarrow Q' = self :: *[ancestor :: c[d \land /parent :: a[b]]]/g$
Optimize

- Result from Relax step can be evaluated on ref attribute of view
- If **data**, **path** or **copy** are available in view, construct algebra expression that avoids access to original data
- If ref is not available, ’Optimize’ step may find out that view cannot be used at all
Optimize Example

- \( V = //e[f] \), **path and copy extracted**
- \( Q' = self :: *[ancestor :: c[d \land /parent :: a[b]]]/g \)
- use selection with regular expression predicate on path attribute to answer predicate
  - regexp is */a/b/c/d/*
- use copy to find /g
**Conclusion**

- **XPath View Framework**
  - Detect parts of query that can be assisted by view
  - Use as much information from view as possible to answer query
  - Polynomial algorithm

- **Missing from talk**
  - Predicate treatment
  - Using multiple views in single query plan
  - Numbers

- **Missing from paper**
  - Cost-based selection of view and matching
  - Evaluate queries with just their algebra and ref?