# View Maintenance for Hierarchical Semistructured Data

Hartmut Liefke and Susan B. Davidson
University of Pennsylvania

Presenter: Junyi Xie

## Recap

- View Maintenance in Relational Database
  - Re-computation
  - Incremental view maintenance
    - Compute and apply only the incremental changes
    - Insertion: Multi-linearity law
      $V(R_1 \cup \Delta R_1, R_2) = V(R_1, R_2) \cup V(\Delta R_1, R_2)$
    - Deletion: Counting technique
  - "Incremental Maintenance of Views with Duplicates," by Griffin and Libkin, *SIGMOD*, 1995.
- What is the difference for semi-structured data?
  - Different underlying data model (Tree for XML doc.)
  - Re-definition of union operation

## Paper Contribution

- Warehouse Architecture for XML
  - WHAX data model embedding both semi-structured and relational data source
  - Deep union operator
- WHAX-QL based on XML-QL
- Multi-linearity under constraints
- Extended counting technique for delete updates

## Outlines

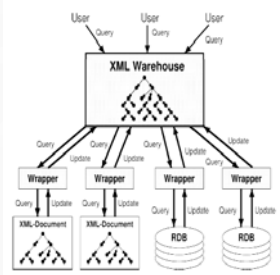
- Related Work
- WHAX Data Model
- View Definition over WHAX
- Incremental View Maintenance
- Aggregations in WHAX

## Related Work

- Abiteboul, et.al. "*Incremental Maintenance for Materialized Views over Semistructured Data",* VLDB 98'
  - Restricted version of Lorel.
  - Need additional auxiliary data structures
- Zhuge, et.al. "*Graph Structured Views and Their Incremental Maintenance*", ICDE 98'
  - Simple path expressions
- Common drawbacks:
  - Updates are always atomic: any single insertion/deletion/change of atomic values causes view maintenance process
  - No group and aggregation operations can be performed over the views

## WHAX Architecture

- Source Data
  - Relational database
  - XML repositories
- XML Warehouse
  - A materialized view on the source data
  - Typically much smaller than underlying data source

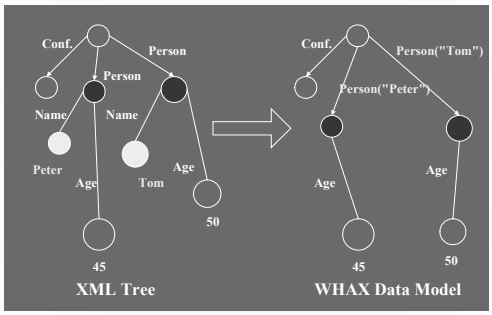



XML Data Warehouse Architecture

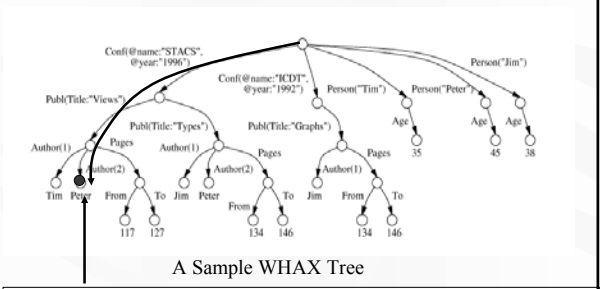
# WHAX Data Model

- Typical XML File
  - An edge labeled tree
    - Edge: labeled with tag or attribute
    - Leaf: associated value
- WHAX Data Model
  - Each node identified with *local identifier*(key)
  - Each node globally identified with a sequence of *local identifiers*
    - Each path from root to a node is unique

# WHAX Data Model




# WHAX Data Model



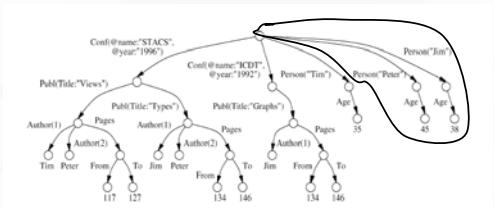

A Sample WHAX Tree

**Globally identified by: {Conf(@name:"STACS", @year: "1996"), Pub(Title: "Views"), Author(2)}**

# WHAX Data Model

- Formal Data Model
  - *L:* set of all element tags and attribute labels
    - Conf, Author, Person, Age, etc.
  - *V:* set of all values(atomic values or sub-trees)
    - Tim, Peter, 35, {Age: 45}, etc.
  - Key value $l(k) \in (L \times V)$ denotes a local identifier
    - Person("Tim"), Conf(@name: "STACS", @year: "1996")
  - Tree in WHAX: $\{l_1(k_1) : v_1, \ldots, l_n(k_n) : v_n\}$
    - If local key k is empty, then use l as $l(\{\})$
      - {Age:45}

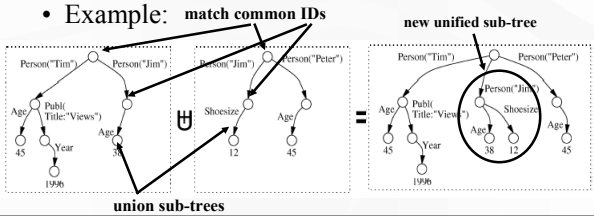
# WHAX Data Model: Example




- Sub-tree of two rightmost edges can be written as:
  {Person(Name: "Peter"): {Age:"45"}, Person(Name: "Jim"): {Age: "38"}}

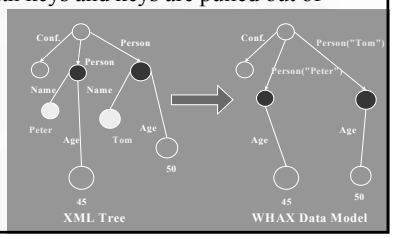
# Deep Union

- Fundamental Operation in WHAX
  - Deep Union($v_1$, $v_2$): match the common ids of $v_1$ and $v_2$, recursively union their respect sub-trees
- Example:



# Data Mapping in WHAX

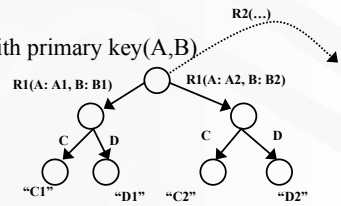
- XML As WHAX Trees
  - Given info about keys, labels in XML tree are annotated with keys and keys are pulled out of XML tree
  - Example:




# Data Mapping in WHAX

- Relational Database in WHAX
  - Natural translation: denote each tuple by an outgoing edge from the root using key *k* as local id.
  - Example: R1 with primary key(A,B)

| R1 | A | B | C | D |
|---|---|---|---|---|
| | A1 | B1 | C1 | D1 |
| | A2 | B2 | C2 | D2 |




# View Definition

- Query Language: WHAX-QL
  - Example: Select the name and the age of all authors older than 36

    V1($db) = where <Person($n).Age> $a </a> in $db, $a > 36
    Construct <MyPerson($n).Age> $a </a>
  - Difference from XML-QL: local id matched against patterns
    - XML-QL: <tag> $x </> in $db
      - <person>$n</> in $db
    - WHAX-QL: <tag<Kpat>> $x </> in $db
      - <Kpat>: Key pattern, matched against with local identifiers
      - <person($n)></> in $db, <person("Tom")></> in $db

# View Definition

- Another Example
  - For each author, return the book title and ISBN

    V2($db) = where <Book(ISBN: $a)>
    <Title>$t</>
    <Author(ID: $k)>$p</>
    </> in $db
    Construct <Author(ID:$k).Book(ISBN:$a).Title>$t </>
  - Re-grouping power
    - View tree: a tree with authors at root and titles at leaves
    - Automatically coalesced on author, no Skolem function used

# View Definition

- Still Another Example
  - For each person, return their age and all STACS publications

    V3($db) =
    where <Person(Name:$n).Age> $a </> in $db
    <Conf(@name:"STACS", @year: $y).Publ(Title: $t).Author($n)>
    </> in $db
    Construct <Author(Name:$n).Age>$a</>,
    <Author(Name:$n).STACS(Year:$y).Title($t)></>
  - Join Power of WHAX-QL
    - Between *persons* and *authors* over variable $n

# Syntax of WHAX-QL

- Where-construct clause
  - For query: Path pattern: <*PPat*>
    - *PPat* ::= $LPat_1(KPat_1),\ldots,LPat_n(KPat_n)$
  - For output: Path expression <*PExpr*>
    - *PExpr* ::= $LPat_1(e_1),\ldots, LPat_n(e_n)$
  - Label pattern <*LPat*>
    - *LPat* ::= *l* | *$x*
- Example

  V1($db) = where <Person($n).Age> $a </a> in $db, $a > 36
  Construct <MyPerson($n).Age> $a </a>

Q ::= **where** <$PPat_1$> $x_1$ </> **in** $d_1$
…
<$PPat_m$> $x_m$ </> **in** $d_m$;
$cond_1, \ldots, cond_n$
**construct**
<$PExpr_1$> $e_1$ </>, … , <$PExpr_p$> $e_p$ </>

# Syntax of WHAX-QL

- XML-QL vs WHAX-QL
  - WHAX is based on deterministic tree model
    - Each node is uniquely identified
  - WHAX-QL requires no Skolem functions
  - XML-QL needs Skolem functions to do grouping
    - Example: Return titles of all book grouped by year
      WHERE <book year = $y><title> $t</title></book> in "book.xml"
      CONSTRUCT <bookByYear id = F1($y)> <bookTitle>$t</bookTitle> </bookByYear>

Skolem function

# Incremental View Maintenance

- Multi-linear property
  - Function $f$ is called multi-linear in each $R_i$ w.r.t. operation $\cup$ if the following holds

    $f(R_1, ..., R_i \cup \Delta R_i, ..., R_n) = f(R_1, ..., R_i, ..., R_n) \cup f(R_1, ..., \Delta R_i, ..., R_n)$
  - ONLY applicable under insertions
- Multi-linear in WHAX queries

  $V(\$db \cup \Delta\$db) = V(\$db) \cup V(\Delta\$db)$
- How to make WHAX multi-linear?
  - Key variable constraint
  - Base variable constraint

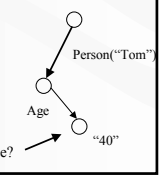
# Key Variable Constraint

- Variable in WHAX-QL
  - *parameter variable*: parameters to query
  - *label/key variable:* variables in Path Patterns bound to labels/keys
  - *value variable:* variable at the leaf of path($x_i$)
  - Example:

V($db) =
**where** <Person(Name:$n).Age> $a </> in $db
**construct …**

# Key Variable Constraint

- Key variable constraint
  - A WHAX-QL query is maintainable if no parameter/value variable $x occurs as a key variable or operand of some base operations $e_1$ op $e_2$
  - Intuition:
    - Insertion may cause deletion!
      - Consider another "age" is inserted for a person
    - Example:
      V($db) = where <Person($n).Age> $a </> in $db,
      $a > 36 <-value variable is an operand
      Construct <MyPerson($n).Age> $a </a>




What about it becomes a sub-tree?

# Key Variable Constraint

- Query-rewrite
  - Query re-write
    - Not always possible to rewrite a query into equivalent maintainable query
    - One way is use similar query that returns same expected results
  - Not maintainable query
    V($db) = where <Person($n).Age> $a </> in $db,
    $a > 36 <-value variable is an operand
    Construct <MyPerson($n).Age> $a </a>
  - Maintainable similar query
    V'($db) = where <Person($n).Age.$a> </> in $db,
    $a > 36 value variable becomes a label variable
    Construct <MyPerson($n).Age> $a </a>

# Base Variable Constraint

- Example query
  - V($db) = where
    <Person(Name:$n).Age> $a </> in **$db**
    <Conf(@name:"STACS", @year: $y).Publ(Title: $t).Author($n)></> in **$db**
    **Construct …**
  - V is not maintainable
    - V($db $\cup$ $\Delta$$db ) $\neq$ V($db) $\cup$ V($\Delta$$db)
    - $\Delta$*$db* may join with *$db,* multi-linearity does not hold here.
  - Solution: use distinct base variable
    - V'($db, $db') = where
      <Person(Name:$n).Age> $a </> in **$db**
      <Conf(@name:"STACS", @year: $y).Publ(Title: $t).Author($n)></> in $db'
      **Construct …**
    - V'($db, $db) = V($db)
    - V'($db $\cup$ $\Delta$$db, $db $\cup$ $\Delta$$db)
      = V($db, $db) $\cup$ V($\Delta$$db, $db) $\cup$ V($db, $\Delta$$db) $\cup$ V($\Delta$ $db, $\Delta$$db)

# Base Variable Constraint

- Summary
  - A maintainable WHAX-QL view V ($\$d_1$ , ... $\$d_n$) is multi-linear in its parameters $\$d_1$ , ... $\$d_n$ if all base variables $db of the same where-construct expression are distinct and do not occur in the construct-clause.
- For detail, check out the longer version of this paper
  - **"Efficient View Maintenance in XML Data Warehouses"** *Technical Report MS-CIS-99-27* (1999), Hartmut Liefke and Susan Davidson

# Deletions

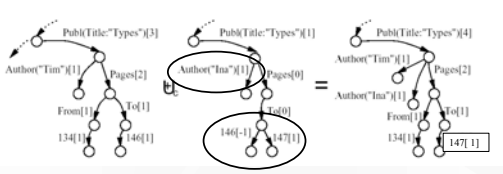
- No Multi-linearity in deletion
  - For deletion
    - $f(R_1, ..., R_i - \nabla R_i, ..., R_n) \neq f(R_1, ..., R_i, ..., R_n) - f(R_1, ..., \nabla R_i, ..., R_n)$
  - Reason: Union operation is not invertible
  - Example: V1(A, B) = {(1,2),(2,3)} ΔV1(A, B)= {(2,3), (4,5)}
    - V2 = V1 ∪ ΔV1 = {(1,2),(2,3),(4,5)}
    - But, V2 - ΔV1 = {(1,2)} ≠ V1
- Solutions
  - View analysis
    - Each tuple in view has only one derivation in base relation
  - Multi-set semantics/counting
    - Allow duplicates by counting

# WHAX w/ Counting

- *Support*: Count Value in WHAX
  - Each edge is annotated with *support*
  - *full support = direct support + indirect Support*
    - *d.support:* support for edge itself(possibly zero)
    - *i.support:* sum of supports of child edges
  - Assignment for data source
    - Leaf edge: f.support = d.support = 1
    - Inner edge: d.support = 0, i.support = # of leaf edges reached from this edge, f.support = i.support

# WHAX w/ Counting

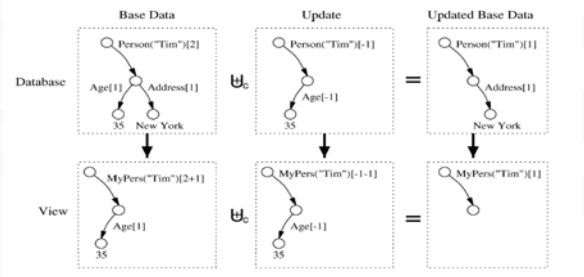
- Deep Union with Counting




Edge with 0 support(node 146) is deleted from the result

# View Maintenance in WHAX

- Simple Example



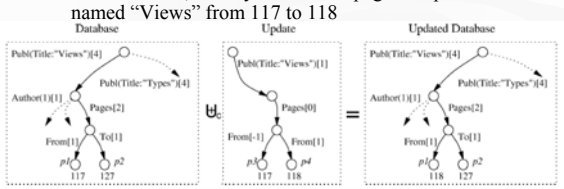

# View Maintenance in WHAX

- Computing Support for Views
  - Direct support for each output path $Pexpr_j$ is the product of full support of each input path $PPat_i$
    - $d.supp(Pexpr_j) = \prod f.supp(PPat_i)$
  - Example view

```
V($db, $db') = where
<Conf(@name: "VLDB", @year: "2000").Publ(Title: $t).Pages.From>
   $from </> in $db
<Conf(@name: "VLDB", @year: "2000").Publ(Title: $t).Pages.To> $to
   </> in $db'
Construct <publ(Title:$t).PageCount> $to - $from +1 </>
```

# View Maintenance in WHAX

- Example
  - In data source, modify the FROM-page of a publication named “Views” from 117 to 118

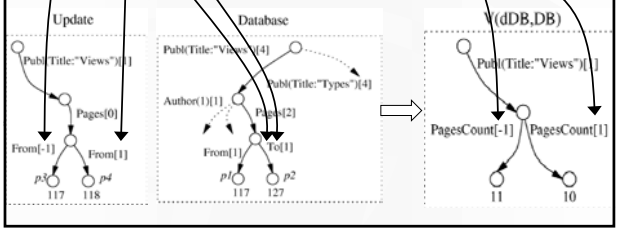



- Propagate update to the view
  - V(DB ∪ dDB, DB ∪ dDB) = V(DB, DB) + V(DB,dDB) + V(dDB, DB) + V(dDB,dDB)

Empty!

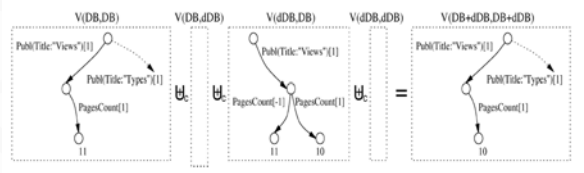
# View Maintenance in WHAX

- Path binding in V(dDB, DB)
  - (p3, p2), d.support = -1
  - (p4, p2), d.support = 1




# View Maintenance

- Propagate update to the view




# Aggregation

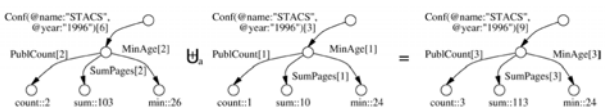
- Aggregation expression in construct-clause
  - sum(e), avg(e), count(), min(e), max(e), etc.
- Example

V = where <Conf(@name: “VLDB”, @year: “2000”).Publ(Title:$t) >$a</> in $db construct <Conf(“VLDB-2000”).PublCount>count( )</>

  - For each path binding, output path is constructed and aggregate count( ) simply sums up the number of output path

# Aggregation

- Deep Union for Aggregates
  - Aggregates can merged through deep union



# Summary & Future Work

- WHAX data model
  - A hierarchical data model with key constraints
  - Comprise both relational and semi-structured data source
  - WHAX-QL based on XML-QL
- Incremental view maintenance
  - Multi-linearity law for insertion
  - Counting method(*support*)for deletion
- Future Work
  - Ordered data structures
  - Other extensions: e.g, negation and recursion

**Both borrowed from RDBMS!**