

# **Index Structures for Matching XML Twigs Using Relational Query Processors**

Divesh Srivastava

AT&T Labs–Research

<http://www.research.att.com/~divesh/>

Joint work with Zhiyuan Chen, Johannes Gehrke, Flip Korn, Nick Koudas, Jayavel Shanmugasundaram

## Outline of Talk

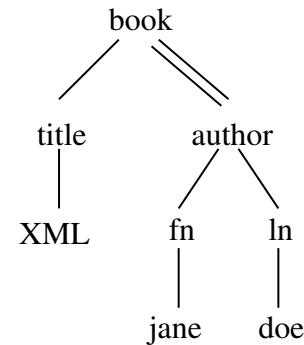
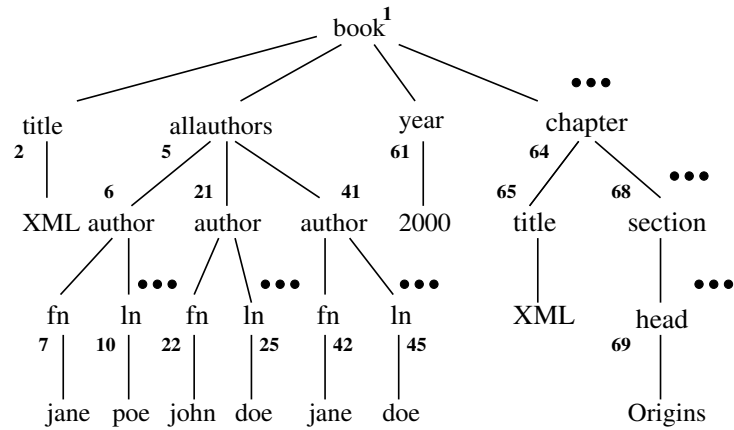
- Desiderata, indexing problems
- XML indexes, realized using  $B^+$ -trees
- Experimental results

# XML Document, XML Tree, Query Twig

```

<book>
  <title> XML </title>
  <allauthors>
    <author>
      <fn>jane</> <ln>poe</>
    </author>
    <author>
      <fn>john</> <ln>doe</>
    </author>
    <author>
      <fn>jane</> <ln>doe</>
    </author>
  </allauthors>
  <year> 2000 </year>
  <chapter>
    <title> XML </title>
    <section>
      <head> Origins </head> ...
    </section> ...
  </chapter> ...
</book>

```



## Desiderata

- XML data often stored in relational database systems
- Goal: Develop  $B^+$ -tree index structures that support
  - single index lookup for simple (parent-child) XML path queries
  - efficient evaluation of XML ad hoc, recursive, twig queries
  - tight coupling with relational query processors

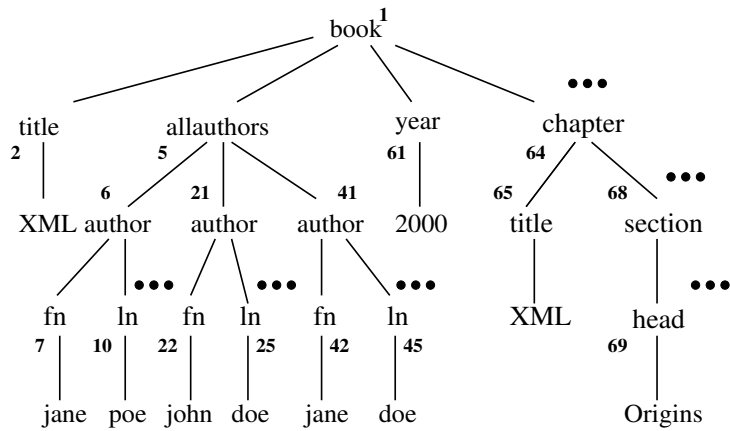
## Related Work

- APEX [4], DataGuide [10], etc. index paths and data separately
- Index Fabric [5] indexes XML paths and data values together
  - no efficient support for ad hoc, recursive, twig queries
- ViST [25], PRIX [21] support ad hoc, recursive, twig queries
  - uses expensive sub-sequence matching even for simple path queries
- Join Index [24], ASR [14] in OODBMS
  - targeted towards single path queries without recursion
- ToXin [22], XRel [29] index XML paths using relational database
  - ToXin like ASR, XRel requires multiple index lookups

## Problems: FreeIndex, BoundIndex

- PCsubpath: Query subpath with no intermediate “//”
- FreeIndex: Return all matches of PCsubpath in one index lookup
  - allows merge or hash join processing in RDBMS
  - /book/allauthors/author[fn = ‘jane’]: ([1,5,6,7], [1,5,41,42])
  - /book/allauthors/author[ln = ‘doe’]: ([1,5,21,25], [1,5,41,45])
- BoundIndex: Return all matches rooted at  $d$  in one index lookup
  - allows index nested loop join processing in RDBMS
  - /book[title = ‘XML’]: ([1,2])
  - //author[ln = ‘doe’] rooted at 1: ([21,25], [41,45])

# Relational Representation of All Data Paths



HeadId	SchemaPath	LeafValue	IdList
1	<b>B</b>	null	[ ]
1	<b>BT</b>	null	[2]
1	<b>BT</b>	XML	[2]
1	<b>BU</b>	null	[5]
1	<b>BUA</b>	null	[5, 6]
1	<b>BUAF</b>	null	[5, 6, 7]
1	<b>BUAF</b>	jane	[5, 6, 7]
1	<b>BUAL</b>	null	[5, 6, 10]
1	<b>BUAL</b>	poe	[5, 6, 10]
...	...	...	...
5	<b>U</b>	null	[ ]
5	<b>UA</b>	null	[6]
5	<b>UAF</b>	null	[6, 7]
5	<b>UAF</b>	jane	[6, 7]
5	<b>UAL</b>	null	[6, 10]
5	<b>UAL</b>	poe	[6, 10]
...	...	...	...

# ROOTPATHS Index

HeadId	SchemaPath	LeafValue	IdList
1	<b>B</b>	null	[ ]
1	<b>BT</b>	null	[2]
1	<b>BT</b>	XML	[2]
1	<b>BU</b>	null	[5]
1	<b>BUA</b>	null	[5, 6]
1	<b>BUAF</b>	null	[5, 6, 7]
1	<b>BUAF</b>	jane	[5, 6, 7]
1	<b>BUAL</b>	null	[5, 6, 10]
1	<b>BUAL</b>	poe	[5, 6, 10]
	...		

- $B^+$ -tree on  $\text{LeafValue} \cdot \text{reverse}(\text{SchemaPath})$ , returns IdList
  - solves FreeIndex problem in one index lookup
  - `//author[fn = 'jane']`: use prefix match key ('jane', FA\*)
  - `/book/allauthors/author/fn`: use prefix match key ('null', FAUB)
  - IdList is key to evaluating branching queries efficiently

# DATAPATHS Index

HeadId	SchemaPath	LeafValue	IdList
1	<b>BUAF</b>	null	[5, 6, 7]
1	<b>BUAF</b>	jane	[5, 6, 7]
1	<b>BUAL</b>	null	[5, 6, 10]
1	<b>BUAL</b>	poe	[5, 6, 10]
	...		
5	<b>UAF</b>	null	[6, 7]
5	<b>UAF</b>	jane	[6, 7]
5	<b>UAL</b>	null	[6, 10]
5	<b>UAL</b>	poe	[6, 10]
	...		

- $B^+$ -tree on LeafValue·HeadId·reverse(SchemaPath)
  - solves FreeIndex and BoundIndex problems in one index lookup
  - /book as a FreeIndex problem with virtual root as HeadId
  - //author[fn = 'jane'] as a BoundIndex problem for each book id
  - DATAPATHS index is bigger than ROOTPATHS index

## Family of Indexes

Index	Subset of SchemaPath	Sublist of IdList	Indexed Columns
Value [17]	paths of length 1	only last ID	SchemaPath, LeafValue
Forward link [17]	paths of length 1	only last ID	HeadId, SchemaPath
DataGuide [10]	root-to-leaf path prefixes	only last ID	SchemaPath
Index Fabric [5]	root-to-leaf paths	only first or last ID	SchemaPath, LeafValue
ROOTPATHS	root-to-leaf path prefixes	full IdList	LeafValue, reverse SchemaPath
DATAPATHS	all paths	full IdList	LeafValue, HeadId, reverse SchemaPath

- Given 4-ary relational representation, each index in family
  - stores a subset of all possible SchemaPaths
  - stores a sublist of IdList
  - indexes a subset of HeadId, SchemaPath, and LeafValue

## Experimental Setup

- XML data: Stored as Edge Table in IBM's DB2 version 7.2
  - 100MB scaled XMark data, 50MB DBLP data
- Query workload: 15 XMark queries, 3 DBLP queries

<i>Query</i>	<i>Branches</i>	<i>Result Size Per Branch</i>	<i>Depth of Branches</i>	<i>Recursions</i>
$Q1_x$ to $Q3_x$	1	1-11062	–	0
$Q1_d$ to $Q3_d$	1	1-10258	–	0
$Q4_x$ to $Q9_x$	2-3	1-7519	High	0
$Q10_x$ to $Q11_x$	2-3	3-59486	Low	0
$Q12_x$ to $Q15_x$	2-3	41-20946	Low	1

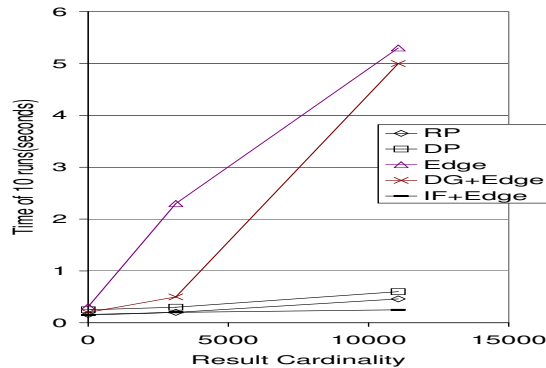
- Experimental platform: 1.7GHz Pentium, Windows 2000
  - 1GB memory, 37GB disk, 40MB buffer pool, OS cache turned off

## Alternative Indexing Strategies: Space Used

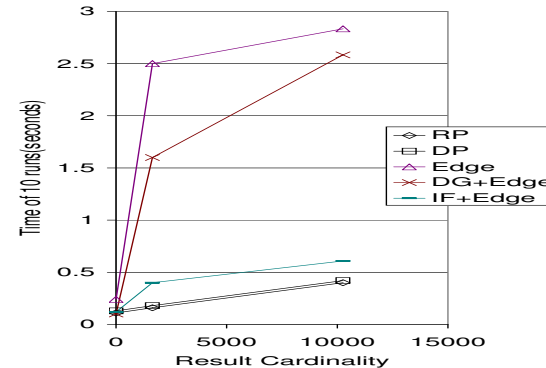
<i>Data set</i>	<i>RP</i>	<i>DP</i>	<i>Edge</i>	<i>DG+Edge</i>	<i>IF+Edge</i>	<i>ASR</i>	<i>JI</i>
<i>XMark</i>	119	431	127	169	167	464	822
<i>DBLP</i>	80	83	106	133	151	93	318

- ROOTPATHS, DATAPATHS, with differential IdList encoding
- Edge Table index, with value, forward link, backward link
- DataGuide + backward link on Edge Table
  - backward link to identify branch point id from leaf id
- Index Fabric + backward link on Edge Table
  - backward link to identify branch point id from leaf id

## Results: Indexing Schema Paths and Data Values



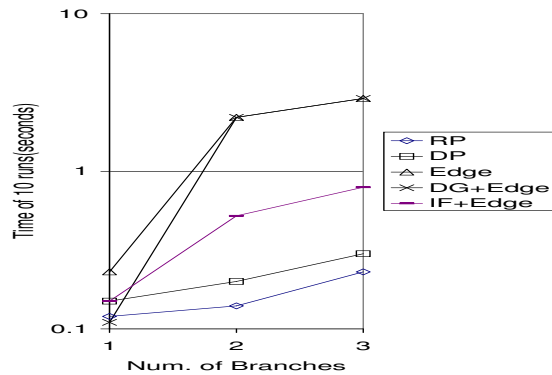
(a) XMark



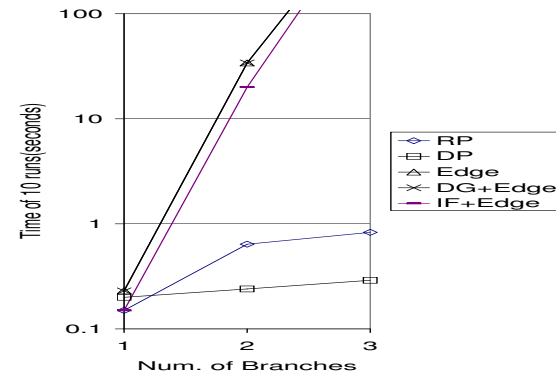
(b) DBLP

- Use fully specified path queries with varying selectivity
- Index Fabric, ROOTPATHS, DATAPATHS are best approaches
- Edge, DataGuide perform badly, as selectivity decreases
  - multiple expensive join operations need to be performed

## Results: Benefit of Returning IdLists



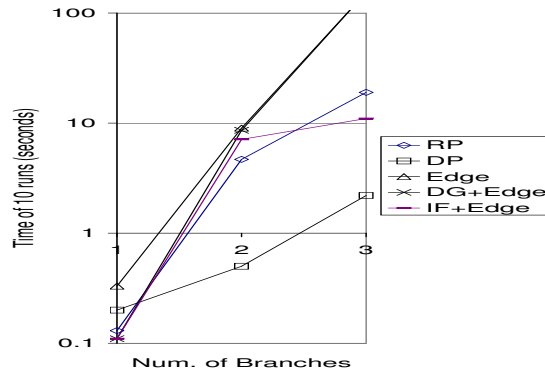
(a) Selective branches



(b) Selective & unselective branches

- Twig queries with varying number, selectivity of branches
- ROOTPATHS, DATAPATHS scale gracefully
- Edge, Index Fabric, DataGuide perform badly
  - 5-way expensive join needs to be performed for each branch

## Results: Benefit of Supporting BoundIndex



Twig queries with low branch points.

- Twig queries with selective and unselective branches
- DATAPATHS performs uniformly well
- Edge, Index Fabric, DataGuide, ROOTPATHS perform badly
  - ROOTPATHS does not support index nested loop join (BoundIndex)

# Conclusions

- Technical contributions of paper
  - framework for classifying a family of XML indexes
  - propose ROOTPATHS, DATAPATHS with different space-time tradeoffs
  - instantiate indexes using B<sup>+</sup>-trees in RDBMSs
  - tightly integrate indexes with relational query processor
  - evaluate performance tradeoff between index space and matching time
- Future work: Efficient update techniques