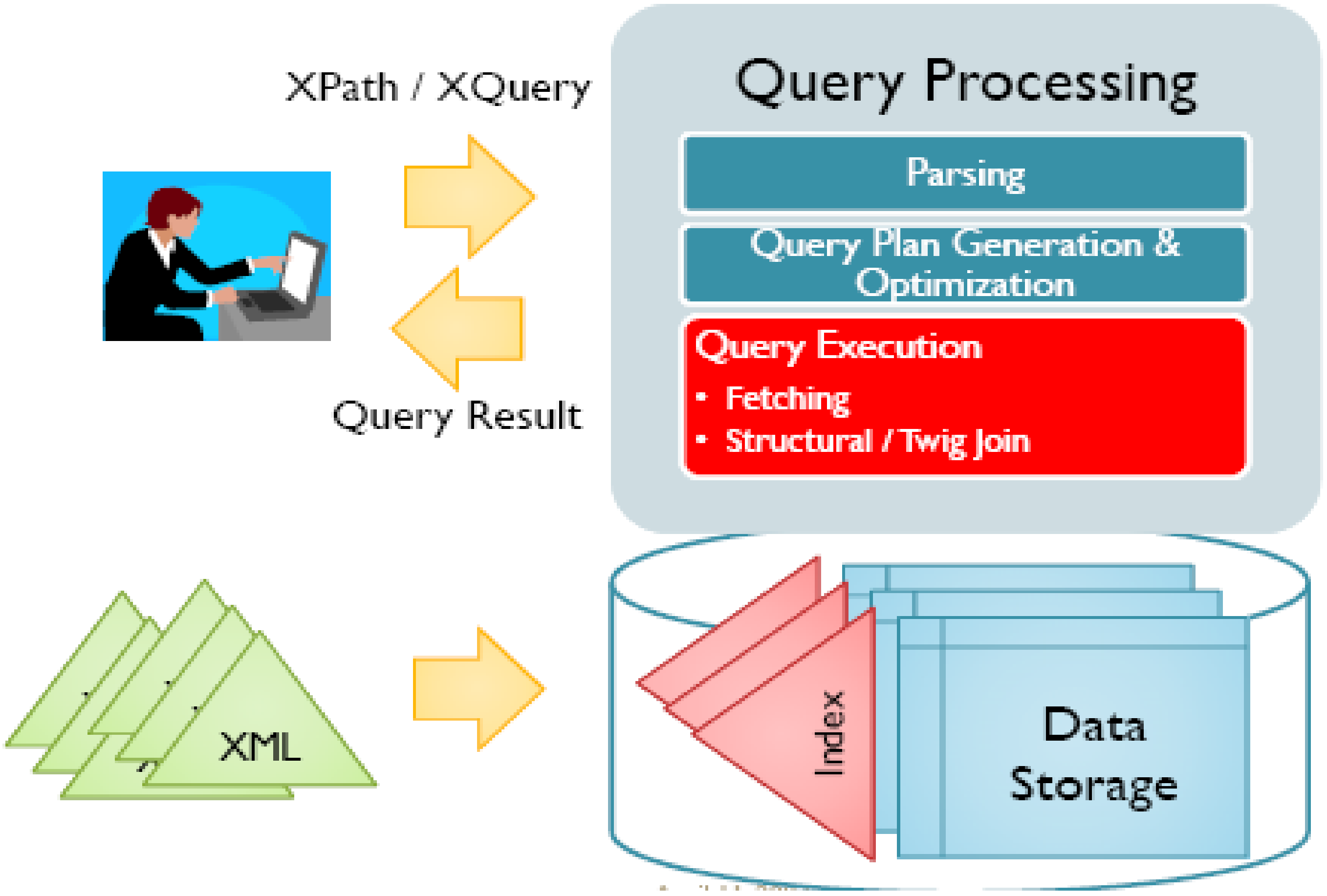
The Storage for Semi-structured Data

Three types of XML database systems:

- **Native XML databases**
  - XML data often is stored in proprietary object repositories or in text files, in which tags encode the schema
  - Lorel and Tsimmis store their data as graphs; the schema is stored as attributes labeling the graph's edges
  - Strudel stores the data externally as structured text, and internally as a graph
  - Use labeling scheme, index, structural joins
- **XML-enable relational databases**
  - Map XML data into relational tables
  - Many different mapping methods
- Hybrid database systems
  - Support both relational and XML data types
  - DB2 9 (IBM), Oracle 10g rel 2, SQL Server 2005 (Microsoft)

An XML Database System



Native XML Databases:

In the file system

- store each XML document as a separate operating system file and use a DOM parser to create a DOM tree in main memory whenever the document is accessed by a query.
- Advantages
  - It is easy to implement
  - It does not require the use of a database system or storage manager.

In the file system (cont.)

- Disadvantages
  - XML files in ASCII format need to be parsed every time when they are accessed for either browsing or querying.
  - the entire parsed file must be memory-resident during query processing.
  - it is hard to build and maintain indices on documents stored this way.
  - update operations are difficult to implement.

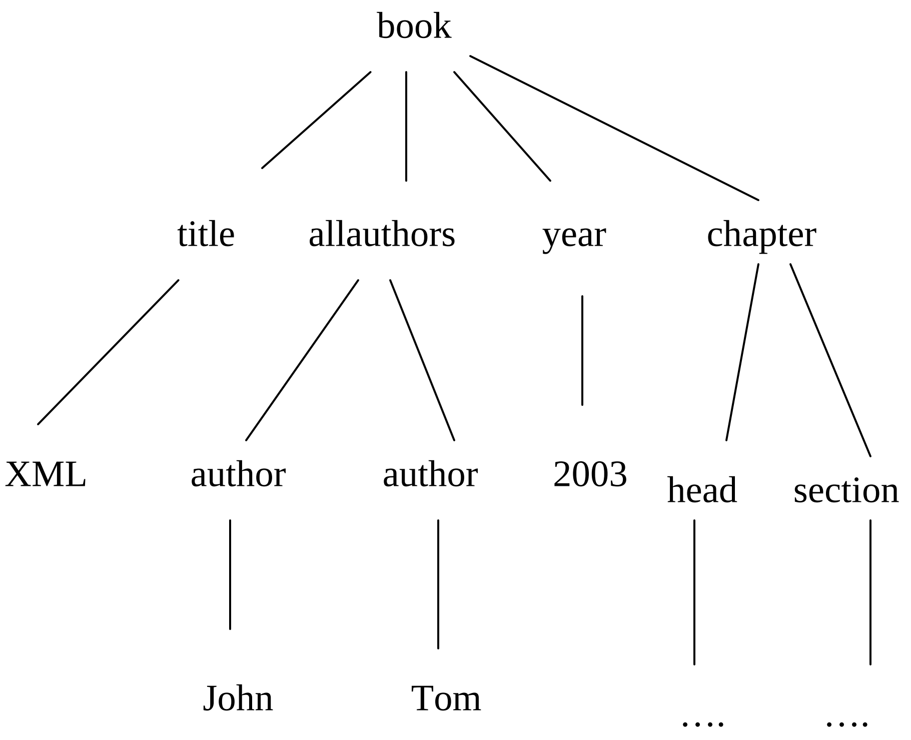
Native XML Databases:

Stored as Lists of Elements (streams)

- Use some labeling scheme to number the nodes of the XML document
- For query processing using structural join or twig pattern query
- XML employs tree-structured model for representing data
- XML query can be decomposed into a set of basic structural (parent-child, ancestor-descendant, following, etc.) relationships between pairs of nodes or some twig patterns

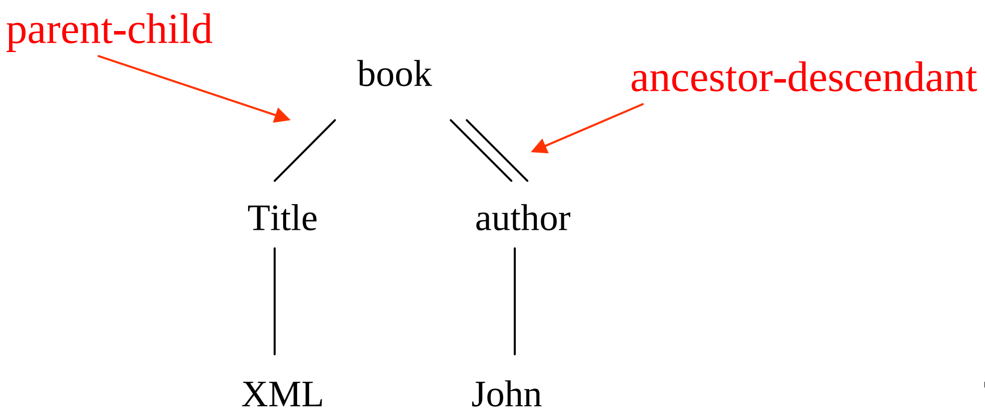
```
<book title="XML">
    <allauthors>
        <author>John</author>
        <author>Tom</author>
     </allauthors>
    <year>2003</year>
    <chapter>
      <head>….</head>
      <section>…</section>
    </chapter>
</book>
```

a) XML source

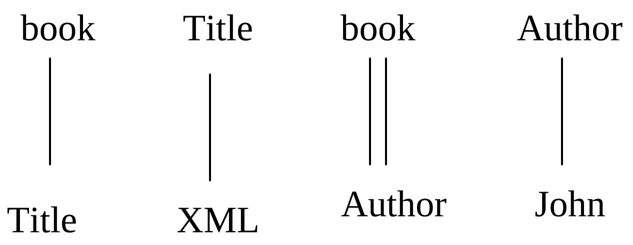


b) XML tree

Any node in XML tree may be an element, attribute, value of XML source.



c) Twig Pattern Query



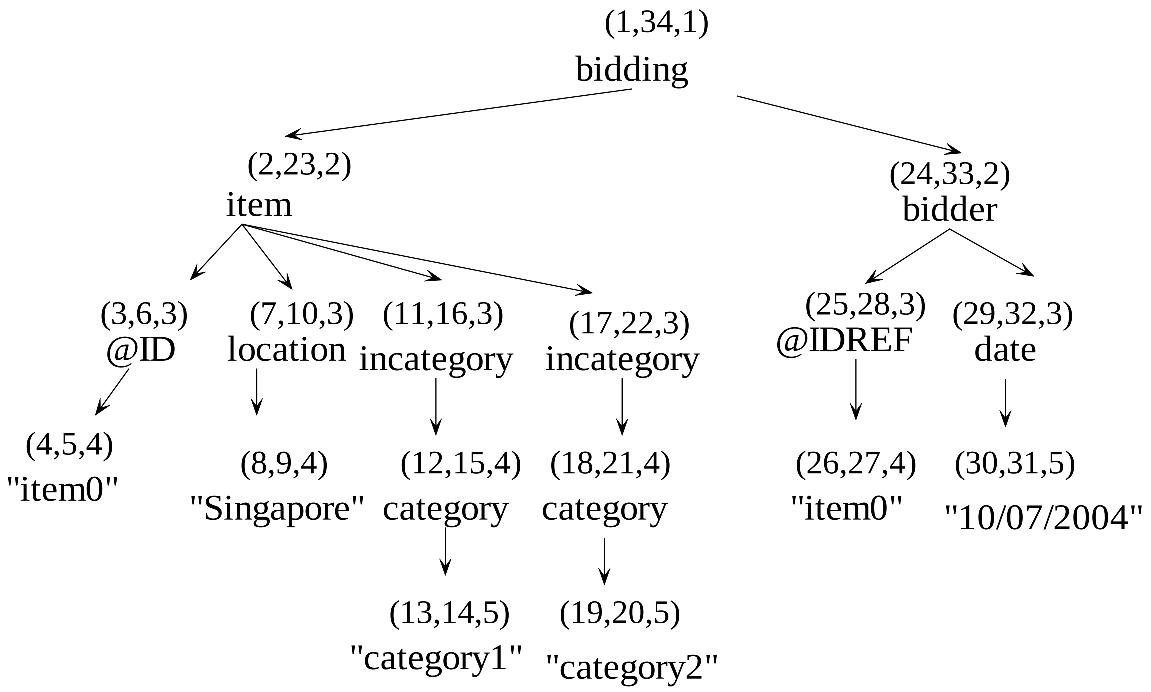
d) Basic Structural relationship

Node Labeling schemes

- The method of assigning the labels to nodes of the XML trees is called a node labeling scheme.
- Labels of the same tag name are stored in a stream in document order.
- Given 2 labels of 2 nodes, we can determine parent-child, ancestor-descendant, following, or preceding relationships, etc. of the 2 nodes. These relationships are important parts of XML queries.

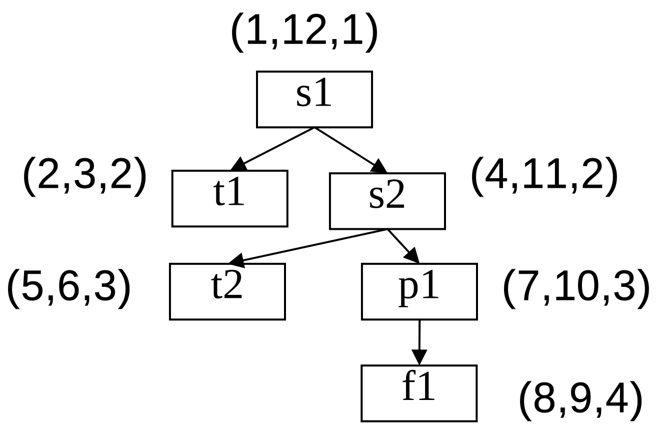
(1) Containment (Range) labeling scheme

- Each node is assigned with three values, i.e. (**start, end, level**).
- Property 1: Node a is an **ancestor** of node b if and only if a.start<b.start and a.end>b.end
- Property 2: Node a is the **parent** of node b if and only if a.start<b.start, a.end>b.end, and a.level=b.level-1.
- Labels of the same tag name are stored as a **stream** in document order.



Another example: An XML tree with containment labels and its **streams.**

An XML tree:

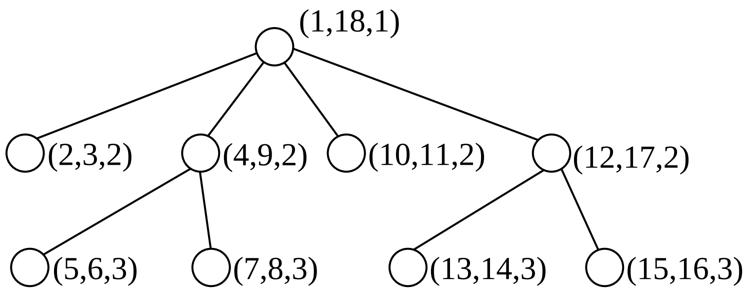


Data streams:

T_s	(1,12,1), (4,11,2)
T_t	(2,3,2), (5,6,3)
T_f	(8,9,4)
T_p	(7,10,3)

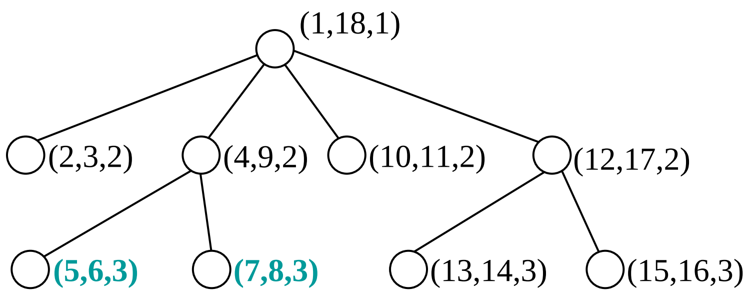
(1) Containment (Range) labeling scheme (cont.)

- **Ancestor-descendant.** (5,6,3) is a descendant of (1,18,1) because interval [5,6] is contained in interval [1,18]
- **Parent-child.** (5,6,3) is a child of (4,9,2) because interval [5,6] is contained in interval [4,9], and levels 3-2=1
- **Ordering.** (5,6,3) is before (10,11,2) in document order because the "start" of (5,6,3) i.e. 5 is smaller than the "start" of (10,11,2) i.e. 10.



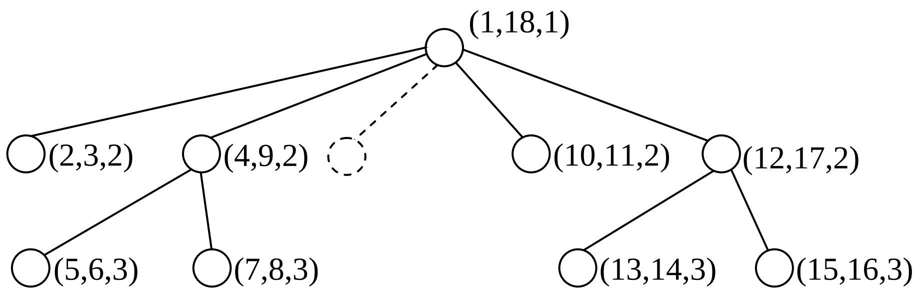
Containment is bad to determine the sibling relationship

- Sibling.
  - To determine whether (7,8,3) is a sibling of (5,6,3), containment scheme needs to search the parent of (5,6,3) firstly, then determine whether (7,8,3) is a child of this parent.
  - The containment scheme needs to determine many parent-child relationships to get the sibling relationship.
  - Very expensive.



Containment is bad to process updates

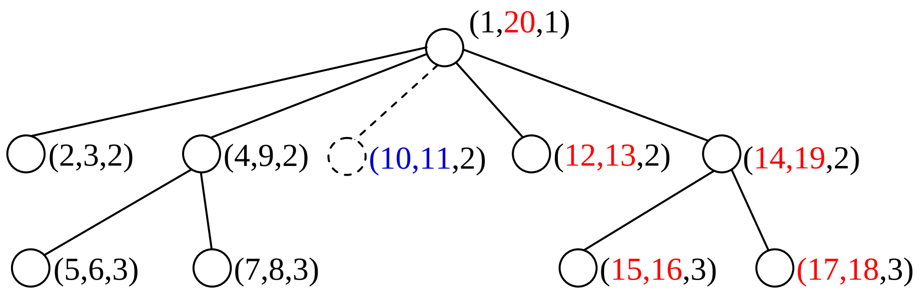
- Need to re-label all the ancestor nodes and all the nodes after the inserted node in document order



Containment is bad to process updates

- Need to re-label all the ancestor nodes and all the nodes after the inserted node in document order

- All the red color numbers were changed from their original values, very *expensive*



Existing approaches to process the updates in containment scheme

- Increase the interval size and leave some values unused for the future insertions [Li et al VLDB01]
  - When unused values are used up, have to re-label
- Use float-point value [Amagasa et al ICDE03]
  - Float-point value represented in a computer with a fixed number of bits
  - Due to float-point precision, have to re-label
- They both can not completely avoid the re-labeling

[Li et al VLDB01] Q. Li and B. Moon. Indexing and Querying XML Data for Regular Path Expressions. In *Proc. of VLDB*, pages 361-370, 2001.

[Amagasa et al ICDE03] T. Amagasa, M. Yoshikawa, and S. Uemura. QRS: A Robust Numbering Scheme for XML Documents. In *Proc. of ICDE*, pages 705-707, 2003.

(2) Prefix labeling scheme

- Three main prefix schemes
  - DeweyID [Tatarinov et al SIGMOD02]
  - BinaryString [Cohen et al PODS02]
  - OrdPath [O'Neil et al SIGMOD04]
- Determine different relationships based on the prefix property
- We will only discuss DeweyID scheme

[Tatarinov et al SIGMOD02] I. Tatarinov, S. Viglas, K.S. Beyer, J. Shanmugasundaram, E.J. Shekita, and C. Zhang. Storing and querying ordered XML using a relational database system. In *Proc. of SIGMOD*, pages 204-215, 2002.

[Cohen et al PODS02] E. Cohen, H. Kaplan, and T. Milo. Labeling Dynamic XML Trees. In *Proc. of PODS*, pages 271-281, 2002.

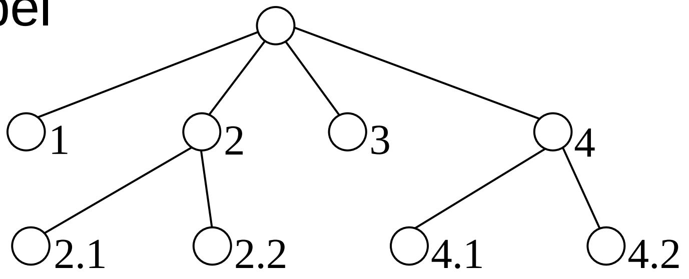
[O'Neil et al SIGMOD04] P.E. O'Neil, E.J. O'Neil, S. Pal, I. Cseri, G. Schaller, and N. Westbury. ORDPATHs: Insert-Friendly XML Node Labels. In *Proc. of SIGMOD*, pages 903-908, 2004.

DeweyID [Tatarinov et al SIGMOD02]

- Label the nth child of a node with an integer n
- This n should be concatenated to the prefix (i.e. its parent’s label) and delimiter (e.g. “.”) to form the complete label of this child node.
- The label of the root of the XML tree is an empty string (for all the prefix labeling schemes).

- Ancestor-descendant. “2.1” is a descendant of the root, because the label of the root is empty which is a prefix of “2.1”

- Parent-child. “2.1” is a child of “2” because “2” is an immediate prefix of “2.1”, i.e. when removing “2” from the left side of “2.1”, “2.1” has no other prefixes.

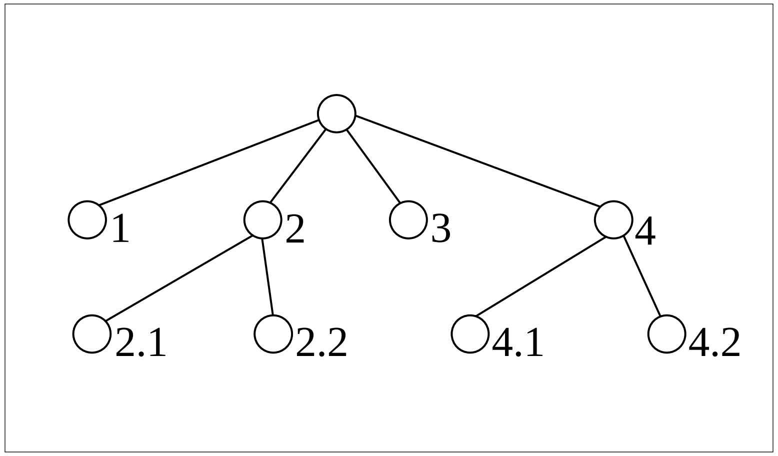


[Tatarinov et al SIGMOD02] I. Tatarinov, S. Viglas, K.S. Beyer, J. Shanmugasundaram, E.J. Shekita, and C. Zhang. Storing and querying ordered XML using a relational database system. In *Proc. of SIGMOD*, pages 204-215, 2002.

DeweyID (Cont.)

- Ordering. “2.1” is before “3” in document order because the “2” in “2.1” is smaller than of “3”.

- Sibling. “2.2” is a sibling of “2.1” because their prefixes are the same, i.e. their prefixes are both “2”.

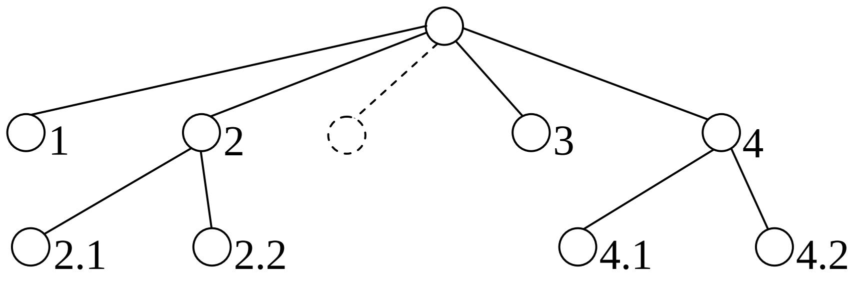


Bad to process all the four basic relationships if the XML tree is tall.

DeweyID is bad to process order-sensitive updates

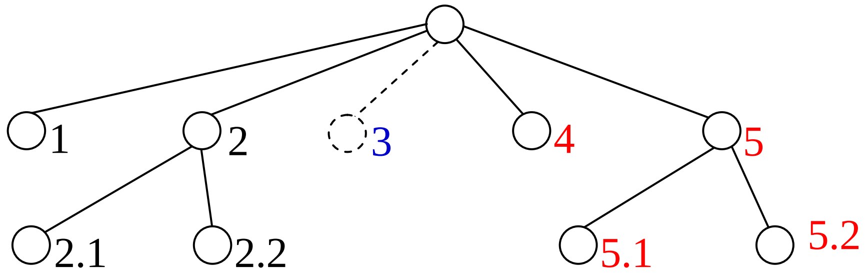
- Order-sensitive updates ---- to maintain the document order when updates are performed

- Need to re-label all the sibling nodes after the inserted node and all the descendants of these siblings



DeweyID is bad to process order-sensitive updates (cont.)

- Order-sensitive updates ---- to maintain the document order when updates are performed
- Need to re-label all the sibling nodes after the inserted node and all the descendants of these siblings
- All the red color labels were changed from the original values, very *expensive*



XML query processing : XML Twig Pattern approach

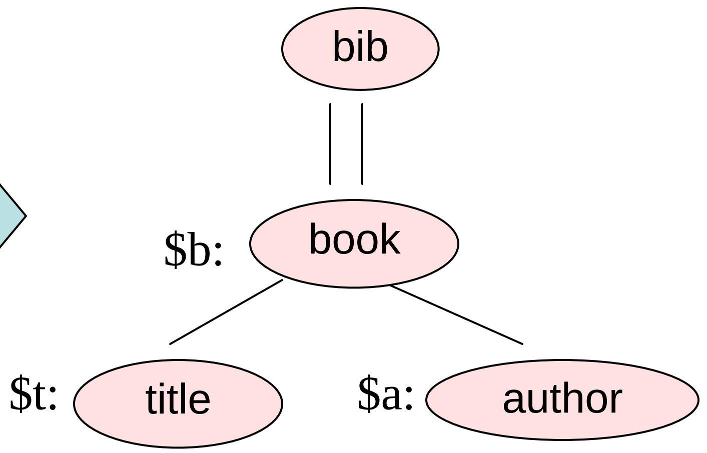
- XML twig pattern matching is a core operation in XPath and XQuery
- Definition of XML Twig Pattern Query: an XML twig pattern query is a small tree whose nodes are tags, attributes or text values; and edges are either parent-child (P-C) or ancestor-descendant (A-D) relationships, etc.

An XML twig pattern example

Create a flat list of all the title-author pairs for every book in bibliography.

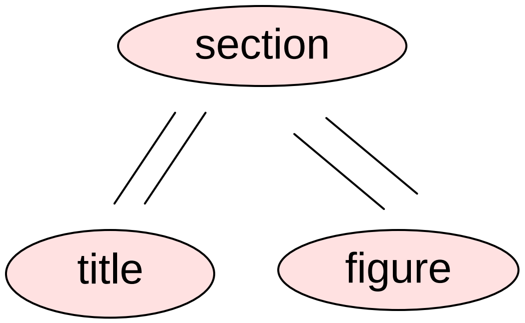
To answer the XQuery, we need to match the following XML twig pattern:

```
XQuery:
<results>
{
  for $b in doc("bib.xml")/bib//book,
       $t in $b/title,
       $a in $b/author,
  return
  <result> { $t } { $a } </result>
 }
</results>
```

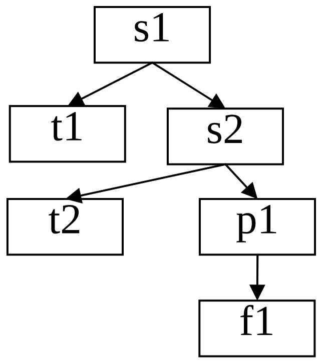


- Given an XML twig pattern *Q*, and an XML database *D*, we need to find **all** the matches of *Q* on D efficiently.
- E.g. Consider the following twig pattern and document:

Twig pattern:



An XML tree:



Query solutions:

(s1, t1, f1)
(s2, t2, f1)
(s1, t2, f1)

Notation: s1 denote the first element of section in XML tree by pre-order,...etc

Using a relational DBMS

- XML data is stored in relations and the XML query language (for example, XML-QL or XQuery) is translated to SQL and executed by the underlying relational database system.
- Some approaches:

  -- The Edge Approach
  -- The Attribute Approach
  -- Universal Table
  -- Normalized Universal Approach
  -- XRel
  -- Vectorization technique
  -- STORED
  -- Shore
  -- B-Tree approach

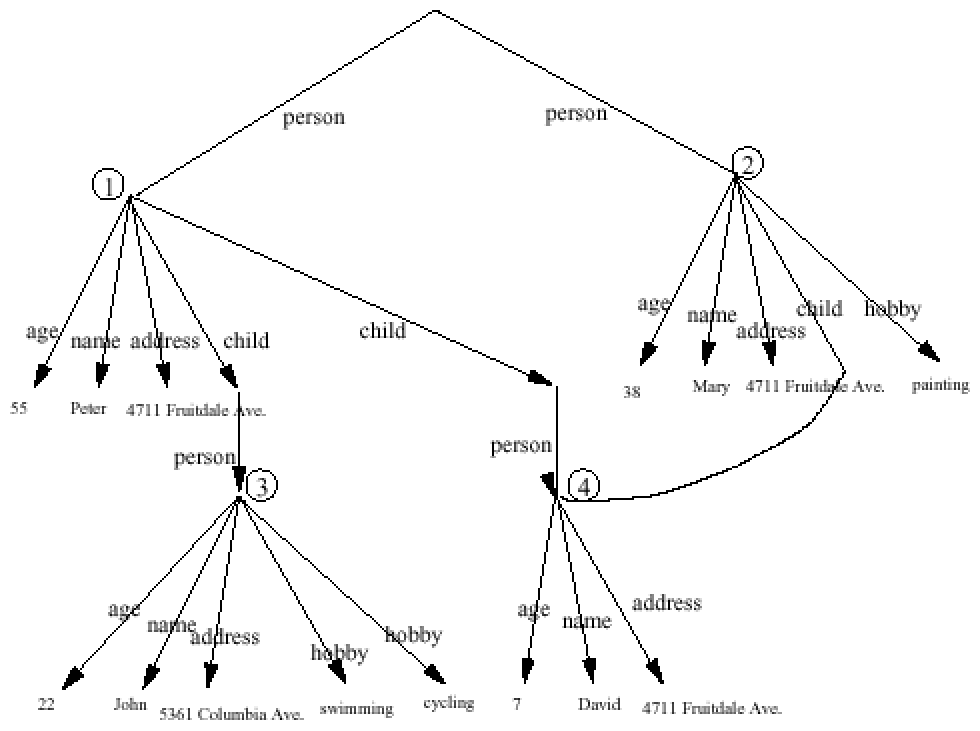
(1) The Edge Approach

- Every node (XML element) in the directed graph is assigned an id
- Each tuple in the edge table corresponds to one directed edge in the directed graph
- Schema –

  Edge (<u>source, ordinal</u>, name, flag, target)

Notes: **name** is the label of the edge;
source is the ID of node in the XML tree;
ordinal with value i indicates it (name) is the ith child of its parent (source) node;
flag value is to indicate whether the target value is a node id or a value.

(1) The Edge Approach (cont.)



(1) The Edge Approach (cont.)

source	ordinal	name	flag	target
1	1	age	1	55
1	2	name	1	peter
1	3	address	1	Fruit
1	4	child	0	3
1	5	child	0	4
2	1	age	1	38
…	…	…		…

Note: flag=**1** means target is a value.
flag=**0** means target is a node id.

(1) The Edge Approach (cont.)

- an index on the *source* column
- a combined index on the {*name, target*} columns.

(2) The Attribute Approach

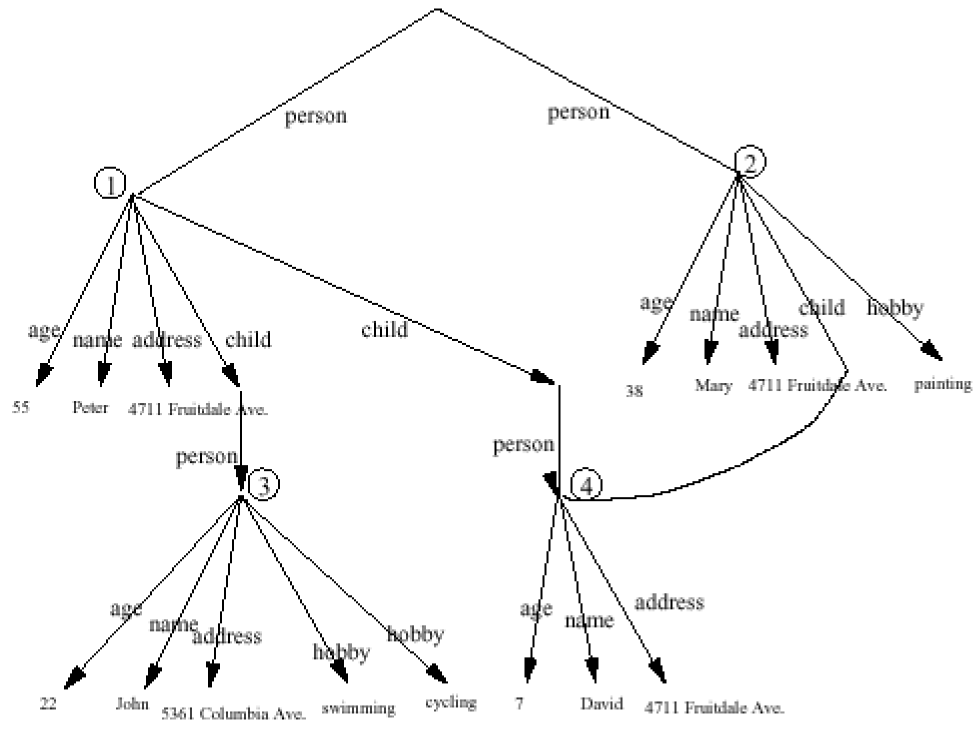
- Group all attributes with the same name into one table
- A horizontal partitioning of the Edge table, using *name* as the partitioning attributes
- A_{age} (<u>source, ordinal</u>, flag, target)

  A_{name} (<u>source, ordinal</u>, flag, target)

  $A_{address}$ (<u>source, ordinal</u>, flag, target)

  …

(2) The Attribute Approach (cont.)



(2) The Attribute Approach (cont.)

A_{hobby}

source	ordinal	target
2	5	Painting
3	4	Swimming
3	5	cycling

A_{child}

source	ordinal	target
1	4	3
1	5	4
2	4	4

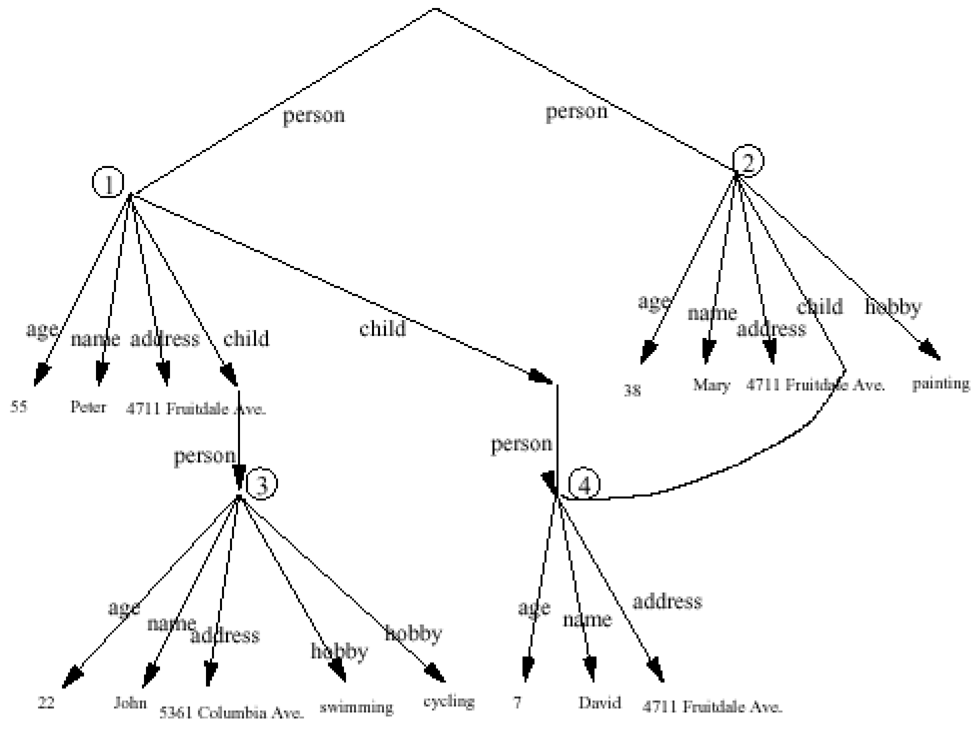
Note: Need a flag attribute for the tables to indicate whether the target is a value or an node id.

(3) Universal Table Approach

- Structure

Universal (source, ordinal$_{n1}$, flag$_{n1}$, target$_{n1}$,
ordinal$_{n2}$, flag$_{n2}$, target$_{n2}$,
. . . ,
ordinal$_{nk}$, flag$_{nk}$, target$_{nk}$)

(3) Universal Table Approach (cont.)



(3) Universal Table Approach (cont.)

source	…	ord_{name}	$\text{targ}_{\text{name}}$	…	$\text{ord}_{\text{child}}$	$\text{targ}_{\text{child}}$	$\text{ord}_{\text{hobby}}$	$\text{targ}_{\text{hobby}}$
1	…	2	Peter	…	4	3	null	null
1	…	2	Peter	…	5	4	null	null
2	…	2	Mary	…	4	4	5	painting
3	…	2	John	…	null	null	4	swimming
3	…	2	John	…	null	null	5	cycling
4	…	2	David	…	null	null	null	null

Note: There is also a flag for each target attribute.
The universal table has a lot of redundant data because of multi-valued attributes.

(4) Normalized Universal Approach

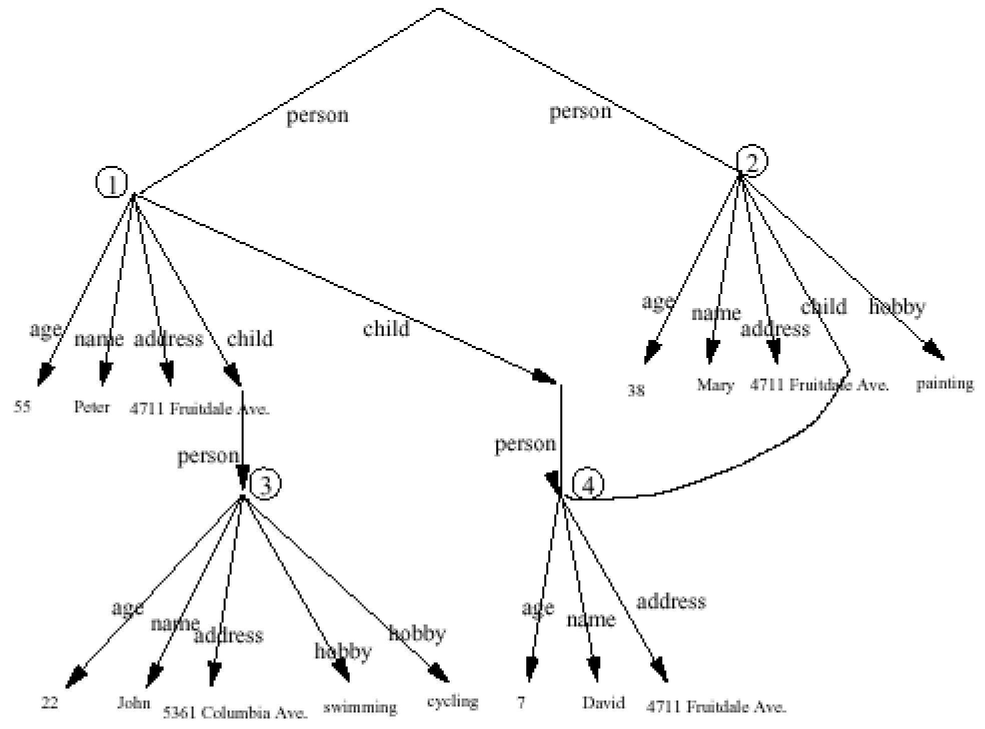
- UnivNorm (<u>source</u>, ordinal_{n1}, flag_{n1}, target_{n1},
  ordinal_{n2}, flag_{n2}, target_{n2},
  . . . ,
  ordinal_{nk}, flag_{nk}, target_{nk})
- Overflow_{n1} (<u>source, ordinal</u>, flag, target),

  . . . ,

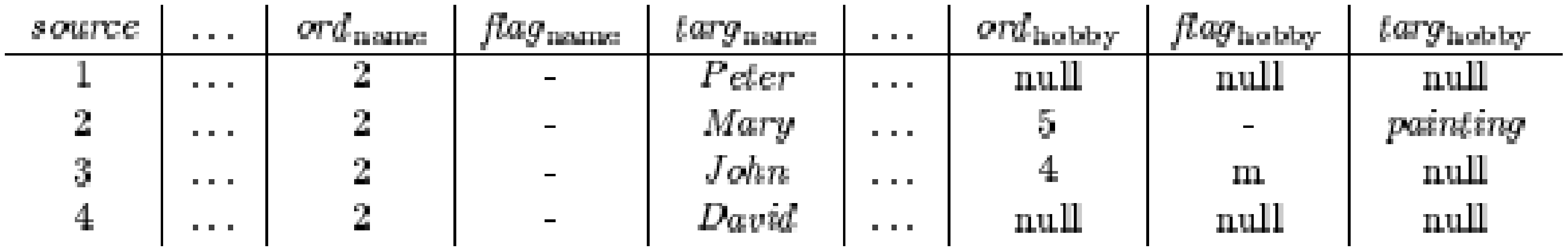
  Overflow_{nk} (<u>source, ordinal</u>, flag, target)

Note: (1) Overflow tables are for multivalued attributes.
(2) flag_{n1} is the flag for target_{n1}.

(4) Normalized Universal Approach (cont.)



(4) Normalized Universal Approach (cont.)



source	...	ord_{name}	$flag_{name}$	$targ_{name}$	...	ord_{hobby}	$flag_{hobby}$	$targ_{hobby}$
1	...	2	-	*Peter*	...	null	null	null
2	...	2	-	*Mary*	...	5	-	*painting*
3	...	2	-	*John*	...	4	m	null
4	...	2	-	*David*	...	null	null	null

$\text{Overflow}_{\text{hobby}}$

source	ordinal	target
3	4	swimming
3	5	cycling

$\text{Overflow}_{\text{child}}$

source	ordinal	target
1	4	3
1	5	4

Note: hobby and child are mutlivalued attributes of person object. Also need a flag attribute for the overflow tables. flag = m in the UR table indicates the target is a multi-valued attribute and has an overflow table.

(5) XRel – a path-based approach

- Labels the XML document using containment scheme and then divide a document into four parts: path, element, text, and attribute, and store them into 4 relation tables

Ref: M. Yoshikawa and T. Amagasa. XRel: a path-based approach to storage and retrieval of XML documents using relational databases. In Proc. of ACM TOIT (2001)

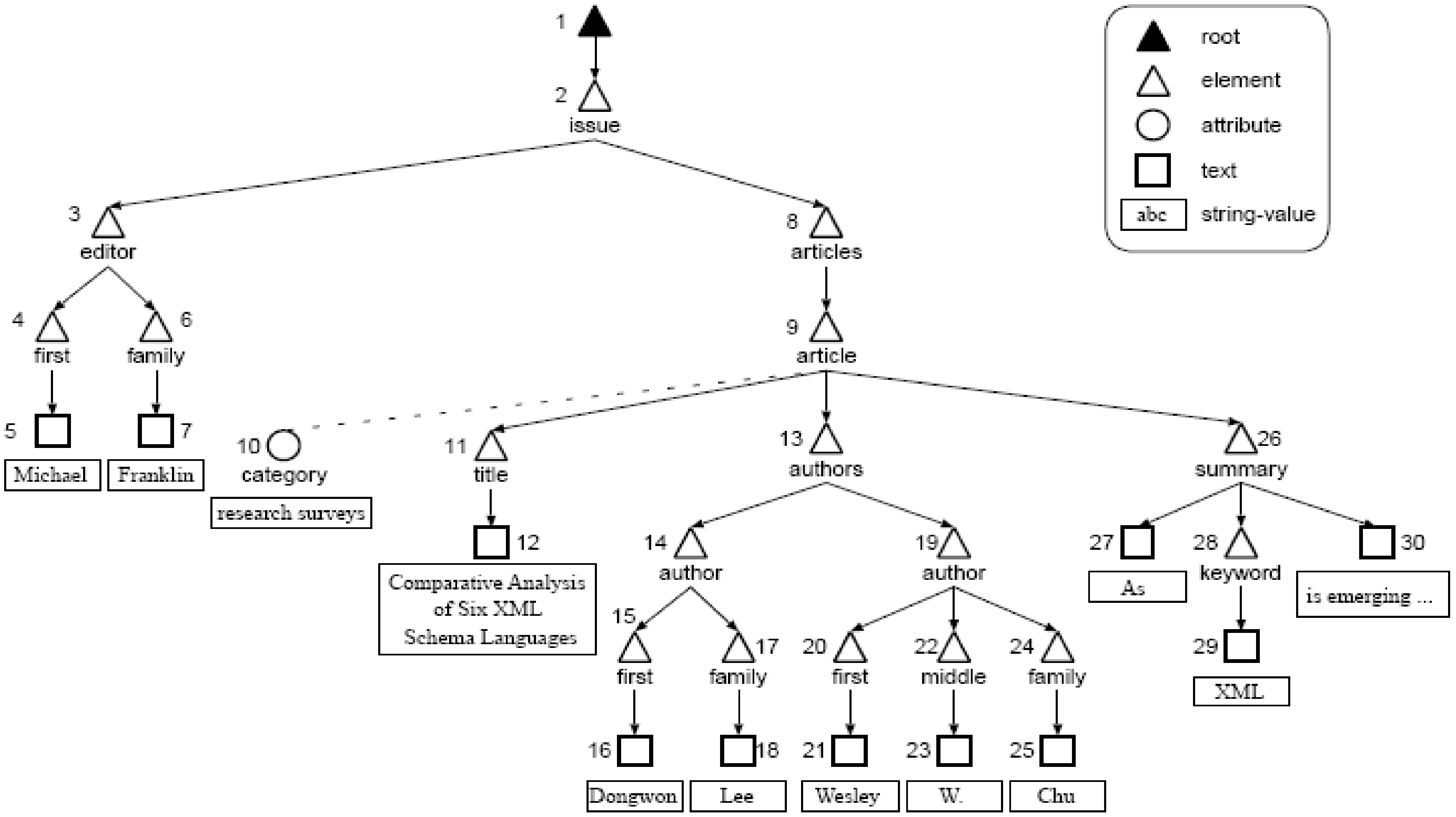


Figure: An XML tree

The four tables of the XML tree are:

(1) Element table

- Elements of the XML data

Node 3: The database attributes index and reindex in the relation Element represent the occurrence order of an element node among the sibling element nodes of the same node type in document order and reverse document order, respectively.

Element

docID	pathID	start	end	index	reindex	NodeID
1	1	0	729	1	1	2
1	2	7	70	1	1	3
1	3	15	36	1	1	4
1	4	37	61	1	1	6
1	5	71	721	1	1	8
1	6	81	710	1	1	9
1	8	118	180	1	1	11
1	9	181	335	1	1	13
1	10	190	248	1	2	14
1	11	198	219	1	1	15
1	12	220	239	1	1	17
1	10	249	325	2	1	19
1	11	257	277	1	1	20
1	13	278	296	1	1	22
1	12	297	316	1	1	24
1	14	336	700	1	1	26
1	15	348	369	1	1	28

Note1: start and end are parts of the containment label

Note2: pathID is a foreign key to the Path table

Note4: NodeID is just for easier reference purpose, not actually required

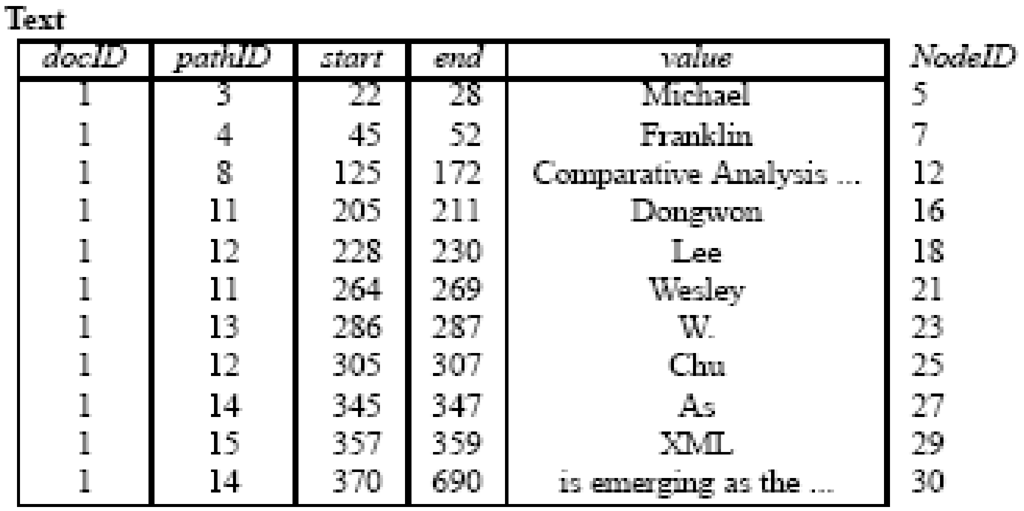
(2) Attribute table

- Attributes of XML elements

Attribute

docID	pathID	start	end	value	NodeID
1	7	82	82	research surveys	10

(3) Text table

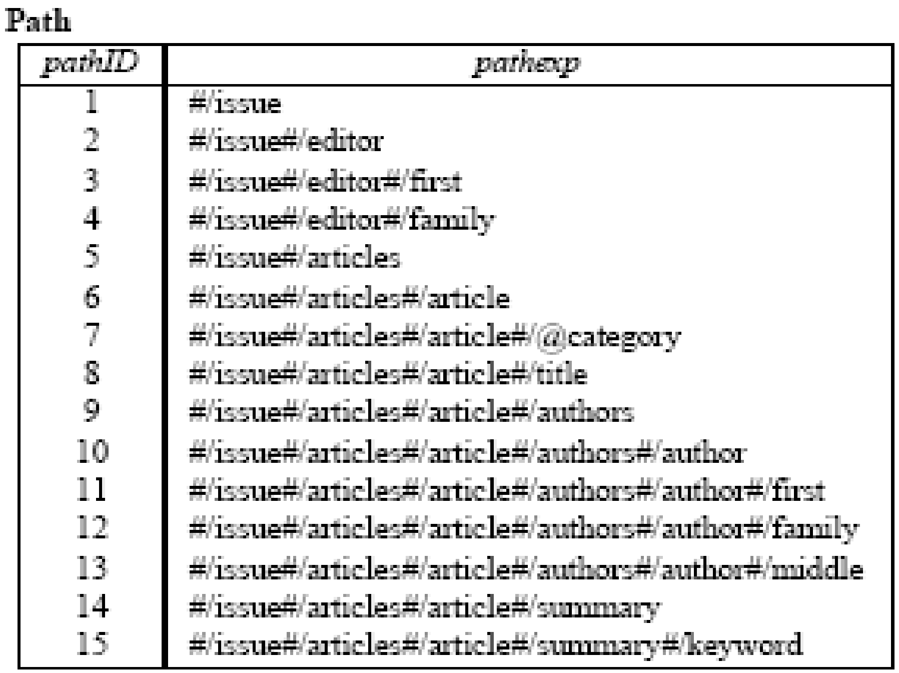


Text

docID	pathID	start	end	value	NodeID
1	3	22	28	Michael	5
1	4	45	52	Franklin	7
1	8	125	172	Comparative Analysis ...	12
1	11	205	211	Dongwon	16
1	12	228	230	Lee	18
1	11	264	269	Wesley	21
1	13	286	287	W.	23
1	12	305	307	Chu	25
1	14	345	347	As	27
1	15	357	359	XML	29
1	14	370	690	is emerging as the ...	30

(4) Path table

- XPaths from root to leaf nodes are given a pathID



Path

pathID	pathexp
1	#/issue
2	#/issue#/editor
3	#/issue#/editor#/first
4	#/issue#/editor#/family
5	#/issue#/articles
6	#/issue#/articles#/article
7	#/issue#/articles#/article#/@category
8	#/issue#/articles#/article#/title
9	#/issue#/articles#/article#/authors
10	#/issue#/articles#/article#/authors#/author
11	#/issue#/articles#/article#/authors#/author#/first
12	#/issue#/articles#/article#/authors#/author#/family
13	#/issue#/articles#/article#/authors#/author#/middle
14	#/issue#/articles#/article#/summary
15	#/issue#/articles#/article#/summary#/keyword

Note: Tag names are not explicitly stored in the element, attribute, and text tables, but only stored in the Path table as the last node of the XPath expressions. Also levels of nodes are not explicitly stored.

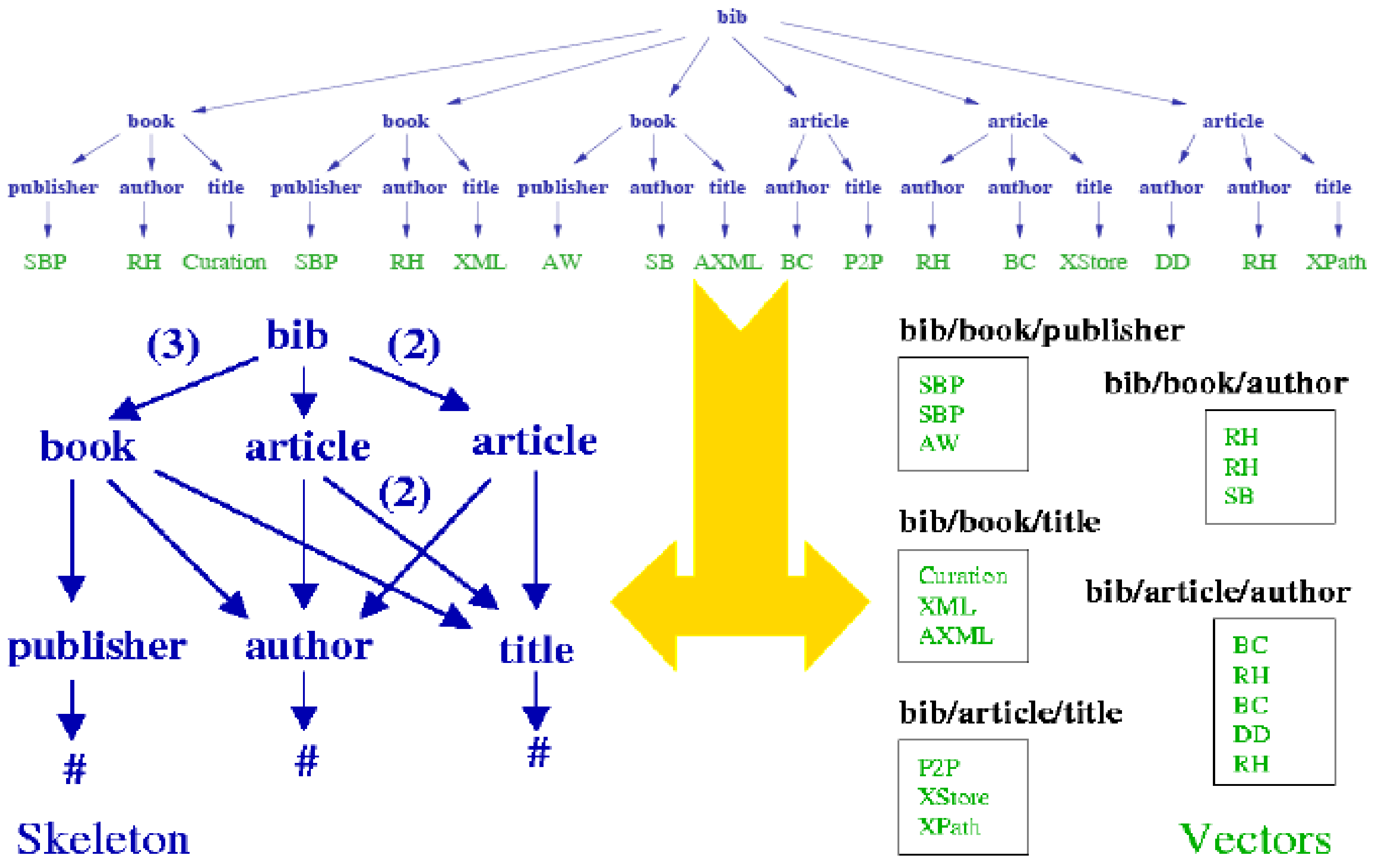
Figure A XRel storage of the XML docuents

(6) Vectorising Large XML repositories

An XML document is decomposed into 2 parts

- a compressed skeleton describing the structure of the document, where string data is replaced by a placeholder "#"
- Data vector files containing the string data. The root-to-leaf path determines the file that the string data is written to.

Ref: Peter Buneman, Byron Choi, Wenfei Fan, Robert Hutchison, Robert Mann, Stratis Viglas: Vectorizing and Querying Large XML Repositories. ICDE 2005: 261-272



Notes:

(1) There are two "article" nodes in the skeleton. One type of articles node has one author, and another type of articles has 2 authors.

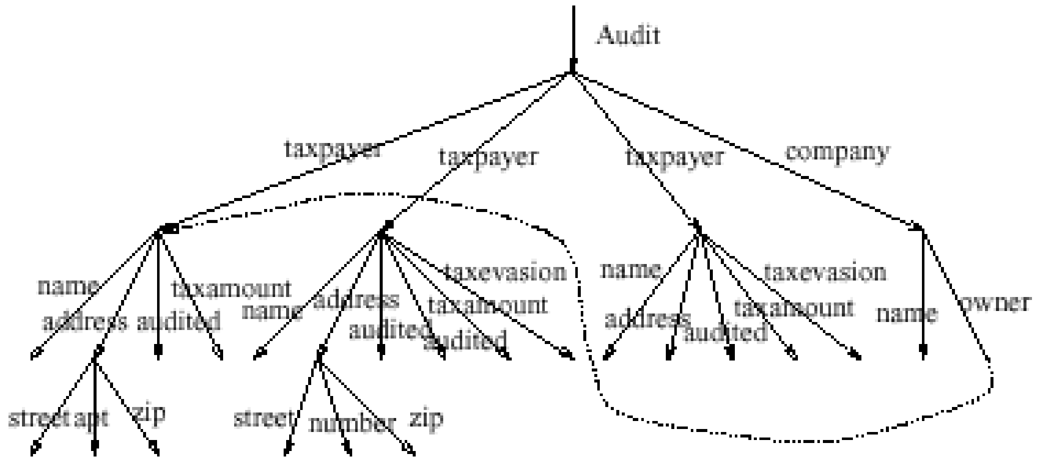
(2) The number (3) indicates bib has 3 child book nodes.

(3) Q: How to find which title is published by which publisher?

(7) STORED

- use relational database management systems to store and manage semistructured data
- a mapping between the semistructured data model and the relational data model
- When a semistrcutured data instance is given, a STORED mapping can be generated automatically using data-mining techniques. Q: How to find the mapping?

(7) STORED (cont.)

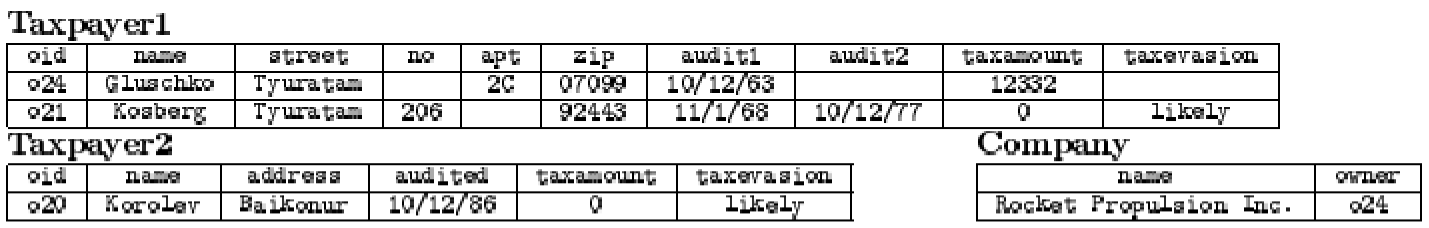


```
Audit: &o1
  {taxpayer: &o24
     {name : &o41 "Gluschko",
      address : &o34 {street : &105 "Tyuratam",
                      appartment : &o623 "2C"
                      zip : &121 "07099"}
      audited : &o46 "10/12/63",
      taxamount : &o47 12332},
   taxpayer : &o21
     {name : &o132 "Kosberg",
      address : &o25 {street : &427 "Tyuratam",
                      number : &928 206,
                      zip : &121 "92443"}
      audited : &o46 "11/1/68",
      audited : &o46 "10/12/77",
      taxamount : &o283 0,
      taxevasion : &o632 "likely"}
   taxpayer : &o20
     {name : &o132 "Korolev",
      address : &o253 "Baikonur, Russia",
      audited : &o46 "10/12/86",
      taxamount : &o283 0,
      taxevasion : &o632 "likely"}
   company : &o26
      {name : &o623 "Rocket Propulsion Inc.",
       owner : &o24}
  }
```

(7) STORED (cont.)

- Model is an ordered version of the OEM model
- A complex object is an ordered set of (attribute, object) pairs
- An atomic object is an atomic value of type *int*, *string*, *video*, etc.

(7) STORED (cont.)



Taxpayer1

oid	name	street	no	apt	zip	audit1	audit2	taxamount	taxevasion
o24	Gluschko	Tyuratam		2C	07099	10/12/63		12332	
o21	Kosberg	Tyuratam	206		92443	11/1/68	10/12/77	0	likely

Taxpayer2

oid	name	address	audited	taxamount	taxevasion
o20	Korolev	Baikonur	10/12/86	0	likely

Company

name	owner
Rocket Propulsion Inc.	o24

Q: How to know there are two types of taxpayers, i.e. Taxpayer1 and Taxpayer2?

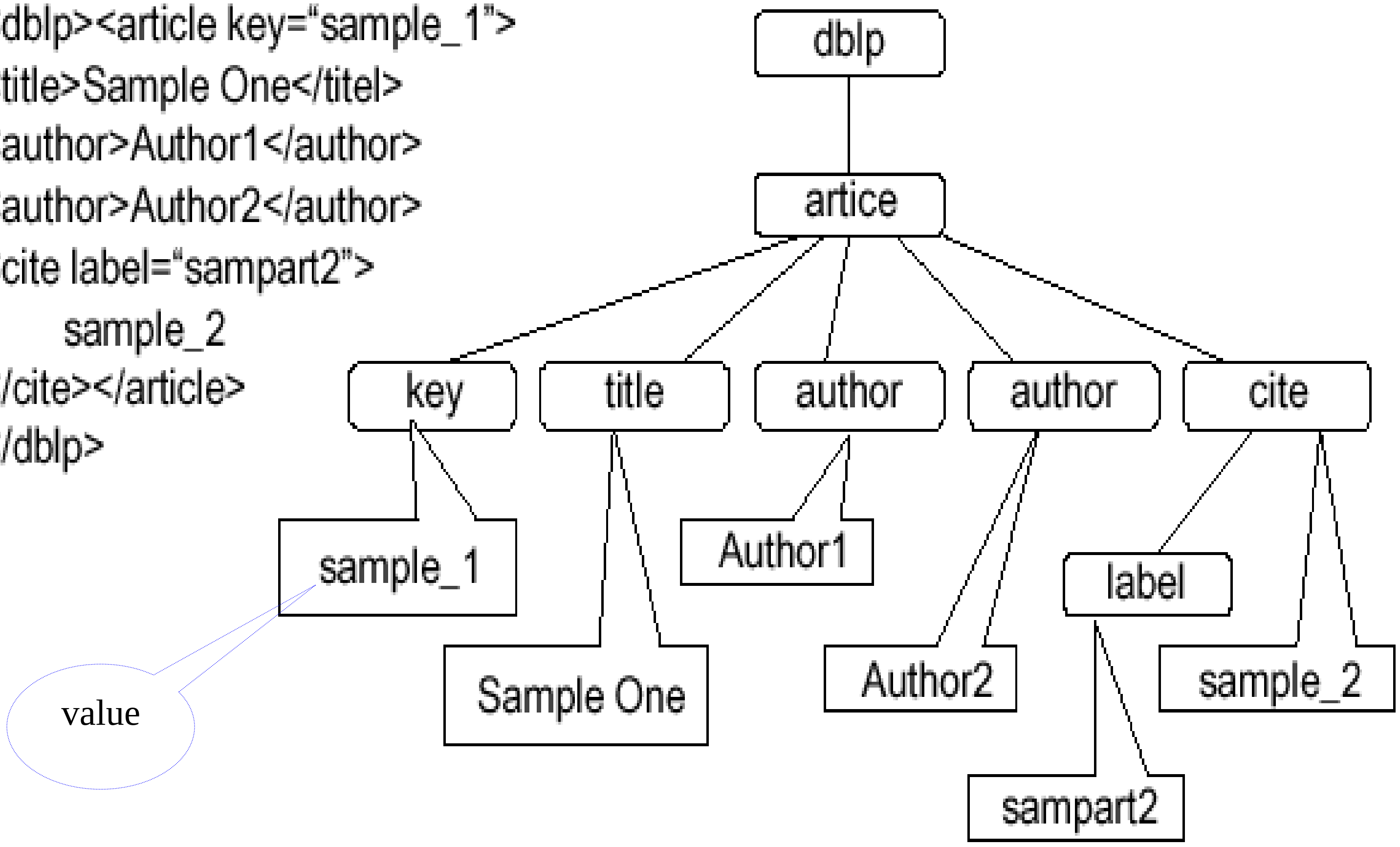
Q: How to find the address (which is a composite attribute) in Taxpayer1?

Q: How to find and store the node ID of the owner of the Company?

Q: In general, how many relations are needed for a given XML doc?

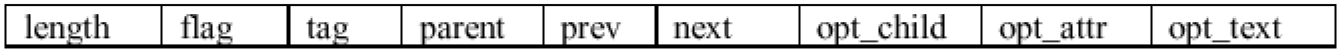
(8) Shore

```
<dblp><article key=“sample_1”>
<title>Sample One</titel>
<author>Author1</author>
<author>Author2</author>
<cite label=“sampart2”>
      sample_2
</cite></article>
</dblp>
```



(8) Shore (cont.)

- store each XML element of the XML file as a separate object
- The format of each *lw_object* (light weight object)

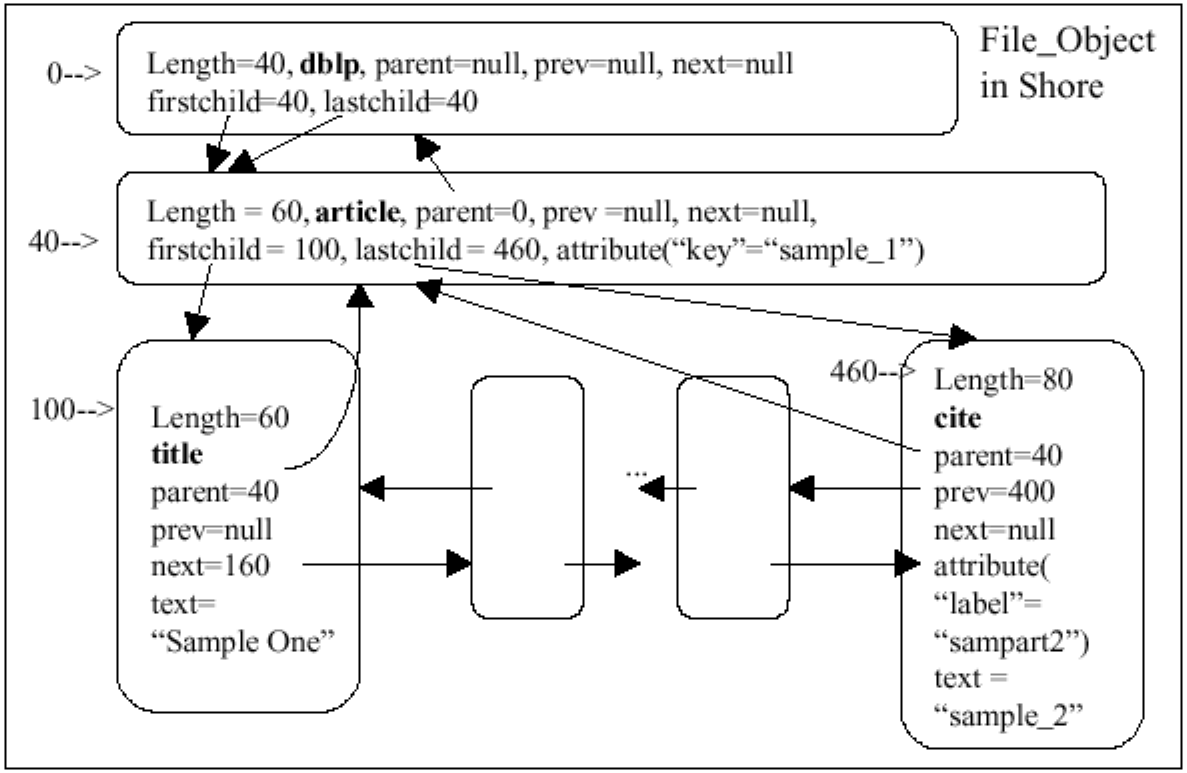


length	flag	tag	parent	prev	next	opt_child	opt_attr	opt_text

(8) Shore (cont.)

Each circular box corresponds to a *lw_object*

File_Object – the manager object corresponding to the XML document


File_Object in Shore
0-->
Length=40, **dblp**, parent=null, prev=null, next=null
firstchild=40, lastchild=40
40-->
Length = 60, **article**, parent=0, prev =null, next=null,
firstchild = 100, lastchild = 460, attribute("key"="sample_1")
100-->
Length=60
title
parent=40
prev=null
next=160
text=
"Sample One"
...
460-->
Length=80
cite
parent=40
prev=400
next=null
attribute(
"label"=
"sampart2")
text =
"sample_2"

Note: Bad! It uses length and offset as id.

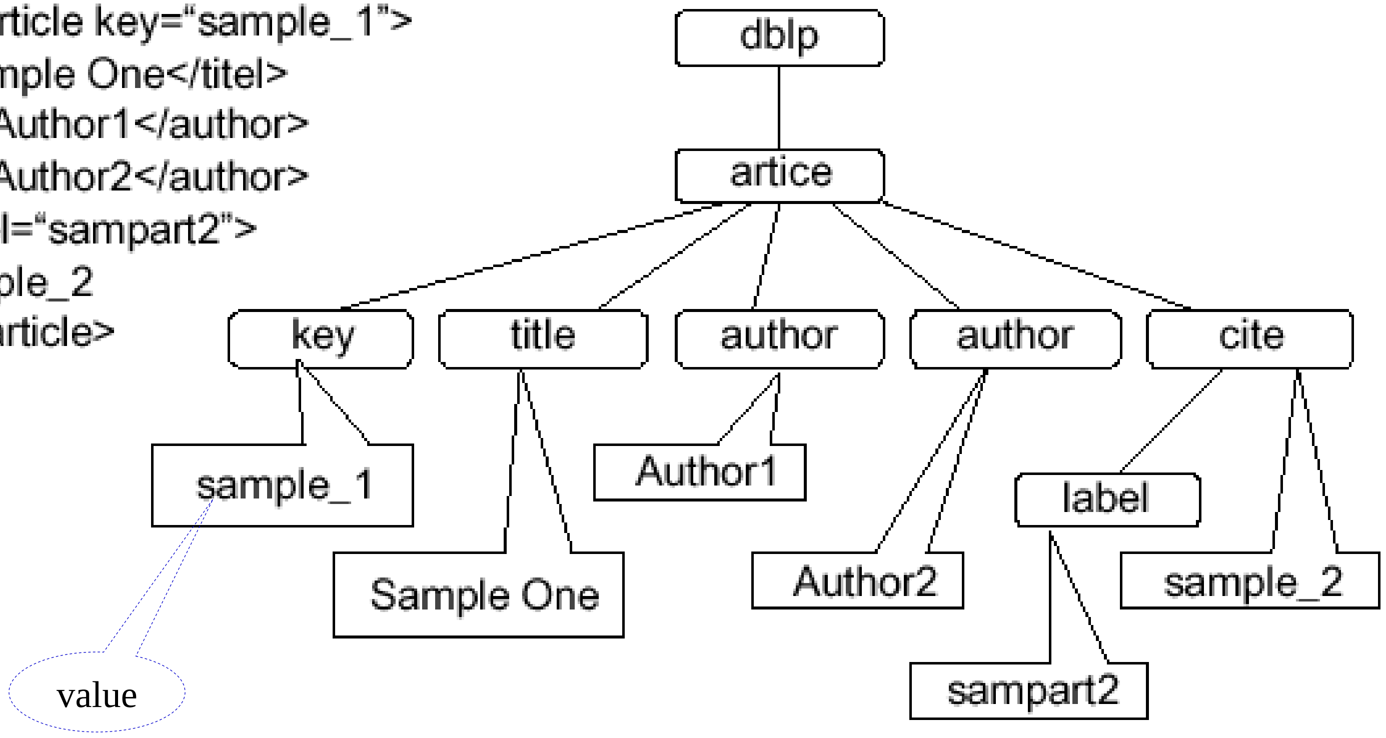
(8) Shore (cont.)

- drawback

  Since updated lw_objects that grow in size must be validated and then appended to the end of the file_object, the file_object tends to be fragmented, and space utilization deteriorates. Such updates also need to update linking information of other lw_objects.

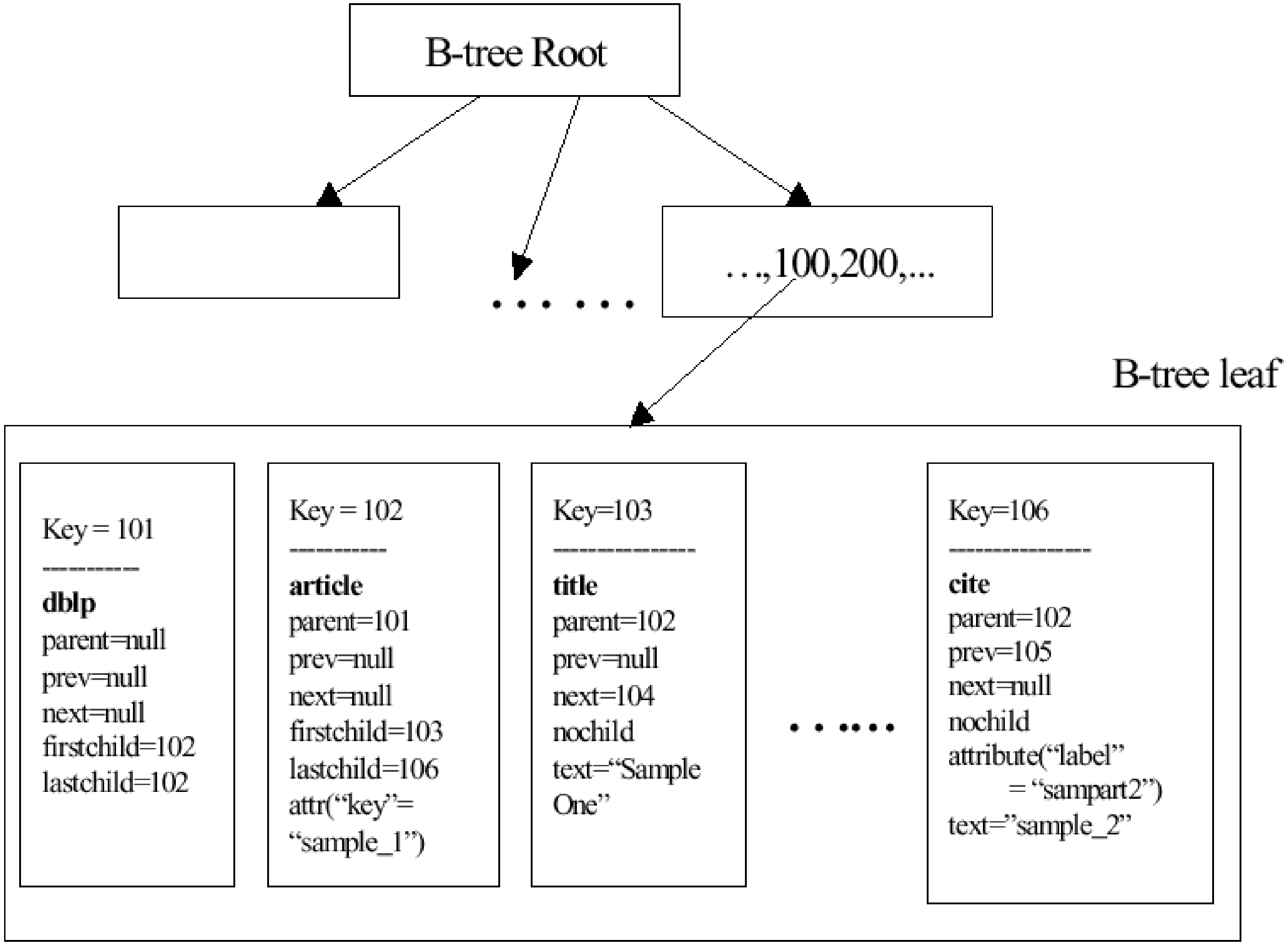
(9) B-tree Approach

```
<dblp><article key="sample_1">
<title>Sample One</titel>
<author>Author1</author>
<author>Author2</author>
<cite label="sampart2">
      sample_2
</cite></article>
</dblp>
```



(9) B-tree Approach (cont.)

This method uses B-tree to store the nodes of the XML tree, easier for XML data updates.


B-tree Root
…,100,200,...
B-tree leaf
Key = 101
dblp
parent=null
prev=null
next=null
firstchild=102
lastchild=102
Key = 102
article
parent=101
prev=null
next=null
firstchild=103
lastchild=106
attr("key"= "sample_1")
Key=103
title
parent=102
prev=null
next=104
nochild
text="Sample One"
Key=106
cite
parent=102
prev=105
next=null
nochild
attribute("label" = "sampart2")
text="sample_2"