Обработка и исполнение запросов в СУБД (Лекция 9)

XML СУБД

v6

Георгий Чернышев

Высшая Школа Экономики

chernishev@gmail.com

18 ноября 2020 г.

План лекции

- Модель данных XML;
- Языки запросов к XML;
- Три подхода к выполнения запросов к XML:
  1. Реляционные XML процессоры;
  2. Нативные XML процессоры;
  3. Смешанная схема.
- Индексирование для XPath;
- Выполнение XPath запросов на примере алгоритмов PathStack и TwigStack;
- Итог: настоящее XML и будущее древовидных моделей данных.

XML I

- XML — eXtensible Markup Language;
- Бесплатное упрощение ISO Standard General Markup Language (SGML); первая версия вышла в 1998, W3C;
- Идеи:
  1. добавим к тексту теги, для возможности придания семантики;
  2. дать возможность пользователю задавать свои теги;
  3. текст должен быть читабельным пользователем;
  4. возможность указания обработки;
- Пришел не один, а с “обвесом” других языков и технологий;
- Формирует базис многих технологий: ODF, XHTML, OOXML, DocBook.

XML II

Успешные приложения:

- Собственно, разметка документов;
- Обмен данными в слабо связанных системах:
  - DTD или XML Schema задает формат сообщений при общении различных компьютерных систем;
  - То есть, XML описывает формат, структуру и данные сообщений;
  - Примеры: SOAP, RSS/Atom.
- Моделирование слабо-структурированных данных:
  - Суть: менее строгая нежели реляционная, ER и OO модели;
  - Иерархическая, гибкая, позволяет разреженность в данных;
  - Позволяет иметь гетерогенную структуру данных;
  - Позволяет иметь свои свойства для каждого объекта;
  - Способна к быстрым изменениям.

XML III

- Элемент (открывающий и закрывающий тег + тело):

```
<t1> body </t1>
```

- Элементы могут быть вложенными:

```
<t1> body1 <t2> body2 </t2> </t1>
```

- Тег может иметь атрибуты:

```
<t1 attr1=``value1''> body </t1>
```

- Элементы могут быть пустыми:

```
<t1/>
```

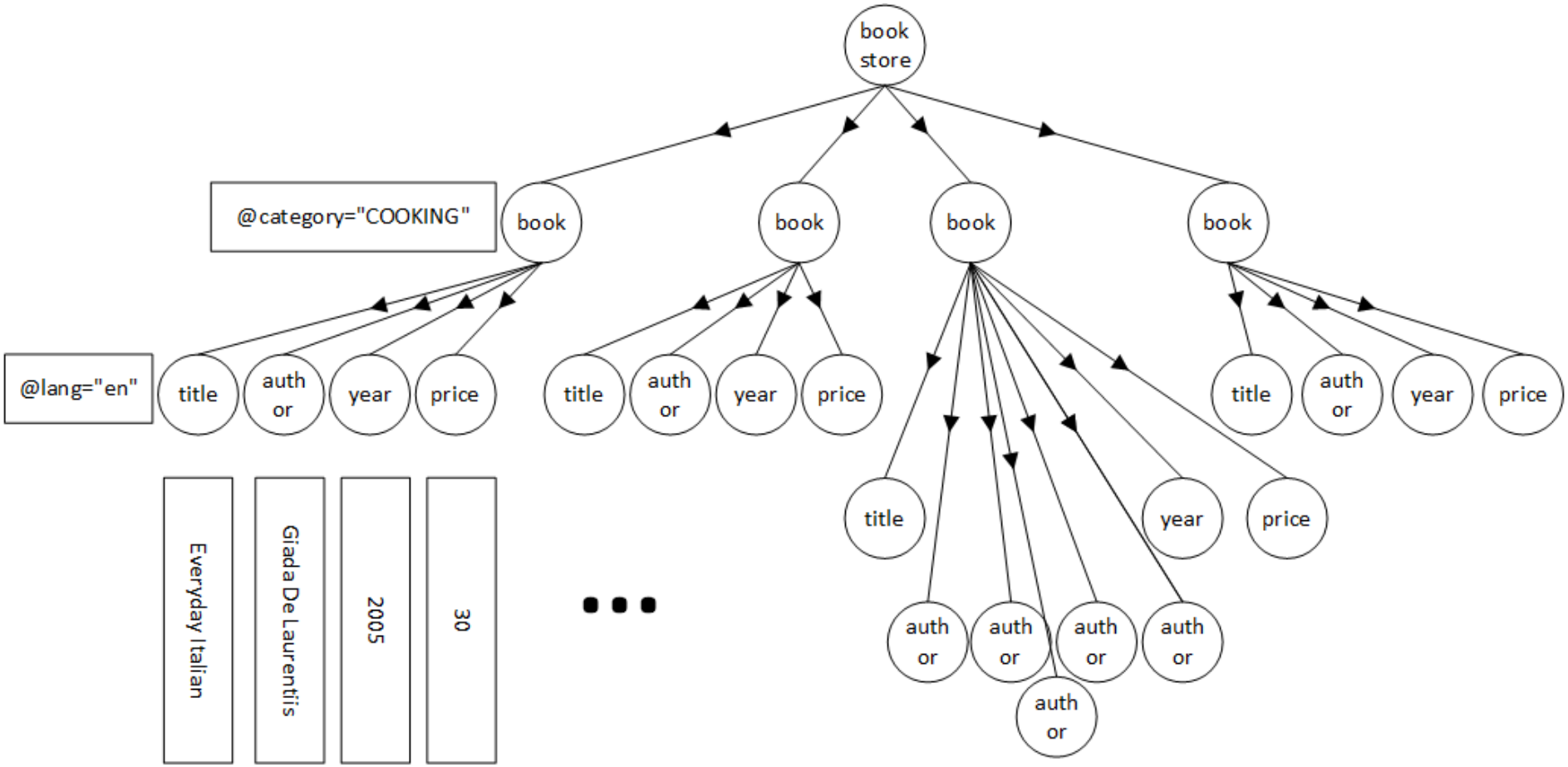
- DTD и XML schema ограничивают возможные деревья;
- В дереве есть порядок, левосторонний обход в глубину.

Пример I

Пример взят из
http://www.w3schools.com/xml/xquery_example.asp

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book category="COOKING">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="CHILDREN">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="WEB">
    <title lang="en">XQuery Kick Start</title>
    <author>James McGovern</author>
    <author>Per Bothner</author>
    <author>Kurt Cagle</author>
    <author>James Linn</author>
    <author>Vaidyanathan Nagarajan</author>
    <year>2003</year>
    <price>49.99</price>
  </book>
  <book category="WEB">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

Пример II


book store
@category="COOKING"
book
book
book
book
@lang="en"
title
auth or
year
price
title
auth or
year
price
title
auth or
year
price
Everyday Italian
Giada De Laurentiis
2005
30
title
year
price
auth or
auth or
auth or
auth or
auth or

Языки запросов к XML [Hidders and Paredaens, 2009]

- XPath (XML path language) — язык навигационных выражений
  - разрабатывался W3C, с 1999 года;
  - поиск шаблона в графе;
  - последовательность шагов для выборки узлов.
- XQuery (XML query language) — декларативный язык для запросов к коллекциям XML документов;
  - тоже W3C, 1998;
  - появился не на пустом месте: Quilt, Lorel, YATL;
  - подобен SQL, формула FLWOR;
  - включает в себя XPath.
- Множество других, нишевых: XUpdate, NEXI, XQuery Full-Text, ...

XPath

Последовательность location steps, формируем путь наподобие пути к каталогу в linux:

- Может быть абсолютным, а может быть относительным; Пример:

```
book/title/@lang
/bookstore/book/title/@lang
```

- * и @* для всех элементов и атрибутов;
- другие: //, .., |
- предикаты: 1) сравнение строк 2) существование 3) позиционные

```
//title[@lang=''en'']/..
//book[author=''J K. Rowling'' and price<30]
//book/author[2]
//book[/author]
//book[/author]/price
```

XPath: еще примеры

Запрос 1: doc("books.xml")/bookstore/book/title

```
<title lang="en">Everyday Italian</title>
<title lang="en">Harry Potter</title>
<title lang="en">XQuery Kick Start</title>
<title lang="en">Learning XML</title>
```

Запрос 2: doc("books.xml")/bookstore/book[price<30]

```
<book category="CHILDREN">
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
```

XQuery I

FLOWR нотация:

- FOR — выборка последовательности узлов;
- LET — привязка последовательности к переменной;
- WHERE — фильтрация узлов;
- ORDER BY — сортировка узлов;
- RETURN — что возвращать (вычисляется один раз для каждого узла).

Пример 1:

```
for $x in doc("books.xml")/bookstore/book
where $x/price>30
return $x/title
```

Результат:

```
<title lang="en">XQuery Kick Start</title>
<title lang="en">Learning XML</title>
```

XQuery II

Пример 2:

```
for $x in doc("books.xml")/bookstore/book
where $x/price>30
order by $x/title
return $x/title
```

Результат:

```
<title lang="en">Learning XML</title>
<title lang="en">XQuery Kick Start</title>
```

XPath аналог так не может:

```
doc("books.xml")/bookstore/book[price>30]/title
```

XQuery III

FLOWR нотация:

- В FOR и LET может быть несколько последовательностей;
- В RETURN может быть IF или CASE;
- В FLOWR узлы можно создавать на лету;
- ...

Пример 3:

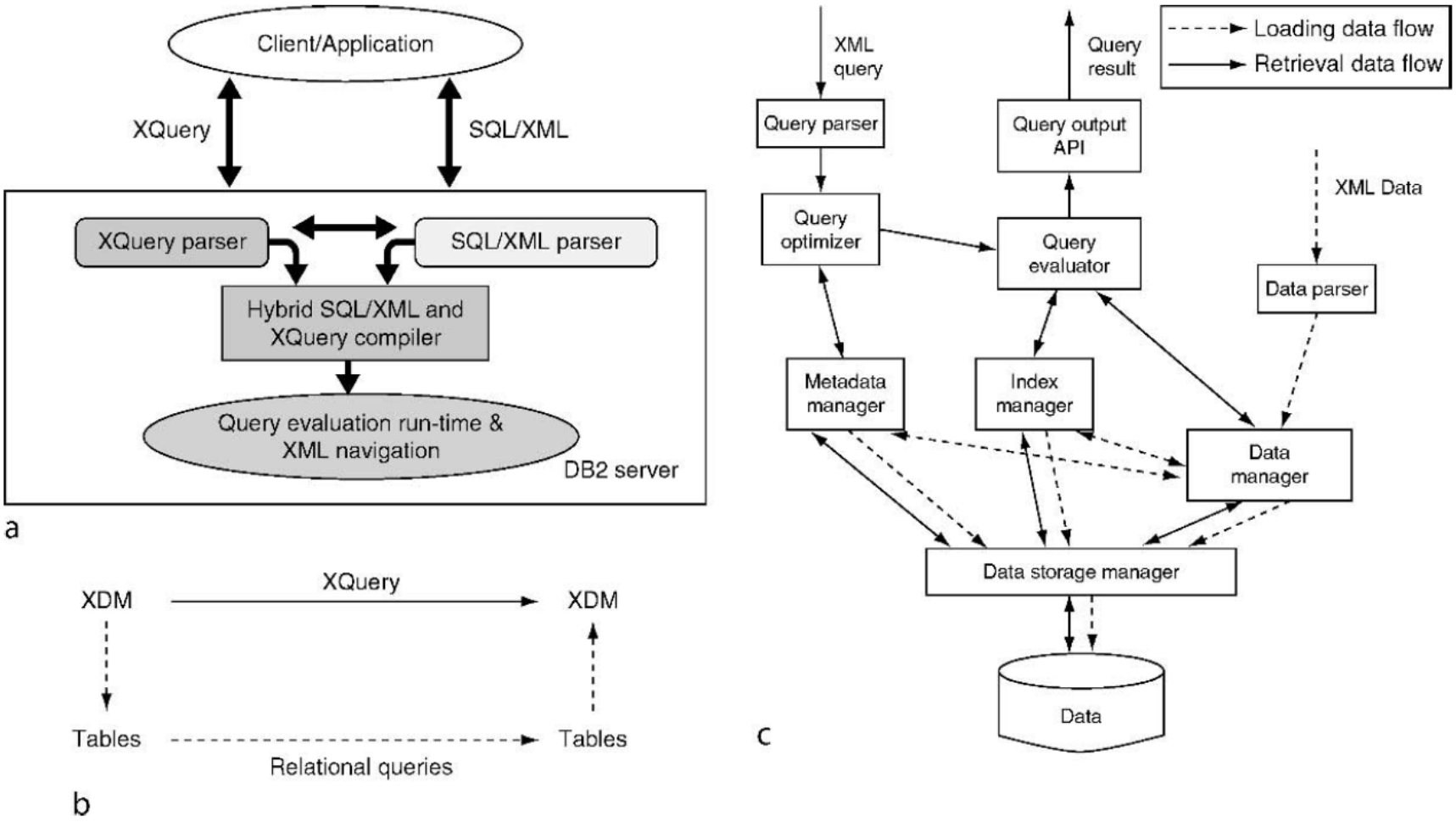
```
for $s in fn:doc(''students.xml'')//student,
    $e in fn:doc(''enrollments.xml'')//enrollment
let $cn := fn:doc(''courses.xml'')//course[@crs-code=$e/@crs-
    code]/name
where $s/@stud-id = $e/@stud-id
order by $cn
return <enroll> {$s/name, $cn} </enroll>
```

XQuery: выполнение [Grust et al., 2009]

Как можно строить XML СУБД:

1. Использовать реляционную СУБД: оттранслировать XML в таблицы, оттранслировать запрос в SQL, выполнить, оттранслировать результаты в XML и выдать обратно;
2. Выполнять нативно. Требуются специальные методы индексирования и выполнение запросов;
3. Гибридная схема, смесь реляционных подходов и XML технологий.

Примеры систем



XQuery Processors. Figure 1. (a) *DB2 XML* system architecture. (b) Pathfinder: purely relational XQuery on top of vanilla database back-ends. (c) Timber architecture overview [11]. [1]

[1] Изображение взято из [Grust et al., 2009]

Реляционные СУБД и XML

Pathfinder: чисто реляционное выполнение XQuery

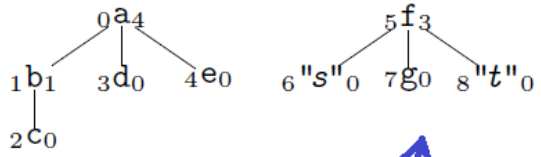
- Идея: существующие реляционные системы могут эффективно исполнять XQuery;
- В основе XQuery лежит понятие секвенции — последовательность состоящая из значений и деревьев. Для кодирования секвенций:
- Модель XDM (XQuery Data Model) — каждый узел это запись в таблице, возможно, с “хитрой” системой кодирований (pre/post region[2], ORDPATH) [Grust et al., 2009];
- Вычисление запроса: результаты FOR (для каждой переменной) выкладываем в таблицу, “разворачивая” их;
- Далее, на эту таблицу надстраиваем операторы специальной алгебры (Flat Relational Algebra).

[2] `http://db.in.tum.de/~grust/files/xquery-on-sql-hosts.pdf` материалы школы EDBT

Pathfinder: кодирование последовательностей

```
<a>
  <b><c/></b>
  <d/>
  <e/>
</a>
```

```
<f>
  s<g/>t
</f>
```



pre	*size*	*level*	*kind*	*prop*	*frag*
0	4	0	"elem"	"a"	0
1	1	1	"elem"	"b"	0
2	0	2	"elem"	"c"	0
3	0	1	"elem"	"d"	0
4	0	1	"elem"	"e"	0
5	3	0	"elem"	"f"	1
6	0	1	"text"	"s"	1
7	0	1	"elem"	"g"	1
8	0	1	"text"	"t"	1

(a) Two XML fragments. (b) Fragment trees. (c) Tree encoding (table doc).

Figure 1: Relational encoding of two XML fragments. Nodes in the fragment trees (b) have been annotated with their *pre* and *size* properties. Both trees are encoded as independent fragments 0 and 1 in (c).

pos	*pre*	*val*
1	NULL	"1.0"
2	NULL	"x"
3	0	NULL
4	5	NULL

Figure 2: Relational sequence encoding.

3

[3] Изображение взято из [Grust et al., 2004]

Pathfinder (выполнение FLOWR): пример I

Nested `for` blocks

s_0 `for $v0 in (10,20)`

s_1 `for $v1 in (100,200)`

s_2 `return $v0 + $v1`

$loop(s_0)$

iter
1

$loop(s_1)$

iter
1
2

$loop(s_2)$

iter
1
2
3
4

- Derive $\$v_0$, $\$v_1$ as before (uses row numbering operator ϱ):

$\$v_0$ in s_1:

iter	pos	item
1	1	10
2	1	20

$\$v_1$ in s_2:

iter	pos	item
1	1	100
2	1	200
3	1	100
4	1	200

4

[4] Изображение взято из http://db.in.tum.de/~grust/files/xquery-on-sql-hosts.pdf

Pathfinder (выполнение FLOWR): пример II

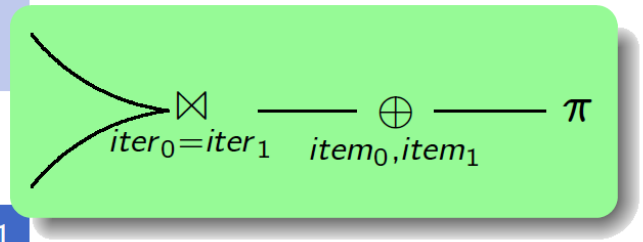
```
for $v0 in (10,20)
   for $v1 in (100,200)
      s2 [ return $v0 + $v1
```

v_0

$iter_0$	pos_0	$item_0$
1	1	10
2	1	10
3	1	20
4	1	20

v_1

$iter_1$	pos_1	$item_1$
1	1	100
2	1	200
3	1	100
4	1	200



=

iter	pos	item
1	1	110
2	1	210
3	1	120
4	1	220

[5]

Кому непонятно имеет смысл глянуть еще вот сюда: "Pathfinder: XQuery - The Relational Way".

[5] Изображение взято из http://db.in.tum.de/~grust/files/xquery-on-sql-hosts.pdf

DB2 XML: гибрид реляционной и XML СУБД

Идея: реляционные и XML данные могут успешно дополнять друг с друга, но XML данные потребуют свого хранилища и процессора. Реляционные компоненты остаются и переиспользуются. $\longrightarrow$ Можно использовать оба типа данных вместе (SQL + XQuery или SQL/XML).

- Хранит двоичное представление XML дерева, не надо парсить;
- Запросы не в SQL, а в свой Query Graph Model (QGM);
- Была перезапись запроса и стоимостная оптимизация для XQuery;
- Не сильно зависит от схемы, но использует для выбора индексов;
- Индексы для XPath выражений:
  - строится для выражения, умеет /, //, [ ];
  - представлен как два B^+-дерева — path, value index;
  - path index — все различные пути, в path id;
  - value index — содержит path id, значения, и node id — для каждого узла, удовлетворяющего индексируемому выражению;
  - так как индексы строятся на сложных XPath выражениях, предложили алгоритм проверки вложенности деревьев для выбора индекса;

Нативные XML СУБД [Grust et al., 2009]

- XML данные хранятся в их естественном виде, нет затрат на конвертацию в SQL и обратно;
- Примеры: Timber [Jagadish et al., 2002] и Sedna [Taranov et al., 2010]
- В Timber — своя алгебра для работы с XML: access methods, методы перезаписи запроса и стоимостная оптимизация;
- При загрузке документа узлу сопоставляется (D, S, E, L): Document, Start Key, End Key, Level (node);
- Эта схема позволяет быстро устанавливать отношения между узлами:
  - $< d_1, s_1, e_1, l_1 >$ предок $< d_2, s_2, e_2, l_2 > \Longleftrightarrow (s_1 < s_2) \,\&\, (e_1 > e_2)$
- Узлы хранятся в порядке Start key.

Timber

- Представление XQuery — алгебра TLC (Tree Logical Class), оперирует последовательностями упорядоченных, помеченных деревьев;
  - операторы: filter, select, project, join, ... , flatten, shadow/illuminate.
- TIX — алгебра-расширение для систем информационного поиска, добавили функции оценки;
- Новые физические операторы: материализация узла (по id узла получить поддерево), структурное соединение (вычисление XPath) — набор stack-based алгоритмов;
- Оптимизатор: 1) когда материализовывать, 2) последовательность структурных соединений — выбор порядка их проведения;
- Оптимизатор — стоимостной, оценка результата основывается на специализированной гистограмме для XML (positional histogram).

Sedna

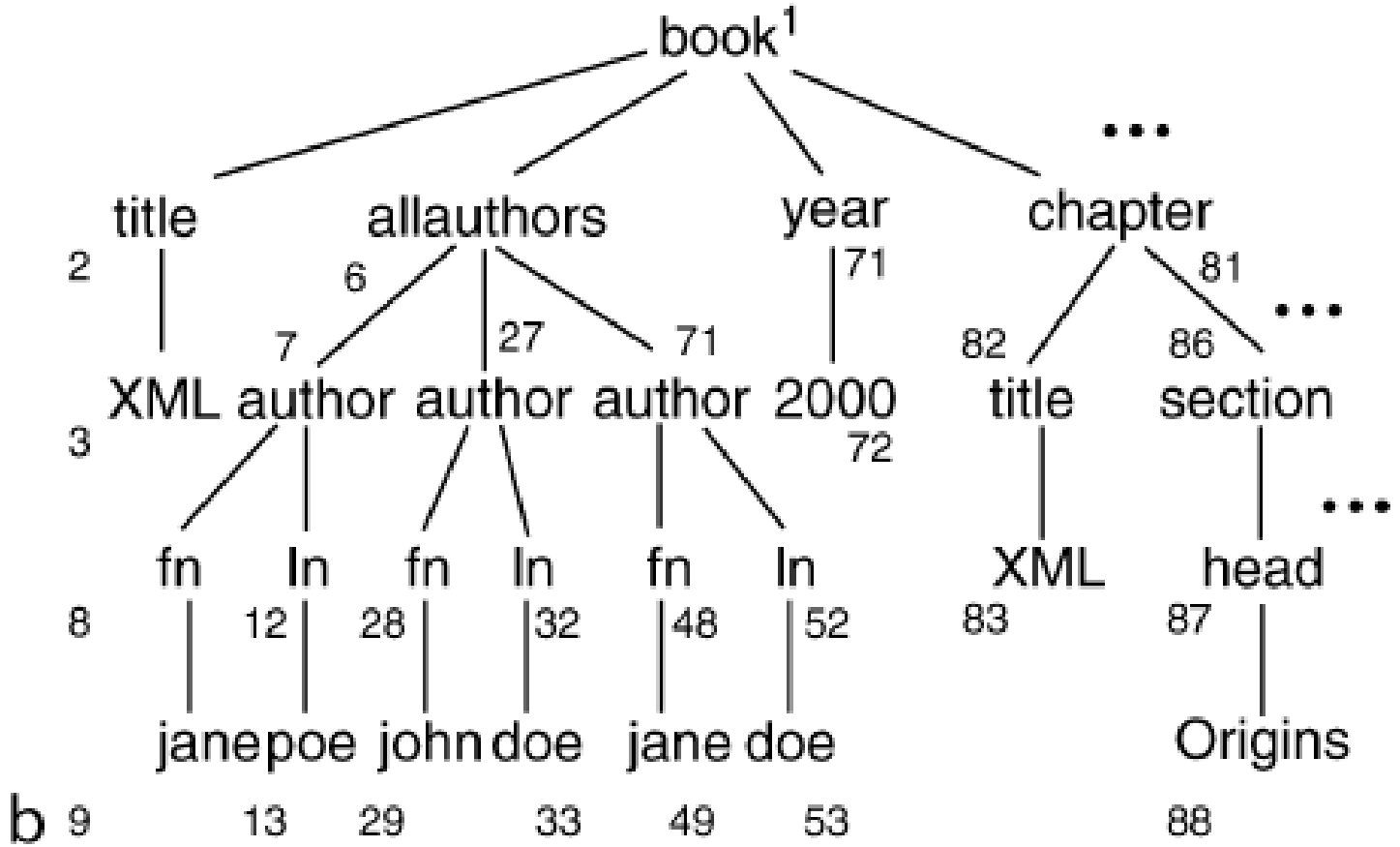
- Делал коллектив в ИСПРАН, руководитель — С.Д. Кузнецов (его переводы я отсылал вас читать на citforum).
- Активная разработка 2004–2008, 2006/7 – OpenSource.
- Централизованная, дисковая (делали pointer swizzling).
- Было:
  - 100% поддержка XQuery,
  - транзакции,
  - внедрения (Большая Российская Энциклопедия, для нее вкрутили полнотекстовый поиск).
- На мой взгляд это самый большой контрибьшен в исследовательское сообщество по базам данных, за всю историю страны.
- Их пытался купить Оракл (!).

Немного про индексирование XPath

По [Luna Dong and Divesh Srivastava, 2009]:

- Индексирование вершин (схемы нумерации вершин):
  - ID — в лоб: подобие инвертированного индекса, но это дорого :(
  - Dewey — аналог библиотечной классификации, проверка подстроки (делаем trie), плоха если дерево глубокое;
  - Интервальная схема — тройки из (первая нумерация, вторая нумерация, первая нумерация родителя) при левостороннем обходе;
    - проверять вложенность для потомка, для ребенка равенство;
    - потом на интервалах можно сделать R-tree;
- Индексирование путей: записать путь от одной вершины и до другой.

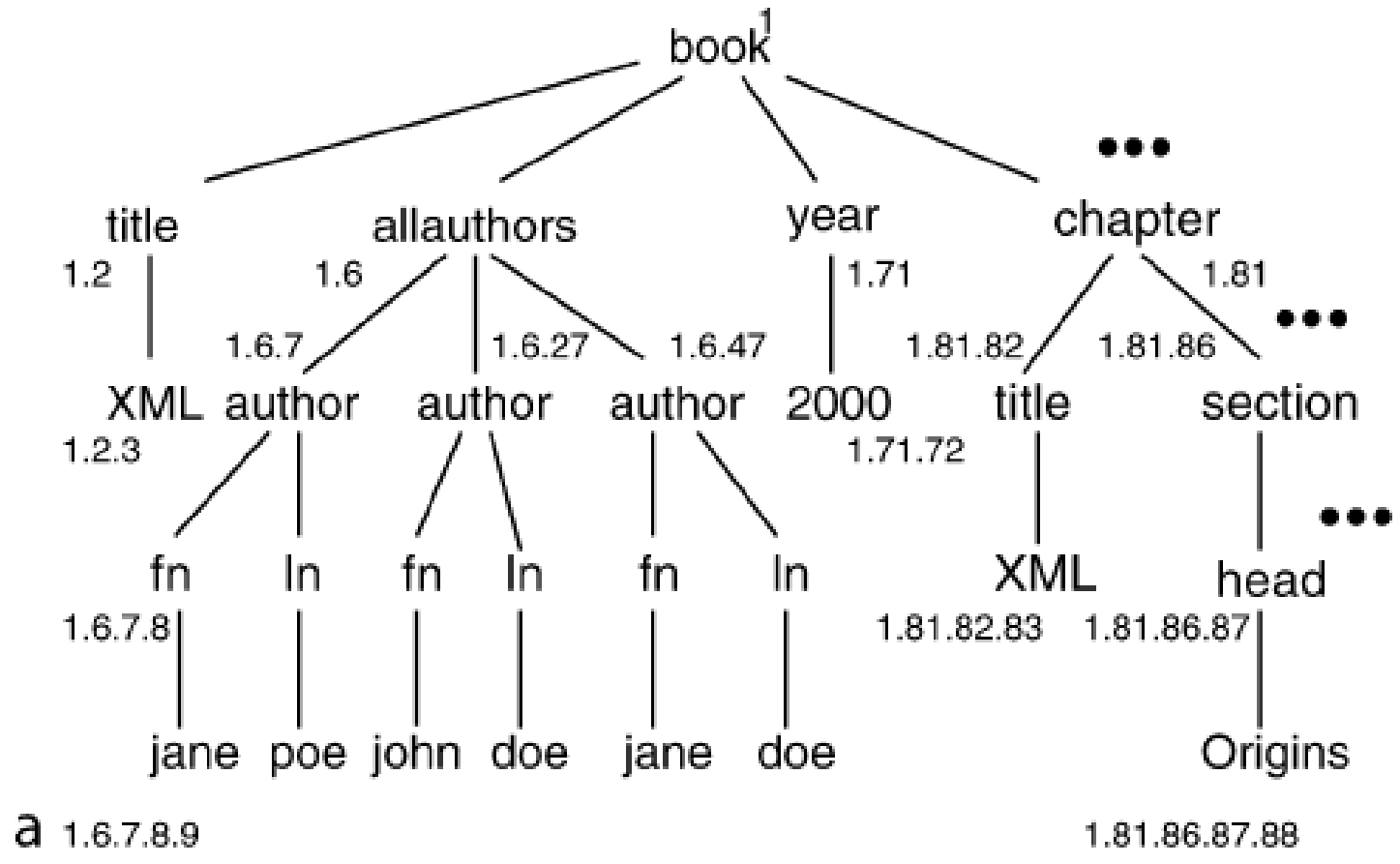
Исходные данные:



6

[6] Изображение взято из [Luna Dong and Divesh Srivastava, 2009]

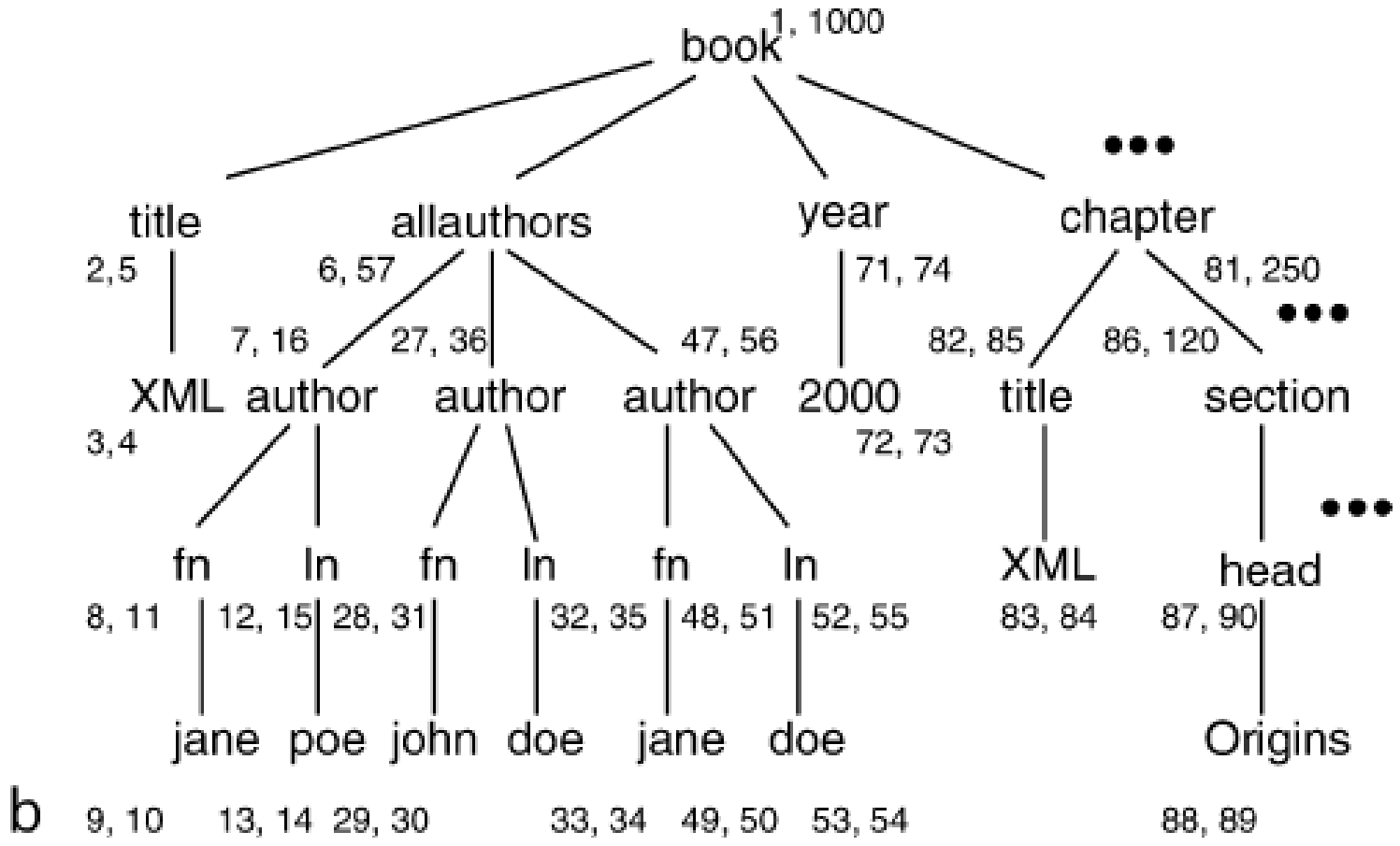
Схема Dewey



7

[7] Изображение взято из [Luna Dong and Divesh Srivastava, 2009]

Интервальная схема



[8]

[8] Изображение взято из [Luna Dong and Divesh Srivastava, 2009]

Индексирование путей

XML Indexing. Table 1. The 4-ary relation for path indexes

HeadId	SchemaPath	LeafValue	IdList
1	**B**	null	[]
1	**BT**	null	[2]
1	**BT**	XML	[2]
1	**BU**	null	[6]
1	**BUA**	null	[6,7]
1	**BUAF**	null	[6,7,8]
1	**BUAF**	jane	[6,7,8]
1	**BUAL**	null	[6,7,12]
1	**BUAL**	poe	[6,7,12]
	...		
6	**U**	null	[]
6	**UA**	null	[7]
6	**UAF**	null	[7,8]
6	**UAF**	jane	[7,8]
6	**UAL**	null	[7,12]
6	**UAL**	poe	[7,12]
	...		

9

[9] Изображение взято из [Luna Dong and Divesh Srivastava, 2009]

Схема методов для подхода индексирования путей

XML Indexing. Table 2. Members of family of path indexes

Index	Subset of `SchemaPath`	Sublist of `IdList`	Indexed Columns
Value [11]	paths of length 1	only last Id	`SchemaPath, LeafValue`
Forward link [11]	paths of length 1	only last Id	`HeadId, SchemaPath`
DataGuide [7]	root-to-leaf path prefixes	only last Id	`SchemaPath`
Index Fabric [6]	root-to-leaf paths	only first or last Id	`reverse SchemaPath, LeafValue`
ROOTPATHS [3]	root-to-leaf path prefixes	full IdList	`LeafValue, reverse SchemaPath,`
DATAPATHS [3]	all paths	full IdList	`LeafValue, HeadId, reverse SchemaPath`

10

[10] Изображение взято из [Luna Dong and Divesh Srivastava, 2009]

Вычисление XPath запросов

Пересказ части статьи “Holistic Twig Joins: Optimal XML Pattern Matching” [Bruno et al., 2002] про вычисление twig queries.

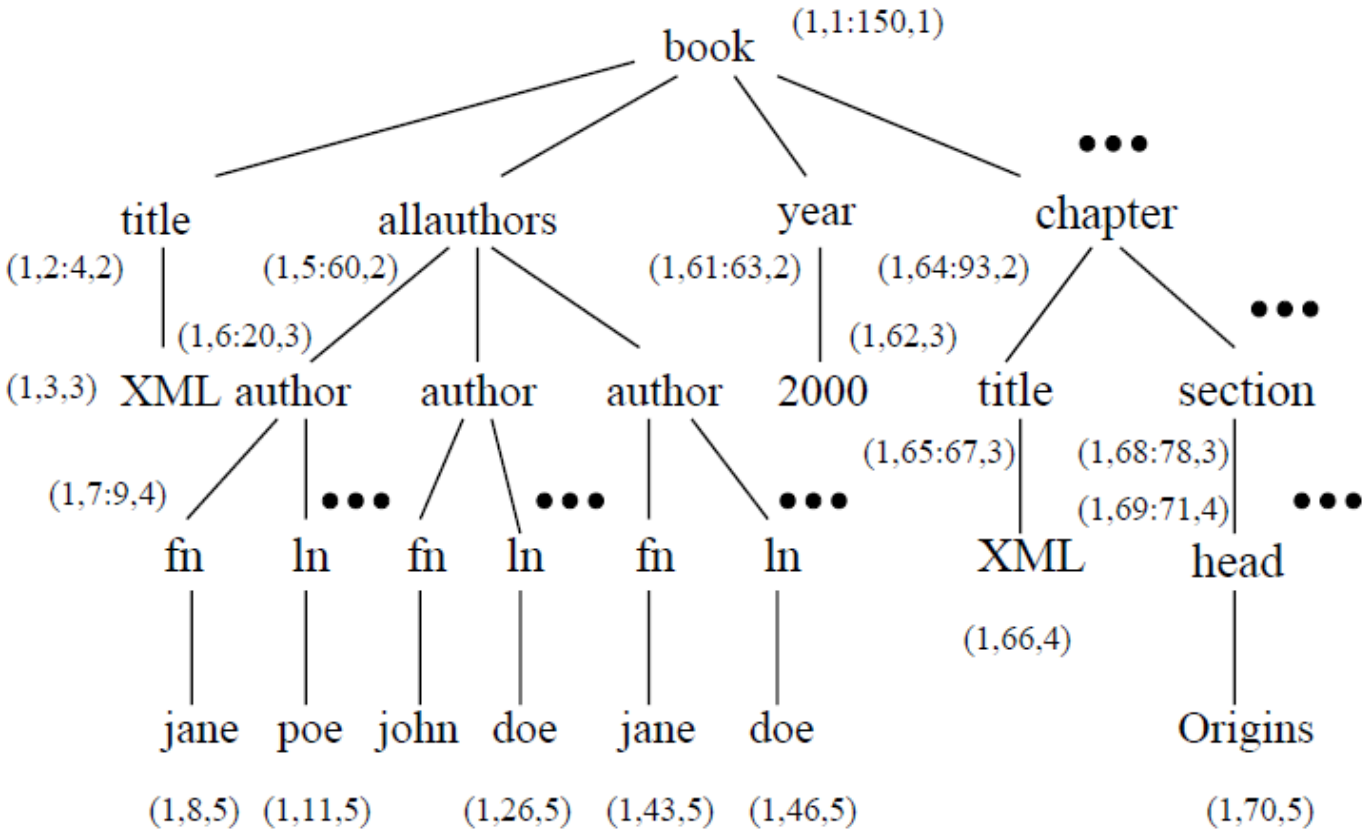
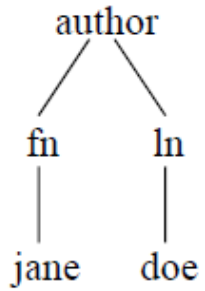


Figure 1: A sample XML Tree representation



A Sample XML Query

11

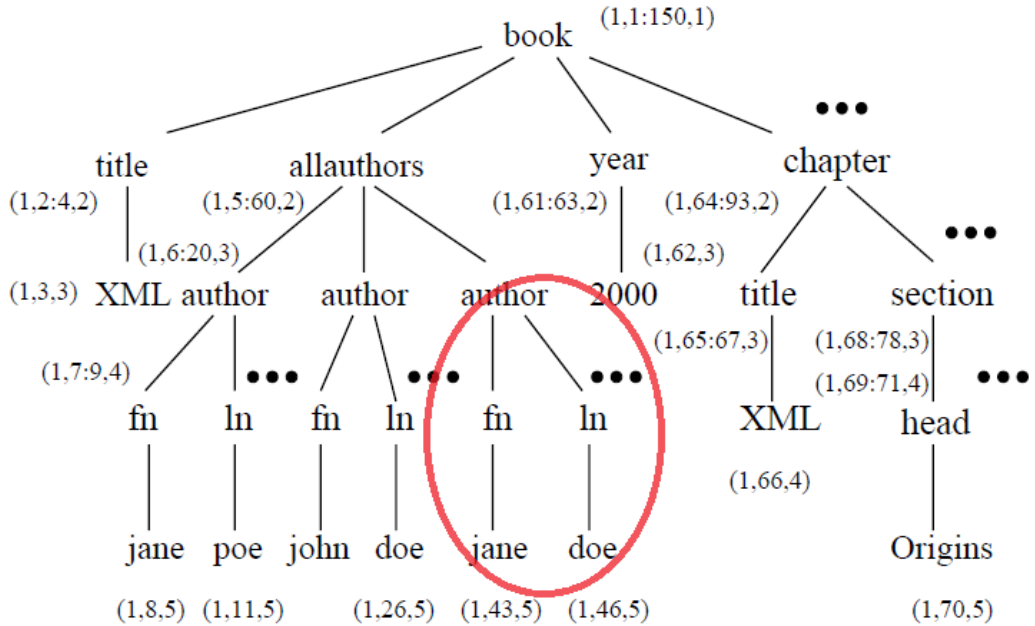
[11] Изображение взято из [Bruno et al., 2002]

Наивный подход

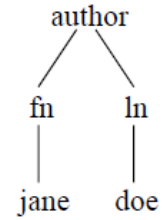
"В лоб":

1. Разбить запрос на бинарные отношения;
2. Найти такие отношения в базе данных;
3. Склеить результат.

плохо, ибо промежуточных результатов может быть ОЧЕНЬ МНОГО, при том что входных и результирующих данных небольшое количество.



author – fn	3
author – ln	3
fn – jane	2
ln – doe	2



A Sample XML Query

А результат -- один!

a

[a] Изображение взято из [Bruno et al., 2002]

Альтернатива: алгоритм PathStack

Идея:

1. Используем набор связанных друг с другом стеков;
2. Каждый стек содержит частичные результаты;
3. Полный результат получится “правильным” обходом стеков.

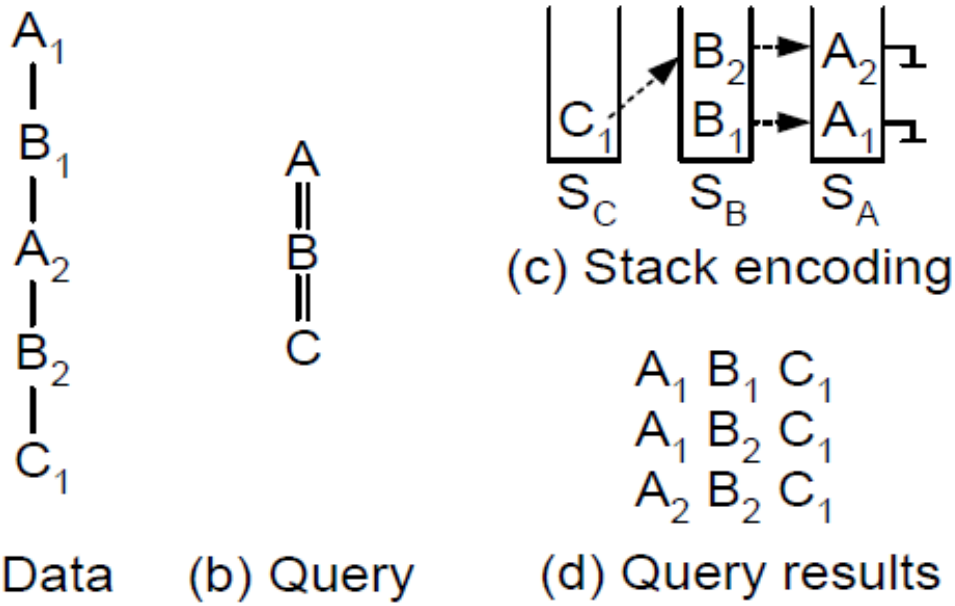
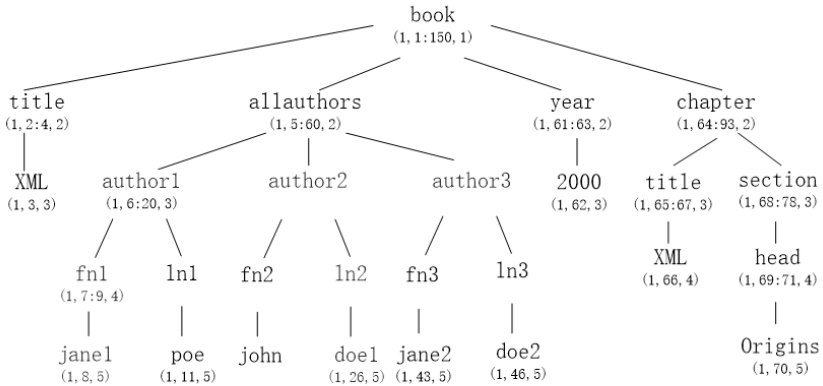


Figure 3: Compact encoding of answers using stacks. 12

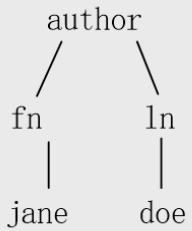
[12] Изображение взято из Bruno2002

Notation

XML document



Query



isLeaf (*author*) = false

isRoot (*author*) = true

parent (*fn*) = *author*

children (*author*) = {*fn*, *ln*}

subtreeNodes (*author*) = {*fn*, *ln*, *jane*, *doe*}

Streams

$\mathbf{T_a}$: a_1, a_2, a_3

$\mathbf{T_{fn}}$: fn_1, fn_3

$\mathbf{T_{ln}}$: ln_2, ln_3

$\mathbf{T_j}$: j_1, j_2

$\mathbf{T_d}$: d_1, d_2

eof (T_a) = false

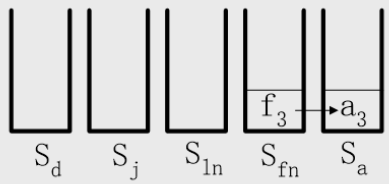
advance (T_a) => $\mathbf{T_a}$: a_1, a_2, a_3

next (T_a) = a_1

nextL (T_a) = 6

nextR (T_a) = 20

Stacks



empty (S_a) = false

pop (S_f)

push (S_{ln}, ln_3, *pointer to* a_3)

topL (S_a) = LeftPos of a_3

topR (S_a) = RightPos of a_3

13

[13] Изображение взято из http://www.cs.ucr.edu/~tsotras/cs236/W15/HolisticTwigJoin.ppt

Идея в общих чертах. В цикле, пока потоки не опустеют, брать узел с минимальным LeftPos и пушить в соответствующий стек. Как только получим лист — выдаем решения.

```
Algorithm PathStack(q)
01 while ¬end(q)
02   q_min = getMinSource(q)
03   for q_i in subtreeNodes(q) // clean stacks
04     while (¬empty(S_{q_i}) ∧ topR(S_{q_i}) < nextL(T_{q_min}))
05       pop(S_{q_i})
06   moveStreamToStack(T_{q_min}, S_{q_min}, pointer to
                                   top(S_{parent(q_min)}))
07   if (isLeaf(q_min))
08     showSolutions(S_{q_min}, 1)
09     pop(S_{q_min})

Function end(q)
   return ∀q_i ∈ subtreeNodes(q) : isLeaf(q_i) ⇒ eof(T_{q_i})

Function getMinSource(q)
   return q_i ∈ subtreeNodes(q) such that nextL(T_{q_i})
     is minimal

Procedure moveStreamToStack(T_q, S_q, p)
01 push(S_q, (next(T_q), p))
02 advance(T_q)
```

Figure 4: Algorithm PathStack

[a]

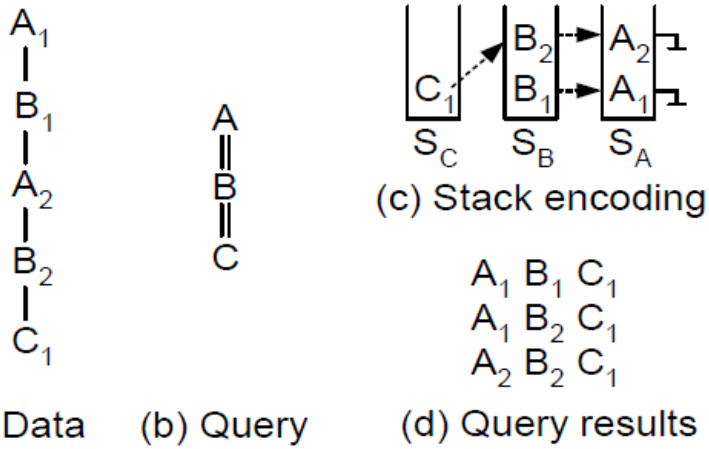
[a] Изображение взято из [Bruno et al., 2002]

```
Procedure showSolutions(SN, SP)
// Assume, for simplicity, that the stacks of the query
//   nodes from the root to the current leaf node we
//   are interested in can be accessed as S[1], ..., S[n].
// Also assume that we have a global array index[1..n]
//   of pointers to the stack elements.
// index[i] represents the position in the i'th stack that
//   we are interested in for the current solution, where
//   the bottom of each stack has position 1.

// Mark we are interested in position SP of stack SN.
01 index[SN] = SP
02 if (SN == 1) // we are in the root
03   // output solutions from the stacks
04   output (S[n].index[n], ..., S[1].index[1])
05 else // recursive call
06   for i = 1 to S[SN].index[SN].pointer_to_parent
07     showSolutions(SN - 1, i)
```

Figure 5: Procedure showSolutions

14



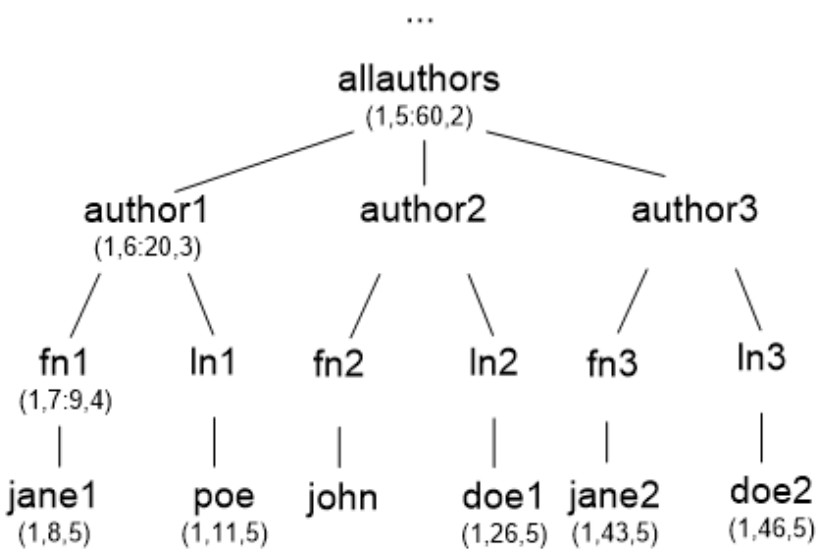
1) Стеки кодируют ответы
2) Выдача идет от листа к корню
3) Для P/C оси надо модифицировать алгоритм: а) рекурсию "укоротить" до 1 уровня и б) сверяться по LevelNum.

[14] Изображение взято из [Bruno et al., 2002]

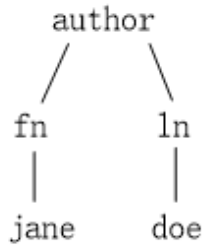
Анализ алгоритма PathStack

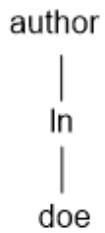
- Корректный: возвращает все значения, не возвращает ничего лишнего.
- В худшем случае PathStack имеет вычислительную сложность по I/O и CPU линейную от суммы размеров входных списков и выходного списка.
- Но! Лишние промежуточные результаты таки бывают :(

Пример:



Query:





Query result:

(a3, fn3, ln3, j2, d2)

(a1, fn1, j1)

(a3, fn3, j2)

(a2, ln2, d1)

(a3, ln3, d2)

15

Идея TwigStack

Идея: класть в стек только такие ноды, которые потом могут быть соединены с кем-то.

- Перед тем как взять ноду h_q из потока T_q и положить в стек S_q TwigStack используем getNext() чтобы проверить:
  1. что узел h_q имеет потомка h_{q_i} в каждом потоке T_{q_i} для каждого $q_i \in children(q)$;
  2. каждая из нод h_{q_i} рекурсивно удовлетворяет 1).

Анализ алгоритма TwigStack

- Корректный: возвращает все значения, не возвращает ничего лишнего.
- В худшем случае TwigStack имеет вычислительную сложность по I/O и CPU линейную от суммы размеров входных списков и выходного списка.
- В худшем сучае TwigStack имеет пространственную сложность вида минимум из 1) сумма размеров n входных списков и 2) n * максимальную длину пути от корня до листа в D.

Итог по XML

- Скорее мертв чем жив, отдельные СУБД на нем никого не интересуют, исследования завершены;
- Причины на 70% связаны со сложностью XQuery, люди не стали его учить $\longrightarrow$ он не пошел в массы, нет пользователей;
- Оставшиеся 30% это сложность вычисления запросов;
- Однако функционал включен во многие большие СУБД: есть и тип данных, и язык запросов поддерживается;
- Близится очередное пришествие языков для древовидных структур — для JSON, хотя там всё немного по-другому. Подробнее: JSONiq (кажется не взлетел), SQL++, PartiQL, n1ql.

Ссылки I

H. V. Jagadish, S. Al-Khalifa, A. Chapman, L. V. S. Lakshmanan, A. Nierman, S. Paparizos, J. M. Patel, D. Srivastava, N. Wiwatwattana, Y. Wu, and C. Yu. 2002. TIMBER: A native XML database. The VLDB Journal 11, 4 (December 2002), 274–291. DOI=http://dx.doi.org/10.1007/s00778-002-0081-x

Xin Luna Dong, Divesh Srivastava. XML Indexing. Encyclopedia of Database Systems. Springer US, 2009. 3585–3591. http://dx.doi.org/10.1007/978-0-387-39940-9_779

Torsten Grust, H. V. Jagadish, Fatma Ozcan, Cong Yu. XQuery Processors. Encyclopedia of Database Systems. Springer US, 2009. 3671–3675.

Torsten Grust, Sherif Sakr, and Jens Teubner. 2004. XQuery on SQL hosts. In Proceedings of the Thirtieth international conference on Very large data bases - Volume 30 (VLDB '04), Mario A. Nascimento, M. Tamer Özsu, Donald Kossmann, Renée J. Miller, José A. Blakeley, and K. Bernhard Schiefer (Eds.), Vol. 30. VLDB Endowment 252-263.

Ссылки II

Jan Hidders and Jan Paredaens. XPath/XQuery. Encyclopedia of Database Systems. Springer US, 2009. 3659–3665.http://dx.doi.org/10.1007/978-0-387-39940-9_774

Ilya Taranov, Ivan Shcheklein, Alexander Kalinin, Leonid Novak, Sergei Kuznetsov, Roman Pastukhov, Alexander Boldakov, Denis Turdakov, Konstantin Antipin, Andrey Fomichev, Peter Pleshachkov, Pavel Velikhov, Nikolai Zavaritski, Maxim Grinev, Maria Grineva, and Dmitry Lizorkin. 2010. Sedna: native XML database management system (internals overview). In Proceedings of the 2010 ACM SIGMOD International Conference on Management of data (SIGMOD '10). ACM, New York, NY, USA, 1037-1046. DOI=http://dx.doi.org/10.1145/1807167.1807282

Nicolas Bruno, Nick Koudas, and Divesh Srivastava. 2002. Holistic twig joins: optimal XML pattern matching. In Proceedings of the 2002 ACM SIGMOD international conference on Management of data (SIGMOD '02). ACM, New York, NY, USA, 310-321. DOI: https://doi.org/10.1145/564691.564727