XML Technologies
XML Databases: Storage

Christian Grün
Database & Information Systems Group
Universität Konstanz

XML Storage

Challenge: map logical data structures to bits and bytes

country	city	population	pop_year
f0_136	Tirane	192000	87
f0_136	Shkoder	62000	87
f0_136	Durres	60000	87
f0_136	Vlore	56000	87
f0_136	Elbasan	53000	87
f0_136	Korce	52000	87

Tabular Data

```
<mondial>
  <country name="Albania" population="3249136">
    <name>Albania</name>
    <city id="f0_1461" country="f0_136">
      <name>Tirane</name>
      <population year="87">192000</population>
    </city>
    <city id="f0_36498" country="f0_136">
      <name>Shkoder</name>
      <population year="87">62000</population>
      <located_at type="lake" water="f0_39058"/>
    </city>
    ...
```

XML Data

Tabular Storage

Example: relational table

ID	Name	Code	City	Country	Date
1	Yoshinari Matsuo	1088477	Tokyo	JP	2010-12-12
2	Giovanni Ciampa	80142	Napoli	IT	2010-11-03
3	Jean-Clause Risset	69007	Lyon	FR	2011-05-03
4	Aliona Siniovas	2600	Vilnius	LT	2009-03-21
5	Johannes Schmid	80327	Munich	DE	2009-02-21
6	Juan Movellan	18004	Granada	ES	2008-02-26

✎ Which *entities* can you spot that need to be stored?

Tabular Storage

ID	Name	Code	City	Country	Date
1	Yoshinari Matsuo	1088477	Tokyo	JP	2010-12-12

Contents

- (unordered) set of tuples (rows)

Meta data (mandatory)

- no. of rows and columns
- column names and types
- name of table

Meta data (optional)

- index structures
- min/max values, distribution
- flags: ordered, sparse, empty

✎ What *data types* could be used for storing the above data?

Tabular Storage

ID	Name	Code	City	Country	Date
1	Yoshinari Matsuo	1088477	Tokyo	JP	2010-12-12

PostgreSQL Data Types (extract)

http://www.postgresql.org/docs/current/static/datatype.html

Name	Description
`varchar[n]`	variable-length character string
`char[n]`	fixed-length character string
`date`	calendar date (year, month, day)
`int`	signed four-byte integer
`text`	variable-length character string

✎ How many *bytes* are required to store the above data?

Tabular Storage

Storage types: row store

1	Yoshinari Matsuo	1088477	Tokyo	JP	2010-12-12
2	Giovanni Ciampa	80142	Napoli	IT	2010-11-03
3	Jean-Clause Risset	69007	Lyon	FR	2011-05-03
4	Aliona Siniovas	2600	Vilnius	LT	2009-03-21
5	Johannes Schmid	80327	Munich	DE	2009-02-21
6	Juan Movellan	18004	Granada	ES	2008-02-26

Idea: contiguous storage of *row entries*

+ *well-established* RDBMS approach
+ adding or deleting records is a *local operation*

– *schema updates* are expensive (e.g. adding a new column)

Tabular Storage

Storage types: column store

1	Yoshinari Matsuo	1088477	Tokyo	JP	2010-12-12
2	Giovanni Ciampa	80142	Napoli	IT	2010-11-03
3	Jean-Clause Risset	69007	Lyon	FR	2011-05-03
4	Aliona Siniovas	2600	Vilnius	LT	2009-03-21
5	Johannes Schmid	80327	Munich	DE	2009-02-21
6	Juan Movellan	18004	Granada	ES	2008-02-26

Idea: contiguous storage of *column entries*

+ *increasingly popular* in NoSQL DBs
+ quick *column-based* operations (scanning, batch updates)

– single tuples are scattered over *multiple disk pages*

Tabular Storage

Column Store: Optimizations

Boncz & Kersten [1999]: MIL Primitives for Querying a Fragmented World

1. *fixed-size* values:
   - directly stored in the column file
2. *variable-sized* values:
   - value is stored in separate heap
   - byte offset is stored in the column file
3. *implicit* storage (`void` type):
   - densely ascending integers
   - need not be stored (*materialized*) at all

1	1088477	Tokyo	JP
2	80142	Napoli	IT
3	69007	Lyon	FR
4	2600	Vilnius	LT
5	80327	Munich	DE
6	18004	Granada	ES

✎ How many *bytes* are required to store the above data?

XML Storage

Example document

```
<addressbook>
 <!-- Addresses -->
 <?php echo "Addresses:"; ?>
 <address id="1" date="2010-12-12">
   <name>Yoshinari Matsuo</name>
   <code>1088477</code>
   <city>Tokyo</city>
   <country>JP</country>
 </address>
 <address id="2" date="2010-11-03">
   <name>Giovanni Ciampa</name>
   <code>80142</code>
   <city>Napoli</city>
   ...
```

- Once again: what *entities* can you spot that need to be stored?

XML Storage

Example document

```
<addressbook>
 <!-- Addresses -->
 <?php echo "Addresses:"; ?>
 <address id="1" date="2010-12-12">
   <name>Yoshinari Matsuo</name>
   <code>1088477</code>
   <city>Tokyo</city>
   <country>JP</country>
 </address>
 <address id="2" date="2010-11-03">
   <name>Giovanni Ciampa</name>
   <code>80142</code>
   <city>Napoli</city>
   ...
```

Contents

- element, attributes, PIs: *names*
- attributes, texts, comments, PIs: *values*
- elements, attributes: *namespaces*

☞ entities depend on *node kind*

Meta Data

- *name* of document
- *number* of XML nodes

XML Storage: Node Kinds

Element

- *name*, *namespace* declarations
- *descendants* (count, references)

```
<address> ... </address>
```

Attribute

- *key/value* pairs, *namespaces*

```
<address id="1">
```

Document

- *name* (document URI)

Text

- *value* (possibly typed)

```
<code>1088477</code>
```

Comment

- *value* (always untyped)

```
<!-- Addresses -->
```

Processing Instruction

- *name* (target), *content*

```
<?php echo "Addresses:"; ?>
```

XML Storage: Node Kinds

XML Data Model

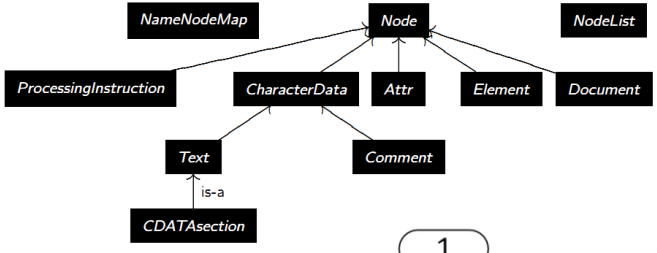
- Strings (names, values) can be stored in *indexes* or *heap files*
- Stored data needs to be *referenced*
- Number of references depends on *node kind*:

node kind	*parent*	*children*	*attr*	*target*	*content*	*value*	*uri*	*name*	*ns*
document		+					✓		
element	✓	+	+					✓	+
attribute	✓					✓		✓	
text	✓				✓				
proc.-instr.	✓			✓	✓				
comment	✓				✓				

✓: fixed size
+: variable size

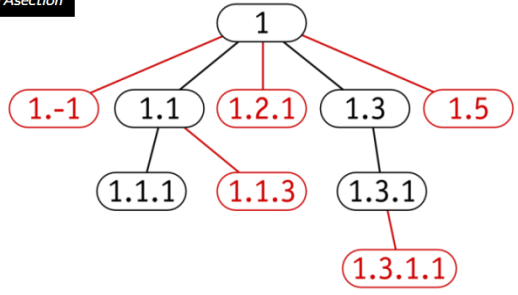
- Variable *size* means variable *number of references*

XML Storage: Encodings



DOM: Document Object Model

+ straightforward mapping to *node objects*
– lots of *random disk access* (PDOM, variants)

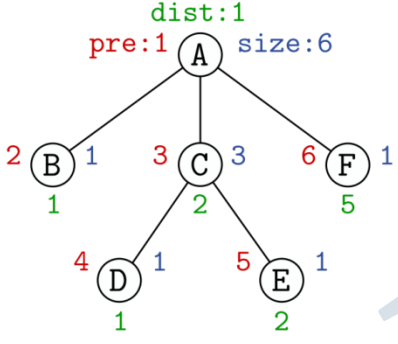


Hierarchic Labeling: ORDPATH

+ *update-friendly* node ids
– labels must be stored in *additional index structure*

Numbering Scheme: pre/dist/size

+ *fixed-size* encoding, implicit support of XPath axes



XML Storage

Example document

```
<addressbook>
 <!-- Addresses -->
 <?php echo "Addresses:"; ?>
 <address id="1" date="2010-12-12">
   <name>Yoshinari Matsuo</name>
   <code>1088477</code>
   <city>Tokyo</city>
   ...
```

Tabular storage

PRE	DIST	SIZE	ENTITY
0	-	127	addressbook.xml
1	1	126	<addressbook>
2	1	1	<!-- Addresses -->
3	2	1	<?php echo ...
4	3	11	<address>
5	1	1	id="1"
...			

Solution: document is mapped to a *flat, tabular* representation

- the pre/dist/size values of all nodes are stored in the table
- references to *children* and *attributes* get obsolete

XML Storage

node kind	*parent*	*children*	*attr*	*target*	*content*	*value*	*uri*	*name*	*ns*
document		+					✓		
element	✓	+	+					✓	+
attribute	✓					✓		✓	
text	✓				✓				
proc.-instr.	✓			✓	✓				
comment	✓				✓				

✓: fixed size
+: variable size

Pre/dist/size storage

PRE	DIST	SIZE	KIND	TARGET	CONTENT	VALUE	NAME
0	-	127	document			addressbook.xml	
1	1	126	element				addressbook
2	1	1	comment		Addresses		
3	2	1	proc-instr	php	echo "Addresses:";		
4	3	11	element				address
5	1	1	attribute			1	id
6	2	1	attribute			2010-12-12	date
7	3	2	element				name
8	1	1	text			Yoshinari Matsuo	
...							

- due to simplicity, some properties (such as *namespaces*) are excluded

✎ Table has many unused fields... Which columns could be *merged*?

XML Storage

node kind	*parent*	*children*	*attr*	*target*	*content*	*value*	*uri*	*name*	*ns*
document		+					✓		
element	✓	+	+					✓	+
attribute	✓					✓		✓	
text	✓				✓				
proc.-instr.	✓			✓	✓				
comment	✓				✓				

✓: fixed size
+: variable size

Pre/dist/size storage

PRE	DIST	SIZE	KIND	TEXT	NAME
0	-	127	document	addressbook.xml	
1	1	126	element		addressbook
2	1	1	comment	Addresses	
3	2	1	proc-instr	echo "Addresses:";	php
4	3	11	element		address
5	1	1	attribute	1	id
6	2	1	attribute	2010-12-12	date
7	3	2	element		name
8	1	1	text	Yoshinari Matsuo	
...					

- *element/attribute names* and *pi targets* have similar semantics
- *contents* and *values* can be aggregated as well (no overlaps)

XML Storage: Address Ranges

Motivation

- next, we need to assign *data types* to all columns
- data will occupy *variable amount* of size

☞ the *less bits* we need for representing a single node,
the better our system will perform

- once again, required size differs, depending on *node kind*

Node Kind

- *6 node kinds* exist

☞ node kind can be represented in *3 bits* (provides 2^3 choices)

XML Storage: Address Ranges

Pre/dist/size storage

PRE	DIST	SIZE	KIND	TEXT	NAME
0	-	127	doc	addressbook.xml	
1	1	126	elem		addressbook
2	1	1	comm	Addresses	
3	2	1	pi	echo "Addresses:";	php
4	3	11	elem		address
5	1	1	attr	1	id
6	2	1	attr	2010-12-12	date
7	3	2	elem	1	name
8	1	1	text	Yoshinari Matsuo	
...					

- What shall we do... with *variable-sized data* (texts and names)?

```
<addressbook>
 <!-- Addresses -->
 <?php echo "Addresses:"; ?>
 <address id="1" date="2010-12-12">
   <name>Yoshinari Matsuo</name>
   <code>1088477</code>
   <city>Tokyo</city>
   <country>JP</country>
 </address>
 <address id="2" date="2010-11-03">
   <name>Giovanni Ciampa</name>
   <code>80142</code>
   <city>Napoli</city>
   <country>IT</country>
 </address>
 <address id="3" date="2011-05-03">
   <name>Jean-Claude Risset</name>
   <code>69007</code>
   <city>Lyon</city>
   <country>FR</country>
 </address>
   ...
```

XML Storage: Address Ranges

Pre/dist/size storage

PRE	DIST	SIZE	KIND	TEXT	NAME
0	-	127	doc	0	
1	1	126	elem		1
2	1	1	comm	16	
3	2	1	pi	26	2
4	3	11	elem		3
5	1	1	attr	0	1
6	2	1	attr	2	2
...					

Off	Texts/Comments/PI
0	addressbook.xml
16	Addresses
26	echo "Addresses:";
44	Yoshinari Matsuo

Off	Attr.Values
0	1
2	2010-12-12
13	1

ID	Element/PI Names
1	addressbook
2	address
3	php
4	name
5	code

ID	Attr.Names
1	id
2	date

- names are indexed and replaced by their *numeric references*
- texts are replaced by *file offsets* (What about *indexing them* as well?)

XML Storage: Address Ranges

Observations

- *attributes*, *texts*, *processing instructions* and *comments* have no children
- *documents* have no parents
- number of distinct *element/ attribute/pi* names is limited
- distance to parent of *attributes* is comparatively small

☞ bytes/short values are sufficient to store *small values*

☞ constant values need *not be stored* at all!

node kind	*dist*	*size*	*text*	*name*
document	c	+	+	
element	+	+		–
attribute	–	c	+	–
text	+	c	+	
proc.-instr.	+	c	+	–
comment	+	c	+	

+/–: large/small address space
c: constant value

XML Storage: Address Ranges

Large documents: required space

Instances	file size	#nodes	#eln	#atn	ltxt	latr
RuWikiHist	421 GiB	324,848,508	21	6	**411 GiB**	186 MiB
iProClass	36 GiB	**1,631,218,984**	245	4	14 GiB	102 MiB
Inex2009	31 GiB	1,336,110,639	**28,034**	451	9.3 GiB	6.0 GiB
InterPro	14 GiB	860,304.235	7	15	19 B	**6.2 GiB**
Wikicorpus	4.4 GiB	157,948,561	1,257	**2,687**	1.5 GiB	449 MiB

#nodes	#eln	#atn	ltxt	latr
29	5	3	**39**	28
31	8	2	34	27
31	**15**	9	34	33
30	3	4	5	**33**
28	11	**12**	31	29

- #nodes: number of *XML nodes* in the document
- #elm and #atn: number of *distinct element/attribute names*
- ltxt and latr: *byte length* of all texts and attribute values, resp.
- largest value is marked *bold*

☞ table on the right shows *number of bits* needed to store largest value

XML Storage: Address Ranges

Large documents: required space

#nodes	#eln	#atn	ltxt	latr
29	5	3	**39**	28
31	8	2	34	27
31	**15**	9	34	33
30	3	4	5	**33**
28	11	**12**	31	29

node kind	*dist*	*size*	*text*	*name*
document	c	+	+	
element	+	+		−
attribute	−	c	+	−
text	+	c	+	
proc.-instr.	+	c	+	−
comment	+	c	+	

Correlations between *document statistics* and *node property ranges*:

- #nodes affects pre, dist and size property
- #elm and #atn affect name property
- ltxt and latr affect text property

XML Storage: Address Ranges

Bit layout in BaseX (simplified)

#nodes	#eln	#atn	ltxt	latr
29	5	3	**39**	28
31	8	2	34	27
31	**15**	9	34	33
30	3	4	5	**33**
28	11	**12**	31	29

node kind	*kind*	*dist*	*size*	*text*
document	3	0	31	40
element	3	31	31	
attribute	3	5	0	40
text	3	31	0	40
proc.-instr.	3	31	0	40
comment	3	31	0	40

- pre value is *implicitly given* by the table position (row number)
- 31 bits are reserved for dist and size value (= positive signed integer)
- 5 bits are reserved for the dist value of attributes
- 40 bits are reserved for text offsets

XML Storage: Address Ranges

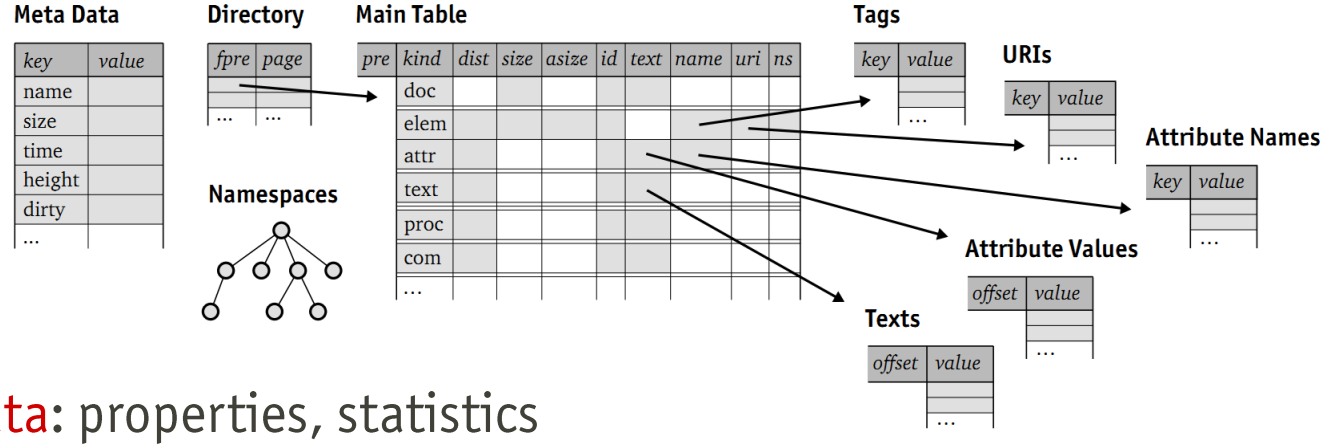
Bit layout in BaseX (simplified)

	0				32				64				96				128
document		k						text				size				id	
element	a	k	name			uri			dist			size				id	
attribute	d	k	name					text								id	
text		k						text				dist				id	
proc.-instr.		k						text				dist				id	
comment		k						text				dist				id	

- fixed number of bits per node: 128
- additional node properties:
  - id: persistent node id, which will not change after updates
  - a: number of attributes of an element; name, uri: names, namespaces

XML Storage

BaseX storage architecture (simplified)



- Meta Data: properties, statistics
- Main Table: tabular node representation
- Directory: references to disk pages (required for updates)

XML Storage: Updates

Example

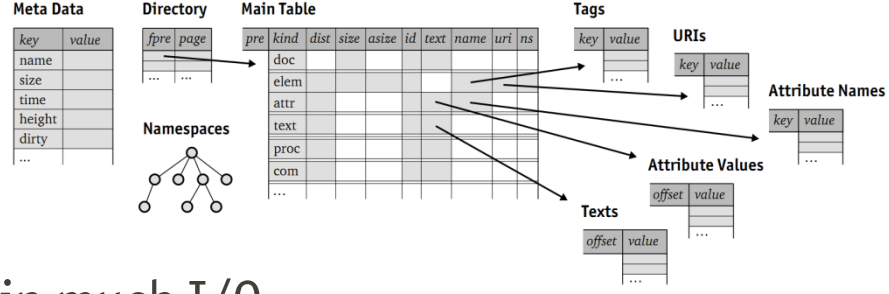
Delete node with *pre* = 5, *s* := *size(pre)*:

1. move all table entries in the range [*pre* + *s*, *#nodes*] by −*s*
2. subtract *s* from the *size* value of all *ancestors* nodes of *pre*
3. subtract *s* from the *dist* value of the *following siblings* of all *ancestor-or-self* nodes of *pre*

✎ Which step may be most expensive?

PRE	DIST	SIZE	ENTITY
0	-	127	addressbook.xml
1	1	126	<addressbook>
2	1	1	<!-- Addresses -->
3	2	1	<?php echo ...
4	3	11	<address>
5	1	1	id="1"
6	2	1	date="2010-12-12"
7	3	2	<name>
8	1	1	Yoshinari Matsuo
9	5	2	<code>
10	1	1	1088477
11	7	2	<city>
12	1	1	Tokyo
...			

XML Storage: Updates

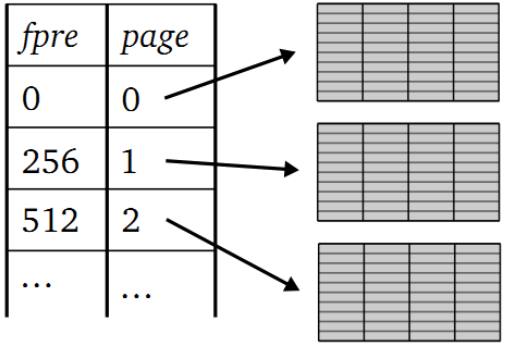


Problem

- copying table entries has linear complexity – O(#nodes) – and results in much I/O

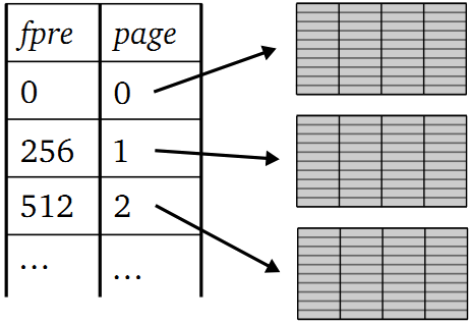
Solution: blockwise storage

- introduce *logical pages* (= blocks on disk)
- add *directory* with two columns:
  – *fpre*: pre value of the first entry of a page
  – *page*: pointers to pages
- within a page, nodes are *contiguously stored*

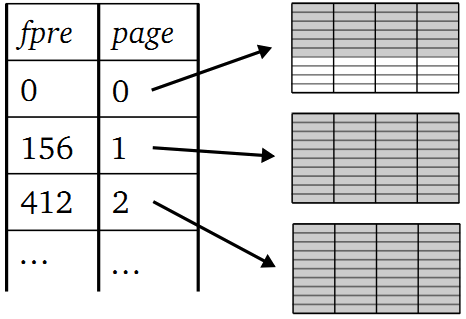


XML Storage: Updates

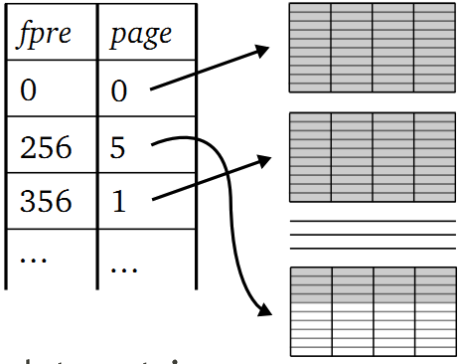
Examples



Initial state:
- page size: 4096 bytes
- 256 entries per page



Delete entries:
- *update* existing blocks
- *update* directory



Delete entries:
- ideally: *fill up* existing pages
- otherwise, *create* new pages
- *update* directory

✎ If 1000 nodes are inserted, how many blocks are written?

XML Storage: Optimizations

Compactification

- remember: bits can be *distributed differently*, depending on node kind

	0				32			64			96			128
document		*k*				*text*			*size*			*id*		
element	*a*	*k*	*name*		*uri*		*dist*		*size*			*id*		
attribute	*d*	*k*	*name*			*text*						*id*		
text		*k*				*text*			*dist*			*id*		
proc.-instr.		*k*				*text*			*dist*			*id*		
comment		*k*				*text*			*dist*			*id*		

Integer values

- integers can be *inlined*, i.e., stored instead of the file offset
- a single bit is used as *inlining flag*
- applicable to values without *spaces*, *decimals*, etc. (" 1.0 " != "1")

String values

- short texts can be *inlined* as well
- *compression algorithms* can be used to reduce text size
- additional bits are used as flags

✎ Ideas are always welcome…

XML Storage

Summary

- *native storage* allows for a wide range of customs optimizations
- *sequential* and *direct access* to nodes can be faster, and take less memory, than index-based access (à la PDOM or ORDPATH)
- looking at *real data instances* helps to define reasonable address limits
  Note, however, that limits will need to be surpassed one day
- examples: Y2K bug; switch from 8 to 16 to 32 to 64 to... 128? bits
- outlook: the presented byte representation of a row could also be stored in a *relational database* or any other data store!