Trends in DBMSs

5. XDBMS Architecture

Theo Härder
www.haerder.de

Increased flexibility of XML

XDBMS requirements

Labeling of XML tree nodes

Which node processing services are required?

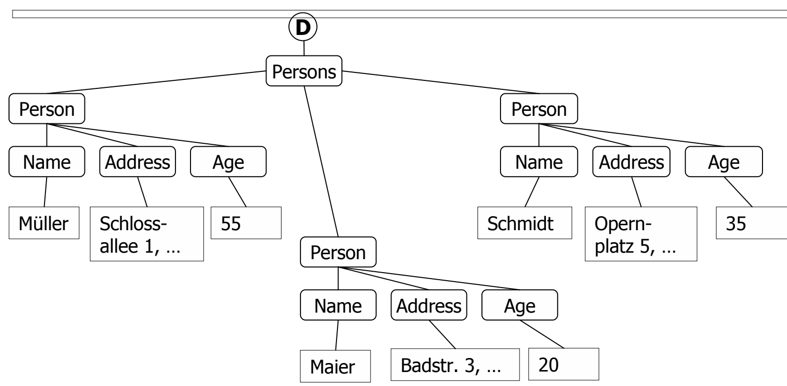
How to support fine-grained XML locking?

Main reference:
Haustein, M. P., Härder, T.: An Efficient Infrastructure for Native Transactional XML Processing, appears in: Data & Knowledge Engineering, 2006.

Technische Universität Kaiserslautern

Trends in DBMSs

Mapping: XML ←→ Relational Model



Person		
Name	Address	Age
Müller	Schlossallee 1, …	55
Maier	Badstraße 3, …	20
Schmidt	Opernplatz 5, …	35

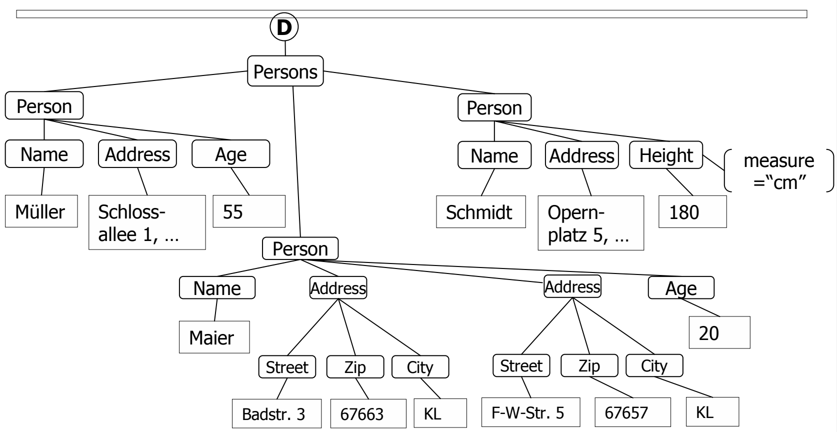
Perfect mapping in the RM only, if
- uniform objects (entities)
- exactly three-level description O/A/W
- atomic values
- no aspects (attributes of XML elements)
- order does not matter.

XML Documents are to be Stored in Databases

- Conceptual representation: trees with nodes and edges
- Document order must be preserved / recoverable: **node order matters!**
- **LOBs** don't enable fine-granular management, no content-based search and no multi-user operation
- **Mapping onto relational tables?**
  - many solutions: "shredding"
  - XML query language (e.g., XQuery, XPath, DOM, SAX) must be mapped to SQL
  - use of the SQL optimizer!
  - but: concurrency control (locking) very cumbersome, because a document is distributed over n tables

Trends in DBMSs
RM vs. XML
Architectural requirements
Node labeling
Indexing
Node processing services
Concurrency control
Lock manager
Performance measurement
DBIS Datenbanken und Informationssysteme
© 2005 AG DBIS

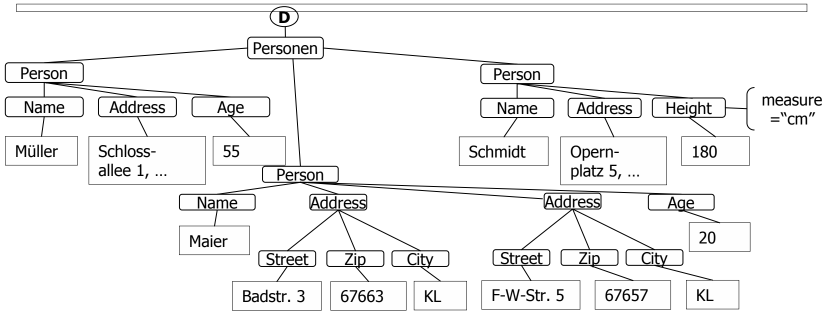
Mapping Flexibility: XML ←→ Relational Model



Person			
Name	Address	Age	Height
Müller	Schlossallee 1,..	55	--
Maier	?	20	--
Schmidt	Opernplatz 5, …	--	180

Mapping in RM in a single table fails, if object description
- has more than three levels (composed attributes),
- owns multi-valued or relation-valued attributes (n-level repeating groups)

Mapping Flexibility : XML ←→ Relational Model (2)



Person

No	Name	Address	AdNo	Age	Height
1	Müller	Schlossallee 1, …	--	55	--
2	Maier	- -	1	20	--
3	Schmitt	Opernplatz 5, …	--	--	180

Address

AdNo	No	Street	Zip	City
1	1	Bachstr. 3	67663	KL
1	2	F-W-Str. 5	67657	KL
…				

mapping in RM across several tables
- at least becomes very complex and hard to understand
- additionally must preserve order
- is also called "shredding"

Which Model Wins?

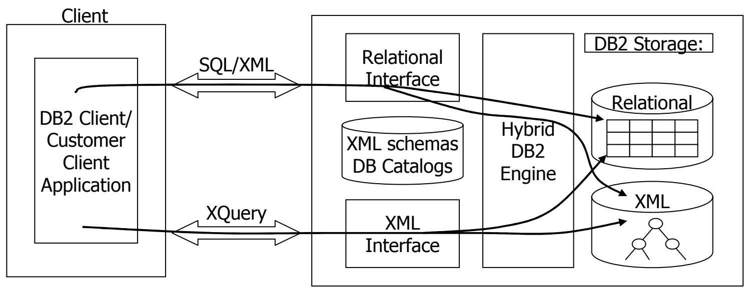
- Increasing development of distributed applications
  - via the Internet, too (e-Commerce, e-Business)
  - message formats are standardized by XML and predefined for application classes
  - but **messages are data** (to be stored in DBs), too!
  - permanent conversions: RM ←→ XML?

- Native XML processing
  - hierarchical structure of XML documents is preserved in the storage structure (on disk)
  - they can be faster processed in most cases than after a transformation into a relational schema
  - in this way, poor performance, poor scalability, and restricted flexibility can be avoided
  - access is directly performed via XML query languages

Which Model Wins? (2)

- Coexistence
  - native XML processing does not replace RM
  - both models will exist and complement each other!
- XML is already used intensively for communication and data description (Web Services, SOA), because it is
  - much more flexible and it simplifies data integration
  - self-describing, extensible, manufacturer- and platform independent

User 1 sees a sophisticated relational database which supports XML, too



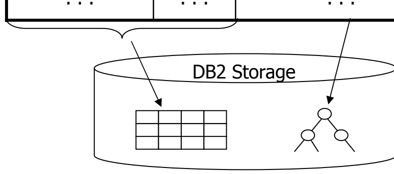
User 2 sees a sophisticated XML database which supports SQL, too

Example: DB2 Viper

- XML processing is implemented in all DB2 components
  - SQL queries are executed in the relational mode
  - XQuery expressions referring to the native XML store are directly translated and evaluated (without transformation to SQL)
  - hybrid queries are compiled and optimized in in a shared query execution plan (QEP)

create table dept (ID char(8), …, xmldoc xml)

ID	. . .	xmldoc
“V18”	. . .	<department> … <name> … </name> </department>
. . .	. . .	. . .



XQuery: FLWOR – Example

- Simple FLWOR expression (For, Let, Where, Order, Return)
  - find all persons (name, age) younger than 25

```
for    $p in //Person
let    $n := $p/Name, $a := $p/Age
where $a < 25
order by $n
return <YoungPerson> {$n, $a } </YoungPerson>
```

  - equivalent query without LET, WHERE:

```
for    $p in //Person[Age < 25]
order by $p/Name
return <YoungPerson> {$p/Name, $p/Age} </YoungPerson>
```

- Variables begin with "$"-prefix

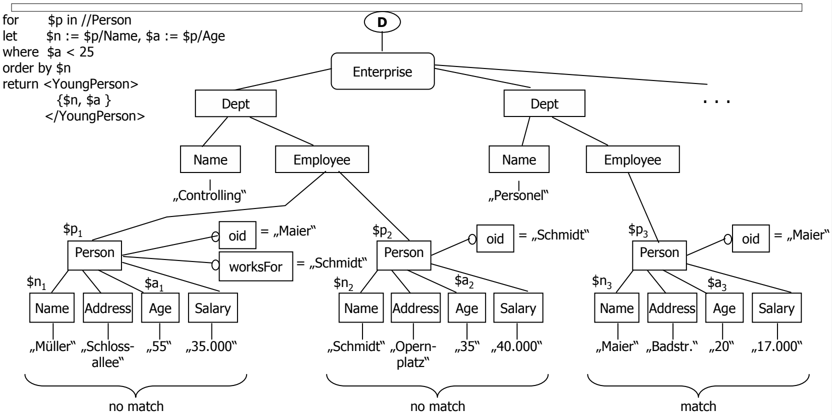
Trends in DBMSs

RM vs. XML
Architectural requirements
Node labeling
Indexing
Node processing services
Concurrency control
Lock manager
Performance measurement

DBIS Datenbanken und Informationssysteme
© 2005 AG DBIS

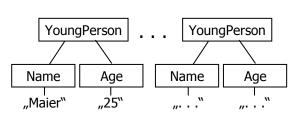
Evaluation Example – FLWOR Expression

```
for    $p in //Person
let    $n := $p/Name, $a := $p/Age
where  $a < 25
order by $n
return <YoungPerson>
         {$n, $a }
       </YoungPerson>
```



1) $p iterates over all "Person" elements and binds the children "Name" and "Age" to $n resp. $a.
2) it is checked in each iteration step, whether the "where" condition is true
3) matches are stored in a sequence and sorted according to $n (Name)
4) output is generated

output



Is the DBMS Layer Model Suitable for XML?

- Layer model
  - designed for declarative and set-oriented evaluation of records
  - layer architecture is comparable to a set of "abstract machines" cooperating in a "use hierarchy"
  - hierarchical layers and information hiding guaranteed long-term system evolution
  - "self-*" properties require much more information flow across system layers

Important observation:
The invariants in database management determine the mapping steps of the supporting architecture

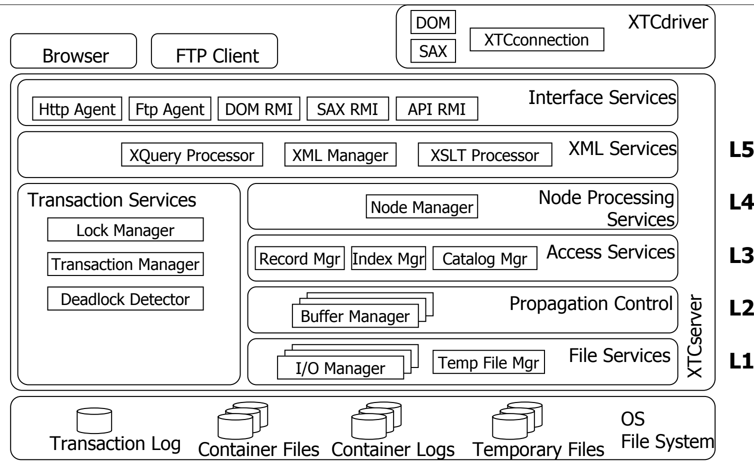
Reuse and adaptation of
- storage system
- access system
- data system?

Native XDBMSs

- Improved solution needed
  - fine-granular management and storage of the XML documents as native tree-like storage structures
  - navigational and direct access to all document nodes
  - indexing of nodes to accelerate both types of requests
  - modification of documents also required under multi-user operations (cooperative processing)
  - fine-granular locking: nodes, edges, and subtrees

- How to store and address tree nodes, which can be arbitrarily displaced by later insertions?
  - how do XML documents appear at the user level?
  - which storage structures are adequate?
  - which labeling scheme should be used for the nodes?

A Native XDBMS Architecture



- XTC – architectural overview
  - reuse of layer model is possible, but needs substantial adjustments and, in particular, new functionality in higher layers
  - variations: relational, hybrid, ROX

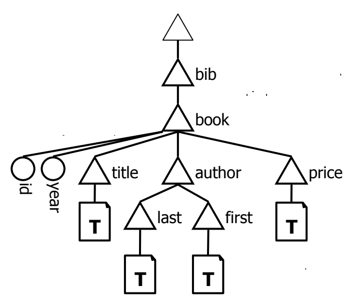
Trends in DBMSs
RM vs. XML
Architectural requirements
Node labeling
Indexing
Node processing services
Concurrency control
Lock manager
Performance measurement

Introduction to DOM (Document Object Model)

- XML fragment

```
<bib>
  <book year="1994" id="1">
    <title>TCP/IP Illustrated</title>
    <author>
      <last>Stevens</last>
      <first>W.</first>
    </author>
    <price>65.95</price>
  </book>
</bib>
```

- Representation as DOM tree

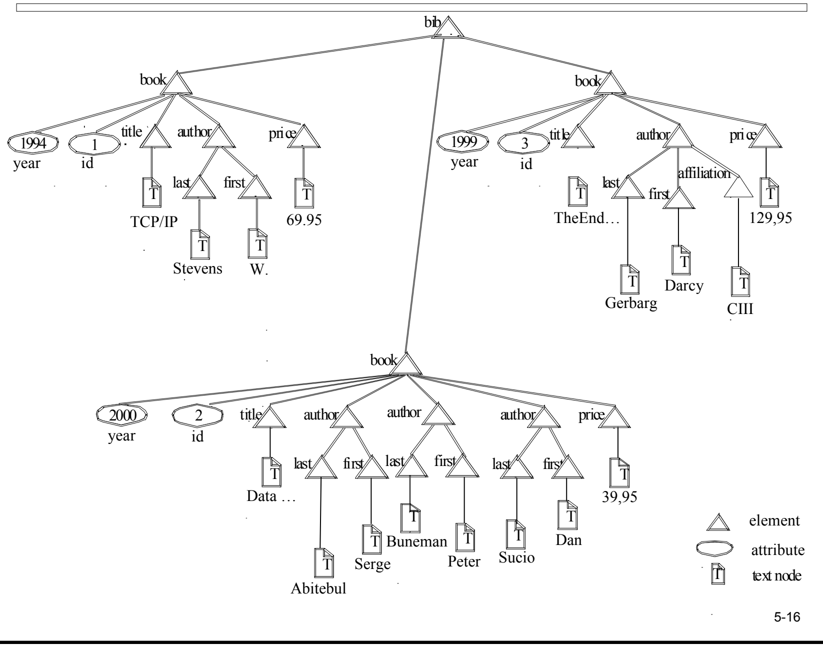


- DOM API
  - navigation
    - getFirstChild()
    - getLastChild()
    - getNextSibling()
    - getPreviousSibling()
    - getAttributes()
    - getNodeValue()
  - modification
    - appendChild (...)
    - insertBefore (...)
    - removeChild (...)
    - setNodeValue (...)
    - setAttribute (...)
  - query
    - getElementById (...)
    - getElementsByTagName (...)
    - hasAttribute (...)

Running Example of an XML Document

```
<bib>
        <book year="1994" id="1">
                <title>TCP/IP Illustrated</title>
                <author>
                        <last>Stevens</last>
                        <first>W.</first>
                </author>
                <price>65.95</price>
        </book>
        <book year="2000" id="2">
                <title>Data on the Web</title>
                <author>
                        <last>Abiteboul</last>
                        <first>Serge</first>
                </author>
                <author>
                        <last>Buneman</last>
                        <first>Peter</first>
                </author>
                <author>
                        <last>Suciu</last>
                        <first>Dan</first>
                </author>
                <price>39.95</price>
        </book>
        <book year="1999" id="3">
                <title>The Economics of . . . </title>
                <editor>
                        <last>Gerbarg</last>
                        <first>Darcy</first>
                        <affiliation>CITI</affiliation>
                </editor>
                <price>129.95</price>
        </book>
</bib>
```


Example of a DOM Tree: Conceptual Representation



Holistic Support of all Internal XDBMS Operations

- Node Labeling
  - representation of an XML document: ordered, labeled tree with nodes of type element, attribute, text
- Support of
  - declarative query processing
    - all core operations
    - indexing support
  - navigational processing
    - in combination with XML document representation and
    - additional access path structures
  - concurrency control
    - most operations jump into the document tree
    - intention locks up to the document root required

⇨ **without accessing the XML document on disk**

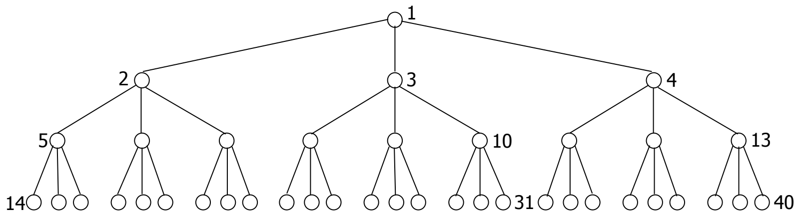
Trends in DBMSs

RM vs. XML
Architectural requirements
Node labeling
Indexing
Node processing services
Concurrency control
Lock manager
Performance measurement

Node labeling – the key to fine-grained management of XML documents

Node Labeling – Early Requirements

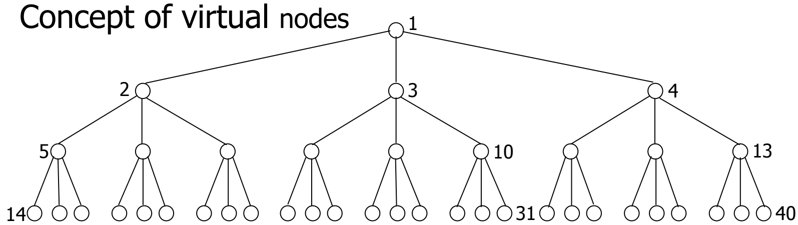
- Declarative access of **static** XML documents
  - efficient evaluation of the 13 axes of the XQuery and the XPath 2.0 language model (sequence semantics)
  - most important axes: **parent/child, ancestor/descendant, preceding-sibling/following-sibling**
- Complete k-ary trees (example: k = 3)



  - pre-analysis required to determine max (k)
  - real documents are incomplete k-ary trees

Node Labeling – Early Requirements (2)

- Concept of virtual nodes

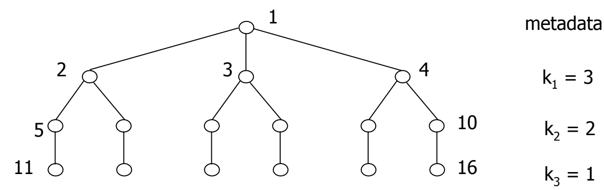


  - parent (cn, k) = ceil ((cn – 1)/k)
  - child (cn, k) = cn*k – (k-1) + 1, cn*k – (k-2) + 1,
    cn*k – (k-i) + 1, …, cn*k – 1 + 1, cn*k + 1
  - ancestor (cn, k) = parent (cn, k), parent (parent (cn, k), k), …
  - descendant (cn, k) = child (cn, k), child (child (cn, k), k), …
  - sibling (cn, k) = child (parent (cn, k), k), …
  - previous/following …
- KO criterion
  - any computed label may correspond to a virtual node
  - tree representation has to be accessed to check whether a node is real or virtual

⇨ **a document may have a very large k and very many levels**

Node Labeling – Early Requirements (3)

- Improvements (see eXist prototype): use pre-analysis to
  - determine max (k_i) per level l_i
  - build complete trees (k_i, l_i)
  - reduce the set of virtual nodes



⇨ **relationships among nodes may still be computed**

- KO criterion
  - order-preserving insertion (replacement of virtual nodes) not always possible
  - subtree insertions may violate the labeling scheme
  - insertions may enforce the relabeling of the entire tree

Trends in DBMSs
RM vs. XML
Architectural requirements
Node labeling
Indexing
Node processing services
Concurrency control
Lock manager
Performance measurement

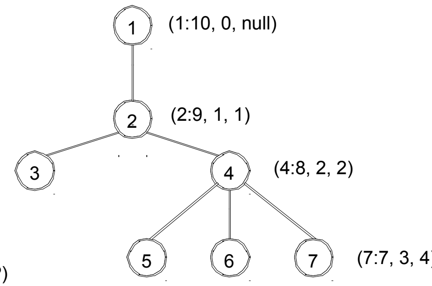
Node Labeling – New Requirements

- Support of **dynamic** XML documents
  - all axes relationships should be evaluated without accessing the document
  - internal navigation operations should help to optimize declarative queries
  - multi-lingual XML interfaces require navigational support (e.g., DOM and SAX)
  - labeling scheme should be insensitive to insertions
  - most important for intention **locking**: a node label should allow for the determination of the node labels (IDs) of all its ancestors
- Principal Approaches to a Solution
  - **two classes: range-based and prefix-based schemes**

Range-based Schemes

- positions of nodes marked by (DocNo, LeftPos:RightPos, LevelNo)
- LP and RP describe the labeling range in each node with its subtree; generated by a depth-first traversal of the tree
- ancestor-descendant containment (DocNo is omitted):
  a node n1 (LP1:RP1, lv1) contains a node n2 (LP2:RP2, lv2),
  iff LP1 < LP2 and RP1 > RP2.
- additional condition for parent-child containment: lv1 = lv2 - 1
- supporting preceding-sibling/following-sibling relationship?

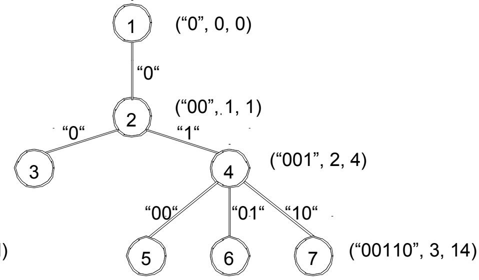
- simple example



label template (LP:RP, lv, P_LP)

Prefix-Based Schemes

- Each node is encoded with a unique string S such that
  - S(v) is before S(u) in lexicographic order iff node v is before node u in the document order
  - S(v) is a prefix of S(u) iff node v is the ancestor of node u
- Simple example:
  - assign to the outgoing edges of each node a set of prefix-free binary strings in lexicographical order from left to right
  - the label of each node is the concatenation of the parent's label and the string assigned to its incoming edge
  - record the level of a node
  - add the edge string length esl to each node descriptor to derive the ancestor label



label template (S, lv, esl)

DeweyIds Embody a Special Prefix Labeling Scheme

- Labels must
  - be immutable for the lifetime of the nodes
  - preserve the document order, when inserting new nodes
  - easily reveal the level and the ID for all ancestor nodes
- DeweyID consists of **several divisions** separated by dots
  - overflow mechanism: **even** division values

    $\mathbf{d_1 = 1.3.17.2.2.3.4.9}$ $\mathbf{d_2 = 1.3.17.2.3.7}$
  - level determination

    $\mathbf{d_1 = 1.3.17.2.2.3.4.9}$
  - ancestor IDs: $a_0 = 1$; $a_1 = 1.3$; $a_2 = 1.3.17$; $a_3 = 1.3.17.2.2.3$
  - ordering

    $\mathbf{d_2 \ ? \ d_1}$

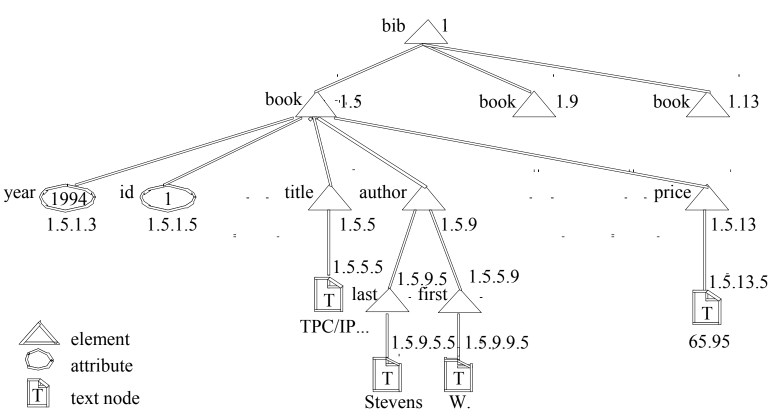
    $\mathbf{d_1 < d_2 : \ 1.3.17.2.2.3.4.9 < 1.3.17.2.3.7}$

Trends in DBMSs

RM vs. XML
Architectural requirements
Node labeling
Indexing
Node processing services
Concurrency control
Lock manager
Performance measurement

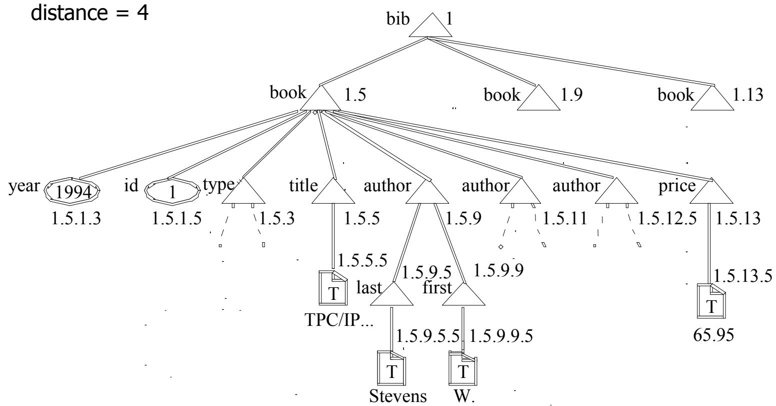
Initial Assignment of DeweyIds

- assignment of division values is affected by parameter *distance* (= 4)
- on initial loading, only **odd** division values are assigned
- odd division value indicates level transition

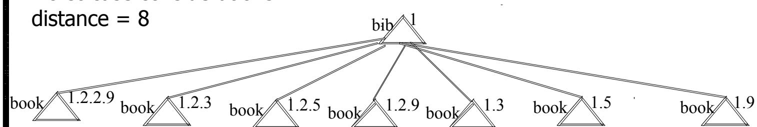


DeweyIDs: Insertion of Subtrees

distance = 4



worst-case considerations:
distance = 8



Benefits of DeweyID Use

- Existing DeweyIDs allow the assignment of new IDs without the need to reorganize the IDs of nodes present. Relabeling only in case of violations of implementation restrictions
- The DeweyID of each ancestor node can be determined in a very simple way
- Comparison of two DeweyIDs delivers the order of the respective nodes in the left-most depth-first stored document.
- Checking whether node d1 is an ancestor of d2 only requires to check whether DeweyID of d1 is a prefix of DeweyID of d2.
- High distance values reduce the probability of reorganization. They have to be balanced against increased storage space

but: DeweyIDs may become very long

ORDPATHs and DLN schemes have similar properties.
We call the generic form SPLIDs (Stable Path Labeling IDs)

Encoding of DeweyIDs

- Fixed length field

TL	L_0	E_0	L_1	E_1	. . .	L_k	E_k

TL = total length
l_i = length of L_i
L_i = length of i-th division (O_i)
E_i = value of i-th division

$l_i = 6$: $L_{Oi} < 64$: $O_i < 2^{64}$ bits

$O_i = 7$ needs 6+3 bits

- Fixed- and variable-length length fields

TL	L_{f0}	L_{v0}	E_0	L_{f1}	. . .	L_{vk}	E_k

l_f = length of L_{fi}
L_{fi} = length of L_{vi}
L_{vi} = length of the i-th division

length of $L_{vi} < 2^{Lfi}$: value of $O_i < 2^{Lvi+1}$ using range expansion

$l_f = 2$: $O_i < 2^{31}$

$l_f = 3$: $O_i < 2^{511}$

but penalty for small division values: $O_i = 7$ needs 3+2+3 bits

Encoding of DeweyIDs (2)

- **k-based representation**
  - m = ceil (log (k + 1))
  - reserve one code of length m to represent the separator "."
  - interpret a sequence of m-bit codes as a number with base k

k = 3: "0": 00, "1": 01, "2": 10, ".": 11

1.7.11 : TL 01 11 10 01 11 01 00 10

$1*3^0$ $2*3^1 + 1*3^0$ $1*3^2 + 0*3^1 + 2*3^0$

good space efficiency: $O_i = 7$ needs 6 bits, but no adaptation to value distributions

is there a better k: k = 1 or k = 7?

k = 7: "0": 000, "1": 001, "2": 010, "3": 011, …., ".": 111

1.7.11 : TL 001 111 001 000 111 001 100

$1*7^0$ $1*7^1 + 0*7^0$ $1*7^1 + 4*7^0$

$O_i = 7$ needs 9 bits

KO criterion: comparison of DeweyIDs at the bit/byte level not possible

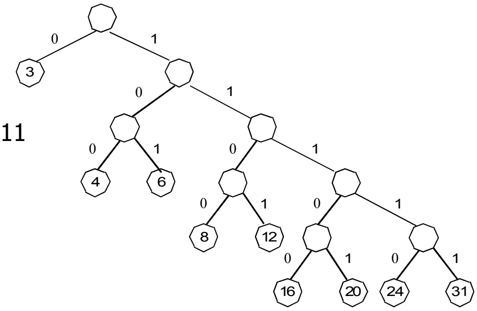
Encoding of DeweyIDs (3)

- Huffman codes

TL	C_0	E_0	C_1	E_1	. . .	C_k	E_k

1.7.11: TL 0001 0111 1000011

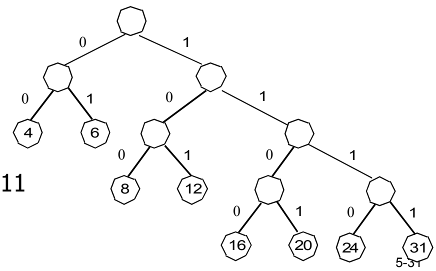
O_i = 7 needs 4 bits



- Degrees of freedom
  range weights and
  length assignments

1.7.11: TL 000001 000111 001011

O_i = 7 needs 6 bits



Trends in DBMSs

RM vs. XML
Architectural requirements
Node labeling
Indexing
Node processing services
Concurrency control
Lock manager
Performance measurement

Characteristics of XML Documents Considered

file name	description	size (bytes)	number of element nodes	number of text nodes	number of attributes	max. depth	⌀-depth	max. fanout	⌀-fanout of elements
1) treebank_e.xml	Encoded DB of English records of Wall Street Journal	86,082,517	2,437,666	1,391,845	1	**37**	**8.44**	**56,385**	**1.58**
2) nasa.xml	Astronomical data	25,050,288	476,646	303,676	56,317	**9**	**6.08**	**2,435**	**1,76**
3) psd7003.xml	DB of protein sequences	**716,853,016**	21,305,818	15,955,109	1,290,647	**8**	**5.68**	**262,529**	**1.81**
4) SwissProt.xml	DB of protein sequences	114,820,211	2,977,031	2,013,844	2,189,859	**6**	**4.07**	**50,000**	**2.41**
5) dblp.xml	Computer Science Index	**284,994,162**	6,662,623	6,013,355	1,375,832	**7**	**3.39**	**649,080**	**2.11**
6) customer.xml	Customers from TPC-H benchmark	515,660	13,501	12,000	1	**4**	**3.41**	**1,501**	**1.89**
7) ebay.xml	Ebay auction data	35,562	156	107	0	6	4.26	12	1.90
8) lineitem.xml	Line items from TPC-H benchmark	32,295,475	1,022,976	962,800	1	4	3.45	60,176	1.94
9) mondial-3.0.xml	Geographical DB of diverse sources	1,784,825	22,423	7,467	47,423	6	4.15	955	3.45
10) orders.xml	Orders from TPC-H Benchmark	5,378,845	150,001	135,000	1	4	3.42	15,001	1.90
11) uwm.xml	Courses of a University Website	2,337,522	66,729	40,234	6	6	4.37	2,112	1.91

Encoding of DeweyIDs

Huffman code	L_i	value range of O_i
0	3	1-7
100	4	8-23
101	6	24-87
1100	8	88-343
1101	12	344-4,439
11100	16	4,440-69,975
11101	20	69,976-1,118,551
11110	24	1,118,552-17,895,767
11111	31	17,895,768-2,165,379,414

range expansion

Optimization potential

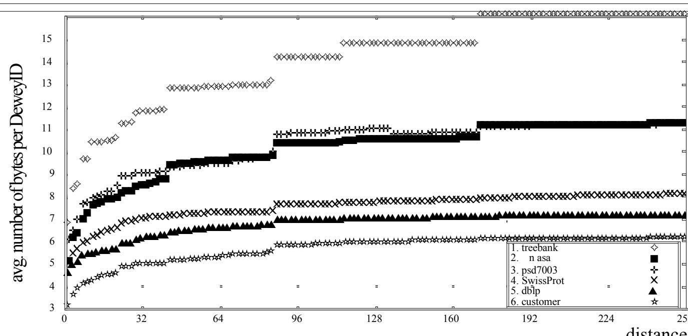
- analysis phase, if possible: determine DOM tree parameters for optimized Huffman code assignment (even level-wise applicable)
- cut prefix 1.
- apply prefix compression to DeweyIDs

Trends in DBMSs

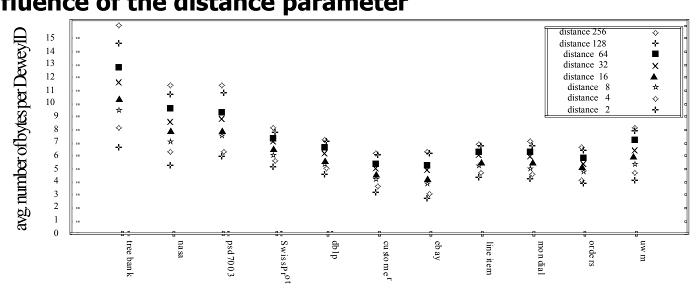
RM vs. XML
Architectural requirements
Node labeling
Indexing
Node processing services
Concurrency control
Lock manager
Performance measurement

DBIS Datenbanken und Informationssysteme
© 2005 AG DBIS

Avg. Sizes of DeweyIDs Grouped by the Documents Avg. Depth



influence of the distance parameter

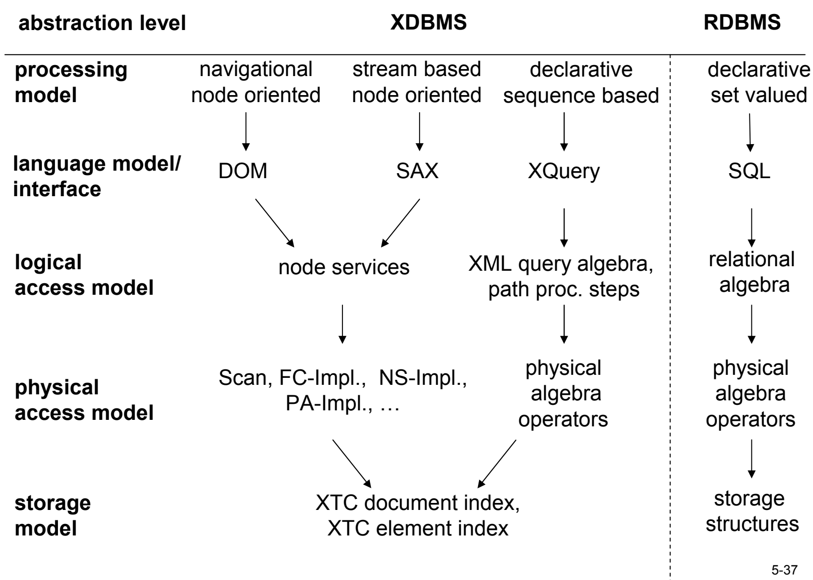


DeweyIDs – Comparison of Avg. Sizes to Max. Sizes

Document	∅-size			max-size		
	dist(2)	dist(32)	dist(256)	dist(2)	dist(256)	dist(256)
1. tree-bank_e	**6.67**	11.57	**15.94**	**22**	46	**72**
2. nasa	5.19	8.54	11.30	8	13	18
3. psd7003	5.61	8.84	11.30	8	13	17
4. SwizzProt	5.10	7.04	8.14	8	11	13
5. dblp	4.58	6.12	7.16	7	10	13
6. customer	**3.17**	5.04	**6.19**	**4**	6	**7**

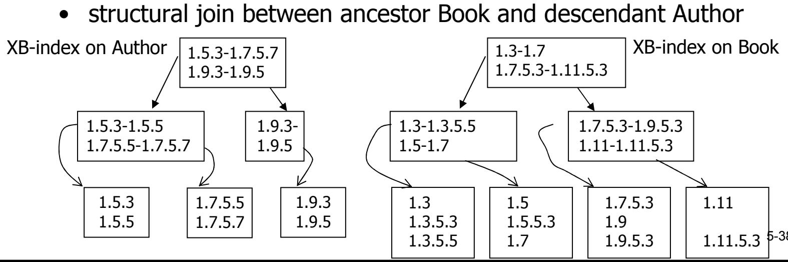
Node processing services – indexing support for and evaluation of navigational operations and path processing steps of set-oriented operations

Levels of Abstraction



XML Indexing Methods

- Indexing of documents or element sets
  - organization of the document on disk using a document container and provision of additional access paths
  - goals: effective support of
    - navigation operations (node-at-a-time)
    - structural join operations (set-at-a-time) (TreeStack, PathStack, TwigStack, …)
  - examples (of a gold rush area)
    - XB-Tree, XR-Tree (tailored to holistic twig joins)
    - XTC document index, XTC element index
  - structural join between ancestor Book and descendant Author



XML Indexing Methods (2)

- Content or text indexing (known from IR)
  - goals:
    - support of predicates on textual content of a document (contains, near, matches, …)
    - indexing of (typed) values such as Integer, Double, …
  - examples
    - inverted lists
    - value index (VIndex in Lore)
- Hybrid Indexing
  - for structure and text
  - goal:
    - evaluation of queries on structure and content
  - examples
    - IndexFabric, SPHINX, BUS

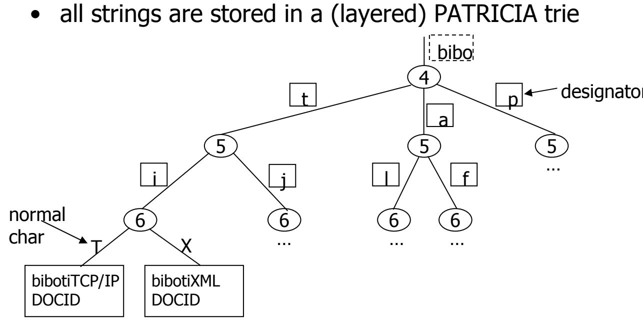
Trends in DBMSs

RM vs. XML
Architectural requirements
Node labeling
Indexing
Node processing services
Concurrency control
Lock manager
Performance measurement

DBIS Datenbanken und Informationssysteme
© 2005 AG DBIS

XML Indexing Methods (3)

- IndexFabric as an example
  - designators: special characters for encoding paths
  - a unique designator is assigned to each tag
    - bib = bi, type = ty, author = a, last = l
    - book = bo, title = ti, price = p, first = f
    - example: "bibotiTCP/IP"
  - all strings are stored in a (layered) PATRICIA trie



query evaluation: translate query to designator string and lookup in the trie

XML Indexing Methods (4)

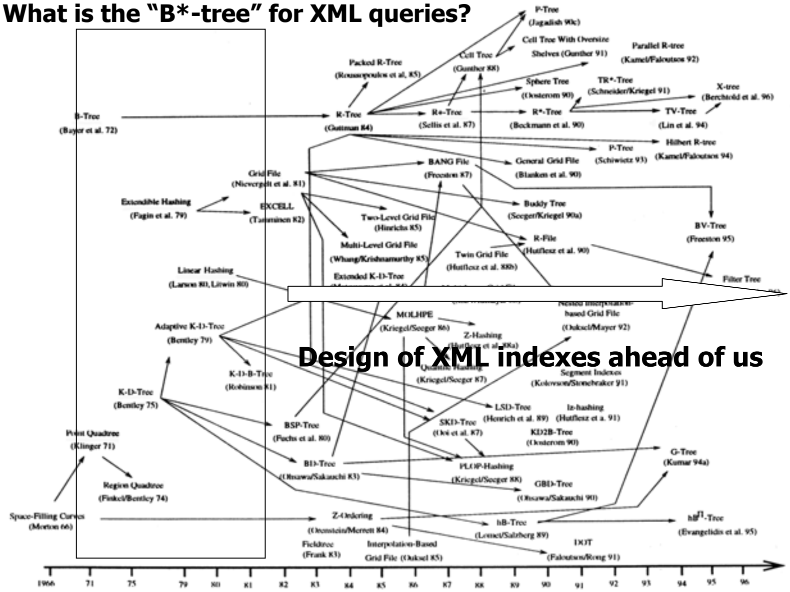
- Path indexing
  - goals: effective support for
    - query rewriting, query pruning based on structural summaries, e.g., to check the existence of paths
    - availability of statistical data (container for metadata)
    - evaluation of (root-to-leaf) path queries through collections of element sets
  - examples
    - DataGuide, 1-Index (identical for tree-based documents)
    - APEX, D(k)-Index (adaptive methods), A(k)-Index

- Many types of path fragments in queries to be supported
  - parent/child: a/b/c
  - ancestor/descendants: a//x//y
  - enhanced by predicates: a/b[pred1]//x[pred2]
  - twigs: a/b[.//c]//x[d]

⇨ What is the ubiquitous index – the silver bullet – for XML queries?

Genealogy of XML Indexes not yet Known

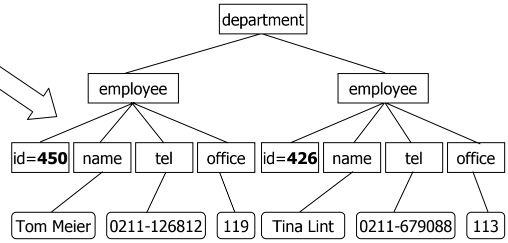
What is the "B*-tree" for XML queries?

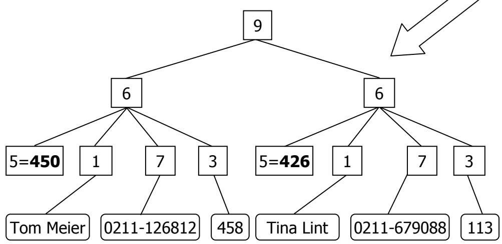


Transformation of an XML Document (Example DB2)

Transformation into an internal XML tree; element names are replaced by means of a dictionary

```
<department>
 <employee id="450">
          <name> Tom Meier</name>
          <tel>0211-126812</tel>
          <office>119</office>
 </employee>
 < employee id="426">
          <name>Tina Lint</name>
          <tel>0211-679088</tel>
          <office>113</office>
 </employee>
</department>
```





SYSIBM:SYSXMLSTRINGS

String table	
9	dept
6	employee
1	name
5	id
7	tel
3	office

Trends in DBMSs

RM vs. XML
Architectural requirements
Node labeling
Indexing
Node processing services
Concurrency control
Lock manager
Performance measurement

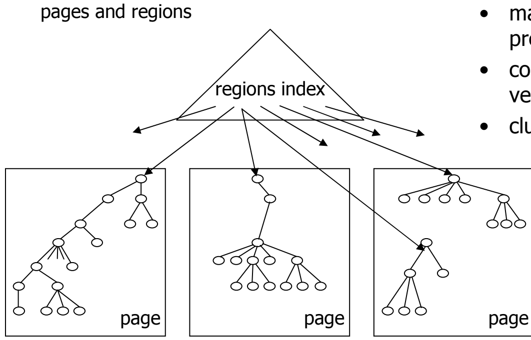
DBIS Datenbanken und Informationssysteme
© 2005 AG DBIS

XML Indexes (Example Viper)

- path-specific indexes (structure)
- full-text indexes (content)

→ indexing of elements, attributes, text

mapping of tree structure to pages and regions

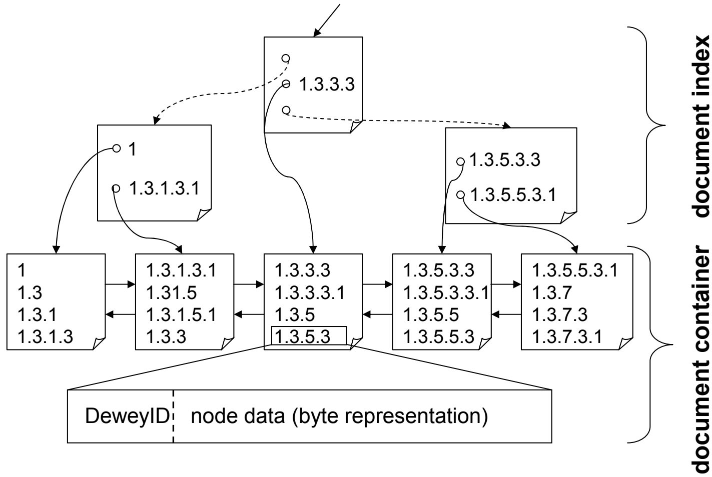


- XML index entry consists of indexed
  - path to node (PathID)
  - value
  - table row for the document (RowID)
  - Nodes and Regions within the document (NodeID)

→ can refer to an atomic value or to a subtree

- mapping enables intelligent prefetching
- concurrency control using versions on pages/subtrees
- clustering?

XTC Document Index

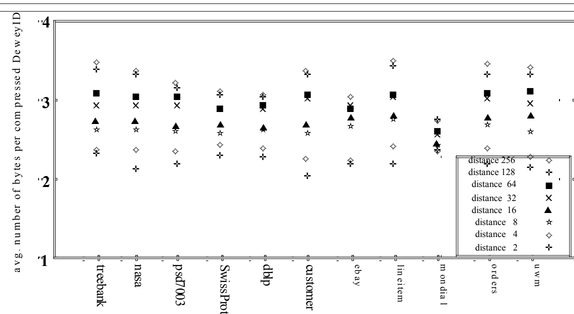


document index is a B-tree for the document(s) stored in the doubly-chained pages of the document container
text values exceeding a given threshhold are stored in referenced mode

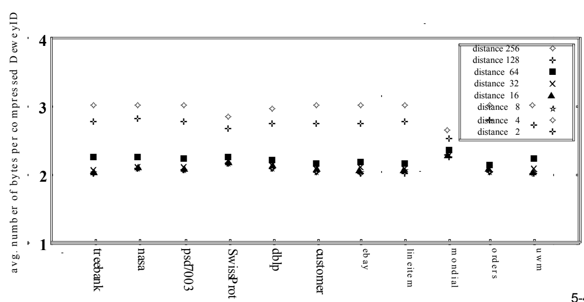
prefix compression works!

XTC Document Index – ∅-Size per Compressed DeweyID

Prefix compression

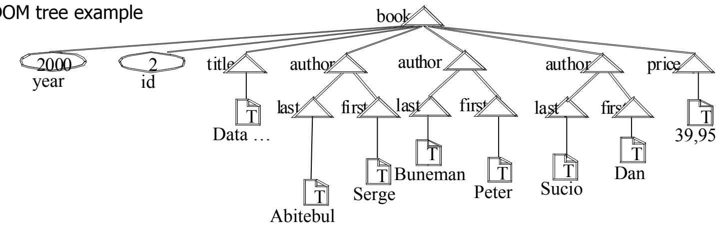


Optimized prefix compression/ encoding scheme

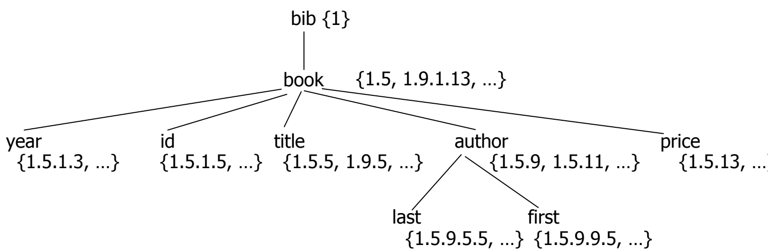


Datagraph – Supporting Query Optimization and Evaluation

DOM tree example

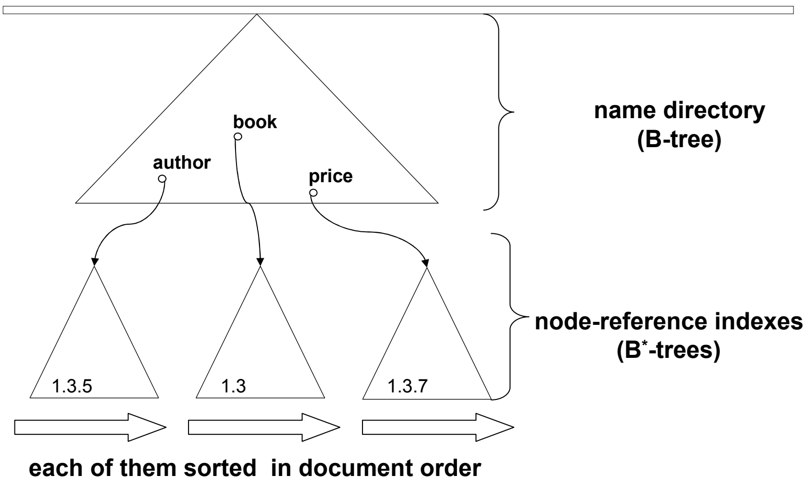


A data guide d has for every label path in the source document s a data path instance in d and every label path of d is a label path of s



A data guide provides a structural summary
XTC stores the DeweyIDs in separate index structures

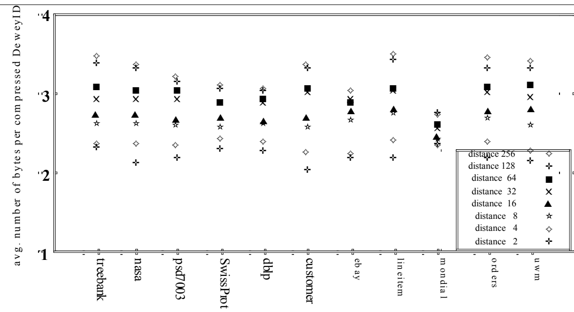
XTC Element Index



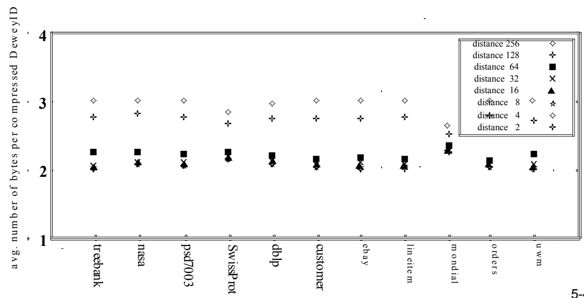
In combination with a data guide, the XTC element index can be used like a path index
- unique occurrence of element names: DeweyIDs are path indexes
- homonyms of element names: additional tests allow the separation of the paths (DeweyIDs)

XTC Document Index – ∅-Size per Compressed DeweyID

Prefix compression



Optimized
prefix compression/
encoding scheme

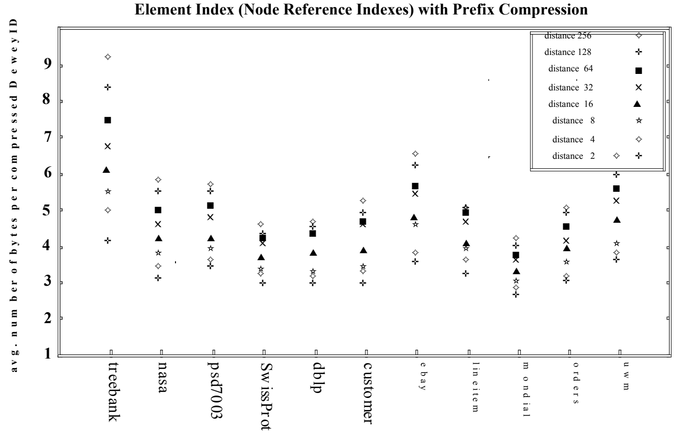


Trends in DBMSs

XTC Element Index – ∅-Size per Compressed DeweyID

Prefix compression of DeweyIDs used as pointers

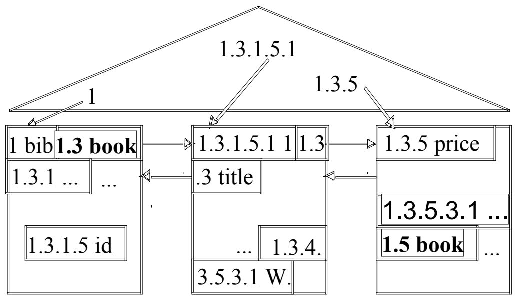
Element Index (Node Reference Indexes) with Prefix Compression



RM vs. XML
Architectural requirements
Node labeling
Indexing
Node processing services
Concurrency control
Lock manager
Performance measurement

Example: DeweyID Support for a Navigational 0peration

- Expressiveness of DeweyIDs allows derivation of ancestors
- Navigational axes
  - parent
  - first/last child
  - next/previous sibling
- Context node: 1.5
  - previous sibling without scanning the container pages



XQuery – XPath: Path Processing Steps

- Problem: XQuery is really complex
  - order-preserving joins, implicit grouping, result construction …
- therefore at the moment "only" XPath

```
doc("sample.xml")//autor
[count
  (parent::buch/preceding-sibling::
   element())>3]/vname[../nname = "Adams"]
```

- still complex
- concentration on path processing steps

```
Axis::name_test
```

core processing units: sequence of path processing steps,
evaluation order: bottom-up, top-down, starting in the middle
algorithms: selective access via DeweyIDs, locking-aware

XPath Axes

1. **parent**
2. **child**
3. **ancestor**
4. **descendant**
5. **previous-sibling**
6. **following-sibling**
7. **previous**
8. **following**
9. attribute
10. namespace
11. self
12. ancestor-or-self
13. descendant-or-self

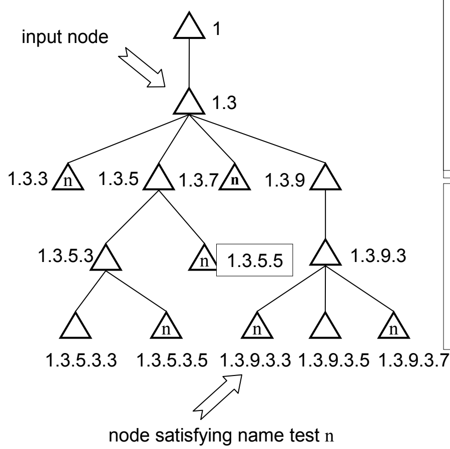
Axes Operators

- Each of the 8 relevant XPath axes has a set of own operators (selected by the optimizer)
- Input
  - name test
  - duplicate-free sequence of DeweyIDs in document order
- Output
  - duplicate-free sequence of DeweyIDs in document order on specified axis: each referenced node satisfies the name test
- Chaining of axis operators to evaluate XPath expressions of the form

$\texttt{axis}_1\texttt{::name\_test}_1\texttt{/…/axis}_n\texttt{::name\_test}_n$

Child Axis

input sequence: dark nodes



observe level information

„probing“ of the parent DeweyIDs

HT(1.3)	1.3.3
HT(1.3.5.3)	1.3.5.3.5
HT(1.3.9.3)	1.3.5.5
	1.3.7
	1.3.9.3.3
	1.3.9.3.7

result

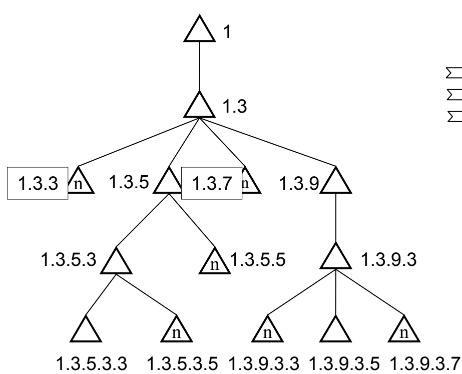
1.3.3
1.3.5.3.5
1.3.7
1.3.9.3.3
1.3.9.3.7

Trends in DBMSs

RM vs. XML
Architectural requirements
Node labeling
Indexing
Node processing services
Concurrency control
Lock manager
Performance measurement

Descendant Axis

may produce very many duplicates
document order avoids duplicates in the result



merge method

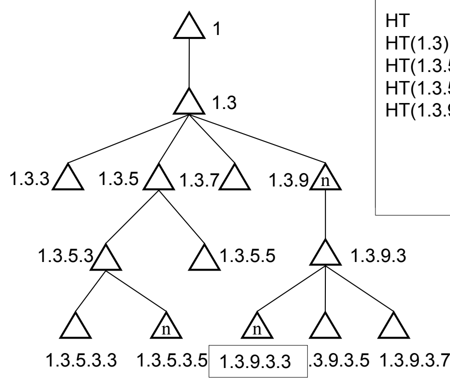
input	descendents
1.3.5	1.3.3
1.3.5.3	1.3.5.3.5
1.3.9.3	1.3.5.5
	1.3.7
	1.3.9.3.3
	1.3.9.3.7

result

1.3.5.3.5
1.3.5.5
1.3.9.3.3
1.3.9.3.7

Following-Sibling Axis

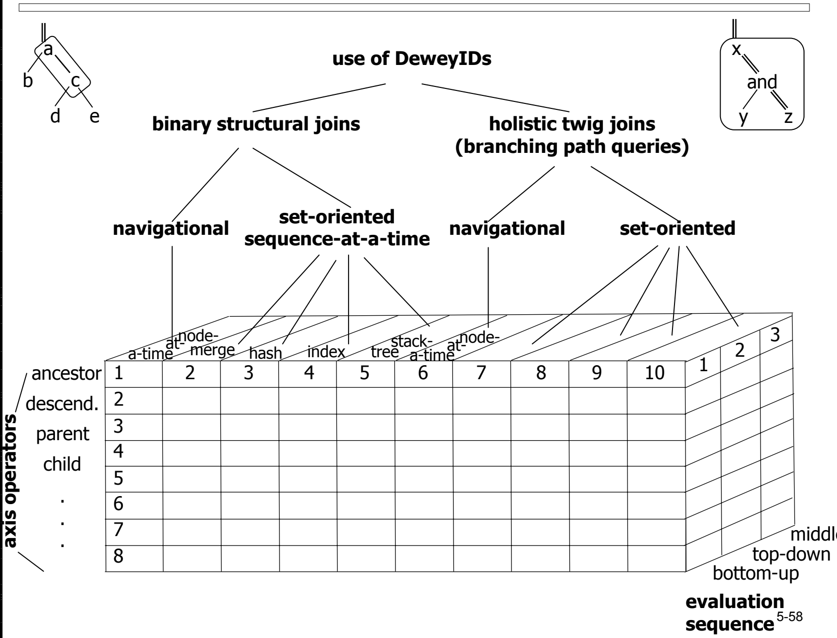
all processing steps avoid duplicates: O(n)



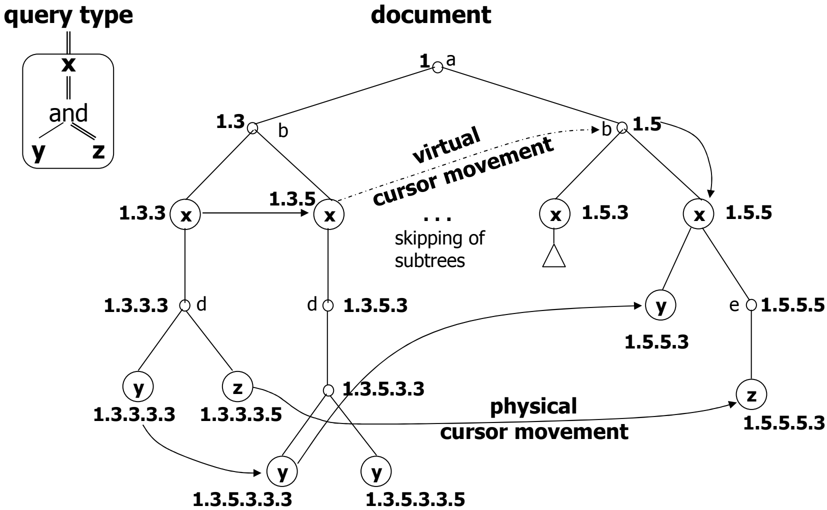
"probing" of predecessor nodes

HT	following-siblings
HT(1.3) = 1.3.3	1.3.5.3.5
HT(1.3.5.3) = 1.3.5.3.3	1.3.9
HT(1.3.5) = 1.3.5.5	1.3.9.3.3
HT(1.3.9.3) = 1.3.9.3.5	

Path Processing Algorithms



Evaluation of Path Processing Steps – Optimization of Cursor Movement



Index access to nodes of element type x, y, z

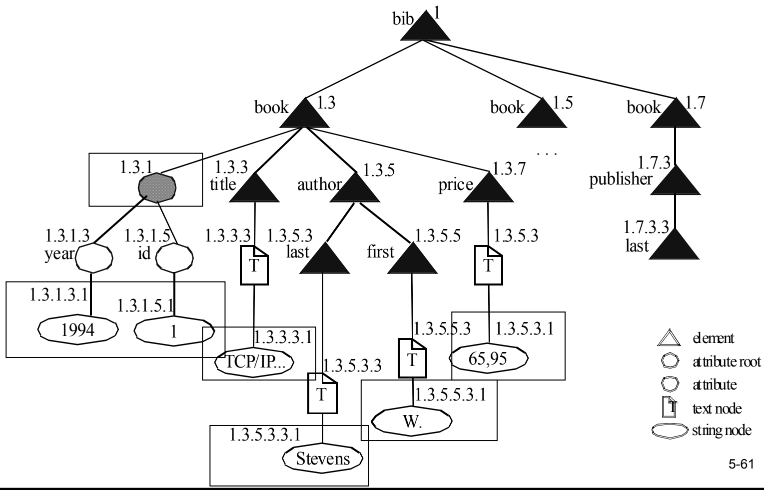
RM vs. XML
Architectural requirements
Node labeling
Indexing
Node processing services
Concurrency control
Lock manager
Performance measurement

The forgotten Need in XML Processing – Fine-grained locking of XML documents

RM vs. XML
Architectural requirements
Node labeling
Indexing
Node processing services
Concurrency control
Lock manager
Performance measurement

Example of a taDOM Tree: Conceptual Representation

- View of the Lock Mgr
  - memory representation is enhanced by virtual node types
  - new node types *attribute root* and *string*
  - optimization of concurrent access (fine-grained locking)



Trends in DBMSs

RM vs. XML
Architectural requirements
Node labeling
Indexing
Node processing services
Concurrency control
Lock manager
Performance measurement

Node Locks

- Node locks and compatibility matrix
  refined from hierarchical locking schemes

Compatibility matrix

	-	IR	NR	LR	SR	IX	CX	SU	SX
IR	+	+	+	+	+	+	+	-	-
NR	+	+	+	+	+	+	+	-	-
LR	+	+	+	+	+	+	-	-	-
SR	+	+	+	+	+	-	-	-	-
IX	+	+	+	+	-	+	+	-	-
CX	+	+	+	-	-	+	+	-	-
SU	+	+	+	+	+	-	-	-	-
SX	+	-	-	-	-	-	-	-	-

Read locks

lock	**effect**
IR (intention read)	Read lock on non-direct child node
NR (node read)	Read lock on the node
LR (level read)	Read lock on context node and all direct-child nodes
SR (subtree read)	Read lock on entire subtree

Write locks

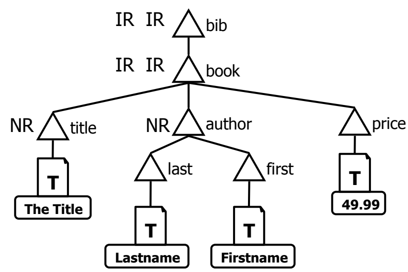
lock	**effect**
SX (exclusive)	Write lock on entire subtree
CX (child excl.)	Write lock on direct child node
IX (intent. excl.)	Write lock on non-direct child node
SU (update)	Read lock with intended update operation on entire subtree

Node Read Lock

- NR
  - requested for reading the context node
  - requires IR locks on the ancestor path

Transaction T_1 is reading <title>

Transaction T_2 is reading <author>



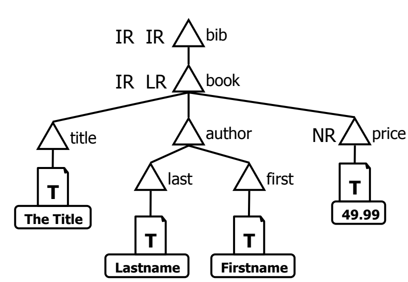
Trends in DBMSs
RM vs. XML
Architectural requirements
Node labeling
Indexing
Node processing services
Concurrency control
Lock manager
Performance measurement
DBIS Datenbanken und Informationssysteme
© 2005 AG DBIS

Level Read Lock

- LR
  - requested for reading the context node and all nodes located in the level below (all direct-child nodes)
  - requires NR locks on the ancestor path

Transaction T_1 is reading <book> and all direct-child nodes (<title>, <author>, and <price>)

Transaction T_2 is reading <price>



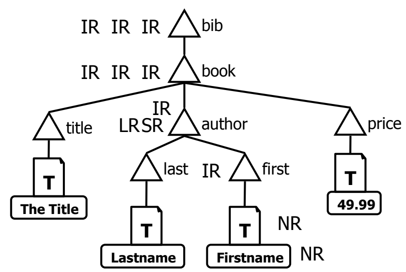
Subtree Read Lock

- SR
  - requested for reading the context node and all nodes located in the subtree below
  - requires IR locks on the ancestor path

Transaction T_1 is reconstructing
<author>
<last>Lastname</last>
<first>Firstname</first>
</author>

Transaction T_2 is reading the value of the text node in <first>

Transaction T_3 is reading all direct-child nodes of <author> (<last> and <first>)



Exclusive Locks

- Intention exclusive lock (IX), child exclusive lock (CX), and exclusive lock (X)
  - requested for updating the context-node's content or deleting the context node and its entire subtree
  - requires CX lock and IX locks on the ancestor path

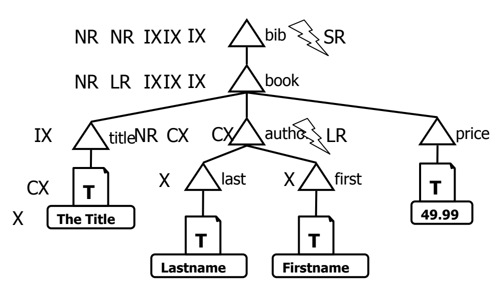
Transaction T_1 is updating the book title

Transaction T_2 is deleting the <first> node and its content

Transaction T_3 is deleting the <last> node and its content

Transaction T_4 is reading the <author> node

Transaction T_5 is reading all direct-child nodes of <book> but is blocked when reading all child nodes of <author>



Transaction T_6 is blocked when reconstructing the entire document

Optimization of Lock Management

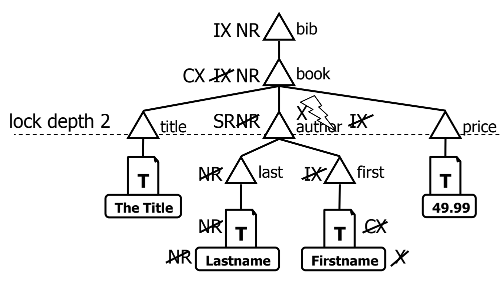
- Reducing the number of locks held
  - tunable lock granularity
    - *lock depth:* number of node levels on which locks are acquired (counted from document root)
    - on exceeding the *lock depth:* acquisition of SR and X locks only on *lock depth* level
  - lock escalation
    - *escalation threshold* and *escalation depth*
    - starting scan of subtrees with *escalation depth* height on taDOM leaves
    - calculating percentage of existing locks on the nodes
    - on exceeding the *escalation threshold:* replacing locks in subtree with SR resp. X locks on subtree root

Escalation Threshold

- Tunable lock depth
  - reduce the number of locks held using coarser lock granularity
  - pay with less concurrent processing
  - on exceeding the lock depth: acquisition of SR and X locks only at *lock depth* level or above

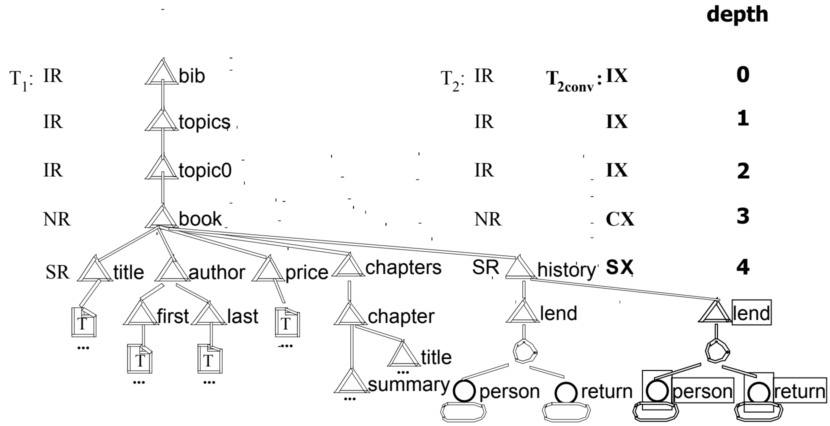
Transaction T_1 is reading the author's last name

Transaction T_2 is updating the author's first name



Optimization Parameter: Lock Depth

T_1 reads this book with lock depth = 4:
options SR on bib, topics, topic0, book, or on each children of book



T2 reads the history subtree (SR)

and decides to attach a new lend subtree

⇒ lock depth < 4 would have prevented the lock conversion

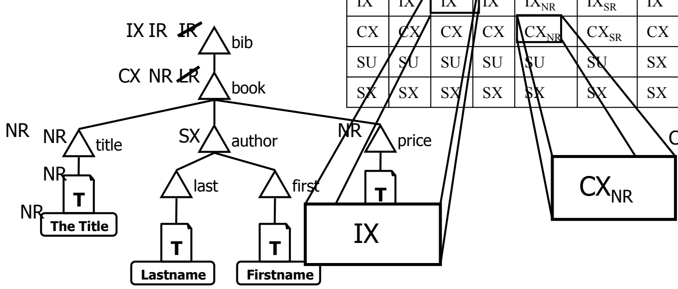
Trends in DBMSs
RM vs. XML
Architectural requirements
Node labeling
Indexing
Node processing services
Concurrency control
Lock manager
Performance measurement

Node Lock Conversion

- Lock conversion in the ancestor path of author

	-	IR	NR	LR	SR	IX	CX	SU	SX
IR	IR	IR	NR	LR	SR	IX	CX	SU	SX
NR	NR	NR	NR	LR	SR	IX	CX	SU	SX
LR	LR	LR	LR	LR	SR	IX_{NR}	CX_{NR}	SU	SX
SR	SR	SR	SR	SR	SR	IX_{SR}	CX_{SR}	SR	SX
IX	IX	IX	IX	IX_{NR}	IX_{SR}	IX	CX	SX	SX
CX	CX	CX	CX	CX_{NR}	CX_{SR}	CX	CX	SX	SX
SU	SU	SU	SU	SU	SU	SX	SX	SU	SX
SX	SX	SX	SX	SX	SX	SX	SX	SX	SX

Conversion matrix



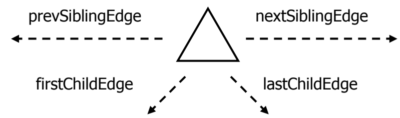
Transaction T_1 is reading <book> and all its direct-child nodes

Transaction T_2 is reading <book>, the first child node <title> and its value

Transaction T_1 is deleting <author> and its entire subtree

Navigation Locks

- Definition of virtual navigational edges for nodes



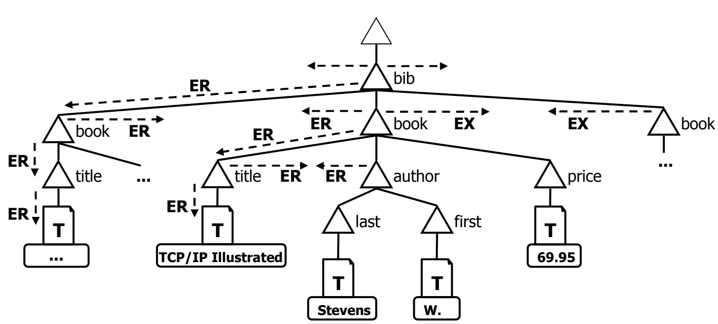
- Acquisition of adequate locks on edges
  - to be done in addition to node locking, but only if node level is not already locked by LR, SR, U, or X
  - while navigating by use of corresponding edge
  - while adding or deleting nodes which are connected to corresponding edge
- Well-known R/U/X compatibility for navigation locks

	-	ER	EU	EX
ER	+	+	-	-
EU	+	+	-	-
EX	+	-	-	-

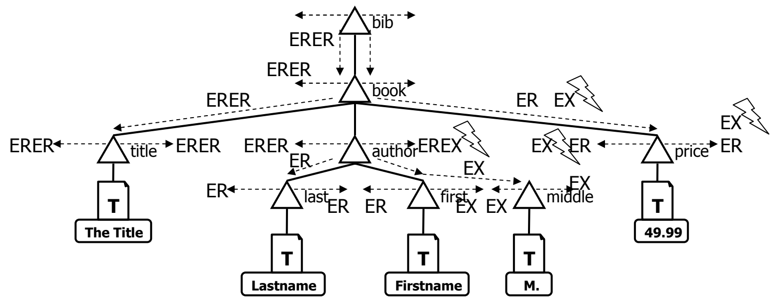
ER: edge read
EU: edge update
EX: edge exclusive

Navigation Locks (2)

- Example: concurrent reading and inserting of books
  - Transaction 1
    - navigating to title value of first book element
    - navigating to title value of second book element
    - navigating to author element of second book element
  - Transaction 2
    - inserting a new book element



Navigation Locks (3)



Transaction T_1 navigates
<bib> <book> <title> <author> <price>
and determines <price> as last child

Transaction T_2 is blocked when appending a new last child to <book> or inserting a new node between <author> and <price>

Transaction T_3 navigates
<bib> <book> <title> <author>
<last> <first> and does NOT determine the next sibling node of <first>

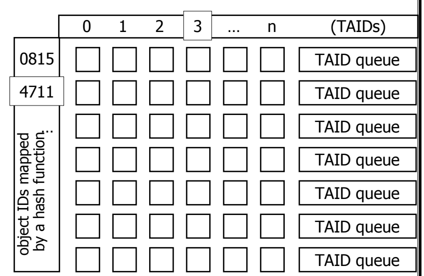
Transaction T_4 is allowed to append a new last-child element to <author>

Lock manager

Use of semaphore tables

- Synchronization of objects of varying types
  - at diverse system layers (pages, nodes, edges, indexes)
  - incomparable lock compatibilities
  - very short to very long lock durations
  - differing access frequencies
- 6 specialized semaphore tables
  - one lock per object and transaction
  - FIFO-managed LRBs (TAID queues)
  - statically configurable
  - needs optimization for high TA loads! use of linked OCBs dynamically allocated

simplified illustration

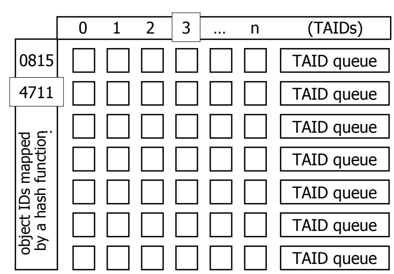


Trends in DBMSs

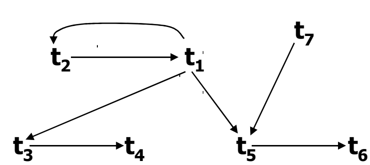
RM vs. XML
Architectural requirements
Node labeling
Indexing
Node processing services
Concurrency control
Lock manager
Performance measurement

Use of semaphore tables (2)

- Lock management
  - allocation and removal of objects
  - assignment, replacement, and removal of semaphores
  - removal of transactions



- Deadlock detection
  - eager cycle checking is not cost-effective
  - lazy and more economic solution: periodic checking

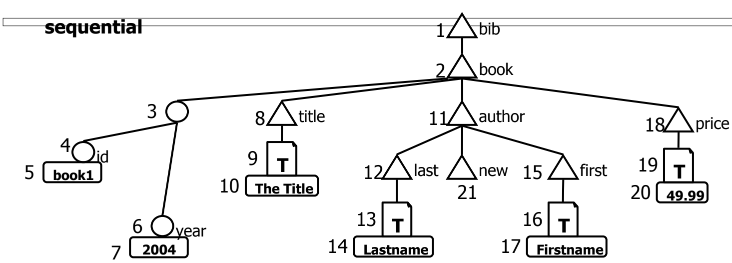


XTCserver – Meta Locking

- Meta-lock requests from node manager to lock manager
  - request *shared node lock*
  - request *shared level lock*
  - request *tree lock* (*shared*, *update*, *exclusive*)
  - request *edge lock* (*shared*, *update*, *exclusive*) on previous sibling edge, next sibling edge, first child edge, and last child edge
  - release locks
- Mapping of meta-lock request to current locking algorithm by lock manager
  - a lock manager implements a certain interface
  - exchange of the lock manager interface implementation exchanges the system's complete XML locking mechanism

Identifying nodes – node numbering schemes

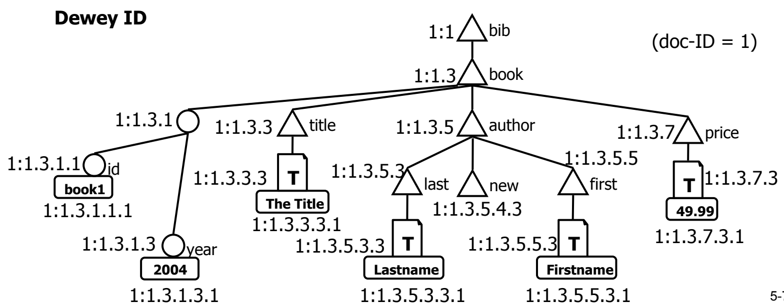
sequential



- very slow, although supported by on-demand indexes
- determination of parent ID and ancestor IDs very frequent

Dewey ID

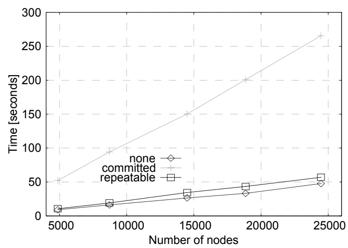
(doc-ID = 1)



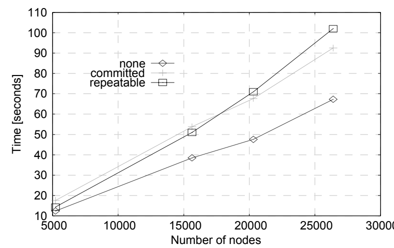
Basic Cost of Lock Management

RM vs. XML
Architectural requirements
Node labeling
Indexing
Node processing services
Concurrency control
Lock manager
Performance measurement

- Experiment
  - reconstruction of entire XML documents (XMark) in depth-first order using DOM API
  - two document traversals within one transaction
  - locking overhead at different isolation levels is revealed



Sequential ID implementation



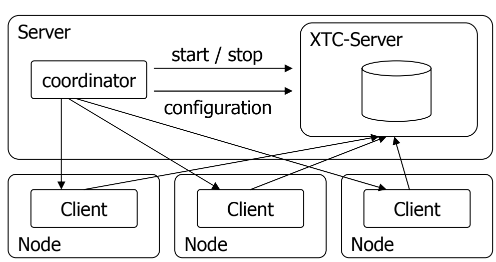
Dewey ID implementation

needs variable-length entries for keys and value fields

TaMix Benchmark Framework

RM vs. XML
Architectural requirements
Node labeling
Indexing
Node processing services
Concurrency control
Lock manager
Performance measurement

- Infrastructure for distributed OLTP benchmarks



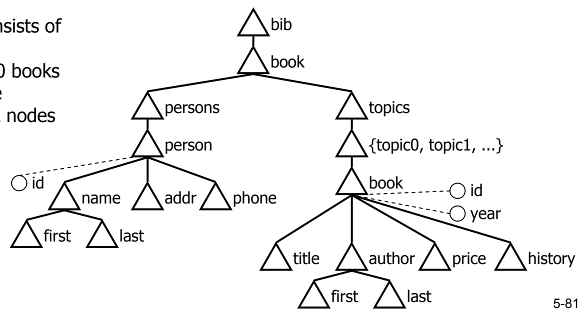
- Automated measurement
  - 3 runs per measurement point
  - runtime 5 minutes each
  - for 12 lock protocols
  - in 6 lock depths
  - ca. 20 hours per measurement

Transaction Throughput

- Experiment
  - variation of isolation level and lock depth
  - tunable lock depth: coarser lock granularity may be chosen
  - library application
    - 10 clients are retrieving books
    - 1 client is retrieving persons
    - 2 clients are updating histories of books

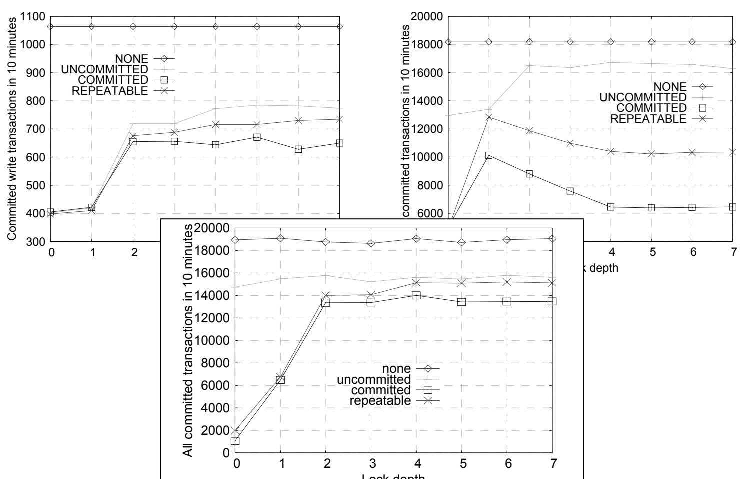
Library document consists of

- 500 persons, 25,000 books
- 6.4 MB data volume
- nearly 500,000 XML nodes



Transaction Throughput (2)

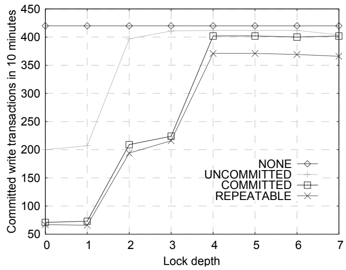
- Batch-processed transactions (no think time!)

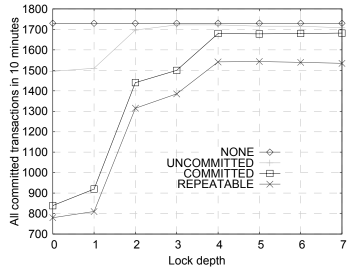


Dewey ID implementation

Transaction Throughput (3)

- Transactions with a "human in the loop"





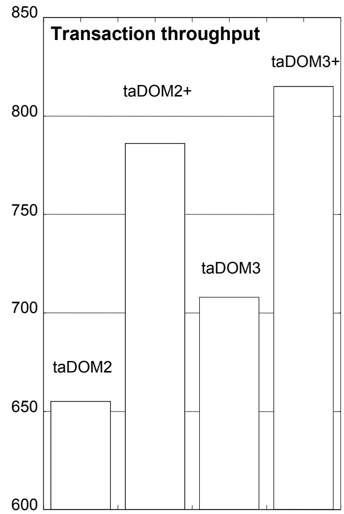
Sequential ID implementation (delay of 5 seconds before commit)

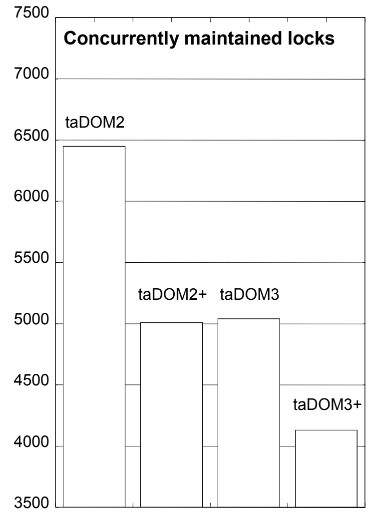
taDOM* Group – Lock Protocol Optimization

- Optimization steps
  - taDOM2+: LRIX, SRIX, LRCX, SRCX
    - determination of ancestor IDs very cheap, but that of child IDs very expensive
    - now lock mgr can ease the problem by setting implicit locks on them
  - taDOM3: NRIX, NRCX, NU, NX
    - special mode allows locking a single node without affecting the subtree att.
    - IX and CX only indicate the intention of write operations on some descendant nodes, but do not reveal information about read accesses
  - taDOM3+: LRIX, SRIX, LRCX, SRCX, LRNU, SRNU, LRNX, SRNX
- Lock conversion in taDOM3

	-	IR	NR	LR	SR	IX	NRIX	CX	NRCX	NU	NX	SU	SX
IR	IR	IR	NR	LR	SR	IX	NRIX	CX	NRCX	NU	NX	SU	SX
NR	NR	NR	NR	LR	SR	NRIX	NRIX	NRCX	NRCX	NR	NX	SU	SX
LR	LR	LR	LR	LR	SR	$NRIX_{NR}$	$NRIX_{NR}$	$NRCX_{NR}$	$NRCX_{NR}$	NU_{NR}	NX_{NR}	SU	SX
SR	SR	SR	SR	SR	SR	$NRIX_{SR}$	$NRIX_{SR}$	$NRCX_{SR}$	$NRCX_{SR}$	NU_{SR}	NX_{SR}	SR	SX
IX	IX	IX	NRIX	$NRIX_{NR}$	$NRIX_{SR}$	IX	NRIX	CX	NRCX	NX	NX	SX	SX
NRIX	NRIX	NRIX	NRIX	$NRIX_{NR}$	$NRIX_{SR}$	NRIX	NRIX	NRCX	NRCX	NX	NX	SX	SX
CX	CX	CX	NRCX	$NRCX_{NR}$	$NRCX_{SR}$	CX	NRCX	CX	NRCX	NX	NX	SX	SX
NRCX	NRCX	NRCX	NRCX	$NRCX_{NR}$	$NRCX_{SR}$	NRCX	NRCX	NRCX	NRCX	NX	NX	SX	SX
NU	NU	NU	NU	NU_{NR}	NU_{SR}	NX	NX	NX	NX	NU	NX	SU	SX
NX	NX	NX	NX	NX_{NR}	NX_{SR}	NX	NX	NX	NX	NX	NX	SX	SX
SU	SU	SU	SU	SU	SU	SX	SX	SX	SX	SU	SX	SU	SX
SX	SX	SX	SX	SX	SX	SX	SX	SX	SX	SX	SX	SX	SX

Performance Comparison of the taDOM* group



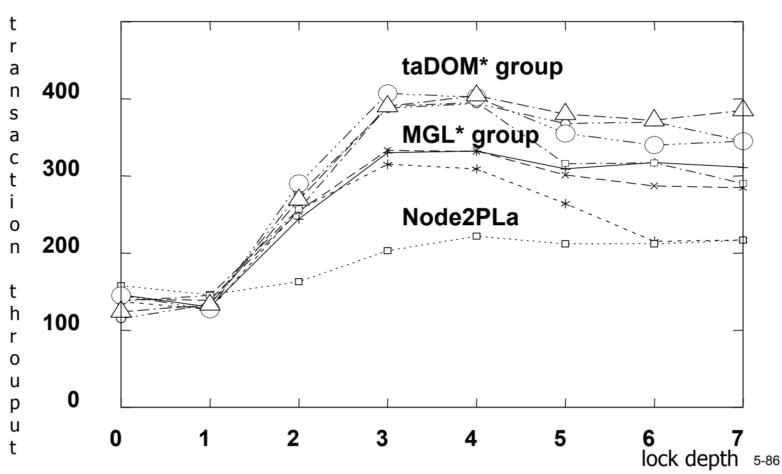


Trends in DBMSs

RM vs. XML
Architectural requirements
Node labeling
Indexing
Node processing services
Concurrency control
Lock manager
Performance measurement

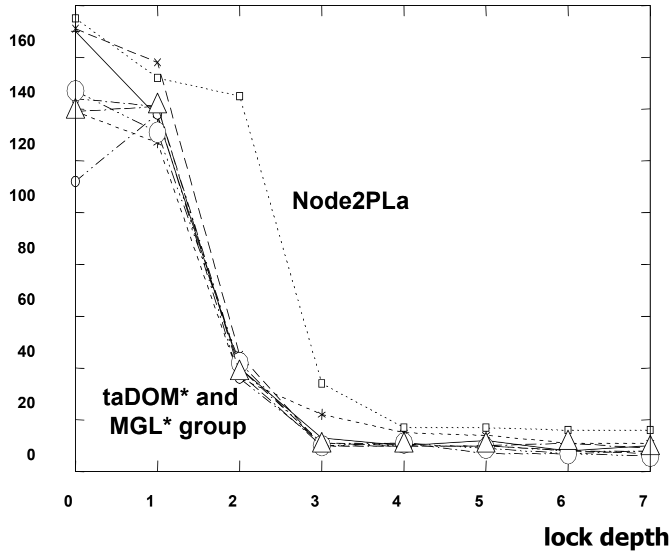
Contest of XML Lock Protocols

- Benchmark: 3 groups of lock protocols
  - *-2PL and MGL* groups
  - taDOM* group: taDOM2, taDOM2+, taDOM3, taDOM3+
  - meta-synchronization allows identical runtime environment



Contest of XML Lock Protocols (2)

- Deadlocks



Summary

- **Invariants of DB management**
  determine the mapping steps in the DBMS architecture
  - XDBMS require substantial adjustments
  - VITA data or data streams?
- DeweyIDs are the key for an efficient infrastructure
  - support for navigational operations
  - core path processing steps
  - concurrency control
- Support of 12 locking protocols
  - taDOM protocols taDOM2, taDOM2+, taDOM3, and taDOM3+
  - classic hierarchical protocols and improvements IRX, IRX+, IRIX, IRIX+, and URIX
  - alternative XML locking protocols Node2PL, NO2PL, and OO2PL
- Processing of declarative queries: various optimizations, adjustments and complements needed
  - index structures
  - query optimization
  - plan operators and cost models
  - evaluation of alternative access plans

Trends in DBMSs

RM vs. XML | Architectural requirements | Node labeling | Indexing | Node processing services | Concurrency control | Lock manager | Performance measurement

Conclusions and Outlook

- XTC is used as a test vehicle for empirical DB research
  - optimization of native XML storage models
  - effectiveness of XML concurrency control
    - fine-granular locking on nodes and edges
    - exchange of lock manager possible
    - meta-synchronization allows comparison of different compatibilities
  - performance evaluation revealed
    - cost of lock management
    - effect of isolation levels on transaction throughput
    - influence of node numbering schemes:
      - node insertions at arbitrary positions needed
      - ancestor path locking without accessing the storage engine desirable
  - mapping different XML language models
    - via access models to the XML storage model
    - to analyze the locking behavior of XQuery processing
    - new locking approach to XQuery?

Trends in DBMSs

RM vs. XML | Architectural requirements | Node labeling | Indexing | Node processing services | Concurrency control | Lock manager | Performance measurement

Further References

- *Beyer, K. S., Cochrane, R., Josifovski, V., Kleewein, J., Lapis, G., Lohman, G. M., Lyle, B., Ozcan, F., Pirahesh, H., Seemann, N., Truong, T. C., van der Linden, B., Vickery, B., Zhang, C.: System RX: One Part Relational, One Part XML. in: Proc. ACM SIGMOD Conf., Baltimore, Maryland, pp. 347-358, 2005.*
- *Härder, T., Haustein, M. P., Mathis, C., Wagner, M.: Node Labeling Schemes for Dynamic XML Documents Reconsidered, appears in Data & Knowledge Engineering, Elsevier, 2006.*
- *Haustein, M.: Eine XML-Programmierschnittstelle zur transaktionsgeschützten Kombination von DOM, SAX und XQuery, in: Proc. BTW-Konferenz, Karlsruhe, March 2005, pp. 265–284.*
- *Haustein, M.: Verhinderung von Phantomen in XML-Datenbanksystemen mit wertbasierten Achsensperren, in: Proc. Berliner XML Tage, Sept. 2005, pp. 79-92.*
- *Haustein, M. P., Härder, T.: Adjustable Transaction Isolation in XML Database Management Systems, in: Proc. 2nd Int. XML Database Symp., Toronto, Canada, Aug. 2004, LNCS 3186, Springer, pp. 173-188.*
- *Haustein, M. P.: Feingranulare Transaktionsisolation in nativen XML-Datenbanksystemen, Verlag Dr. Hut, München, Januar 2006.*
- *Haustein, M. P., Härder, T., Luttenberger, K.: Contest of XML Lock Protocols, appears in: Proc. VLDB Conf., Seoul, Sept. 2006.*
- *Mathis, Ch., Härder, T.: Hash-Based Structural Join Algorithms, appears in: Proc. DATAX'06 Workshop, Munich, March 2006.*
- *Mathis, Ch., Härder, T., Haustein, M.: Locking-Aware Structural Join Operators for XML Query Processing, in: Proc. Sigmod Conf., Chicago, Illinois, pp. 467-478, 2006.*
- *Päßler, M., Nicola, M.: Native XML-Unterstützung in DB2 Viper, Datenbank-Spektrum 17/2006, pp. 42-47.*