Realization of DBS

6. Tree-Based Access Paths

Theo Härder
www.haerder.de

Optimization techniques that reduce the number of physical I/Os are generally more efficient than those that improve the efficiency in performing the I/Os!

Main reference:
Theo Härder, Erhard Rahm: Datenbanksysteme – Konzepte und Techniken der Implementierung, Springer, 2001, Chapter 7.

Jim Gray, Andreas Reuter: Transaction Processing – Concepts and Techniques, 5th printing, Morgan Kaufmann Publ., 1993, Chapter 15.

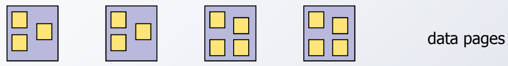
Realization of DBS

Tree-Based Access Paths

- **Goal**
  - Design principles for access paths to the records of a table, for which a search criterion is supported
  - Ways to support hierarchical access
- **Access paths for primary key**
  - Binary search trees?
  - Multi-way trees and digital trees, hash methods (chapter 7)
- **B- and B*-trees** (repetition)
- **Digital trees (**m-ary Trie, binary digital trees)
- **Addressing in trees**
  - Important for fine-granular mapping of XML documents
  - Labeling schemes for nodes should consider structure and order of the document and avoid relabeling in case of arbitrary subtree insertions
  - Support of navigation, declarative query evaluation, and locking
- **Important characteristics**
  - n = #instances of a record type, b = avg. #records/page (blocking factor)
  - q = #hits of a query, N_S = #page accesses, N_B = #leaf pages, h_B = height of B*-tree

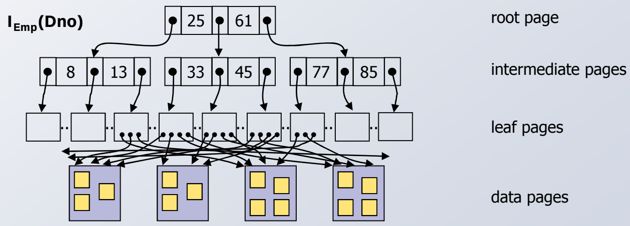
Some Important Access Methods to a Record Type

- **Table scan**



Scan (must be supported by all DBMSs!)
- is sufficient / efficient in case of:
  - small volumes of a record type (e.g., ≤ 5 pages)
  - queries returning large sets of hits (e.g., > 3% for disks)
- DBMS can apply prefetching to optimize scan operations

- **Index scan**



Requirements for Access Paths

- **Following types of accesses must be supported**
  - Sequential access to all records of a record type (scan)

    Select * From Emp
  - Sequential access in sorted sequence of an attribute

    ... Order by Name
  - Direct access via primary key

    ... Where Eno = 0815
  - Direct access via a secondary key

    ... Where Job = 'programmer'
  - Direct access via composed keys and complex search expressions (ranges, ...)

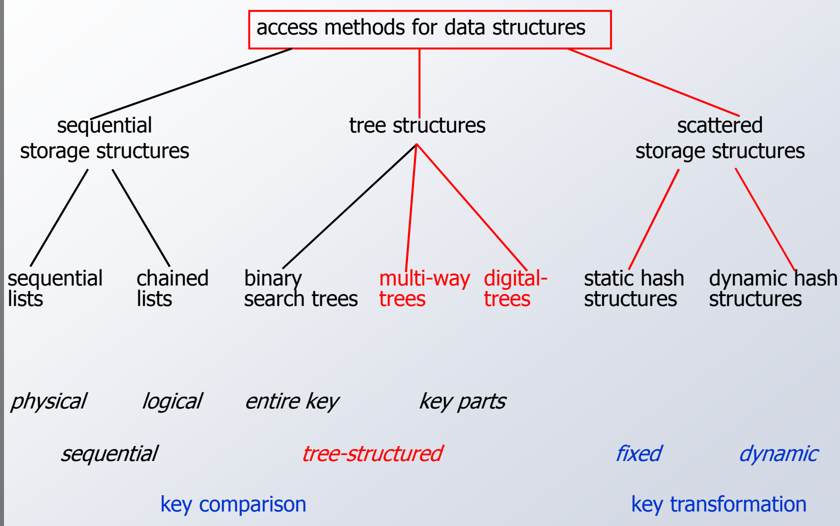
    ... Where Salary Between 50K And 100K
  - Navigational access from a record to a related set of records of the same or of another record type

    ... Where E.Dno = D.Dno

➥ If a suitable access path is missing, sequential search (scan) is needed

Classification
Primary key access
Digital trees
m-ary Trie
Binary digital trees
Why XDBMSs?
Addressing in trees
DeweyIDs for node labeling

Classification of Primary-Key Access Paths



Realization of DBS

Classification
Primary key access
Digital trees
m-ary Trie
Binary digital trees
Why XDBMSs?
Addressing in trees
DeweyIDs for node labeling

Multi-Way Trees

- Base: page = transportation unit to disk (in contrast to binary search trees)
- Ancestor: ISAM (static, periodic reorganization)
- Evolution to B- and B*-tree
  - Referenced and materialized storage of data records
  - Dynamic reorganization by splitting and merging of pages
- **Functions**
  - Direct key access and sorted sequential access (range access)
- **Balanced structure**
  - Independent of set of keys and independent of insertion sequence
- **Realization of index-organized tables**
  - Often ordered according to primary key
  - Clustering by embedded data records
- **Improvement of fan-out**
  - Key compression
  - Use of "separator keys" in B*-trees, Prefix-B-trees
- **Improvement of occupancy degree**
  - ➥ Generalized splitting method

B-Trees

Review

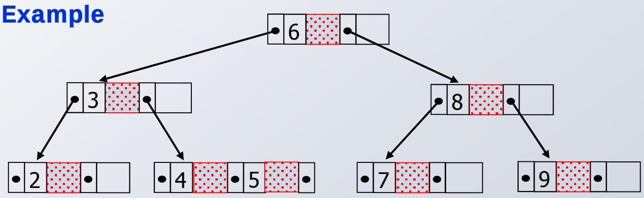
- **Def.: A B-tree of type (k, h) is a tree with the following properties**
  1. Each path from root to leaf has length h
  2. Each inner node has at least k+1 children.
     The root is a leaf or has at least 2 children
  3. Each node has at most 2k+1 children

- **Page format**

Z_0	K_1	D_1	Z_1	K_2	D_2	Z_2	...	K_m	Z_m	D_m	free

Z_i = pointer child page
K_i = key
D_i = data of the record or reference to the record (materialized or referenced)

- **Example**



8KB pages: Z=4 B, K=4 B, D=92 B => 100 B per entry => ca. 80 children
Z=4 B, K=4 B, D=4 B => 12 B per entry => ca. 680 children

B*-Trees

Review

- **Def.: A B*-tree of type (k, k*, h) is a tree with following properties**
  - Each path from root to leaf has length h
  - Each inner node has at least k+1 children. The root is a leaf or has at least 2 children.
  - Each leaf has at least k* entries.
  - Each inner node has at most 2k+1 children. Each leaf has at most 2k* entries.

- **Inner node**

Z_0	K_1	Z_1	K_2	Z_2	...	K_m	Z_m	free

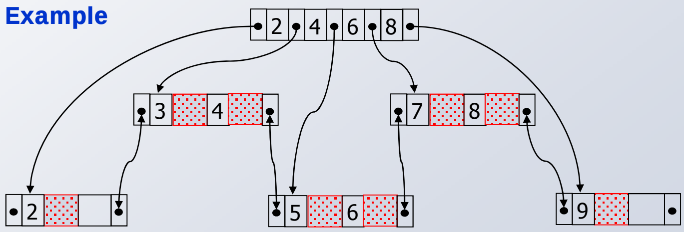
Z_i = pointer child page, K_i = key

Leaf node

V	K_1	D_1	K_2	D_2	...	K_m	D_m	free	N

D_i = reference to record (materialized or referenced)
N = successor pointer, P = predecessor pointer

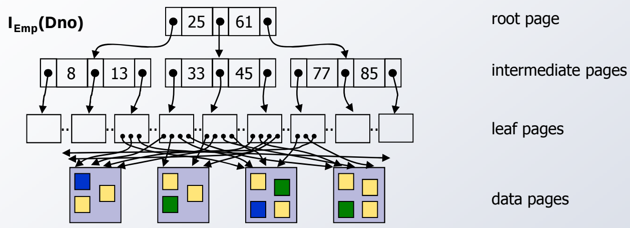
- **Example**



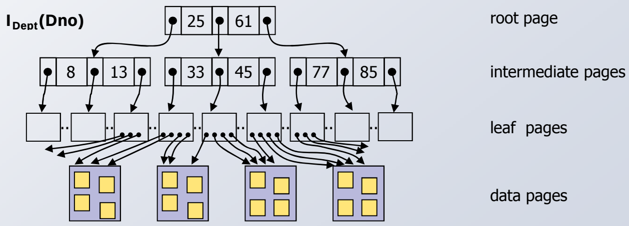
Z=4 B, K=4 B => 8 B per entry => ca. 1000 children for 8 KB pages

Unclustered vs. Clustered Access

- **Index scan without clustering**



- **Index scan with clustering**

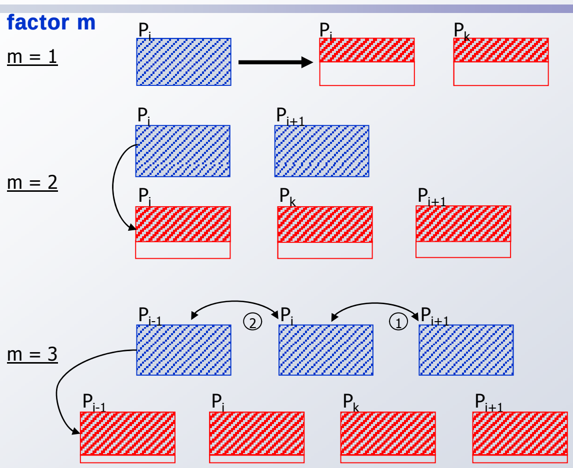


Realization of DBS

Classification
Primary key access
Digital trees
m-ary Trie
Binary digital trees
Why XDBMSs?
Addressing in trees
DeweyIDs for node labeling

Splitting in B*-Trees

- **Split factor m**



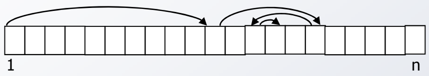
- **Occupancy**

occupancy	m=1	m=2	m
worst case	$\frac{1}{1+1}$	$\frac{2}{2+1}$	$\frac{m}{m+1}$
avg. case:	ln 2 (69%)		$m \cdot \ln\left(\frac{m+1}{m}\right)$

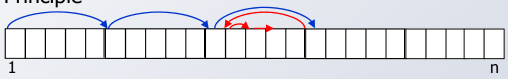
➥ m ≤3:
otherwise too expensive

Search in a Page (Internal structure is a list with n entries)

- **Sequential search**
  - Sorted or unordered set of keys: $C_{avg}(n) \approx n/2$
  - *Only minor improvements for sorted lists (in case of unsuccessful search)*
- **Binary search** essentially more efficient (Divide-and-Conquer strategy)



  - Assumption: sorted order and entries of fixed size
  - $C_{avg}(n) \approx \log_2(n+1)-1$ for large n
- **Jump search**
  - Assumption: sorted order and entries of fixed size
  - Principle



  - At first, the list is traversed in jumps of m entries, to localize the section which potentially contains the requested key
  - Then, the key is searched according to some method in the given section
  - $C_{avg}(n) = \frac{1}{2}a \cdot \frac{n}{m} + \frac{1}{2}b(m-1)$ if a jump costs *a* units and a comparison *b* units
  - What is the optimal jump size m?

Digital Trees

- **So far: always comparison of the entire key**

  In digital search trees or digital trees, for short, comparisons in tree nodes are performed to determine the search path not according to the entire key, but according to subsequent key fractions. Each differing sequence of key fractions results in a separate search path in the tree; all keys with the same prefix have the same search path for the length of the prefix.

  ↪ **Organization of the digital tree and search in the tree occur according to "key fractions"**

- **Digital search trees - principle**
- **m-ary Trie (detour)**

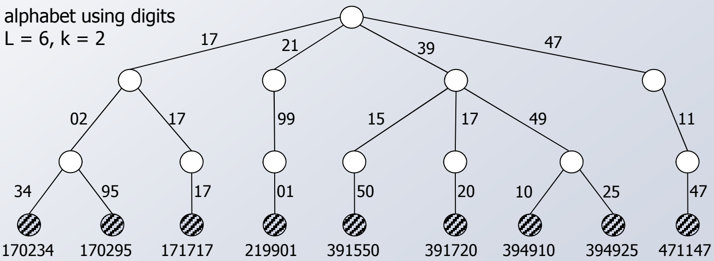
  General alphabet
  - Trie representation
  - Base operations
  - Improvement of space occupancy
  - Digital tree having a variable node format
- **Binary digital tree**

  Binary alphabet
  - Binary digital search tree
  - PATRICIA tree: avoidance of one-way branching
  - Binary Radix tree: improvement of lookup opportunities

Digital Trees – The Idea

- **Principle**
  - Decomposition of the key in fractions
  - Tree construction according to key fractions
  - Search in the tree by comparison of key fractions
- **What are key fractions?**
  - Key consists of L characters of an alphabet
  - Key fractions can be formed by bits, digits, characters as elements of an alphabet
  - But also aggregations of these basic elements can be used (e.g., syllables of length k)
  - Longest path in the tree + 1 = height of the tree = L/k + 1, if L is the key length and k is the length of the key fractions
- **Conceptual representation of a digital tree**

alphabet using digits
L = 6, k = 2

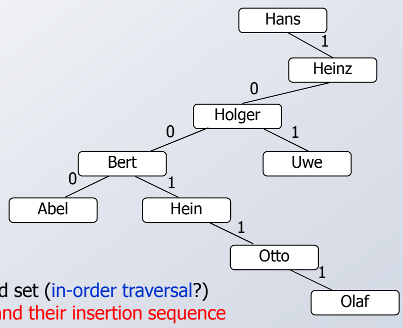


➥ max. degree of the digital tree m = 100

Binary Digital Trees (Binary Alphabet)

- **1. variant: binary digital search tree**
  - A complete key is stored in each node - similar to a binary search tree
  - Upon insertion, a key obtains the first free leaf node located via its bit sequence
  - For the decision, whether the left or right branch is used in a node if the stored key does not match the search key, the single bits of the search key are tested in the sequence they occur

HANS = 1 0 0 1 0 0 0 ...
HEINZ = 1 0 0 1 0 0 0 ...
HOLGER = 1 0 0 1 0 0 0 ...
BERT = 1 0 0 0 0 1 0 ...
...
OTTO = 1 0 0 1 1 1 1 ...
...



- **Evaluation**
  - No representation of an ordered set (in-order traversal?)
  - Dependent on the set of keys and their insertion sequence
  - Long one-way branches, no dynamic balancing

  ➥ **balanced trees are better:** instead of the bit sequence of K_i use a random number with K_i as seed
- **Application:** static set of keys with strongly weighted access frequencies

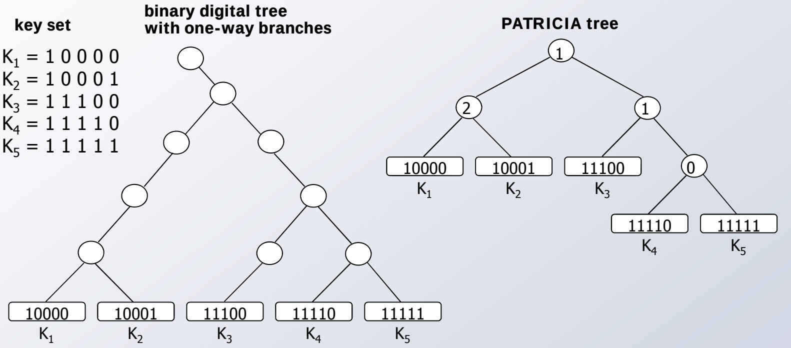
Classification
Primary key access
Digital trees
m-ary Trie
Binary digital trees
Why XDBMSs?
Addressing in trees
DeweyIDs for node labeling

Binary Digital Trees (2)

- **2. variant: PATRICIA tree** (Practical Algorithm To Retrieve Information Coded In Alphanumeric)
  - **Basic idea:** avoidance of one-way branches
  - Storage of keys in the leaves
  - **Inner nodes:** maintain how many bits have to be skipped for the path selection test
  - Construction principle

key set

K_1 = 1 0 0 0 0
K_2 = 1 0 0 0 1
K_3 = 1 1 1 0 0
K_4 = 1 1 1 1 0
K_5 = 1 1 1 1 1



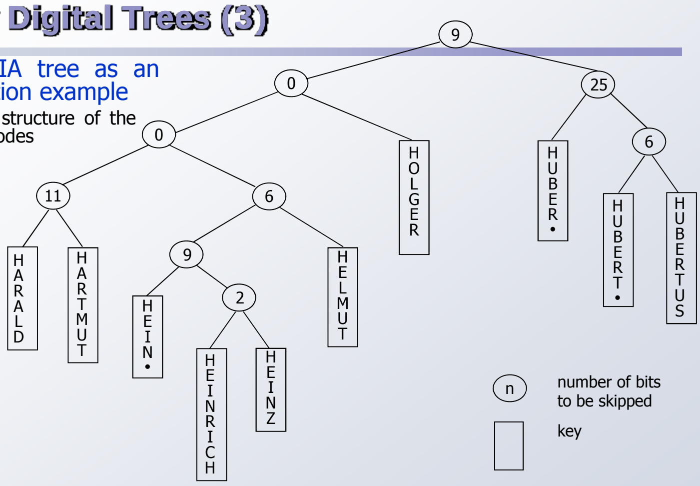
- **Evaluation**
  - There are no one-way branches
  - Otherwise, however, similar to the binary digital search tree
  - Tree structure can be understood as test procedure for search keys. For each key, the test sequence must be completely checked before success or failure is decided

Realization of DBS

Classification
Primary key access
Digital trees
m-ary Trie
Binary digital trees
Why XDBMSs?
Addressing in trees
DeweyIDs for node labeling

Binary Digital Trees (3)

- PATRICIA tree as an application example
  - Simple structure of the inner nodes



- How does search proceed for key
  HEINZ = X’1001000100010110010011001110101101 0’ ?

- How has to be tested if search goes for
  ABEL = X’100000110000101000101100110 0’ ?
  ➥ successful and failed search ends in a leaf node

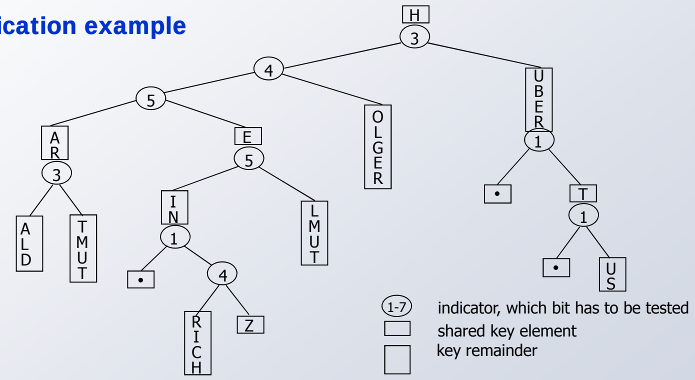
Classification
Primary key access
Digital trees
m-ary Trie
Binary digital trees
Why XDBMSs?
Addressing in trees
DeweyIDs for node labeling

Binary Digital Trees (4)

- **3. variant: binary Radix tree**

  As modification of the PATRICIA Trie
  - Storage of test information
  - Additionally storage of variable-length key fractions in inner nodes, as soon as they can be factored out as prefixes for the keys of the related subtree

- **Application example**



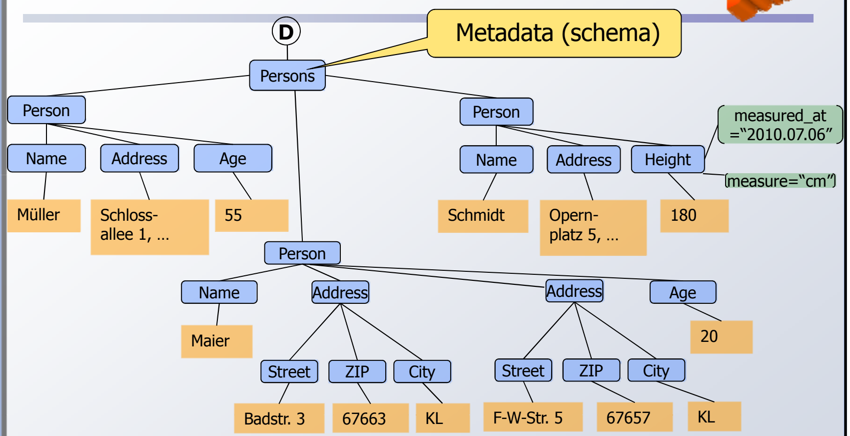
HEINZ = X’10010001000101100100110011101011010’

- More complex node formats and more expensive search and update operations
- Failed search can be frequently stopped in an inner node

Classification
Primary key access
Digital trees
m-ary Trie
Binary digital trees
Why XDBMSs?
Addressing in trees
DeweyIDs for node labeling

Mapping: XML ←→ Relational Model

Detour

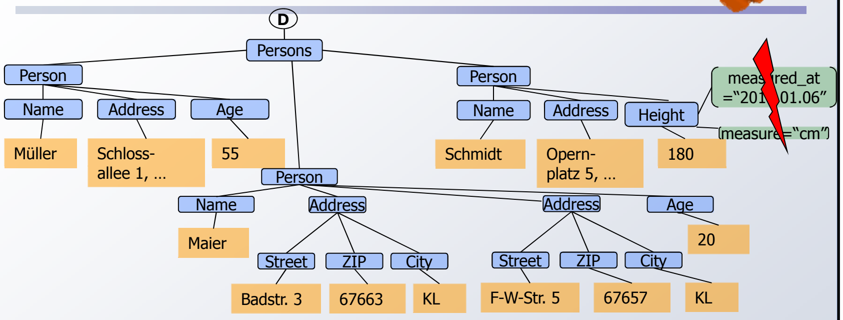


Person			
Name	Address	Age	Height
Müller	Schlossallee 1,..	55	--
Maier	?	20	--
Schmidt	Opernplatz 5, ...	--	180

RM mapping in a table is not possible, if object description has
- more than 3 levels,
- multi- or relation-valued attributes,
- aspects (attributes of elements)

Mapping: XML ←→ Relational Model (2)

Detour



Person

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

Address

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

RM mapping across several tables

- is very complex and incomprehensible,
- must preserve order,
- is also called "Shredding"

Classification
Primary key access
Digital trees
m-ary Trie
Binary digital trees
Why XDBMSs?
Addressing in trees
DeweyIDs for node labeling

Why XML Data Model? It's the Flexibility, Stupid!

Detour

- Flexibility
  - Data mapping
  - Cardinality variations
  - Optional or non-existing structures

No need for atomic values

- Potential for data integration and evolution
  - Every industry uses large and evolving sets of sparsely populated attributes (elements)
  - Financial companies defined >10 XML schemata and vocabularies
    - To standardize data processing
    - To leverage cooperation and data exchange
- Domain- or application-specific standardization
  - Facilitates intra- and inter-organization cooperation
  - With a precise understanding of the data

Classification
Primary key access
Digital trees
m-ary Trie
Binary digital trees
Why XDBMSs?
Addressing in trees
DeweyIDs for node labeling

Why XML DBMSs?

Detour

- XML defined for message exchange
  - Messages are data, too
  - Large volumes of messages and data
  - Avoid conversion

→ XDBMSs: unified management for messages and data

- Transaction-safe document processing
  - Support of cooperative and concurrent multi-user operations
  - Example: Financial Application Logging
    - 10M to 20M inserts of heterogeneous data in 24h
    - 500 peak inserts/sec
    - Concurrently: > 100 users read the data for troubleshooting and auditing tasks
    - Short response times

→ Performance is not everything,
but without performance everything is worth nothing!

XML Applications Need DBMS Support

Detour

xml.cover-pages.org

OWL, TEI, Bergen MLCD Project, MASTER, GDA, EMELD, ETCSL, XSTAR, METS, IMAGES, EAD, EAC, LEAF, Based on XML, RSS, OCS, DocBook, WebML, PSI, DOM, RDDL, ANZLIC, NCIP, EVA, ATLAS, e-GIF, CTML, GovML, TIGERS, OXCI, XCI, EML, Ballots, Elections, Polls, EPA, PIXIT, University of Washington, OMG, CWMI, MDA, OIM, DCMI, VocML, OAI-PMH, PRISM, PICS, XGMML, SGF, GXL, PNML, OPML, WSP, XMTP, OSD, LOGML, Extensible Log Format, RML, XMPP, CPIM, PIDF, IETF, XMSG,

Short Message Service, MAXML, XDNL, DRP, MatML, MoDL, BSML, BIOML, GEML, GeneXML, GAME, LSID, MAGE-ML, MAML, MSAML, SBML, PROXIML, VHG, OMF, XTHML, OPX, OFX/OFE, IFX, IRML, XFRML, XBRL, VRXML, FpML, TWIST, MDDL, MDML, WeatherML, RIXML, daliML, STPML, tpaML, IOTP, JAXM, JAXR, DRM, DPRL, XrML, ODRL, DOI, XACML, EPAL,

XMCL, EBX, ITML, EPP, XNS, DMML, IETF/W3C, XKMS, XCBF, SAML, WS-Security, S2ML, XACL, IDMEF, IODEF,

IOTP, DOMHASH, SDML, FSML, ECML, BIPS, SML, RETML, Real Estate Listing Markup Language, Real Estate Standards, CRTML, CPEX, STAR, SML, ebXML, UBL, XBDL, DRIVE, PML, GCI, COE, EDXL, MathML, RDL, SMIL, MPML, DIDL, CPXe, XMP, SVG,

PGML, VML, IML, Virtual Reality Modeling Language, XML-Based DSL Provisioning, WIDL, GEN, VCML, tXML, TXML, UCC, PML, GUIDE, igML, UDEF, OTA, HITIS, ICE, cXML, mpXML, qbXML, OCP, eCX, Electronic Business Card, HML, ADIS, xNL, xAL, CIML, NAML, HEML, xCal, tML, TCIF/IPI, bcXML, gbXML, PDML, PDX, ECIX, CIDS, TDML, EDA, UXF, JAXB, XLIFF, DESSERT, Bitstream Inc., MPEG-7, CIM, SMI-S, DCML, XTND, Bayesian Networks, PMML, MULECO, RDF, OIL, MDL, XML, ORM-ML, DAML, RoboML, RuleML, BRML, BPML, AORML, XRML, SRML, RFML, IFF, SHOE, DLML, CBML, AIML, PML, PIF-XML, GML, DNF, POIX, XMML, NVML, XDF, ADC, XSIL, OODT, OpenDocument, AIML, PhysicsML, NAA, NITF, NML, NFF, CFML, ESI, DCD, DDML, CharMapML, DASL, DITA XML, DTB, XPP, JDF, PPML, PrintML, PCX, IMS, SCORM, LMML, SIF, TML, DML, CCXML, CPL, CPML, VoiceXML, SALT, TML, MATE, CELLAR, ATLAS, XTML, JSML/JSpeech, PMXML, XRL, ADML, HumanML, ThML, XSEM, OSIS, 'XML for FAX', XFDL, XFA, EFS, BML, BHTML, OSP, DSML, BEEP, OPES, LOTP, SMI, xCBL, UCLP, NAXML, SOX, XBEL, SODL, WS-I, SOAP, UDDI, WS-Addressing, WSIL, WSCL, WSDL, WSCI, WSIA, WSFL, WSUI, WSRP, WSXL, BPEL4WS, DIME, XAML, AML, XER, OOPML, eCTD, NLM, XMLEPR, DTDs, TDL, HRMML, SIDES, BML, KBML, JigXML, Media Object Server - XML, Formal Language for Business Communication, ETD-ML, XUL, XAML, XBL, UIML, PSL, AISI, SML, ETSG, PIDX, POSC, PIPE, MTML, gXML, SM X, ChessML, MRML, …

ACID properties and XQuery eval. have to be guaranteed!

→ here flexible implementation concepts!

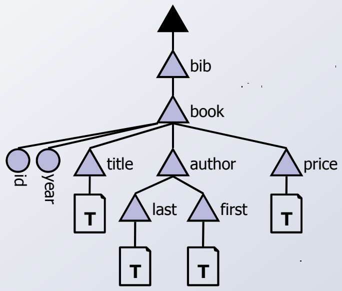
Classification
Primary key access
Digital trees
m-ary Trie
Binary digital trees
Why XDBMSs?
Addressing in trees
DeweyIDs for node labeling

Introduction to DOM (Document Object Model) Detour

- 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 (...)

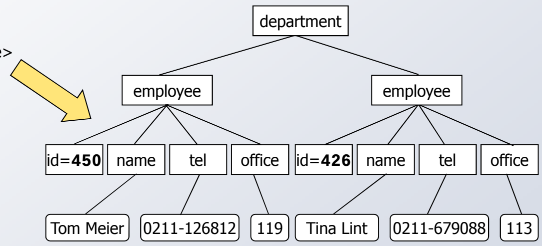
Classification
Primary key access
Digital trees
m-ary Trie
Binary digital trees
Why XDBMSs?
Addressing in trees
DeweyIDs for node labeling

Native XML Storage Structures

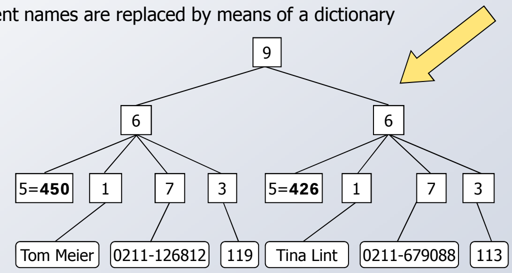
Conceptual XML mapping to a fine-grained storage structure

Transformation into an internal XML tree

```
<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>
```



Element names are replaced by means of a dictionary



SYSIBM:SYSXMLSTRINGS

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

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

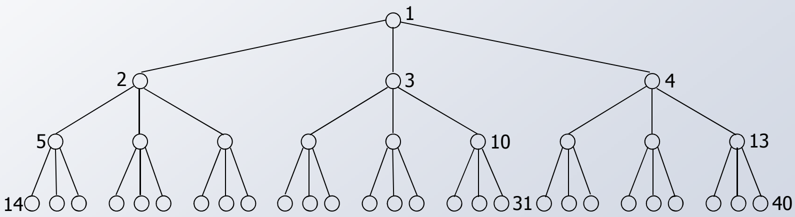
Holistic Support of all Internal XDBMS Operations

- Node Labeling
  - Representation of an XML document: ordered, labeled tree with nodes of type element, attribute, text
- Specific support needed
  - 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**

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)



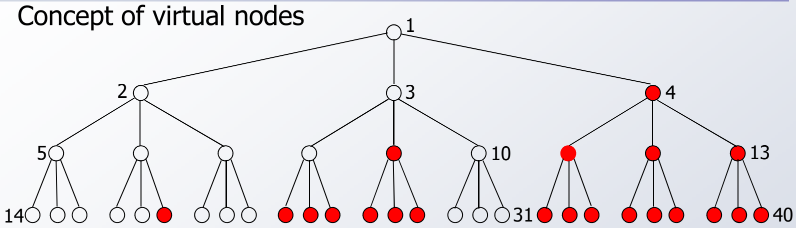
  - Pre-analysis required to determine max (k)
  - Real documents are incomplete k-ary trees

Realization of DBS

Classification
Primary key access
Digital trees
m-ary Trie
Binary digital trees
Why XDBMSs?
Addressing in trees
DeweyIDs for node labeling

Node Labeling – Early Requirements (2)

- Concept of virtual nodes

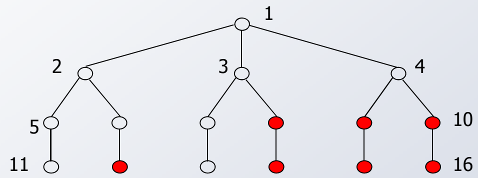


  - parent (cn, k) = ceil ((cn – 1)/k)
  - child (cn, k) = cn*k – (k-1) + 1, cn*k – (k-2) + 1,
    cn*k – (k-i) + 1, …, cn*k – 1 + 1, cn*k + 1
  - ancestor (cn, k) = parent (cn, k), parent (parent (cn, k), k), …
  - descendant (cn, k) = child (cn, k), child (child (cn, k), k), …
  - sibling (cn, k) = child (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 if 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



metadata

$k_1 = 3$

$k_2 = 2$

$k_3 = 1$

⇨ **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

Node Labeling – New Requirements

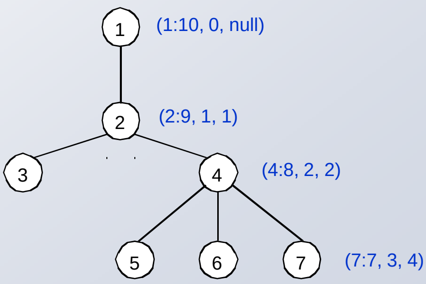
- Support of **dynamic** XML documents
  - All axes relationships should be evaluated without accessing the document
  - Internal navigation operations should help to optimize declarative queries
  - Multi-lingual XML interfaces require navigational support (e.g., DOM and SAX)
  - Labeling scheme should 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**

Classification
Primary key access
Digital trees
m-ary Trie
Binary digital trees
Why XDBMSs?
Addressing in trees
DeweyIDs for node labeling

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



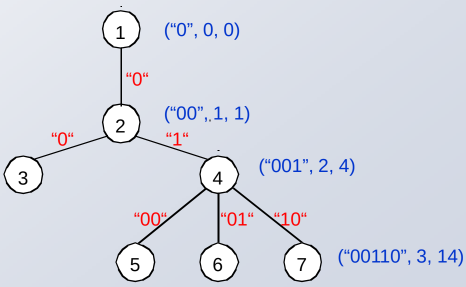
label template (LP:RP, lv, P_LP)

Realization of DBS

Classification
Primary key access
Digital trees
m-ary Trie
Binary digital trees
Why XDBMSs?
Addressing in trees
DeweyIDs for node labeling

Prefix-Based Schemes

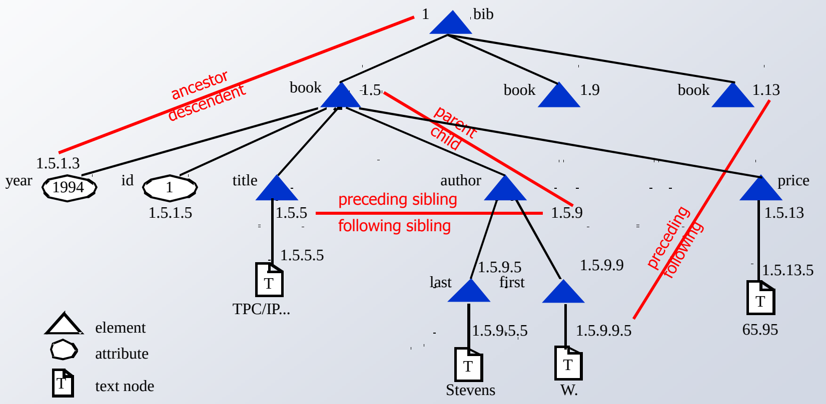
- Each node is encoded with a unique string S such that
  - S(v) is before S(u) in lexicographic order iff node v is before node u in the document order
  - S(v) is a prefix of S(u) iff node v is the ancestor of node u
- Simple example:
  - Assign to the outgoing edges of each node a set of prefix-free 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)

Prefix-Based Labeling Scheme – DeweyIDs (SPLIDs)

- DeweyIDs consist of **several division values** separated by dots
- On initial loading, only **odd** division values are assigned
- Initial assignment is controlled by parameter *distance* (= 4)
- Computation of XPath axes relationships



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

Realization of DBS

Classification
Primary key access
Digital trees
m-ary Trie
Binary digital trees
Why XDBMSs?
Addressing in trees
DeweyIDs for node labeling

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

    d_1 = **1.3.17.2.2.3.4.9** d_2 = **1.3.17.2.3.7**

  - Level determination

    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

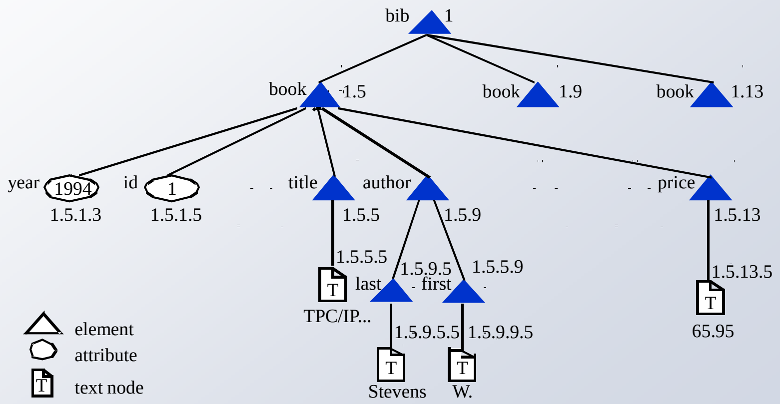
  - Ordering d_2 **?** d_1

    d_1 **<** d_2 : **1.3.17.2.2.3.4.9 < 1.3.17.2.3.7**

Classification
Primary key access
Digital trees
m-ary Trie
Binary digital trees
Why XDBMSs?
Addressing in trees
DeweyIDs for node labeling

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

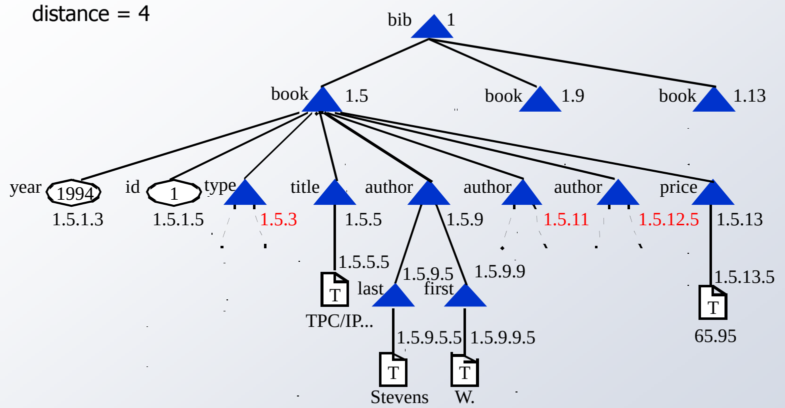


Realization of DBS

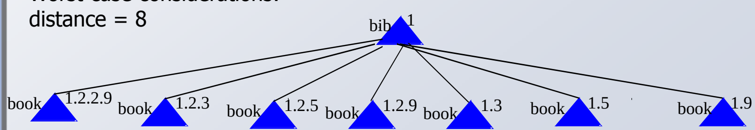
Classification
Primary key access
Digital trees
m-ary Trie
Binary digital trees
Why XDBMSs?
Addressing in trees
DeweyIDs for node labeling

DeweyIDs: Insertion of Subtrees

distance = 4



Worst-case considerations:
distance = 8



Classification
Primary key access
Digital trees
m-ary Trie
Binary digital trees
Why XDBMSs?
Addressing in trees
DeweyIDs for node labeling

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)

Classification
Primary key access
Digital trees
m-ary Trie
Binary digital trees
Why XDBMSs?
Addressing in trees
DeweyIDs for node labeling

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
E_i = encoding of i-th division
O_i = value of the 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

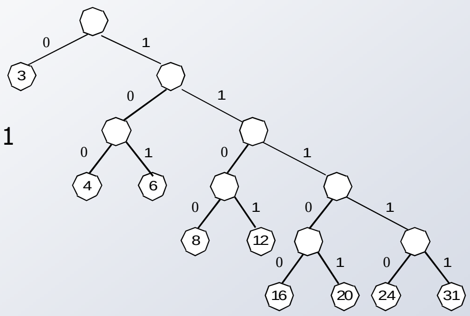
Encoding of DeweyIDs (3)

- Huffman codes

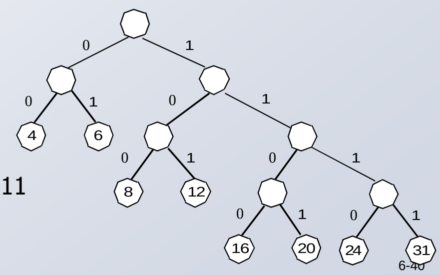
TL	C_0	E_0	C_1	E_1	. . .	C_k	E_k

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

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

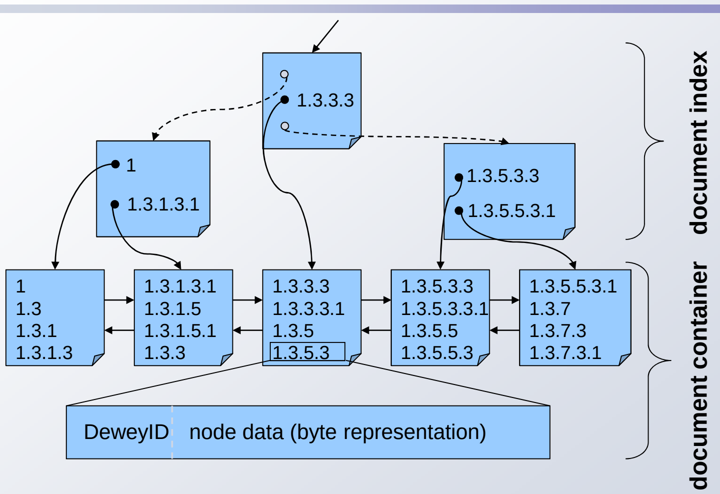
DeweyIDs – Comparison of Avg. Sizes to Max. Sizes

Document	∅-size			max-size		
	dist(2)	dist(32)	dist(256)	dist(2)	dist(256)	dist(256)
1. treebank	**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. SwissProt	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**

Realization of DBS

Classification
Primary key access
Digital trees
m-ary Trie
Binary digital trees
Why XDBMSs?
Addressing in trees
DeweyIDs for node labeling

Native XML Document Storage (XTC Approach)



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 threshold are stored in referenced mode

prefix compression works!

Summary

- **Clustering optimizes (sorted) sequential accesses**
- **Access behavior of AVL tree with $O(\log_2 n)$ is not good enough**
- **Standard access path: B*-tree** (the ubiquitous B*-tree)
  - Is not missing in any DBMS
  - Materialized and referenced storage of data records
  - Index-organized table with clustering
- **Index structure as B*-trees**
  - Can be specified with and without clustering
  - Balanced structure independent of set of keys and insertion sequence

  ➥ Dynamic reorganization by splitting and merging of pages
  - Direct key access to an indexed record
  - Sorted sequential access to all records (supports range queries, join operations, etc.)

  ➥ How many Index structures/tables?
- **Digital trees**
  - No "built-in" balancing criterion
  - Proposed as path indexes for XML documents
  - Mapping onto external storage is difficult for dynamic documents
- **DeweyIDs (SPLIDs) as preferred node labeling scheme for trees**
  - Order preserving and stable in case of insertions, but variable-length entries
  - Expressive power with effective support for DB operations

Access Paths in Commercial Database Systems

DB2(IBM)	B*tree (clustered, non-clustered), partitioned tables, ...
Informix	B-tree, static hashing, ISAM, HEAP, ...
Oracle	B*-tree (with prefix-/suffix compression), (join-) clustering, ...
Sybase	B*-tree (clustered, non-clustered), ...
RDB (DEC)	B*tree (clustered, non-clustered), hashing, join clustering, ...
NonStop SQL (Tandem)	B*-tree (clustered, non-clustered) with prefix compression, ...
UDS (Siemens)	B*tree, static hashing, clustering (LIST), Inverted pointer list (Pointer-Array), CHAIN

Classification
Primary key access
Digital trees
m-ary Trie
Binary digital trees
Why XDBMSs?
Addressing in trees
DeweyIDs for node labeling

Addressing in Trees Using DeweyIDs

Initial document loading*

While a new document is loaded—typically bulk-loaded in left-most depth-first order—, the DeweyIDs for its nodes are dynamically assigned which is guided by the following rules:

1. Element root node: It always obtains DeweyID 1.
2. Element nodes: The first node at a level receives the DeweyID of its parent node extended by a division of *distance + 1*. If a node N is inserted after the last node L at a level, DeweyID of L is assigned to N where the value of the last division is increased by *distance*.
3. Attribute nodes: A node N having at least one attribute, obtains (in taDOM) an attribute root R for which the DeweyID of N extended by a division with value 1 is assigned. The attribute node yields the DeweyID of R extended by a division. If it is the first attribute node of R, this division has the value 3. Otherwise, the division receives the value of the last division of the last attribute node increased by 2. In this case, the distance value does not matter, because the attribute sequence does not affect the semantics of the document. Therefore, new attributes can always be inserted at the end of the attribute list.
4. Text nodes: A node containing text is represented in taDOM by a text node and a string node. For text nodes, the same rules apply as for element nodes. The value of an attribute or a text node is stored in a string node. This string node obtains the DeweyID of the text node resp. attribute node, extended by a division with value 1.

* T. Härder, M. Haustein, C. Mathis, M. Wagner: Node Labeling Schemes for Dynamic XML Documents Reconsidered, Data & Knowledge Engineering 60:1, pp. 126-149, Elsevier 2007; http://wwwlgis.informatik.uni-kl.de/cms/index.php?id=9

Realization of DBS

Classification
Primary key access
Digital trees
m-ary Trie
Binary digital trees
Why XDBMSs?
Addressing in trees
DeweyIDs for node labeling

Addressing in Trees Using DeweyIDs (2)

DeweyID assignment when new nodes are inserted

When new nodes are inserted at arbitrary logical positions, their DeweyIDs must reflect the intended Document order as well as position, level, and type of node without enforcing modifications of DeweyIDs already present. For element nodes and text nodes, the same rules apply. In contrast to them, attribute roots, attribute nodes, and string nodes do not need special consideration by applying rule 3, because order and level properties do not matter.

Assignment of a DeweyID for a new last sibling is similar to the initial loading. Here, the last level only consists of one division. Hence, when inserting element node *year* after price, addition of the distance value yields 1.9.33. In case, the last level consists of more than one division (indicated by even values), the first division of this level is increased by *distance - 1*. For example, the successor of 1.3.14.6.5 is 1.3.21.

If a sibling is inserted before the first existing sibling, the first division of the last level is halved and, if necessary, ceiled to the next integer or increased by 1 to get an odd division. This measure secures that the "before-and-after gaps" for new nodes remain equal. Hence, inserting a *type* node before *title* would result in DeweyID 1.9.5. If the first divisions of the last level are already 2, they have to be adopted unchanged, because smaller division values than 2 are not possible, e.g., the predecessor of 1.9.2.2.8.9 is 1.9.2.2.5. In case the first division of the last level is 3, it will be replaced by *2.distance+1*. For example, the predecessor of 1.9.3 receives 1.9.2.9.

The remaining case is the insertion of node d_2 between two existing nodes d_1 and d_3. Hence, for d_2 we must find a new DeweyID which is between the DeweyIDs of d_1 and d_3. Because they are allocated at the same level and have the same parent node, they only differ at the last level (which may consist of arbitrary many even divisions and one odd division, in case a weird insertion history took place at that position in the tree). All common divisions before the first differing division are also equal for the new DeweyID. The first differing division determines the division becoming part of DeweyID for d_2. If possible, we prefer a median division to keep the before-and-after gaps equal. Assume for example, d_1 = 1.9.5.7.5 and d_3 = 1.9.5.7.16.5, for which the first differing divisions are 5 and 16. Hence, choosing the median odd division result in d_2 = 1.9.5.7.11.

If d_4 = 1.5.6.7.5 and d_6 = 1.5.6.7.7, only even division 6 would fit. Remember, we have to recognize the correct level. Hence, having distance value 8, d_5 = 1.5.6.7.6.9.