XML Databases
10. XML Storage 1 – Overview
Silke Eckstein
Andreas Kupfer
Institut für Informationssysteme
Technische Universität Braunschweig
http://www.ifis.cs.tu-bs.de

10. XML Storage 1

10.1 Motivation

• Applications require different types of XML documents
  – Structure vs. content
  – Regular vs. irregular
• Thus, XML documents are
  – Data-centric
  – Document-centric
  – or somewhere in-between
• Questions
  – Storage of XML documents
  – Efficient processing of queries on the stored documents or data
• There are several methods for storage
  – 1st goal: Learn and understand methods
  – 2nd goal: Classify methods
    • Principles
    • Advantages and disadvantages
    • Usage

10.2 Text-based storage

10.2.1 Index structures

10.3 Model-based storage

10.4 Schema-based storage

10.5 Conclusion

10.6 Overview and References
10. XML Storage

10.1 Motivation

10.2 Text-based storage

10.2.1 Index structures

10.3 Model-based storage

10.4 Schema-based storage

10.5 Conclusion

10.6 Overview and References

10.2 Text-based storage

- The whole XML document text is stored as character data
  - File in the file system
  - CLOB (Character Large Object) in the DBS
- Operations documents as a whole are very efficient
  - Reading and writing the whole document
  - But the content is monolithic and opaque with respect to the relational query engine (query can't inspect a fragment)
- Getting granular access requires additional support
  - Full text index
  - Path index

10.2.1 Index structures

- Index structures for XML documents allow efficient access for specific queries
  - Different types of indexes are optimized for different types of queries
- Generate redundancy
  - Index has to be up-to-date by propagating data changes
- Index structures can be storage structures as well
  - They define the storage method

- Full text index
  - Indexes single words from the full text
  - Index format for unstructured parts of XML documents
  - Already known from Information Retrieval (inverted lists, tries, suffix trees, …)
- Path index
  - Indexes subtrees/paths in an XML document
  - Index format for semistructured parts of XML documents
  - Already known from object-databases (access support relations, …)

10.2.1 Index structures

- B-tree as value index for an XML fragment

- Types of index structures
  - Value index
    - Indexes atomic values of an XML document, like element content or attribute values
    - Index format for structured parts of XML documents
    - Already known from databases (B-trees, hash index, …)
  - Full text index
    - Indexes single words from the full text
    - Index format for unstructured parts of XML documents
    - Already known from Information Retrieval (inverted lists, tries, suffix trees, …)
  - Path index
    - Indexes subtrees/paths in an XML document
    - Index format for semistructured parts of XML documents
    - Already known from object-databases (access support relations, …)

- Full text index
  - Not limited to exact matches
    - Keyword-based search and boolean retrieval
    - Pattern search (with regular expressions)
  - Use of
    - Statistical, word-based methods
      - Stop word removal
      - Elimination of uncommon items
    - Linguistic methods
      - Normalization of words (e.g. capitalization, hyphenation)
      - Word decomposition by rules (engr) or dictionaries (german)
    - Stemming
    - Knowledge-based methods
      - Use of ontologies and thesauri to search for synonyms, hypernyms and hyponyms
10.2.1 Index structures

- Inverted list as full text index for XML

<table>
<thead>
<tr>
<th>word</th>
<th>occurrence</th>
<th>word position in the text</th>
</tr>
</thead>
<tbody>
<tr>
<td>book</td>
<td>2</td>
<td>123, 126, 142</td>
</tr>
<tr>
<td>text</td>
<td>3</td>
<td>124, 125, 137</td>
</tr>
<tr>
<td>genre</td>
<td>1</td>
<td>127</td>
</tr>
<tr>
<td>title</td>
<td>2</td>
<td>128, 130</td>
</tr>
<tr>
<td>author</td>
<td>3</td>
<td>129, 131, 136</td>
</tr>
<tr>
<td>place</td>
<td>1</td>
<td>133</td>
</tr>
<tr>
<td>year</td>
<td>2</td>
<td>134, 137</td>
</tr>
<tr>
<td>language</td>
<td>1</td>
<td>138</td>
</tr>
<tr>
<td>publisher</td>
<td>1</td>
<td>139</td>
</tr>
</tbody>
</table>

10.2.1 Index structures

- Path index
  - Structure information must be identifiable and reconstructable
    - Assigning the markup to the content as well as
    - Representing the hierarchical nesting and order of elements/attributes
  - Especially suited for keyword search with regard to structure or path expressions

FOR $b$ IN //book
WHERE CONTAINS($b/author, "Benjamin")
RETURN $b$

10.2.1 Index structures

- Types of path indexes
  - Nested path index
    - Access to root node from every node
  - Multi-index
    - Accessing parent nodes
  - Join-index
    - Access parent and child nodes
  - Access Support Relations (ASR)
    - Generalization of indexes above, by listing all paths in a table

bookstore -> book -> author

10.2.1 Index structures

- Conclusion
  - Efficient query processing on XML documents requires different types of index structures
  - Value index
    - For efficient access to structured parts
  - Keyword search, value search
  - Full text index
    - For efficient access to unstructured parts
  - Path index
    - Using the document structure
    - Navigating queries

10.2.1 Index structures

- Summary text-based storage
  - Schema definition:
    - Not required
  - Document reconstruction:
    - Documents stay in their original format
  - Queries:
    - Information retrieval queries
    - Processing the markup of the queries
    - XPath queries possible
  - Special features:
    - Full text functions
  - Efficiency:
    - Character string must be parsed on every access with XML processors
    - Expensive
    - No concurrency on read or write
    - No parallel processing
  - Usage:
    - Suitable to only a limited extent also for semi-structured applications
### 10. XML Storage I

#### 10.1 Motivation

#### 10.2 Text-based storage

10.2.1 Index structures

#### 10.3 Model-based storage

#### 10.4 Schema-based storage

#### 10.5 Conclusion

#### 10.6 Overview and References

---

### 10.3 Model-based storage

- **Idea:** generic storage of the graph structure
  - XML elements, XML attributes,... are nodes of a graph
  - Nesting of elements defines edges
  - Nodes get an (internal) ID based on graph traversal
- **Using relations or object classes to store elements and attributes**

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Element name</th>
<th>Value</th>
<th>Reference to preceding</th>
<th>Rank</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Attribute name</th>
<th>Value</th>
<th>Reference to element</th>
</tr>
</thead>
</table>

- **Document structure can be restored completely**
- **Extension for data type adapted storage is possible**

---

### 10.3 Model-based storage

- **Example:** list bargain buy with prices

```sql
SELECT a.content, b.content FROM Edge a, Edge b
WHERE (a.label = 'price') AND (a.content < 10.00)
AND (b.label = 'description')
AND (b.parent = a.parent) AND (a.key = b.key)
```

---

### 10.3 Model-based storage

- **DOM-based storage**
  - Information from the Document Object Model are stored in the database
  - **Storage alternatives**
    - (Object-)relational databases
    - Object-oriented databases
    - Developing own data structure

---

### 10.3 Model-based storage

- **XML queries**

```
XML queries (XPath, XQuery) are mapped to SQL queries (taking storage structures into account)
Result of XML query is generated from result of database query
- "Labeling" of the result tuples
- Result is in XML format
```

---

### 10.3 Model-based storage

- **The EDGE approach [FK99]**

![EDGE diagram](image)

- Variant BINARY: horizontal partition of EDGE based on label

---

### 10.3 Model-based storage

- **XML documents**
  - XML queries
    - XML queries are mapped to SQL queries (taking storage structures into account)
    - Result of XML query is generated from result of database query
      - "Labeling" of the result tuples
      - Result is in XML format

---

### 10.3 Model-based storage

- **DOM-based storage**
  - Information from the Document Object Model are stored in the database
  - **Storage alternatives**
    - (Object-)relational databases
    - Object-oriented databases
    - Developing own data structure

---

### 10.3 Model-based storage

- **DOM-based storage**

```
Key | id  | parent | label | content
---|-----|--------|-------|---------
D_1 | 1   | NULL   | store | NULL;
D_2 | 2   | 1      | auction| NULL;
D_3 | 3   | 2      | NULL  | NULL;
D_4 | 4   | 3      | auction| NULL;
D_5 | 5   | 4      | 6     | price  | $20;
D_6 | 6   | 5      | description | space and st...;
```
10.3 Model-based storage

DOM-based storage – example

Nodetype: ELEMENT

Nodetype: ATTRIBUTE

Nodetype: TEXT

The Autobiography of Benjamin Franklin

book

text

10.3 Model-based storage

XML Queries

– XML queries (DOM method invocations) are mapped to SQL queries (taking storage structures into account)
– Result of method invocation is generated from result of database query

Summary model-based storage

– Schema definition:
  • not required for storage
– Document reconstruction:
  • Possible, but expensive
– Queries:
  • XML queries possible
  • Adapted database queries
– Special features:
  • Querying many elements/attributes is expensive
– Efficiency:
  • Navigation from the given context is efficient
  • Restoring the document and evaluating path expressions is inefficient
– Usage:
  • For data- and document-centric as well as for semi-structured XML applications

10.4 Schema-based storage

• Motivation
  • XML content shall be stored in a conventional database
  • Accepting the loss of native access
• Problem
  • Generate DB schema automatically
  • Thereby use as much structure information as possible
• General approach for mapping from a DTD
  • Transform DTD into a tree representation
  • Nodes: element types, attributes, etc. (type layer)
  • Edges: nesting relationships of element types and their restrictions
  • Traverse tree in order to transform nodes and edges into database tables (according to certain rules)

• Generating the DB schema for a DTD:
  – Rules to map element types:
    - XML element type
    - Sequence of element types
    - Alternative of element types
    - Element type with quantifier
    - Nested element types
  – Rules to map attributes:
    - XML attribute
    - IMPLIED
    - REQUIRED
    - Default value

10.4 Schema-based storage

XML element type

Sequence of element types

Alternative of element types

Element type with quantifier

Nested element types

- column of a table
- column of a table
- column of a table
- column with null values
- setlist of columns (SET OF LIST OF)
- TUPLE OF

XML attribute

IMPLIED

REQUIRED

Default value

- column of a table
- null values allowed
- null values not allowed
- DEFAULT constraint
10.4 Schema-based storage

• Mapping to relational databases
  – DTD is usually required
  – Queries use SQL functionality
  – RDBMS data types are used (e.g. prices are NUMERIC)
  – Problem: Mapping of collection types
    • Subdivide into additional relations
      – Example:

<table>
<thead>
<tr>
<th>Comment_ID</th>
<th>Customer_info</th>
<th>Feedback</th>
</tr>
</thead>
<tbody>
<tr>
<td>44901</td>
<td>C0001</td>
<td>F0001</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0001</td>
<td>Charles Sanchez</td>
<td><a href="mailto:C.Sanchez@hotmail.com">C.Sanchez@hotmail.com</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Type</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>F001</td>
<td>opinion</td>
<td>DarjeelingSpecial</td>
</tr>
</tbody>
</table>

10.4 Schema-based storage

• Mapping with STORED (Semistructured TO RElational Data)
  – Basic Idea: Use data mining techniques on the XML structure to find a good mapping to tables [DFS99]
    – Input
      • XML documents (or an average sample of the collection)
      • Query workload
      • Restrictions of storage space, number of tables, …
      • No DTD or XML schema is required!
    – Output
      • Relational schema
      • STORED-queries: Mapping instructions for XML documents to DB tables
  – Procedure
    • Determine the XML subtrees with the largest support in the collection and in the queries
    • These subtrees are materialised in tables
    • Irregular data is stored in overflow tables according to the EDGE approach

10.4 Schema-based storage

• Mapping with STORED – example

10.4 Schema-based storage

• Mapping to object relational databases
  – DTD is usually required
  – Queries use SQL functionality
  – “Natural” mapping to tuple types, collection types
  – In case of irregular document structure databases contain many null values.

<table>
<thead>
<tr>
<th>Comment_ID</th>
<th>&lt;Customer_info&gt;</th>
<th>Feedback</th>
</tr>
</thead>
<tbody>
<tr>
<td>44901</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

10.4 Schema-based storage

• Mapping of recursive data definitions
  – DTDs can be recursive
  – Infinite recursion is impossible on instance layer of a database
  – Procedure:
    • Marking the nodes
    • Subdividing into separate tables
    • Use primary and foreign keys in RDBMS
    • Use reference types in ORDBMS
### 10.4 Schema-based storage

**Mapping of element sequences**
- Sequence can be important
  - Use an additional attribute in these cases
  - Example:

  ```xml
  <lecture>
  <lesson>Introduction</lesson>
  <lesson>XML basics</lesson>
  ...
  </lecture>
  ```

**Example:**
- Use an additional attribute in these cases

<table>
<thead>
<tr>
<th>Order</th>
<th>Lesson</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Introduction</td>
</tr>
<tr>
<td>2</td>
<td>XML basics</td>
</tr>
</tbody>
</table>

**Example:**
- Use an additional attribute in these cases

### 10.4 Schema-based storage

**Variant 1 – all alternatives in one table**

```xml
<car>
  <category>Compact Car</category>
  < ผู้
```

- Problem: many null values (wasting storage space)

### 10.4 Schema-based storage

**Variant 2 – subdivided into multiple tables**

```xml
<pension>
  <name>Hotel</name>
  <name>Restaurant</name>
  <name>Car Hire</name>
</pension>
```

- For queries, combination of tables is needed

### 10.4 Schema-based storage

**Variant 3 – Using column type XML**

```xml
<Extension>
  <name>Hotel</name>
  <name>Restaurant</name>
  <name>Car Hire</name>
</Extension>
```

- XML type allows XML queries or DOM methods
### 10.4 Schema-based storage

**Mapping of mixed content**
- Mapping to plain tables is ill-suited
- Use variant 3 from above or

**Content model ANY is not representable at all**
- Arbitrary content, arbitrary element types
- Often the fitting storage structure can only be decided on instance layer

**Content model ANY is not representable at all**
- Arbitrary content, arbitrary element types
- Often the fitting storage structure can only be decided on instance layer

**Mapping solutions with different specializations**
- Algorithms, middleware, commercial applications, ...
- Varying amount of required input or user decisions
- Many algorithms create different database schemas

**Two phases**
- Mapping
  - Assign a place for each node type in the DB
- Shredding
  - Import the XML data as DB tuples

<table>
<thead>
<tr>
<th>Algorithm/product</th>
<th>based on: n/a</th>
<th>DTD schema</th>
<th>restrictions key, cardin. types</th>
<th>DTD optimisation</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM DB2 XML Access</td>
<td><img src="image_url" alt="image" /></td>
<td><img src="image_url" alt="image" /></td>
<td><img src="image_url" alt="image" /></td>
<td><img src="image_url" alt="image" /></td>
</tr>
<tr>
<td>Apache Xerces and Xindice XMarkit</td>
<td><img src="image_url" alt="image" /></td>
<td><img src="image_url" alt="image" /></td>
<td><img src="image_url" alt="image" /></td>
<td><img src="image_url" alt="image" /></td>
</tr>
<tr>
<td>Apache Xindice XMarkit + XQuery</td>
<td><img src="image_url" alt="image" /></td>
<td><img src="image_url" alt="image" /></td>
<td><img src="image_url" alt="image" /></td>
<td><img src="image_url" alt="image" /></td>
</tr>
<tr>
<td>Oracle Xerces and Xindice XMarkit</td>
<td><img src="image_url" alt="image" /></td>
<td><img src="image_url" alt="image" /></td>
<td><img src="image_url" alt="image" /></td>
<td><img src="image_url" alt="image" /></td>
</tr>
<tr>
<td>Oracle Xerces and Xindice XMarkit + XQuery</td>
<td><img src="image_url" alt="image" /></td>
<td><img src="image_url" alt="image" /></td>
<td><img src="image_url" alt="image" /></td>
<td><img src="image_url" alt="image" /></td>
</tr>
<tr>
<td>Informix XML Access</td>
<td><img src="image_url" alt="image" /></td>
<td><img src="image_url" alt="image" /></td>
<td><img src="image_url" alt="image" /></td>
<td><img src="image_url" alt="image" /></td>
</tr>
<tr>
<td>Manual</td>
<td><img src="image_url" alt="image" /></td>
<td><img src="image_url" alt="image" /></td>
<td><img src="image_url" alt="image" /></td>
<td><img src="image_url" alt="image" /></td>
</tr>
</tbody>
</table>

**Algorithmic approaches**
- Shredder can be part of the DB
  - Usually requires an XML schema
  - In the IBM Data Studio, the shredder is part of the "annotated XML schema decomposition"
- Direct approach in DB2:
  - register the XML schema and call the stored procedure:
    ```sql
    register xmlschema http://our.org/custacc from
doc_files/custacc.xsd as cust_schema ;
    complete xmlschema cust_schema enable decomposition ;
call SYSPROC.XDBDECOMPXML ('VRODRIG', 'CUST_SCHEMA', ? ,
?, 1, null, null, null)
    ```

**Shredding without XML schema in DB2**
- XMLTABLE function in combination with an INSERT
  ```sql
  INSERT INTO ENVELOPEXT (MAILFROM, MAILTO, MAILDATE, SUBJECT)
  SELECT MAILFROM, MAILTO, MAILDATE, SUBJECT
  FROM XMLTABLE(
      XMLNAMESPACES('http://www.sal.com/mails' AS "email"),
      '$doc/email:mails/mail' (: some xquery-expression :)
    PASSING xml-source AS "doc"
    COLUMNS
    MAILFROM VARCHAR (100)PATH 'envelope/from',
    MAILTO VARCHAR (100)PATH 'envelope/to',
    MAILDATE VARCHAR (30) PATH 'envelope/email:Date',
    SUBJECT VARCHAR (100)PATH 'envelope/Subject') AS T;
  ```
10.4 Schema-based storage

- **Summary Schema-based storage with automatic mapping**
  - **Schema definition**
    - Is usually required and analysed
    - Not required, e.g. for STORIED
  - **Document reconstruction**
    - Limited (requires logging of the mapping process)
  - **Queries**
    - Database queries
    - XML queries possible, but the XPath horizontal axes, e.g. following, preceding, in.
  - **Special features**
    - Federation with existing databases is possible
    - **High efficiency** by using the DB-engine
  - **Usage**
    - For data-centric XML applications, but with limited nesting

## User defined mapping

- **Idea**
  - In all previously shown methods it is not possible to affect the storage in the DB.
  - With user defined mappings the user defines the storage structure.
  - The structure of XML documents and database schema can be designed independently.
  - Also possible storing XML documents in existing databases.
- **Annotation of DTD and XML schema, respectively**
  - In many cases the mapping definition is combined with existing schema information.
  - Only limited XML queries possible.
  - Logging of the mapping process from XML documents to databases.
  - For a given query all relevant data has to be stored (lossless mapping)

## Example

### XML document

**mapping instruction**

- **XML document**
  - Name
  - Number
- **mapping instruction**
  - Account
  - Balance

### Example for XML-DBMS (Roland Bourret)

```
<SchemaMap>
  <ElementType Name="sales:SalesOrder"/>
  <ToClassTable>
    <ToColumn Name="SONumber"/>
    <ToColumn Name="Number"/>
  </ToClassTable>
  <PropertyMap>
    <ClassMap>
      <PropertyMap>
        <ClassMap>
          <AttrMap>
            <AttrMap>
              <AttrMap>
              </AttrMap>
            </AttrMap>
          </AttrMap>
        </ClassMap>
      </PropertyMap>
    </ClassMap>
  </PropertyMap>
</SchemaMap>
```

### Remarks

- Many different mapping languages or schema annotations
  - Automatic mappings usually have an internal mapping language.
  - Remember the mapping constructs from lecture 5 and 6. The SQL/XML annotations are a mapping language, too.
  - DB2 uses similar annotations as SQL/XML.
  - On the next slide, the example from lecture 6 is shown with DB2 syntax.
10.4 Schema-based storage

- Summary schema-based storage with user defined mapping
  - Schema definition:
    - Depends on mapping language
  - Document reconstruction:
    - Not possible in most cases (requires logging of the mapping process)
  - Queries:
    - Database queries
    - XML queries in rare cases only!
  - Special features:
    - Integration with existing databases is possible
  - Efficiency:
    - High efficiency by using the DB-engine
  - Usage:
    - For data-centric XML applications

10.5 Conclusion

- Different methods for storage of XML documents
  - Text-based
    - Storing whole XML documents as string
    - Can use full text index or path index
  - Model-based
    - Generic mapping of the tree structure
  - Hybrid approaches
    - Combination of some of those methods
  - No algorithm has the optimal solution for all kind of XML documents
  - Reasonable solution is heavily dependent on the application

10.6 References

- "XML und Datenbanken" [Tür08]
  - Can Türker
  - Lecture, University of Zurich, 2008
- "XML und Datenbanken" [KM03]
  - M. Klettke, H. Meier
  - dpunkt.verlag, 2003
- "Generierung eines adaptiven Datenbankschemas für datenzentrierte XML-Dokumente" [Bus08]
  - Carsten Busche
  - Diplomarbeit, TU Braunschweig, 2008
- [FK99]
- [DFS99]