XML Data Management (2)

Mapping XML documents to databases
- Generic
  - XML type
  - DTD oriented mapping
  - User Defined Mappings

The mapping problem
- Tree model of XML documents does not fit to table model of RDB
- Variability of structures makes uniform mapping difficult
  <text> ...This is a very long text.......... </text>
- “data centric”
  <orders>
    <order ID= 4330>
      <itemList>
        <item ID>
          <price>
            <quantity>
              <description>...A not so long txt</description>
            </quantity>
            </price>
          </item>
        </itemList>
      </order>
    </orders>

Why mapping to a DB?
- Non functional characteristics of DBS
  - Fault tolerance
  - Concurrent access
  - Stability
- Searching and transforming data (to some extent) are at the heart of every DBS
- Performance
- .....depends on types of operations and data
  - Searching in large text strings?
  - Tree traverse?
  - Joms?

Classification

Example document

```
<Example document>
  <Movie Title="American Beauty" RunningTime="121"
imdbCode="0169547" Rating="R">
    <Director>
      <First>Sam</First>
      <Last>Mendes</Last>
    </Director>
    <PlotSummary>Lester Burnham is in a mid mid-life crisis...</PlotSummary>
    <Cast>
      <Actor Role="Lester Burnham">
        <First>Kevin</First>
        <Last>Spacey</Last>
        <Award From="Oscar" Category="Best Actor"></Award>
      </Actor>
      <Actress Role="Carolyn Burnham">
        <First>Annette</First>
        <Last>Bening</Last>
        <Award From="BAFTA" Category="Best Actress"></Award>
      </Actress>
      <AwardFrom="Oscar" Category="Best Film">
      </Award>
      <Award From="BAFTA" Category="Best Film"></Award>
    </Cast>
    <AwardFrom="Oscar" Category="Best Director">
    </Award>
    <Award From="BAFTA" Category="Best Director"></Award>
    <AwardFrom="Oscar" Category="Best Film">
    </Award>
    <Award From="BAFTA" Category="Best Film"></Award>
  </Movie>
</Example document>
```

DTD for Movie DB

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT Actor (First,Last,Award+)>
<!ATTLIST Actor Role CDATA #REQUIRED>
<!ELEMENT Actress (First,Last,Award)>
<!ATTLIST Actress Role CDATA #REQUIRED>
<!ELEMENT Award EMPTY>
<!ATTLIST Award Category CDATA #REQUIRED
   From NMTOKEN #REQUIRED>
<!ATTLIST Movie Category CDATA #REQUIRED
   From NMTOKEN #REQUIRED>
<!ELEMENT Cast (Actor,Actress)>
<!ELEMENT Director (First,Last,Award)>
<!ELEMENT Movie (Director,PlotSummary,Cast,Award+)>
<!ATTLIST Movie Rating NMTOKEN #REQUIRED
   RunningTime NMTOKEN #REQUIRED
   Title CDATA #REQUIRED
   imdbCode NMTOKEN #REQUIRED
   >
<!ELEMENT PlotSummary (#PCDATA)>
```
Generic Mapping

Simple relational:
- element- attribute- and edges relations preserve order of elements

DOM oriented:
- map class structure of DOM model onto relations or classes (oo or OR DBS)

Simple

- Element table
  - An element row for each element in the document
  - A generated document id
  - A generated element id
  - Predecessor of element node
  - Order of children
  - Value, if element has a value

- Attribute table
  - Row for each (attribute, value) pair in document
  - Element id of this attribute
  - Attribute name
  - Attribute value
  - Order (if needed)

Simple generic mapping

- Advantage
  - Independent of document structure
  - Simple database structure

- Disadvantage
  - MANY joins to reconstruct the document
  - Attribute types
    - No problem if only "string" type represented
    - Cast to other types: inlining with one value column per type or value table for each type

Alternative representation of order

- Coding of node order with parent-successor relation
  - preorder
    - does not represent parent child relationship
    - \( (\text{preorder}(\text{y}) | \ y \ \text{successor of this}) \)
  - \( y \ \text{successor of} \ x \ \Leftrightarrow \ \text{pre}(x) < \text{pre}(y) \ \text{and bound}(x) \geq \text{pre}(y) \)
  - very easy successor test
XML document as one DB object

- BLOB or CLOB
  - no reconstruction effort, no query support useful for document centric objects?
- (User) defined type "text"
  - no reconstruction effort, indexing support queries like "find all occurrences of 'actor' " in a DB storing movies, no separation of tags and data (f).
  - More sophisticated queries like: "find occurrences of 'Monroe' where 'actress' occurs in a window of n words before/after"
  - locates "actress" elements with (partial) value 'Monroe'

Indexed "text" attribute

- American
  - last
  - Monroe
- Cast ...
- Award ...

Querying

- only boolean queries on keywords
- no difference between structure and content
- only useful in document centric applications

Search predicates (Oracle)

CREATE Table MovieTab AS (id INTEGER, txt CLOB)
SELECT id, txt, SCORE(1)
FROM MovieTab
WHERE CONTAINS (txt, 'Monroe WITHIN Last',1)>0 ;

SELECT id
FROM MovieTab
WHERE CONTAINS (txt, 'Monroe INPATH(//Actress/Last')..);
... CONTAINS (txt, 'HASPATH(//Actress/Last="Monroe" '..);
// exact match required

For performance reasons separation of structure index and data index

Relational type of XML attribute

- American
  - Monroe
- Carolyn
- Oscar
- Actor
- Cast
- Award ...

Automatic approach

- Map each element to a relation
  - Object relational / object oriented: to a class
- Attribute of element -> attribute of a relation / class
- Tree structure by some ordering scheme
- Easier to map into an object oriented / object relational data model

User defined Mapping approach

- Define a mapping between DTD and database schema
- More flexible, more effort

See also R. Bourret: mapping DTDs to Databases
### DTD-oriented approach

**Automatic:**
- Simple elements (not nested) -> attributes
- Nested elements -> object types (classes)

#### DTD

```
<!ELEMENT A (B, C)> class A {
<!ELEMENT B (#PCDATA)> String b;
<!ATTLIST A C c;
F CDATA #REQUIRED> String f; }

<!ELEMENT C (D, E)> class C {
<!ELEMENT D (#PCDATA)> String d;
<!ELEMENT E (#PCDATA)> String e; }
```

- Sibling order is lost
- Could be retained by artificial order attributes

---

### DTD-oriented approach: order

```
<!ELEMENT A (#PCDATA | B | C)*>
<!ELEMENT B (#PCDATA)>
<!ELEMENT C (#PCDATA)>
```

```
class A {
String[] pcdata;
int[] pcdataOrder;
String[] b;
int[] bOrder;
String[] c;
int[] cOrder; }
```

- Not supported by most systems
- +, * can also be mapped to SET or LIST – if supported by data model

---

### DTD-oriented approach

**Some issues**
- Data types
- DTD: manual adaption of schema necessary
- XML schema: adapt data types to database types (automatically)
- Mixed content

```
<desc> The values are
  <simple> like 'X'</simple> or
  <compound> like '(x,y)' </compound>
</desc>
```

- Map onto different attributes and define an order
  - Tab (order, desc, simple, compound)
  - Order

---

### Advantages

- Object oriented / Object relational allow fine-granular, "natural" mapping of object structure
- DTD is needed (of course) – compared to generic mapping
- SQL and XQuery / XPath as query languages as opposed to 'XMLtype' : basically operations on text

---

### Disadvantages

- No document reconstruction possible in principle, but expensive

---

### User defined mapping

**Define your own mapping called Data access Definition in DB2**

```
<ClassMap>
  <ElementType name = "Movies"/>
  <ToClassTable>
    <Table Name="MovieTable"/>
  </ToClassTable>
  <PropertyMap>
    <Attribute Name = "Title"/>
    <ToColumn>
      <Column Name = "Title"/>
    </ToColumn>
  </PropertyMap>
</ClassMap>
```

- Very flexible
- Makes sense if XML docs are mapped to an existing DB schema

---

### Mapping guidelines

<table>
<thead>
<tr>
<th>XML</th>
<th>Database Def</th>
</tr>
</thead>
<tbody>
<tr>
<td>element root</td>
<td>Relation</td>
</tr>
<tr>
<td>simple</td>
<td>attribute</td>
</tr>
<tr>
<td>sequence attributes</td>
<td>attributes</td>
</tr>
<tr>
<td>alternative</td>
<td>attributes</td>
</tr>
<tr>
<td>elem with ?</td>
<td>attribute, NULL</td>
</tr>
<tr>
<td>elem +,*</td>
<td>SET, LIST</td>
</tr>
<tr>
<td>complex</td>
<td>Class (Object type)</td>
</tr>
<tr>
<td>attribute</td>
<td>Attribute of Rel</td>
</tr>
<tr>
<td>IMPLIED</td>
<td>^</td>
</tr>
<tr>
<td>NOT REQUIRED</td>
<td>NOT NULL</td>
</tr>
<tr>
<td>default</td>
<td>default</td>
</tr>
</tbody>
</table>

---

### Some issues

- DTD: manual adaption of schema necessary
- XML schema: adapt data types to database types (automatically)
- Mixed content

```xml
<desc> The values are
  <simple> like 'X'</simple> or
  <compound> like '(x,y)' </compound>
</desc>
```

- Map onto different attributes and define an order
  - Tab (order, desc, simple, compound)
  - Order

---

### Advantages

- Object oriented / Object relational allow fine-granular, "natural" mapping of object structure
- DTD is needed (of course) – compared to generic mapping
- SQL and XQuery / XPath as query languages as opposed to 'XMLtype' : basically operations on text

---

### User defined mapping

**Define your own mapping called Data access Definition in DB2**

```
<ClassMap>
  <ElementType name = "Movies"/>
  <ToClassTable>
    <Table Name="MovieTable"/>
  </ToClassTable>
  <PropertyMap>
    <Attribute Name = "Title"/>
    <ToColumn>
      <Column Name = "Title"/>
    </ToColumn>
  </PropertyMap>
</ClassMap>
```

- Very flexible
- Makes sense if XML docs are mapped to an existing DB schema