Storing XML Data in Relational/Object-Oriented Databases
An Overview

Tree Vs. Content

There are two basic approaches to storing XML documents:

1. **Structure-Based Approaches.** Here the *XML tree* is stored. Content is not analyzed.
2. **Content-Based Approaches.** The *content* of XML document is analyzed and main database enities are identified. The XML document is then broken into such objects which are stored in the database.

Example XML

```
<db>
<person>
 <name> Alex </name>
 <phone> 859 453 3223</phone>
</person>
<dept>
 <name> Computer Science</name>
 <location> Anedson Hall, 773</location>
</dept>
</db>
```

For some apporoaches we assume that each element occurrence has a unique id.

Structure Based Storage

Advantages:

- Works for **any** XML document.

- Does not need a DTD (or other schema description).
- Does not require any additional analysis.

Disadvantages:

- Storage is not always optimal.
- "Natural" queries may be hard to rewrite.
- Query processing may take longer.

Edge Approach

Idea: Store all elements in single table Edge which has the schema:

Edge(source, ordinal, name, flag, target)

Here

source : OID of the parent element;

ordinal : position in the parent element's children list occupied by current element;

name : element tag;

flag : indicator of the content type (#PCDATA vs. structure)

target : if flag=1, OID of the current element.
if flag =0, content of the element.

Note: The Edge table preserves information about the elements of the XML markup only. If XML attributes are present, they need to be stored in a separate table that uses the element OID as the foreign key:

Attribute(element, name, value)

The sample XML instance is represented in the Edge table as follows:

source	ordinal	name	flag	target
0	1	`db`	1	**1**
1	1	`person`	1	**2**
1	2	`department`	1	**3**
2	1	`name`	0	`Alex`
2	2	`phone`	0	859 453 3223
3	1	`name`	0	`Computer Science`
3	2	`location`	0	`773 Anderson Hall`

Note: It may be a good idea to store OID of each element even if its content is only #PCDATA. In this case, Edge table will have the schema **Edge(source, ordinal,name,flag,target,content)**, where target now will always be the OID of the current element and content will be the content of the element if flag=0.

Note: In the real implementation of the Edge approach, target field was a reference to a separate location in the database where either content, or the OID was stored.

Attribute Approach

Attribute approach is similar to the Edge approach. Here, instead of one Edge table, a separate table for each element is created. Each "attribute"[1] table has the same schema as the Edge table in the Edge approach.

In our example, the following tables would be created: Edge-db, Edge-person, Edge-name, Edge-departement, Edge-location and Edge-phone.

In this form, attribute approach is very inconvenient for determining parent and child elements (especially without DTDs). In order to find a parent of a particular element, one must scan **all** the tables, as it is not known what type of element has particular OID. By storing the name of parent element, one can somewhat alleviate this problem, but it still remains for children.

Universal Table

Here, we "join" the information from all attribute tables about the content of a single element. If there are n element types in the he information is stored in the table Universal with the following structure

> Universal(source, name, ordinal-1, flag-1, target-1,. . . ,ordinal-n, flag-n, target-n);

In our example, the schema for the Universal table would be

> Universal(source, name, ordinal-db, flag-db, target-db, ordinal-person, flag-person, target-person, ordinal-name, flag-name, target-name, ordinal-phone, flag-phone, target-phone, ordinal-department, flag-department, target-department, ordinal-location, flag-location, target-location)

Omitting the values of flag fields, the sample XML instance will be represented as follows:

source	name	ord-db	trg-db	ord-person	trg-person	ord-name	trg-name	ord-phone	trg-phone	odr-dept	trg-dept	ord-loc	trg-loc
0	db	*null*	*null*	1	**2**	*null*	*null*	*null*	*null*	2	**3**	*null*	*null*
1	person	*null*	*null*	*null*	*null*	**1**	`Alex`	**2**	859 453 3223	*null*	*null*	*null*	*null*
2	department	*null*	*null*	*null*	*null*	1	`Computer Science`	*null*	*null*	*null*	*null*	2	`773, Anderson`

This is not a very convenient representation. It is not (in general) normalized and null values are present.

Normalized Universal table

Basically, a Universal table and an Overflow table that contains occurrences of repeated tags. For example in representing the following fragment:

```
<person>
   <name> Alex</name>
   <name> Alexander</name>
 </person>
```

The first occurence of ⟨`name`⟩ tag will be put into Universal table, while the second occurrence – into Overflow table.

[1]In XML notation, a misnomer. Should have been "element". But we keep original notation for historic purposes.

Object-oriented approach

XML document is stored as a single object. Each element is stored as an embedded light-weight object.

Offset	Record
0	Length=40, `db`, parent=`nil`, prev=`nil`, next=`nil`, firstChild=40, lastChild= 80
40	Length=40, `person`, parent = 0, next = 80, firstChild = 120, lastChild = 140, no attribute
80	Length=40, `dept`, parent=0, prev= 40, next=`nil`, firstChild = 160 , lastChild = 180, no attribute
120	Length=20, `name`, parent=40, prev=`nil`, next=140, no children, no attribute, #PCDATA = ``Alex''
140	Length=20, `phone`, parent=40, prev=120, next=`nil`, no children, no attribute, #PCDATA = ``859 453 3223''
160	Length=20, `name`, parent=80, prev=`nil`, next=180, no children, no attribute, #PCDATA = ``Computer Science''
180	Length=20, `location`, parent=80, prev=160, next=`nil`, no children, no attribute, #PCDATA = ``Anderson Hall, 773''

Querying XML Data Stored in Relational Databases

Sample XML

Consider the following DTD:

```
<!DOCTYPE stacks [
  <!ELEMENT stacks (section*, librarian*)>

  <!ELEMENT section (shelf*)>
  <!ELEMENT shelf (book*)>
  <!ELEMENT book (ISBN, title, author+)>
  <!ELEMENT ISBN (#PCDATA)>
  <!ELEMENT title (#PCDATA)>
  <!ELEMENT author (#PCDATA)>

  <!ELEMENT librarian (name, shift+)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT shift (day, start, end)>
  <!ELEMENT day (#PCDATA)>
  <!ELEMENT start (#PCDATA)>
  <!ELEMENT end (#PCDATA)>

  <!ATTLIS section nm  ID    #REQUIRED
                   loc CDATA #IMPLIED>
  <!ATTLIS shelf pos ID #REQUIRED>
  <!ATTLIS book  id  ID #REQUIRED>
  <!ATTLIS librarian id ID #REQUIRED
                      sections IDREFS #IMPLIED>
 ]>
```

This DTD describes part of the library XML database called `<stacks>`. This database consists of descriptions of two high-level objects: sections and librarians. Each section consists of a number of shelves, each shelf in its turn containing a number of books. Each librarian covers a particular set of sections (describe as

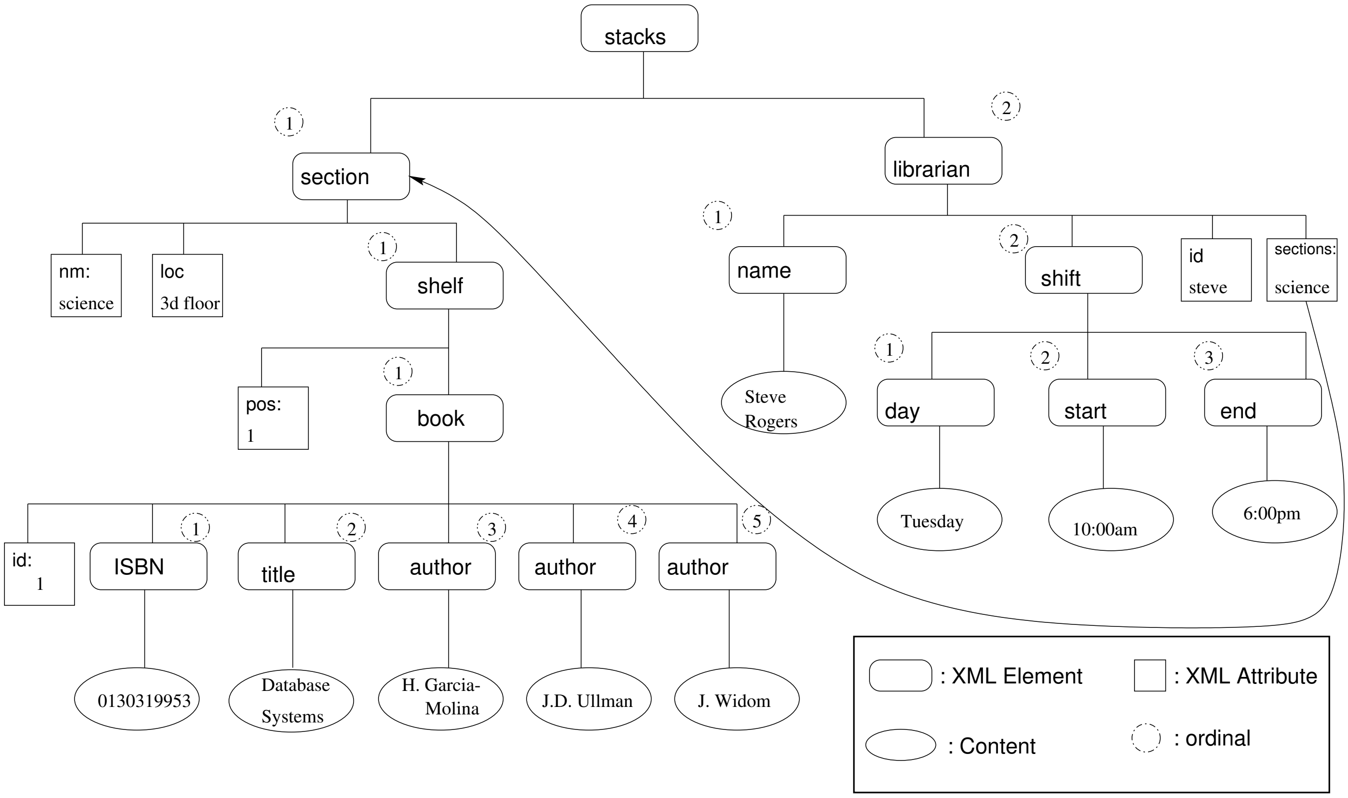

Figure 1: XML Data Model of sample XML

an XML attribute `sections` of type `IDREFS`, and works a number of `shifts` per week.

Conisder the following sample XML conforming to the DTD above.

```
<stacks>
<section nm= ‘‘science’’, loc= ‘‘3d floor’’>
  <shelf pos= ‘‘1’’>
     <book id = ‘‘1’’>
        <ISBN> 0130319953</ISBN>
        <title> Database Systems</title>
        <author> H. Garcia-Molina </author>
        <author> J.D.  Ullman </author>
        <author> J. Widom </author>
     </book>
  </shelf>
</section>
<librarian id= ‘‘steve’’, sections= ‘‘science’’>
  <name> Steve Rogers </name>
  <shift> <day> Tuesday </day>
          <start> 10:00am</start>
          <end>   6:00pm</end>
  <shift>
</librarian>
</stacks>
```

Figure 1 shows the OEM diagram of this XML instance.

Edge Approach

We need two tables to represent our XML instance: the Edge table and the Attributes table. The relational schemas of these tables are:

Edge(source, ordinal, name, flag, target)
Attribute(source, name, type, value)

The sample XML instance is represented in the Edge table as follows:

source	ordinal	name	flag	target
0	1	`stacks`	1	**1**
1	1	`section`	1	**2**
1	2	`librarian`	1	**3**
2	1	`shelf`	1	**4**
4	1	`book`	1	**5**
5	1	`ISBN`	0	0130319953
5	2	`title`	0	`Database Systems`
5	3	`author`	0	`H. Garcia-Molina`
5	4	`author`	0	`J.D. Ullman`
5	5	`author`	0	`J. Widom`
3	1	`name`	0	`Steve Rogers`
3	2	`shift`	1	**6**
6	1	`day`	0	`Tuesday`
6	2	`start`	0	`10:00am`
6	3	`end`	0	`6:00pm`

The accompanying Attributes table will be

source	name	type	value
2	nm	ID	`science`
2	loc	CDATA	`3d floor`
3	id	ID	`steve`
3	sections	IDREFS	`science`
4	pos	ID	1
5	id	ID	1

Queries

Simple:

- What are all ISBN numbers of all the books in collection ?
- What are the names of all librarians ?
- Who are the authors of the “Database Systems” book ?
- Who are the librarians working in the science section ?
- What are the titles of the books located on all shelves with position 1 ?

Getting harder:

- What are the titles of the books located in the science section ?
- Who would you go to on Tuesday at 12:00pm to ask for the “Database Systems” book ?