Chapter 10

Mapping XML to the Relational World

Introduction

- XQuery and other XML query languages operate on XML documents
- Up to now we have assumed that these documents exist in files or network messages
- Often, however, documents are generated on demand from different representations and sources
- One important source of data are relational database management systems (RDBMS)

Introduction (2)

- RDBMS are not going to vanish due to the arrival of the new XML standards
- Quite the contrary, RDBMS are probably going to stay with us for a long time to come
- Building bridges between the XML and the RDBMS world is therefore very important
- In this chapter we are going to have a look at different approaches for mappings between XML and relational data
- SQL/XML is an important ISO standard that addresses these issues

XML Publishing

- Assume that the original data is relational
- The application, however, wants to access this data as XML
- So we have to create an XML representation of the relational data
- This is called XML *publishing* or *composing*

XML Shredding

- The original data may instead be in XML
- The question now is how to store this data in a RDBMS
- The simplest method is to store the XML directly as the value of some attribute/column in a relation
- More generally, this process is called XML *shredding* or *decomposing*
- Shredding can be done in many ways, depending on
  - how structured the data is: ranging from very structured to quite unstructured marked-up text
  - what kind of schema information is available

SQL/XML

- The ISO SQL/XML standard was first produced in 2003
- It was revised in 2006, 2008 and 2011
- It provides a new SQL data type (XML) to store XML in an RDBMS
- SQL/XML provides new SQL functions to generate XML documents or fragments from relational data (called publishing functions)
- In addition to this, there are default mapping rules for SQL datatypes appearing in XML-generating operators
- It also provides additional querying capabilities (using XQuery)

Using the XML Data Type

- The simplest way of storing XML in an RDBMS is to use the SQL/XML data type
- A column of type XML in the RDBMS can contain any XQuery sequence
- Some other columns may also be present
- Example (the `purchaseorder` column is of type XML):

```
id    receivedate  purchaseorder

4023  2001-12-01   <purchaseOrder>
                     <originator billId=’0013579’>
                       <contactName>
                       ...
                   </purchaseOrder>

5327  2002-04-23   <purchaseOrder>
                     <originator billId=’0232345’>
                       ...
```

Using the XML Data Type (2)

- The single column mapping is quite straightforward; the XML document (or sequence) is loaded into the RDBMS “as is”
- A value of type XML can be any valid XQuery sequence or the SQL NULL value
- In fact, a number of parameterised subtypes of the XML type are defined in the standard:
    - XML (SEQUENCE)
    - XML (ANY CONTENT)
    - XML (ANY DOCUMENT)
    - . . .
- We will not study these subtypes

Publishing Techniques

- SQL/XML provides two different techniques for publishing relational data as XML
  - A default mapping from tables to XML
  - Using the SQL/XML publishing functions
- The first of these is very simple, but limited in how useful it is
- The second is much more flexible

Default Mapping

- The default mapping is the simplest publishing technique
- In the default mapping, the names of tables and columns become the names of XML elements, with the inclusion of `row` elements for the each table row
- But the default mapping does not allow for publishing only parts of tables or the result of a query as XML
- Also, many applications may need XML data in specific formats that do not correspond to the result of the default mapping
- These limitations mean that applications may have to perform extensive post-processing on the generated document

Example

Table customer:

name	acctnum	address
Albert Ng	012ab3f	123 Main St., ...
Francis Smith	032cf5d	42 Seneca, ...
...	...	...

XML generated by the default mapping:

```
<customer>
  <row>
    <name>Albert Ng</name>
    <acctnum>012ab3f</acctnum>
    <address>123 Main St., ...</address>
  </row>
  <row>
    <name>Francis Smith</name>
    <acctnum>032cf5d</acctnum>
    <address>42 Seneca, ...</address>
  </row>
  ...
</customer>
```

Default Mapping (2)

- The default mapping can also be used for all tables in a schema, or all schemas in a catalog
- In each case, an extra level is introduced in the output by elements representing schema or catalog names
- The mapping depends on rules for mapping SQL identifiers to XML names, and SQL data types to XML schema data types
- As well as producing an XML document representing the relational data, the default mapping produces an XML schema document

SQL/XML functions for publishing

- `XMLELEMENT()` to produce an XML element
- `XMLATTRIBUTES()` to produce XML attributes
- `XMLFOREST()` which creates a forest of elements
- `XMLCONCAT()` which concatenates a list of XML elements
- `XMLAGG()` which creates a forest of XML elements based on a `GROUP BY` clause in the SQL query
- (We will consider only the first three functions)

Example using XMLELEMENT()

- This example assumes the `customer` table used previously:

```
SELECT c.acctnum,
  XMLELEMENT (NAME "invoice",
              'To ',
              XMLELEMENT (NAME "name", c.name)
             ) AS "invoice"
FROM customer c
```

- This creates an XML element called `invoice` with mixed content:

```
acctnum   invoice

012ab3f   <invoice>To <name>Albert Ng</name></invoice>
032cf5d   <invoice>To <name>Francis Smith</name></invoice>
...
```

Example using XMLATTRIBUTES()

- Once again using the `customer` table:

```
SELECT c.acctnum,
  XMLELEMENT (NAME "invoice",
               XMLATTRIBUTES (c.acctnum AS "id", c.name)
              ) AS "invoice"
FROM customer c
```

- This creates an XML element with attributes and empty content:

```
acctnum   invoice


012ab3f   <invoice id="012ab3f" name="Albert Ng"/>
032cf5d   <invoice id="032cf5d" name="Francis Smith"/>
...
```

- Obviously attributes and nested elements can be combined

XMLFOREST()

- `XMLFOREST()` produces a forest of elements
- Each of its arguments is used to create a new element
- Like `XMLATTRIBUTES()`, an explicit name for the element can be provided, or the name of the column can be used implicitly

Shredding

- There are different ways of shredding XML documents
- If the documents are well-structured and follow a DTD or XML schema:
  - We can extract this schema information and build a relational schema that mirrors this structure
  - Each table in this relational schema stores certain parts of the XML document
- If the documents are irregular and do not follow a common schema:
  - We have to use a very general schema for mapping arbitrary XML trees into an RDBMS

Shredding Unstructured Documents

- One possibility to handle arbitrary documents is to use a relational representation that is totally independent of XML schema information
- This representation models XML documents as tree structures with nodes and edges
- We saw an example of this in Chapter 8 with the Edge relation
- Every single navigation step requires a join on this table
- Alternatives considered in Chapter 8 were
  - Element-partitioned relations
  - Path-partitioned relations

Shredding Structured Documents

- The first step is designing the relational schema
- Some database vendors offer an automated mapping process
- These techniques are often based on annotating an XML schema definition with information about where the corresponding data is to be stored in the RDBMS
- We are going to have a look at some basic techniques for creating a relational schema

Shredding Structured Documents (2)

- Adding extra information:
  - Care has to be taken that we will be able to reassemble the XML document (sometimes more than one document is stored in a table)
  - Usually each node/value stored in a table will have a document id associated with it (regardless of in which table it will end up)
  - Storing positions of a node within its parent will allow us to reconstruct the document order

Shredding Structured Documents (3)

- During shredding we have two basic table layout choices:
  - We can break information across multiple tables
  - We can consolidate tables for different elements
- A simple algorithm for doing this starts scanning at the top of the XML document
- Each time an element is encountered it is associated with a table
- For each child of that element a decision is made whether
  - to put it into the same table (inlining)
  - or start a new table (and find a way to connect the two tables via a join attribute)

Shredding Structured Documents (4)

- There is a simple rule for deciding whether to inline or not:
  - If an element can occur multiple times (e.g. has `maxOccurs > 1`), then put it in a different table
  - If an element has a complex structure (e.g. is of `ComplexType`), then put it in a different table
  - Simple elements (e.g. of `SimpleType`) that occur exactly once are placed in the same table as their parent element
- What about optional elements?
  - Inlining optional elements may lead to many NULL values
  - Putting them into their own table results in expensive join operations
  - Neither choice is optimal in all cases

Example

- Consider our `books.xml` example from Chapter 9
- Since `year`, `title`, `publisher` and `price` each occur once, they can be placed in the same `book` table
- Since `author` can occur many times, it is placed in a different table
- Since `editor` is complex, it is placed in a different table
- The next slide shows the result

Example (2)

book				
id	year	title	publisher	price
1	1994	TCP/IP ...	...	65.95
2	1992	Advanced ...	...	65.95
3	2000	Data on ...	...	39.95
4	1999	The Economics ...	...	129.95

author			
id	last	first	book
5	Stevens	W.	1
6	Stevens	W.	2
7	Abiteboul	Serge	3
8	Buneman	Peter	3
9	Suciu	Dan	3

editor				
id	last	first	affiliation	book
10	Gerbarg	Darcy	CITI	4

Shredding Structured Documents (5)

- After shredding XML documents, it may be possible to consolidate tables
- Some element types may appear multiple times in an XML document at different places (e.g. names or addresses)
- As long as the attributes are used in a consistent way, these different tables can be merged into one
- Shredding, in general, is a complicated process and there are many possible solutions

Conclusion

- The SQL/XML XML data type can handle any kind of XML data
- For the shredding approach some kind of XML schema information is helpful
- It is quite expensive for the shredding approach to reassemble whole documents

Summary

- There are a variety of techniques for mapping between XML and relational data
- Facilities for achieving this mapping are provided by database vendors or third party vendors (e.g. for middleware components)
- Which actual features are necessary depends mostly on the requirements of the application