CHAPTER 23

XML

XML is today widely used in the exchange of data between applications, and for storing data, whether simple or complex, in flat files. All the recent standard formats for storing documents, spreadsheets, presentations, and so on are based on XML, cementing the success of XML. In the context of databases, XML has been quite successful, although not with anything like the success of XML outside of databases; relational databases continue to be used for most mission critical data. However, support for storing XML in databases has improved significantly in the last 5 years.

Our view of XML is decidedly database centric. In this view, XML is a data model that provides a number of features beyond that provided by the relational model, in particular the ability to package related information into a single unit, by using nested structures. Specific application domains for data representation and interchange need their own standards that define the data schema.

Given the extensive nature of XML and related standards, this chapter only attempts to provide an introduction, and does not attempt to provide a complete description. For a course that intends to explore XML in detail, supplementary material may be required. These could include online information on XML and books on XML.

Exercises

23.10 Show, by giving a DTD, how to represent the non-1NF *books* relation from Section 22.2, using XML.

Answer:

```
<!DOCTYPE bib [
    <!ELEMENT book (title, author+, publisher, keyword+)>
    <!ELEMENT publisher (pub-name, pub-branch) >
    <!ELEMENT title ( #PCDATA )>
    <!ELEMENT author ( #PCDATA )>
    <!ELEMENT keyword ( #PCDATA )>
    <!ELEMENT pub-name( #PCDATA )>
    <!ELEMENT pub-branch( #PCDATA )>
] >
```

23.11 Write the following queries in XQuery, assuming the schema from Practice Exercise 23.2.

a. Find the names of all employees who have a child who has a birthday in March.

b. Find those employees who took an examination for the skill type "typing" in the city "Dayton".

c. List all skill types in *Emp*.

Answer:

a. Find the names of all employees who have a child who has a birthday in March.

```
for $e in /db/emp,
    $m in distinct-values($e/children/birthday/month)
where $m = 'March'
return $e/ename
```

b. Find those employees who took an examination for the skill type "typing" in the city "Dayton".

```
for $e in /db/emp
    $s in $e/skills[type='typing']
    $exam in $s/exams
where $exam/city= 'Dayton'
return $e/ename
```

c. List all skill types in *Emp*.

```
for $t in distinct-values (/db/emp/skills/type)
return $t
```

23.12 Consider the XML data shown in Figure 23.3. Suppose we wish to find purchase orders that ordered two or more copies of the part with identifier 123. Consider the following attempt to solve this problem:

```
for $p in purchaseorder
where $p/part/id = 123 and $p/part/quantity >= 2
return $p
```

Explain why the query may return some purchase orders that order less than two copies of part 123. Give a correct version of the above query.

Answer: Reason:
The expression x =y evaluates to true if any of the values returned by the first expression is equal to any one of the values returned by the second expression. In this case, if any of the values in the part/quantity >= 2 then it evaluates to true, so the query returns parts which have id = 123 irrespective of the quantity.

Query:

```
for $b in purchaseorder
where some $p in $b/part satisfies
      $p/id = 123 AND $p/quantity >= 2
return {$b}
```

23.13 Give a query in XQuery to flip the nesting of data from Exercise 23.10. That is, at the outermost level of nesting the output must have elements corresponding to authors, and each such element must have nested within it items corresponding to all the books written by the author.

Answer:

```
<bib>
    for $x in distinct-values(/books/author)
        return
            <author> <name> { $x } </name>
            { for $y in /books[author = $x/author],
                return <book>
                    <title> { $y/title } </title>
                    <publisher> { $y/publisher } </publisher>
                    <keyword> { $y/keyword } </keyword>
                </book>
            } </author>
</bib>
```

23.14 Give the DTD for an XML representation of the information in Figure 7.29. Create a separate element type to represent each relationship, but use ID and IDREF to implement primary and foreign keys.

Answer:

```
<!DOCTYPE bookstore [
    <!ELEMENT basket (contains+, basket-of)>
    <!ATTLIST basket
        basketid ID #REQUIRED >
    <!ELEMENT customer (name, address, phone)>
    <!ATTLIST customer
        email ID #REQUIRED >
    <!ELEMENT book (year, title, price, written-by, published-by)>
    <!ATTLIST book
        ISBN ID #REQUIRED >
    <!ELEMENT warehouse (address, phone, stocks)>
    <!ATTLIST warehouse
        code ID #REQUIRED >
    <!ELEMENT author (name, address, URL)>
    <!ATTLIST author
        authid ID #REQUIRED >
    <!ELEMENT publisher (address, phone, URL)>
    <!ATTLIST publisher
        name ID #REQUIRED >
    <!ELEMENT basket-of >
    <!ATTLIST basket-of
        owner IDREF #REQUIRED >
    <!ELEMENT contains >
    <!ATTLIST contains
        book IDREF #REQUIRED
        number CDATA #REQUIRED >
    <!ELEMENT stocks >
    <!ATTLIST stocks
        book IDREF #REQUIRED
        number CDATA #REQUIRED >
    <!ELEMENT written-by >
    <!ATTLIST written-by
        authors IDREFS #REQUIRED >
    <!ELEMENT published-by >
    <!ATTLIST published-by
        publisher IDREF #REQUIRED >
    <!ELEMENT name (#PCDATA )>
    <!ELEMENT address (#PCDATA )>
    <!ELEMENT phone (#PCDATA )>
```

```
    <!ELEMENT year (#PCDATA )>
    <!ELEMENT title (#PCDATA )>
    <!ELEMENT price (#PCDATA )>
    <!ELEMENT number (#PCDATA )>
    <!ELEMENT URL (#PCDATA )>
] >
```

23.15 Give an XML Schema representation of the DTD from Exercise 23.14.

Answer:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="basket" type="BasketType">
    <xs:element name="customer">
        <xs:complexType>
        <xs:attribute name="email" use="required">
        <xs:sequence>
            <xs:element name="name" type="xs:string">
            <xs:element name="address" type="xs:string">
            <xs:element name="phone" type="xs:decimal">
        </xs:sequence>
            </xs:complexType>
    </xs:element>
    <xs:key name="customerKey">
        <xs:selector xpath="/bookstore/customer">
        <xs:field xpath="email">
    </xs:key>
    <xs:element name="book">
        <xs:complexType>
            <xs:attribute name="ISBN" use="required">
            <xs:sequence>
            <xs:element name="year" type="xs:decimal">
            <xs:element name="title" type="xs:string">
            <xs:element name="price" type="xs:decimal">
            <xs:element name="written-by" type="xs:string">
            <xs:element name="published-by" type="xs:string">
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:key name="bookKey">
        <xs:selector xpath="/bookstore/book">
        <xs:field xpath="ISBN">
    </xs:key>
    <xs:element name="warehouse">
        <xs:complexType>
```

```
                <xs:attribute name="code" use="required">
                <xs:sequence>
                <xs:element name="address" type="xs:string">
                <xs:element name="phone" type="xs:decimal">
                <xs:element name="stocks" type="xs:decimal">
                </xs:sequence>
        </xs:complexType>
</xs:element>
<xs:key name="warehouseKey">
        <xs:selector xpath="/bookstore/warehouse">
        <xs:field xpath="code">
</xs:key>
<xs:element name="author">
        <xs:complexType>
                <xs:attribute name="authid" use="required">
                <xs:sequence>
                <xs:element name="name" type="xs:string">
                <xs:element name="address" type="xs:string">
                <xs:element name="URL" type="xs:string">
                </xs:sequence>
        </xs:complexType>
</xs:element>
<xs:key name="authorKey">
        <xs:selector xpath="/bookstore/author">
        <xs:field xpath="authid">
</xs:key>
<xs:element name="publisher">
        <xs:complexType>
                <xs:attribute name="name" use="required">
                <xs:sequence>
                <xs:element name="address" type="xs:string">
                <xs:element name="phone" type="xs:decimal">
                </xs:sequence>
        </xs:complexType>
</xs:element>
<xs:key name="publisherKey">
        <xs:selector xpath="/bookstore/publisher">
        <xs:field xpath="name">
</xs:key>
<xs:element name="basket-of">
        <xs:attribute name="owner" use="required">
</xs:element>
<xs:keyref name="basketCustomerFKey" refer="customerKey">
        <xs:selector xpath="/bookstore/customer">
        <xs:field xpath="owner">
```

```
        </xs:key>
        <xs:element name="contains">
            <xs:attribute name="book" use="required">
            <xs:attribute name="number" use="required" type="xs:decimal"'>
        </xs:element>
        <xs:keyref name="bookBasketFKey" refer="bookKey">
            <xs:selector xpath="/bookstore/book">
            <xs:field xpath="book/">
        </xs:key>
        <xs:element name="stocks">
            <xs:attribute name="book" use="required">
            <xs:attribute name="number" use="required" type="xs:decimal"'>
        </xs:element>
        <xs:keyref name="bookwarehouseFKey" refer="bookKey">
            <xs:selector xpath="/bookstore/book">
            <xs:field xpath="book/">
        </xs:key>
        <xs:element name="written-by">
            <xs:attribute name="authors" use="required">
        </xs:element>
        <xs:keyref name="authorbookFKey" refer="authorKey">
            <xs:selector xpath="/bookstore/author">
            <xs:field xpath="authors">
        </xs:key>
        <xs:element name="published-by">
            <xs:attribute name="publisher" use="required">
        </xs:element>
        <xs:keyref name="bookpublisherFKey" refer="publisherKey">
            <xs:selector xpath="/bookstore/publisher">
            <xs:field xpath="publisher">
        </xs:key>
            <xs:complexType name="BasketType">
                <xs:attribute name="basketid" use="required">
                <xs:sequence>
                <xs:element name="contains" minOccurs="1"
                        maxOccurs="unbounded">
                <xs:element name="basket-of">
                </xs:sequence>
            </xs:complexType>
        <xs:key name="basketKey">
            <xs:selector xpath="/bookstore/basket">
            <xs:field xpath="basketid">
        </xs:key>
</xs:schema>
```

23.16 Write queries in XQuery on the bibliography DTD fragment in Figure 23.16, to do the following:

a. Find all authors who have authored a book and an article in the same year.

b. Display books and articles sorted by year.

c. Display books with more than one author.

d. Find all books that contain the word "database" in their title and the word "Hank" in an author's name (whether first or last).

Answer:

a. Find all authors who have authored a book and an article in the same year.

```
for $a in distinct-values (/bib/book/author),
      $y in /bib/book[author=$a]/year,
      $art in /bib/article[author=$a and year=$y]
return $a
```

b. Display books and articles sorted by year.

```
for $a in ((/bib/book) | (/bib/article))
order by $a/year
return $a
```

c. Display books with more than one author.

```
for $b in ((/bib/book[author/count()>1])
return $b
```

d. Find all books that contain the word "database" in their title and the word "Hank" in an author's name (whether first or last).

```
for $a in /bibliography/book/
where $b in $a satisfies
      (contains($b/title,"database") AND
      (contains($b/author(@last_name),"Hank")
            OR contains($b/author(@fiest_name),"Hank"))
return $a
```

23.17 Give a relational mapping of the XML purchase order schema illustrated in Figure 23.3, using the approach described in Section 23.6.2.3. Suggest how to remove redundancy in the relational schema, if item identifiers functionally determine the description and purchase and supplier names functionally determine the purchase and supplier address, respectively.

Answer: Removing redundancy:
Item ids functionally determine the desription The table 'item' can be broken down into 2 tables

```
item(item_id, quantity, price, itemlist_id)
and
item_info(item_id, description)
```

Then the item description doesnt get repeated whenever the item appears on the itemlist.
Purchaser_names functionally determine the address The table purchaser can be subdivided as shown below

```
purchaser(name, purcahseorder_id)
and
purchaser_info(name, address)
```

Supplier_names functionally determine the address The table supplier can be subdivided as shown below

```
supplier(name, purchaseorder_id)
and
supplier_info(name, address)
```

The address does not get repeated every time the supplier or purchaser is involved in a purchase.

```
create table purchase_order
    (purchaseorder_id char(20),
    purchaser_name char(20),
    supplier_name char(20),
    itemlist_id char(20),
    total_cost numeric(16,2),
    payment_terms char(20),
    shipping_mode char(20),
    primary key (identifier),
    foreign key (purchaser_name) references purchaser,
    foreign key (supplier_name) references supplier,
    foreign key (itemlist_id) references itemlist)
```

```
create table purchaser
        (purchaser_name char(20),
        address char(20),
        purchaseorder_id char(20),
        primary key (name),
        foreign key (purchaseorder_id) references purchase_order)


create table supplier
        (supplier_name char(20),
        address char(20),
        purchaseorder_id char(20),
        primary key (name),
        foreign key (purchaseorder_id) references purchase_order)


create table itemlist
        (itemlist_id char(20),
        item_id char(20),
        purchaseorder_id char(20),
        primary key (itemlist_id),
        foreign key (item_id) references item,
        foreign key (purchaseorder_id) references purchase_order)

create table item
        (item_id char(20),
        description char(20),
        quantity numeric(16),
        price numeric(16,2),
        itemlist_id char(20),
        primary key (item_id),
        foreign key (itemlist_id) references itemlist)
```

23.18 Write queries in SQL/XML to convert university data from the relational schema we have used in earlier chapters to the *university-1* and *university-2* XML schemas.

Answer: **university-1:**

```
select xmlelement(name "department",
        xmlelement(name "dept_name", d.dept_name),
        xmlelement(name "building" d.building),
        xmlelement(name "budget" d.budget),
        xmlelement(name "courses",
        (select xmlagg(xmlelement(name "course",
                xmlelement(name "course_id" c.course_id),
                xmlelement(name "title" c.title),
                xmlelement(name "credits" c.credits)))
                from course c
                where c.dept_name = d.dept_name
                order by c.course_id)))
from department d
select xmlelement(name "instructor",
        xmlelement(name "IID", i.IID),
        xmlelement(name "name", i.name),
        xmlelement(name "dept_name", i.dept_name),
        xmlelement(name "salary", i.salary),
        (select xmlagg(xmlelement(name "course_id", t.course_id))
                from teaches t
                where teaches.IID = i.IID
                order by t.course_id))
from instructor i
```

university-2:

```
select xmlelement(name "instructor",
        xmlelement(name "IID", i.IID),
        xmlelement(name "name", i.name),
        xmlelement(name "dept_name", i.dept_name),
        xmlelement(name "salary", i.salary),
        xmlelement(name "teaches",
        (select xmlagg(xmlelement(name "course",
                xmlelement(name "course_id" c.course_id),
                xmlelement(name "title" c.title),
                xmlelement(name "credits" c.credits)
                from course c, teaches t
                where c.course_id = t.course_id and t.IID = i.IID
                order by c.course_id)))))
from instructor i
select xmlelement(name "department",
        xmlelement(name "dept_name", d.dept_name),
        xmlelement(name "building" d.building),
        xmlelement(name "budget" d.budget),
        (select xmlagg(xmlelement(name "course_id", c.course_id))
                from course c
                where c.dept_name = d.dept_name)
from department d
```

23.19 As in Exercise 23.18, write queries to convert university data to the *university-1* and *university-2* XML schemas, but this time by writing XQuery queries on the default SQL/XML database to XML mapping.

Answer: **university-1**:

```
<university-1> {
    for $x in /university/department/row
    return
            <department>
                <dept_name> {$x/dept_name} </dept_name>
                <building> {$x/building} </building>
                <budget> {$x/budget} </budget>
                for $c in /university/course/row[dept_name=$x/dept_name]
                return <course>
                    <course_id> {$c/course_id} </course_id>
                    <title> {$c/title} </title>
                    <credits> {$c/credits} </credits>
                </course>
            </department>
    for $i in /university/instuctor/row
    return
        <instructor>
            <IID> {$i/IID} </IID>
            <name> {$i/name} </name>
            <dept_name> {$i/dept_name} </dept_name>
            <salary> {$i/salary} </salary>
                for $t in /university/teaches/row[IID=$i/IID]
                return {
                    <course_id> {$t/course_id} </course_id>
                }
        </instructor>
} </university-1>
```

university-2:

```
<university-2> {
    for $x in /university/instructor/row/
    return
        <instructor>
            <IID> {$i/IID} </IID>
            <name> {$i/name} </name>
            <dept_name> {$i/dept_name} </dept_name>
            <salary> {$i/salary} </salary>
            <teaches> {
                for $t in /university/teaches/row[IID=$i/IID]
                    $c in /university/course/row[$t.course_id=$c/course_id]
                return <course>
                    <course_id> {$c/course_id} </course_id>
                    <title> {$c/title} </title>
                    <dept_name> {$c/dept_name} </dept_name>
                    <credits> {$c/credits} </credits>
                </course>
            } </teaches>
        </instructor>
    for $x in /university/department/row/
    return
        <department>
            <dept_name> {$x/dept_name} </dept_name>
            <building> {$x/building} </building>
            <budget> {$x/budget} </budget>
                for $c in /university/course/row[dept_name=$x/dept_name]
                return
                    <course_id> {$c/course_id} </course_id>
        </department>
} </university-2>
```

23.20 One way to shred an XML document is to use XQuery to convert the schema to an SQL/XML mapping of the corresponding relational schema, and then use the SQL/XML mapping in the backward direction to populate the relation.

As an illustration, give an XQuery query to convert data from the *university-1* XML schema to the SQL/XML schema shown in Figure 23.15.

Answer:

```
<department> {
    for $x in /university-1/department
    return
        <row>
            <dept_name> { $x/dept_name } </dept_name>
            <building> { $x/building } </building>
            <budget> { $x/budget } </budget>
        </row>
} </department>

<course> {
    for $x in /university-1/department/course
    return
        <row> {$x} </row>
} </course>

<instructor> {
    for $x in /university-1/instructor
    return
        <row>
            <IID> { $x/IID } </IID>
            <name> { $x/name } </name>
            <dept_name> { $x/dept_name } </dept_name>
            <salary> { $x/salary } </salary>
        </row>
} <instructor>

<teaches> {
    for $i in /university-1/instructor, $cid in $i/course_id
    return
        <row>
            <IID> { $i/IID } </IID>
            <course_id> { $cid } </course_id>
        </row>
} <teaches>
```

23.21 Consider the example XML schema from Section 23.3.2, and write XQuery queries to carry out the following tasks:

a. Check if the key constraint shown in Section 23.3.2 holds.

b. Check if the keyref constraint shown in Section 23.3.2 holds.

Answer:

a. Check if the key constraint shown in Section 23.3.2 holds.

```
let $x = /university/department
let $y = distinct-values(/university/department/dept_name)
return {
      if fn:count($y) == fn:count($x)
            then 1
            else 0
}
```

b. Check if the keyref constraint shown in Section 23.3.2 holds.

```
return
      (every $c in /university/course satisfies
            (some $d in /university/department satisfies
                  $c/dept_name = $d/dept_name))
```

23.22 Consider Practice Exercise 23.7, and suppose that authors could also appear as top-level elements. What change would have to be done to the relational schema?

Answer: Author id can be added as an attrribute to the author top element. It can be used to refer to the author in the book and article topelements and also keep track of the order.