Fall, 2005 CIS 550

Database and Information Systems

Homework 5 Solutions

November 19, 2007; Due November 26, 2007 at 4:30 pm

For this homework, you will work on a case of *data integration*. Consider the task of integrating different instances of the PBAY auction system, in order to create a "unified PBAY." Two schemas, derived from student answers to the previous homework, will be the basis of this assignment.

Problem 1 [25 points]: Let us begin with a relational version of the problem. Consider the two following relation schemas:

$Seller1(sellerID, rating, email, name, city, state, zip, url, contactPerson, contactEmail)$

$Seller2(sellerID, rating, firstName, lastName, birthDate, affiliation, netValue, email)$

Write a view definition in datalog for *Seller* that combines information from the two tables. This will consist of two rules.

Solution: There are a lot of possible answers. Here's one among them.

```
Sellers(sid, rat, email, name, city, state, zip, url,
cPerson, cEmail, null, null, null)
:-
Seller1(sid, rat, email, name, city, state, zip, url,
cPerson, cEmail)

Sellers(sid, rat, email, fName | lName, null, null, null, null,
null, null, bDate, aff, netValue)
:-
Seller2(sid, rat, fName, lName, bDate, aff,
netValue, email)
```

Problem 2 [25 points]: Write a datalog query, posed over your view, that retrieves the ratings of all sellers.

Take the query and the first rule from your view in Problem 1, and unfold (merge) the query and the view so you arrive at a single query over the base data.

Solution:

```
The query posed over the view is:
SellerRating(sellerID, rating) :-
Sellers(sellerID, rating, -, -, -, -, -, -, -, -, -, -, -)
```

The first rule from our view in Problem 1 is:

```
Sellers(sid, rat, email, name, city, state, zip, url,
cPerson, cEmail, null, null, null)
:-
Seller1(sid, rat, email, name, city, state, zip, url,
cPerson, cEmail)
```

Thus, the query over the base Data is:

```
SellerRating(sellerID, rating) :- Seller1(sellerID, rating, -, -,
-, -, -, -, -, -)
```

Problem 3 [25 points]: Consider the XML version of this problem, given the schemas in **schema-a.xsd** and **schema-b.xsd** (available on the web site, and shown at the end of this document). Write an XQuery that returns an element **sellers**, with the contents of the (imaginary) XML document "a.xml" conforming to schema-a, and "b.xml" conforming to schema-b.

Note you can write an XQuery with a union in the following form:

```
<my-tag>
{    for $x in doc(‘‘a’’)
  return $a   }
{    for $b in doc(‘‘b’’)
  return $b    }
</my-tag>
```

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xs:element name="pbay" type = "pbayType"/>

 <xs:complexType name = "pbayType">
    <xs:element name = "seller" type = "sellerType" maxOccurs="unbounded"/>
<!-- ... -->
 </xs:complexType>

 <xsd:complexType name="SellerType">
  <xsd:attribute name="sellerID" type="xsd:integer"/>
  <xsd:attribute name="rating" type="xsd:string"/>
  <xsd:attribute name="firstName" type="xsd:string"/>
  <xsd:attribute name="lastName" type="xsd:string"/>
  <xsd:attribute name="birthDate" type="xsd:date"/>
  <xsd:attribute name="affiliation" type="xsd:string" minOccurs = "0"/>
  <xsd:attribute name="netValue" type="xsd:integer"/>
  <xsd:sequence>
   <xsd:element name="email" type="xsd:string"/>
  </xsd:sequence>
 </xsd:complexType>

<!-- ... -->
</xsd:schema>
```

Figure 1: Schema A

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:element name="pbay" type = "pbayType"/>

    <xs:complexType name = "pbayType">
        <xs:all>
            <xs:element name = "sellers" type = "sellersType"/>
<!-- ... -->
        </xs:all>
    </xs:complexType>

    <xs:complexType name="sellersType">
            <xs:sequence>
                <xs:element name="seller" type="sellerType" maxOccurs="unbounded"/>
            </xs:sequence>
    </xs:complexType>


    <xs:simpleType name="ratingType">
        <xs:restriction base="xs:string">
            <xs:pattern value="[ABCDFabcdf]"/>
        </xs:restriction>
    </xs:simpleType>

    <xs:complexType name="sellerType">
        <xs:all>
            <xs:element name = "sellerID" type = "xs:int"/>
            <xs:element name="rating" type="ratingType"/>
            <xs:element name="email" type="xs:string"/>
        </xs:all>
        <xs:attribute name="name" type = "xs:string"/>
        <xs:attribute name="city" type = "xs:string"/>
        <xs:attribute name="state" type = "xs:string"/>
        <xs:attribute name="zip" type = "xs:int"/>
        <xs:attribute name="url" type = "xs:string"/>
        <xs:attribute name="contactPerson" type="xs:string"/>
        <xs:attribute name="contactEmail" type = "xs:string"/>
    </xs:complexType>

<!-- ... -->
</xs:schema>
```

Figure 2: Schema B

Solution:

```
 <Sellers>
 {
   for $a in doc(‘‘a.xml’’)/pbay/seller
   return
    <seller>
     <sellerID>{data($a/@sellerID)}</sellerID>
     <rating>{data($a/@rating)}</rating>
     <email>{$a/email/text()}</email>
     <name>{fn:concat(data($a/@firstName), " ", data($a/@lastName)}</name>
     <birthDate>{data($a/@birthDate)}</birthDate>
     <affiliation>{data($a/@affiliation)}</affiliation>
     <netValue>{data($a/@netValue)}</netValue>
    </seller>
 }
 {
   for $b in doc("b.xml")/pbay/sellers/seller
```

```
  return
   <seller>
    <sellerID>{$b/sellerID/text()}</sellerID>
    <rating>{$b/rating/text()}</rating>
    <email>{$b/email/text()}</email>
    <name>{data($b/@name)}</name>
    <city>{data($b/@city)}</city>
    <state>{data($b/@state)}</state>
    <zip>{data($b/@zip)}</zip>
    <url>{data($b/@url)}</url>
    <contactPerson>{data($b/@contactPerson)}</contactPerson>
    <contactEmail>{data($b/@contactEmail)}</contactEmail>
   </seller>
}
</Sellers>
```