CSE 462 Homework #5: Semi-Structured Data

Name: _____ Date: April 22, 2011

***** Due on Due on May 02, 2011 at the beginning of class. *****

This problem set is worth 250 points. You **must** type in your answers. Please, format your XML, DTD, XSchema, and XQuery.

Consider the reference *relational schema* for an online store given below.

> Customers(cemail: *string*, name: *string*, phone: *string*, street: *string*, city: *string*, state: *string*, zip: *string*)
>
> Products(sku: *int*, name: *string*, unitPrice: *real*, unitWeight: *real*)
>
> Orders(oId: *int*, cemail: *string*, orderDate: *date*, shipDate: *date*)
>
> OrderEntries(oId: *int*, sku: *int*, qty: *int*)
>
> Shipments(oId: *int*, street: *string*, city: *string*, state: *string*, zip: *string*)

Observations:

- Keys are underlined.
- `phone` in `Customers` may be `NULL`.
- `unitWeight` in `Products` may be `NULL`.
- `shipDate` in `Orders` may be `NULL`.
- `cemail` in `Orders` references `cemail` in `Customers`.
- `oId` in `OrderEntries` references `oId` in `Orders`.
- `sku` in `OrderEntries` references `sku` in `Products`.
- `oId` in `Shipments` references `oId` in `Orders`.
- The *entry count* of an order is the number of entries associated with the order.
- The *item count* of an order is the sum of product quantities over all entries in the order.
- The *gross value* of an order is the sum of `qty` × `unitPrice` over all entries in the order.
- The *gross weight* of an order is the sum of `qty` × `unitWeight` over all entries in the order.

The online store is in the process of developing an application to provide information to the main office as XML. Your job is to show alternate approaches in which this can be accomplished. For this, you will experiment with both DTD and XSchema for representing the data. You will also show some use cases of how the XML data can be queried using XPath and/or XQuery.

XSchema Type Reference:

- `http://www.w3schools.com/schema/schema_dtypes_string.asp`
- `http://www.w3schools.com/schema/schema_dtypes_date.asp`
- `http://www.w3schools.com/schema/schema_dtypes_numeric.asp`
- `http://www.w3schools.com/schema/schema_dtypes_misc.asp`

DTD, XSchema, and XML Validation:

- `http://www.xmlvalidation.com/`

1. (75pts) Provide a DTD (30pts) and an XSchema (30pts) to encode the reference relational schema based on the requirements described below. Then, provide a minimal XML document (15pts) that is valid with respect to the DTD but not with respect to the XSchema, if one exists.

- The root of your XML document is a `store` element containing `customers`, `products`, and `orders` elements.
- The `customers` element contains any number of `customer` elements. Every `customer` element represents a tuple in the `Customers` relation.
- The `products` element contains any number of `product` elements. Every `product` element represents a tuple in the `Products` relation.
- The `orders` element contains any number of `order` elements. Every `order` element represents a tuple in the `Orders` relation. Each `order` element must contain at least one `orderEntry` element. Every `orderEntry` element represents a tuple in the `OrderEntries` relation associated with the respective `order`. Each `order` element may contain one `shipment` element. Every `shipment` element represents a tuple in the `Shipments` relation associated with the respective `order`.
- Do not include `oId` values for `order`, `orderEntry`, and `shipment` elements. The relationship among these elements are determined from the document structure.
- For all other keys/foreign keys in the relational schema, try to reproduce them in the DTD and XSchema as best as possible, including their data types.

```
<!DOCTYPE store [
  <!ELEMENT store (customers, products, orders)>
  <!ELEMENT customers (customer*)>
  <!ELEMENT customer (name, phone?, street, city, state, zip)>
  <!ATTLIST customer cemail ID #REQUIRED>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT phone (#PCDATA)>
  <!ELEMENT street (#PCDATA)>
  <!ELEMENT city (#PCDATA)>
  <!ELEMENT state (#PCDATA)>
  <!ELEMENT zip (#PCDATA)>
  <!ELEMENT products (product*)>
  <!ELEMENT product (name, unitPrice, unitWeight?)>
  <!ATTLIST product sku ID #REQUIRED>
  <!ELEMENT unitPrice (#PCDATA)>
  <!ELEMENT unitWeight (#PCDATA)>
  <!ELEMENT orders (order*)>
  <!ELEMENT order (orderDate, shipDate?, shipment?, orderEntry+)>
  <!ATTLIST order cemail IDREF #REQUIRED>
  <!ELEMENT orderDate (#PCDATA)>
  <!ELEMENT shipDate (#PCDATA)>
  <!ELEMENT shipment (street, city, state, zip)>
  <!ELEMENT orderEntry (qty)>
  <!ATTLIST orderEntry sku IDREF #REQUIRED>
  <!ELEMENT qty (#PCDATA)>
]>
```

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

 <xs:element name="store">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="customers" type="customerSet"/>
    <xs:element name="products" type="productSet"/>
    <xs:element name="orders" type="orderSet"/>
   </xs:sequence>
  </xs:complexType>

  <xs:key name="customerKey">
   <xs:selector xpath="customers/customer"/>
   <xs:field xpath="@cemail"/>
  </xs:key>

  <xs:key name="productKey">
   <xs:selector xpath="products/product"/>
   <xs:field xpath="@sku"/>
  </xs:key>

  <xs:key name="orderEntryKey">
   <xs:selector xpath="orders/order/orderEntry"/>
   <xs:field xpath="@sku"/>
  </xs:key>

  <xs:keyref name="orderRef_customerKey" refer="customerKey">
   <xs:selector xpath="orders/order"/>
   <xs:field xpath="@cemail"/>
  </xs:keyref>

  <xs:keyref name="orderEntryRef_productKey" refer="productKey">
   <xs:selector xpath="orders/order/orderEntry"/>
   <xs:field xpath="@sku"/>
  </xs:keyref>
 </xs:element>

 <xs:complexType name="customerSet">
  <xs:sequence>
   <xs:element name="customer" type="customerType" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>

 <xs:complexType name="productSet">
  <xs:sequence>
   <xs:element name="product" type="productType" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>

 <xs:complexType name="orderSet">
  <xs:sequence>
   <xs:element name="order" type="orderType" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>

 <xs:complexType name="customerType">
  <xs:sequence>
   <xs:element name="name" type="xs:string"/>
   <xs:element name="phone" type="xs:string" minOccurs="0"/>
   <xs:element name="street" type="xs:string"/>
   <xs:element name="city" type="xs:string"/>
```

```
    <xs:element name="state" type="xs:string"/>
    <xs:element name="zip" type="xs:string"/>
   </xs:sequence>
   <xs:attribute name="cemail" type="xs:string"/>
  </xs:complexType>

  <xs:simpleType name="nonNegativeDecimal">
   <xs:restriction base="xs:decimal">
    <xs:minInclusive value="0"/>
   </xs:restriction>
  </xs:simpleType>

  <xs:complexType name="productType">
   <xs:sequence>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="unitPrice" type="nonNegativeDecimal"/>
    <xs:element name="unitWeight" type="nonNegativeDecimal" minOccurs="0"/>
   </xs:sequence>
   <xs:attrbute name="sku" type="xs:integer"/>
  </xs:complexType>

  <xs:complexType name="orderType">
   <xs:sequence>
    <xs:element name="orderDate" type="xs:date"/>
    <xs:element name="shipDate" type="xs:date" minOccurs="0"/>
    <xs:element name="shipment" minOccurs="0">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="street" type="xs:string"/>
       <xs:element name="city" type="xs:string"/>
       <xs:element name="state" type="xs:string"/>
       <xs:element name="zip" type="xs:string"/>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
    <xs:element name="unitWeight" type="xs:nonNegativeInteger" minOccurs="0"/>
    <xs:element name="orderEntry" minOccurs="0" maxOccurs="unbounded">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="qty">
        <xs:simpleType>
         <xs:restriction base="xs:decimal">
          <xs:minInclusive value="1"/>
         </xs:restriction>
        </xs:simpleType>
       </xs:element>
      </xs:sequence>
      <xs:attribute name="sku" type="xs:integer"/>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
   <xs:attribute name="cemail" type="xs:string"/>
  </xs:complexType>

</xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- This document is valid w.r.t. the DTD but not the XSchema -->
<store xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:noNamespaceSchemaLocation='store.xsd'>
<!-- Should not be able to distinguish on customers -->
 <customers />
<!-- DTDs cannot enforce type constraints: sku, unitWeight, and unitPrice all fail validation -->
 <products>
  <product sku="awesomeSKU">
   <name>Valid w.r.t. DTD but invalid w.r.t. XSchema</name>
   <unitPrice>expensive</unitPrice>
   <unitWeight>heavy</unitWeight>
  </product>
 </products>
<!-- Could have distinguished on orders -->
 <orders />
</store>
```

2. (75pts) Provide a DTD (30pts) and an XSchema (30pts) to encode the reference relational schema based on the requirements described below. Then, provide a minimal XML document (15pts) that is valid with respect to the DTD but not with respect to the XSchema, if one exists.

- The root of your XML document is a `store` element containing `products` and `customers` elements.
- The `products` element contains any number of `product` elements. Every `product` element represents a tuple in the `Products` relation.
- The `customers` element contains any number of `customer` elements. Every `customer` element represents a tuple in the `Customers` relation. Each `customer` element may contain any number of `order` elements. Every `order` element represents a tuple in the `Orders` relation associated with the respective `customer`. Each `order` element must contain at least one `orderEntry` element. Every `orderEntry` element represents a tuple in the `OrderEntries` relation associated with the respective `order`. Each `order` element may contain one `shipment` element. Every `shipment` element represents a tuple in the `Shipments` relation associated with the respective `order`.
- Do not include `oId` or `cemail` values for `order` elements; do not include `oId` values for `orderEntry` and `shipment` elements. The relationship among these elements are determined from the document structure.
- For all other keys/foreign keys in the relational schema, try to reproduce them in the DTD and XSchema as best as possible, including their data types.

```
<!DOCTYPE store [
  <!ELEMENT store (products, customers)>
  <!ELEMENT products (product*)>
  <!ELEMENT product (name, unitPrice, unitWeight?)>
  <!ATTLIST product sku ID #REQUIRED>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT unitPrice (#PCDATA)>
  <!ELEMENT unitWeight (#PCDATA)>
  <!ELEMENT customers (customer*)>
  <!ELEMENT customer (name, phone?, street, city, state, zip, order*)>
  <!ATTLIST customer cemail ID #REQUIRED>
  <!ELEMENT phone (#PCDATA)>
  <!ELEMENT street (#PCDATA)>
  <!ELEMENT city (#PCDATA)>
  <!ELEMENT state (#PCDATA)>
  <!ELEMENT zip (#PCDATA)>
  <!ELEMENT order (orderDate, shipDate?, shipment?, orderEntry+)>
  <!ELEMENT orderDate (#PCDATA)>
  <!ELEMENT shipDate (#PCDATA)>
  <!ELEMENT shipment (street, city, state, zip)>
```

```
  <!ELEMENT orderEntry (qty)>
  <!ATTLIST orderEntry sku IDREF #REQUIRED>
  <!ELEMENT qty (#PCDATA)>
]>
```

If you create types adequately in the first problem, this second schema is almost trivial to define– remove all references to orderSet and add a orderType element to the customer, with cardinality 0–unbounded. Then, fix the keys/keyrefs and remove the keyref from order that referenced the customer.

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

 <xs:element name="store">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="products" type="productSet"/>
    <xs:element name="customers" type="customerSet"/>
   </xs:sequence>
  </xs:complexType>

  <xs:key name="customerKey">
   <xs:selector xpath="customers/customer"/>
   <xs:field xpath="@cemail"/>
  </xs:key>

  <xs:key name="productKey">
   <xs:selector xpath="products/product"/>
   <xs:field xpath="@sku"/>
  </xs:key>

  <xs:key name="orderEntryKey">
   <xs:selector xpath="customers/customer/orders/order/orderEntry"/>
   <xs:field xpath="@sku"/>
  </xs:key>

  <xs:keyref name="orderEntryRef_productKey" refer="productKey">
   <xs:selector xpath="customers/customer/orders/order/orderEntry"/>
   <xs:field xpath="@sku"/>
  </xs:keyref>
 </xs:element>

 <xs:complexType name="customerSet">
  <xs:sequence>
   <xs:element name="customer" type="customerType" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>

 <xs:complexType name="productSet">
  <xs:sequence>
   <xs:element name="product" type="productType" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>

 <xs:complexType name="customerType">
  <xs:sequence>
   <xs:element name="name" type="xs:string"/>
   <xs:element name="phone" type="xs:string" minOccurs="0"/>
   <xs:element name="street" type="xs:string"/>
   <xs:element name="city" type="xs:string"/>
   <xs:element name="state" type="xs:string"/>
   <xs:element name="zip" type="xs:string"/>
   <xs:element name="order" type="orderType" minOccurs="0" maxOccurs="unbounded"/>
```

```
  </xs:sequence>
  <xs:attribute name="cemail" type="xs:string"/>
 </xs:complexType>

 <xs:simpleType name="nonNegativeDecimal">
  <xs:restriction base="xs:decimal">
   <xs:minInclusive value="0"/>
  </xs:restriction>
 </xs:simpleType>

 <xs:complexType name="productType">
  <xs:sequence>
   <xs:element name="name" type="xs:string"/>
   <xs:element name="unitPrice" type="nonNegativeDecimal"/>
   <xs:element name="unitWeight" type="nonNegativeDecimal" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="sku" type="xs:integer"/>
 </xs:complexType>

 <xs:complexType name="orderType">
  <xs:sequence>
   <xs:element name="orderDate" type="xs:date"/>
   <xs:element name="shipDate" type="xs:date" minOccurs="0"/>
   <xs:element name="shipment" minOccurs="0">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="street" type="xs:string"/>
      <xs:element name="city" type="xs:string"/>
      <xs:element name="state" type="xs:string"/>
      <xs:element name="zip" type="xs:string"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
   <xs:element name="unitWeight" type="xs:nonNegativeInteger" minOccurs="0"/>
   <xs:element name="orderEntry" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="qty">
       <xs:simpleType>
        <xs:restriction base="xs:decimal">
         <xs:minInclusive value="1"/>
        </xs:restriction>
       </xs:simpleType>
      </xs:element>
     </xs:sequence>
     <xs:attribute name="sku" type="xs:integer"/>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
  <xs:attribute name="cemail" type="xs:string"/>
 </xs:complexType>

</xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- This document is valid w.r.t. the DTD but not the XSchema -->
<store xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:noNamespaceSchemaLocation='store.xsd'>
 <products />
<!-- '@' character is not allowed in attribute values, but I will accept in this homework -->
 <customers>
  <customer cemail="joe_at_mailinator.com">
   <name>joe</name>
   <street>666 Main St</street>
   <city>Buffalo</city>
   <state>NY</state>
   <zip>14260</zip>
<!-- Distinguish on order entries: invalid type and invalid reference -->
   <order>
    <orderDate>2011-05-02</orderDate>
    <orderEntry sku="joe_at_mailinator.com">
     <qty>2</qty>
    </orderEntry>
   </order>
  </customer>
 </customers>
</store>
```

3. (50pts) Write an XQuery expression that takes as input an XML document valid with respect to the DTD of problem 1 (call it 'input.xml'), and outputs an XML document valid with respect to the DTD of problem 2. All information contained in the input document must be represented in the output document.

```
(:
  Retrieve the document, then the products ($p), which require no further processing. Then, for each customer (they are unique
  since each has a unique cemail attribute), return all customer fields then, before closing the customer tag, traverse all orders
  placed by that customer and, for each one, output one order element. You cannot output the order element directly as you must
  strip the cemail attribute from the element. All order entries can be returned without further processing.
:)
let $d := doc('input.xml')
let $p := $d/store/products
return
<store>
  { $p }
  <customers>
  {
    for $c in $d//customer
    return
    <customer cemail="{$c/@cemail}">
    { $c/name }
    { $c/phone }
    { $c/street }
    { $c/city }
    { $c/state }
    { $c/zip }
    {
      for $o in $d//order[@cemail = $c/@cemail]
      return
      <order>
      { $o/orderDate }
      { $o/shipDate }
      { $o/shipment }
      { $o/orderEntry }
      </order>
    }
    </customer>
  }
  </customers>
</store>
```

4. (50pts) Write XQuery expression to answer each of the problems below. These problems refer to the 'phd.xml' document, which can be found on the course web page with their respective DTDs.

- (25pts) Find all PhD recipients having the largest number (say, *N*) of proper ancestors. Return a `deepest` element containing an attribute `depth` with the value of *N*. The `deepest` element should also contain one child `recipient` element for each recipient having *N* proper ancestors. For each such recipient, return the recipient element with its `id` attribute and its child `name` element; *do not return* the recipient's `phd` or `recipients` child elements.
- (25pts) For every country in which some recipient received a PhD, return a sequence of `country` elements ordered alphabetically by country name. For each such `country` element, include a `name` element with the country's name and one `classOf` element for every year in which the country had a PhD. Each `classOf` element must contain two attributes: the `year` of the "class" and the `total` number of PhD recipients for that year and country. Within the `classOf` element, return the `name` elements of the respective PhD recipients.

```
(:
  Retrieve the document, then find the maximum number of recipient ancestors of any element. In order obtain the maximum depth,
  we must compute the depth of each recipient, obtain the resulting sequence, and then compute the maximum of this sequence.
  Once the maximum is obtained, it is straightforward to return the respective elements.

  If you used simply the unqualified ancestor, I also accepted your answer.
:)
let $d := doc('phd.xml')
let $N := max(for $r in $d//recipient return count($r/ancestor::recipient))
return
  <deepest depth="{$N}">
  {
    for $r in $d//recipient[count(ancestor::recipient) = $N]
    return
      <recipient id="{$r/@id}">
      { $r/name }
      </recipient>
  }
  </deepest>
(:
  Retrieve the document, then find all distinct countries. Output each contry's name and, for each country, find all distinct years
  in which some recipient obtained a PhD. Find all recipients for that country/year pair, then output the year, the count of recipients,
  and their names.
:)
let $d := doc('phd.xml')
for $c in distinct-values($d//country) order by $c
return
  <country>
    <name>{ $c }</name>
    {
      for $y in distinct-values($d//recipient[phd/country = $c]/phd/@year)
      let $rs := $d//recipient[phd/country = $c and phd/@year = $y]
      return
        <classOf year="{$y}" total="{count($rs)}">
        { $rs/name }
        </classOf>
    }
  </country>
```