For this homework, you should test your answers using Galax, the same XQuery processor we used for Homework 4. You can ssh to eniac-l.seas.upenn.edu and run `∼zives/galax/bin/galax-run` on your query source file(s).

Consider how to integrate 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. These schemas are available at `∼zives/galax/schema-a.xsd` and `∼zives/galax/schema-b.xsd`, with corresponding sample data sets `∼zives/galax/data-a.xml` and `∼zives/galax/data-b.xml`.

**Problem 1 [25 points]:** Write an XML Schema capturing an integrated view of the two schemas. The output schema contain the significant concepts from the two schemas, and it should be nested with items in the outer level, and the associated sellers in the lower level.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns="http://www.mystore.org" targetNamespace="http://www.mystore.org"
  elementFormDefault="qualified">
  <xs:element name="PBAY">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="item" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="itemID" type="xs:int" nillable="false"/>
              <xs:element name="itemname" type="xs:string" nillable="false"/>
              <xs:element name="type" type="xs:string" minOccurs="0"/>
              <xs:element name="description" type="xs:string" minOccurs="0"/>
              <xs:element name="condition" type="xs:string" minOccurs="0"/>
              <xs:element name="manufacturer" type="xs:string" minOccurs="0"/>
              <xs:element name="remark" type="xs:string" minOccurs="0"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="sellers">
          ...
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```
</xs:sequence>
</xs:complexType>
<x:key name="itemKey">
  <xs:selector xpath="item"/>
  <xs:field xpath="./itemID"/>
</xs:key>
</xs:element>
</xs:schema>
Problem 2 [25 points]: Write two schema mappings (views) in XQuery, one over schema-a and the other over schema-b, that output XML conforming to your integrated schema in Problem 1. Finally, define an additional view that simply unions together output from the two views under a common root tag. This final view defines the contents of the mediated schema from the output of the schema mappings.

```
declare function local:view-a() as element(item)* {
    let $slist := document("data-a.xml")/Store/Sellers
    let $ilist := document("data-a.xml")/Store/Items
    let $klist := document("data-a.xml")/Store/Stock
    for $i in $ilist
    return
        <item>
            <itemID>{$i/itemID/text()}</itemID>
            <itemname>{$i/itemname/text()}</itemname>
            <type>{$i/type/text()}</type>
            <description>{$i/description/text()}</description>
            <condition>{$i/condition/text()}</condition>
            <manufacturer>{$i/manufacturer/text()}</manufacturer>
            <remark>{$i/remark/text()}</remark>
            <sellers>
            {
                for $k in $klist, 
                $s in $slist
                where $i/itemID = $k/itemID and $k/sellerID = $s/sellerID
                return
                    <seller>
                        <sellerID>{$s/sellerID/text()}</sellerID>
                        <name>{$s/nickname/text()}</name>
                        <address>{$s/address/text()}</address>
                        <phone>{$s/phone/text()}</phone>
                        <email>{$s/email/text()}</email>
                        <membersince>{$s/membersince/text()}</membersince>
                        <memberdue>{$s/memberdue/text()}</memberdue>
                        <rating>{$s/rating/text()}</rating>
                        <saleInfo>
                            <stocksince>{$k/sincedate/text()}</stocksince>
                            <stockdue>{$k/duedate/text()}</stockdue>
                        </saleInfo>
                    </seller>
            }
        </sellers>
    </item>
}
```
</item>
};

declare function local:view-b() as element(item)*
{
  let $slist := document("data-b.xml")/PBAY/seller
  let $ilist := document("data-b.xml")/PBAY/item

  for $i in $ilist
    return
      <item>
        <itemID>{data($i/@itemID)}</itemID>
        <itemname>{$i/name/text()}</itemname>
        <type>{data($i/@type)}</type>
        <description>{$i/description/text()}</description>
        <manufacturer>{$i/madeBy/text()}</manufacturer>
        <sellers>
          {
            for $s in $slist
              where $s/forSale/itemForSale = $i/@itemID
                return
                  <seller>
                    <sellerID>{data($s/@sellerID)}</sellerID>
                    <name>{fn:concat(fn:concat($s/firstName/text(), " "), $s/lastName/text())}</name>
                    <email>{$s/sellerEmail/text()}</email>
                    <positiveFeedback>{$s/positiveFeedback/text()}</positiveFeedback>
                    <negativeFeedback>{$s/negativeFeedback/text()}</negativeFeedback>
                    <rating>{data($s/@rating)}</rating>
                    {
                      for $fs in $s/forSale
                        where $fs/itemForSale = $i/@itemID
                          return
                            <saleInfo>
                              <quantity>{$fs/quantity/text()}</quantity>
                              <unitprice>{$fs/unitprice/text()}</unitprice>
                              <pictureLink>{$fs/pictureLink/text()}</pictureLink>
                            </saleInfo>
                          }
                    }
                  </seller>
          }
      </item>
}
declare function local:view() as element(PBAY)
{
  <PBAY>
  {
    for $a in local:view-a()
      return $a
  }
  {
    for $b in local:view-b()
      return $b
  }
  </PBAY>
};
Problem 3 [25 points]: Write the following query in XQuery over your mediated schema view from the previous problem: Find all names of items sold by the seller with email mike@wharton.upenn.edu.

for $i in local:view()/item
where $i/sellers/seller/email/text() = "mike@wharton.upenn.edu"
return
   $i/itemname
Problem 4 [25 points]: Manually write the unfolding of the previous query into a query directly over Schema A, i.e., merge the query and the mapping to schema-a, such that you have a query that is posed directly over schema-a.

```xml
<answer>
{
  let $j := ( 
    let $slist := document("data-a.xml")/Store/Sellers
    let $ilist := document("data-a.xml")/Store/Items
    let $klist := document("data-a.xml")/Store/Stock
    for $i in $ilist
    return
    <item>
      <itemID>{$i/itemID/text()}</itemID>
      <itemname>{$i/itemname/text()}</itemname>
      <type>{$i/type/text()}</type>
      <description>{$i/description/text()}</description>
      <condition>{$i/condition/text()}</condition>
      <manufacturer>{$i/manufacturer/text()}</manufacturer>
      <remark>{$i/remark/text()}</remark>
      <sellers>
        { 
          for $k in $klist, 
          $s in $slist 
          where $i/itemID = $k/itemID and $k/sellerID = $s/sellerID 
          return 
          <seller>
            <sellerID>{$s/sellerID/text()}</sellerID>
            <name>{$s/nickname/text()}</name>
            <address>{$s/address/text()}</address>
            <phone>{$s/phone/text()}</phone>
            <email>{$s/email/text()}</email>
            <membersince>{$s/membersince/text()}</membersince>
            <memberdue>{$s/memberdue/text()}</memberdue>
            <rating>{$s/rating/text()}</rating>
            <saleInfo>
              <stocksince>{$k/sincedate/text()}</stocksince>
              <stockdue>{$k/duedate/text()}</stockdue>
            </saleInfo>
          </seller>
        }
      </sellers>
    )
  )
}</answer>
```
where $j/sellers/seller/email/text() = "mike@wharton.upenn.edu"
return
  $j/itemname
}
Problem 5 [Extra credit, 10 points]: Suppose that the two schemas use sellerIDs that are assigned using different schemes; some sellers in each schema are shared with the other schema, except that their IDs are different. Briefly explain how one would need to change the mappings (and what other data would be needed).

Answer:
We would need to add a concordance or correspondence table (or the equivalent XML) defining correspondences between sellerIDs in the two schemas. Such a table would need to be populated by someone who knew which sellers were the same.
Figure 1: Schema A  

Figure 2: Schema B