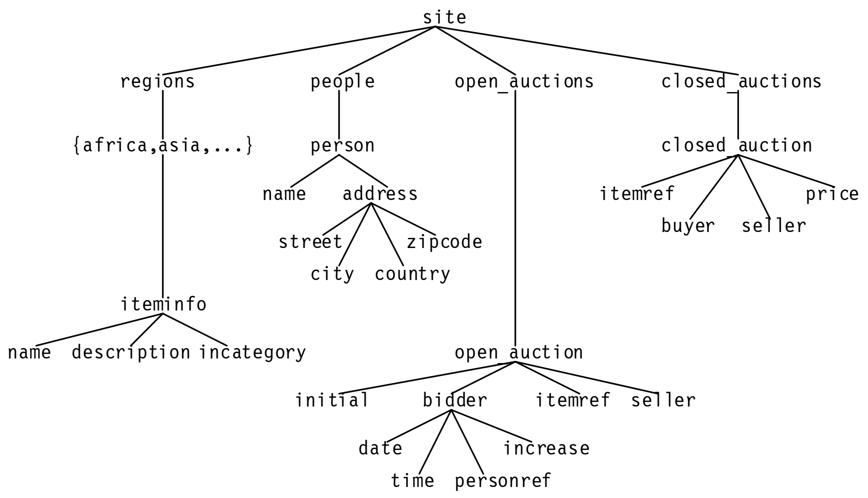
CPS 216 Spring 2004
Homework #3
Assigned: Thursday, March 18
Due: Tuesday, April 6

Problem 1.

Consider an XML document (from a project called XMark) modeling the data maintained by an Internet auction site in `/home/dbcourse/examples/xml-xmark/auction.xml`. The main entities modeled are: items, persons, open auctions, closed auctions, and categories.

- `iteminfo` elements describe items that are for sale or that already have been sold. Each item carries a unique identifier and bears properties like payment method (credit card, money order, etc.), a reference to the seller, a description, etc., all encoded as subelements. Each item belongs to a world region represented by the `iteminfo`'s parent.
- Open auctions are auctions in progress. Their properties include the bid history (i.e., increases over time) along with references to the bidders, a reference to the seller, a reference to the `iteminfo` being sold, etc.
- Closed auctions are auctions that are finished. Their properties include references to the seller and the buyer, a reference to the respective `iteminfo`, the price, the quantity of items sold, etc.
- Persons are characterized by name, email address, phone number, mail address, profile of their interests, a set of open auctions they watch, etc.
- Categories feature a name and a description; they are used to implement classification of items. A category graph links categories into a network.

The figure below illustrates the part of the document structure that is relevant to the problem:



Fire up QuiP's GUI and type in the following query
```
<result>{
  document("/home/dbcourse/examples/xml-xmark/auction.xml")
}</result>
```
to get an overview of the document. Please refer to the document "XML Programming Notes" on the course Web site for instructions on running QuiP.

Write queries in XQuery to answer the following questions. You may test your queries in QuiP's GUI, but please use the batch mode to generate the output to turn in. Because QuiP does not have a sophisticated optimizer, query performance may be heavily influenced by the way you write your queries. If a particular query takes forever to run, consider reordering loops and evaluating selections (filters) as early as possible.

For each question below, say (a), write your XQuery in a file named `hw3-1-a.xquery`, and generate the output file `hw3-1-a.xml` by running `quip hw3-1-a.xquery > hw3-1-a.xml`. Turn in printout of all your `.xquery` and output `.xml` files. ***If the output file is too long, print out only the first page of the output.***

(a) Find names of all items in "`namerica`" region.
(b) Find names of all items that belong to "`category0`."
(c) Find names of all persons whose address is in "`United States`."
(d) Find all buyers who paid for more than $50 in a closed auction. (Note: To get this query working, you might want to consult the "QuiPified" XQuery examples presented in lecture, which can be downloaded from the course Web site under the "Lecture Notes" section.)
(e) Find names of all person who has bidden in an open auction for an item whose name contains the string "`cow`." (Note: To use the XPath built-in function `contains()` in QuiP, you need to convert any node argument to a string explicitly, e.g., `contains(string(name), "cow")`.)
(f) Find names of all persons with address in "`United States`" who never bought anything in closed auctions.
(g) For each open auction whose seller's name is "`Venkatavasu Takano`," print out the following information:

```
<open_auction id="...">
  <bidders total_number="...">
    <bidder_name>...</bidder_name>
    <bidder_name>...</bidder_name>
    ...
  </bidders>
</open_auction>
```

Problem 2.

Consider an XML document describing a thesaurus (from `http://zthes.z3950.org/`) in `/home/dbcourse/examples/xml-zthes/thatt.xml`. The XML document contains a list of terms. Each term has a list of relations with other terms. For example, the following excerpt from `thatt.xml` means that the term `INFORMATION ECONOMICS` is related to terms `INFORMATION`, `INFORMATION INDUSTRY`, `INFORMATION SYSTEMS`, and `INFORMATION THEORY`:

```
<term>
  <termId>663</termId>
  <termName>INFORMATION ECONOMICS</termName>
  <termType>NT</termType>
  <relation>
    <relationType>BT</relationType>
    <termId>662</termId>
    <termName>INFORMATION</termName>
  </relation>
  <relation>
    <relationType>RT</relationType>
    <termId>664</termId>
    <termName>INFORMATION INDUSTRY</termName>
  </relation>
  <relation>
    <relationType>RT</relationType>
    <termId>666</termId>
    <termName>INFORMATION SYSTEMS</termName>
  </relation>
  <relation>
    <relationType>RT</relationType>
    <termId>667</termId>
    <termName>INFORMATION THEORY</termName>
  </relation>
</term>
```

The DTD for `thatt.xml` is `thatt.dtd`, in the same directory. The DTD contains more detailed information on the structure and meaning of `thatt.xml`.

(a) Notice that `thatt.xml` contains lots of redundancy. Within a `relation` element, the `termId` subelement specifies the `id` of the related `term` and `termName` specifies the `name` of the related `term`. However, using the value of `termId`, we can find the `term` element for the related `term` and look up its `termName`. Therefore, it is redundant to store `termName` under `relation`.

To remove this redundancy, and to capture the constraint that a `relation`'s `termId` must be a valid reference to some `term`'s `termId`, we have designed a new DTD in `thatt-new.dtd`. The idea is to make the `termId` of a `term` an attribute of type `ID`, and the `termId` of a `relation` an attribute of type `IDREF`. For example, the term element for `INFORMATION ECONOMICS` would be reformatted as:

```
<term id="term-663" name="INFORMATION ECONOMICS" type="NT">
  <relation type="BT" term="term-662"/>
  <relation type="RT" term="term-664"/>
  <relation type="RT" term="term-666"/>
  <relation type="RT" term="term-667"/>
</term>
```

Your job is to write a program to convert `thatt.xml` into the new format. You have the following options:

1. Write a Java program using SAX API to perform the conversion.
2. Write a Java program using DOM API to perform the conversion.
3. Write an XQuery to perform the conversion.
4. Write an XSLT program to perform the conversion.

Please refer to the document "XML Programming Notes" on the course Web site for instructions on how to compile and run these programs and queries. You can test your code first on a smaller input (`thatt-short.xml`), and compare it with the sample output (`thatt-new-short.xml`). After you are done testing on the small document, test on `thatt.xml`, and validate the output with `thatt-new.dtd`. Instructions on how to validate an XML document can be found in “XML Programming Notes.” Validation will automatically check the consistency of all `IDREF`’s.

You must implement two out of the four options. For each option you implement, turn in a printout of the source code and the first twenty term elements in the result of transforming `thatt.xml`. Show the raw output generated by your code; do not manually edit the output (other than truncating it for printout).

(b) Design and create a relation schema to store the thesaurus data ***using the schema-aware approach discussed in lecture***. Your design should not have the redundancy described in (a). Populate the tables with the data from `thatt.xml`. You should write a program or script to convert the data into flat files and import them into DB2. Do not use individual `INSERT` statements. The choice of programming/scripting language and XML API is up to you. Instructions on how to load a DB2 database can be found in the document “DB2 SQL Programming Notes” on the course Web site; examples can be found in `/home/dbcourse/examples/db-*/` directories.

Turn in the following:

- `CREATE TABLE` statements you used in creating the relational schema, with keys and foreign keys properly declared.
- Printout of the program/script you used to generate DB2 data files for loading.
- Printout of the script you used to import data files into DB2.
- Printout of the result of running the following query for each table you have loaded: `SELECT * FROM table FETCH FIRST 10 ROWS ONLY;`