Homework Assignment 2, HST.950

Handed out:	Lecture 6
Due in:	Lecture 10

Building a Web-based EMR

For this problem, we ask you to explore techniques for how to build a Web-based electronic medical record system. The purpose of the assignment is to convince you, through personal experience, that it is not immensely difficult to do at least a simple job of building such a system, and that the power of the Web and the availability of Web resources makes you very powerful. To keep the assignment within bounds, we make the following assumptions:

- The EMR will be read-only. I.e., there is no way to enter new data or revise existing data.
- We will use the CWS database, with which you should already be somewhat familiar from the first assignment.
- We provide you with a specific implementation environment that already includes some of the basics, from which you can build extensions by re-using patterns and tools already there. The specific environment we provide is one that you are unlikely to have encountered before, though students who have taken 6.171 are likely to have a big leg up. We use the Resin web server, a small and simple Java-based server. The Web site is implemented in JSP (Java Server Pages), which allows you to intersperse HTML fixed content with Java code to compute the dynamic parts of a page. In addition, we have built a few utility classes for Java that you may find helpful (we have) in building this project. And, finally, we have begun a small implementation of the EMR's interface, which you will extend.

There are two (and a half) ways to do this part of the problem set:

- Use a system we have set up especially for the class. <<This option is not available for OCW users.>>
- Configure your own system to use the same technologies we have set up:
  1. Obtain your own copy of the Resin server (which you may use freely for non-commercial development). The current distribution appears to be 2.1.7. Even a "simple" server has an immense number of features and capabilities that lie far outside our intended use. You can find documentation on how to install and run Resin at the Caucho web site or on the "top-level" site on your machine after you have installed it.
  2. Install a copy of the CWS database on your machine. You may already have done this for the first homework assignment. I developed this code using the Access version, but currently I am using MySQL and using the jdbc drivers for MySQL that came with Resin. Presumably other methods would work as well if they are supported on your system. Please recall that this database has been "scrubbed", so none of the names, addresses, dates, etc. in the database should correspond to reality, though the original data were all real. Just in (the unlikely) case that we have mistakenly allowed some identifying data past the scrubbing process, please do not make these data public outside our class.
  3. If you use Access, for which I don't know of direct jdbc drivers, you will need to define a "System DSN" ODBC data source that references the CWS database. Call the data source "cws". This can be done with "Control Panels\Administrative Tools\Data Sources (ODBC)" on your Windows XP, or something similar on other versions of Windows.
  4. Copy the contents of the top-level FTP directory described in the other alternative approach to your machine. You may put it anywhere, but will have to adjust the Resin configuration file to match. I have it at `c:\cws`.
  5. Add definitions to the resin.conf file to define the data source and the cws "web application". The following works on my server:

     ```
     <resource-ref>
       <res-ref-name>jdbc/cws</res-ref-name>
       <res-type>javax.sql.DataSource</res-type>
       <init-param driver-name="com.caucho.jdbc.mysql.Driver"/>
       <init-param url="jdbc:mysql_caucho://localhost:3306/cwsscrubbed"/>
       <init-param user=""/>
       <init-param password=""/>
       <init-param max-connections="20"/>
       <init-param max-idle-time="30"/>
     </resource-ref>
     ```

     and you also need to define the place where the web directory is in your file system:

     ```
     <web-app id='/cws'>
       <app-dir>C:\cws</app-dir>
     </web-app>
     ```

     where `C:\cws` is the location chosen in step 4.

     If you use Access instead of MySQL, I believe you can just replace the corresponding lines above by

     ```
     <init-param driver-name="sun.jdbc.odbc.JdbcOdbcDriver"/>
     ```

```
        <init-param url="jdbc:odbc:cws"/>
```

- *The half-method:* If you are very comfortable building database-backed web sites (e.g., you have done well in 6.171), feel free to build this any way you like.

Explore

First, connect to the skeletal server and note the functionality that we initially provide. The main server page simply lists all the patients in the CWS database alphabetically, along with their "PAT_NUM" identifiers. Each name links to a (dynamic) page that will display information specific to that patient. In the skeleton, this includes only demographics and the problem list, though obviously much other data also exists in the CWS database about these patients.

Take a look at the index.jsp source code for the top page of the site:

```
<h1>CWS Database</h1>
<table border="0" cellspacing="5">
<tr><th>#</th><th>Patient</th></tr>

<%
SqlAccess s = null;
try {
    s = new SqlAccess("jdbc/cws");

    List people = s.retrieveAll("select pat_num,last_name,first_name,title as mid_initl from pat_demograph order by last_name,first_name,mid_initl");

    Iterator it=people.iterator();
    while (it.hasNext()) {
        Entity b = (Entity)it.next();

        EnglishName n = new EnglishName(b.getS("last_name"),
                                        b.getS("first_name"),
                                        b.getS("mid_initl"));

        out.write("<tr><td align=\"right\">" + b.getS("pat_num")
                + "</td><td><a href=\"pt.jsp?id="
                + b.getS("pat_num") + "\">"
                + n.toString()
                + "</a></td></tr>\n");
    }
}
finally {
    if (s!=null) s.close();
}
%>
</table>
```

This code uses three utility classes defined in the edu.mit.lcs package:

1. **SqlAccess**: encapsulates connection and access to the database. Provides methods **retrieve** and **retrieveAll** that return the results of a SQL query. Retrieve yields an **Entity**, and retrieveAll yields a **List** of Entities.
2. **Entity**: An association-list that is used to represent rows from a relational data base, field names/contents from an HTML form, etc. **get** retrieves the value associated with its argument, which is a String. **getS** is like get, but yields an empty string instead of a null. **getH** is like getS, but "HTML-encodes" its contents (e.g., "<" turns into "<"). **getMdyDate** and **getSqlDate** turn various formats of date strings into 12/31/1999-style and 1999-12-31-style dates.
3. **EnglishName**: Understands structured forms of English-style names, which can be constructed from their parts or by parsing a string containing the whole name. **toString** and **toFNF** return the "Smith, Joseph" and "Joseph Smith" versions. **getXXX** and **setXXX** methods exist for each part of the name: first, last, middles, vons, and suffix. This is modeled on BibTeX's analysis of names.

The code above creates a SqlAccess object **s** that holds the database connection. The call to **retrieveAll** selects name parts and patient number from the patient demographics table. We then iterate over all the elements of this list. For each one, we create an **EnglishName** object, and output to the HTML page being constructed one row of a table, which holds the patient number, the last-name-first version of the name, and a hyperlink to **pt.jsp** with this patient number as the **id** argument. (Note that "title as mid_initl" solves an apparent database bug, whereby patients' middle initials are stored under "title".)

You can download the files referenced above (see assignments section of OCW site):

- index.jsp
- pt.jsp
- SqlAccess.java
- Entity.java
- EnglishName.java

Improve PubMed access

You will see in **pt.jsp** that for any patient who has problems associated with his/her case, our skeleton code has hyperlinked each problem name to a complex URL at the National Library of Medicine's PubMed that shows the user "systematic review articles" about the name of the problem. If you look at the NLM's site that this code uses, http://www.ncbi.nlm.nih.gov/entrez/query/static/clinical.html, you will see that in fact there are also four other "research methodology filters" that allow one to search for therapy, diagnosis, prognosis and etiology-related articles about the topic. Interestingly, the filters are actually relatively simple, but have been shown in the Haynes reference to do a good job. *Hint:* Where did I get the large URL embedded in pt.jsp, and the other one in the comment near the end of that file?

Implement an extension to the display generated by pt.jsp that allows the user to determine which of the five kinds of retrieval

PubMed is to perform on the text of the problem name. Think about how your user would perceive various ways you might design and implement this, and argue why you have chosen your approach.

Add Laboratory lookup

The table pat_test_histv contains all the in-house Lab values measured. Design and implement an interface that provides high-level navigation tools to allow a user to look up labs for a specific patient either chronologically (e.g., all lab values measured today) or organized as time series by the quantity measured (e.g., the sequence of serum creatinine values ever measured). Because this table has indications of which values are "normal" or not, be sure you indicate this visually. *Note:* This is certainly an open-ended problem, but I urge you to spend only a limited amount of time on it, and focus on what you consider interesting design issues. For example, you may be tempted to graph various values; unless you are either a wizard Java Applet programmer or have a handy toolkit available, resist the temptation.

Food for Thought

Looking at the rest of the data in CWS, briefly outline what additional capabilities it could support toward having a complete browser for an EMR. This should be just design, not implementation. Estimate how much effort it would take you to actually build it.