Homework 9

Due: 5 Nov 2007

I have a lot of books. While I'm not quite to the level of a lending library, I have enough that they're not all in one place and even if they were I wouldn't always be easily able to find a specific thing; it's big enough that I can't hold every fact about my book collection in my head.

This makes it a prime candidate for a database.

It would probably be instructive from a software engineering angle to make you make me a database, with me as the client you have to satisfy; but that presents too many logistical difficulties and is above and beyond the call for this course. So, instead, I'm going to have you use yourself as a client and make a database for something you have a collection of; it needn't be books, but the collection you have in mind should have the properties I list in the first paragraph above. In particular, the data set should be rich enough to be interesting: probably at least three entity sets and at least four or five relationships. More complexity is fine, although don't make it so big that it becomes infeasible. (You can't use the music-file example we did in class, but there are other music-related data sets that could work.)

If you're stuck for ideas or just want to confirm that your idea will work, feel free to drop me an email about it.

Problem 9.1

Having selected a collection of stuff you own, have access to, or otherwise are familiar with and care about, draw out an E-R diagram to represent the data about this collection in a useful way.

Problem 9.2

Concisely but in prose form (i.e. not a bullet list), explain why your E-R design is appropriate to the task. Take account of the database design principles we discussed in class.

Problem 9.3

Write out a relational database schema based on your E-R diagram. Show your work—that is, explain your reasoning on why each table in the schema is a correct translation of some portion of the E-R design.

Problem 9.4 (×2)

Code up an implementation of your database.

The UI for this can be either an applet or an application, but should have:

- A text box for the filename, and a button to create/load it
- For each table in your schema, one text box per column, and one associated button to add the entry to the table
- For each entity set in your E-R design, a text box where a user can type in a piece of identifying information (e.g., in my example, for an Author, the last name), and an associated button to click which will summon all the relevant information about that entity, including the relationships in which they participate. This should be entirely implementable using simple SELECT ... WHERE ... queries.
- A big text box into which all the query results get dumped. Each new query clears the box before putting its results in.
- A Quit button.

The interface need not be pretty (and indeed in the applet style we've been using, pretty much can't be), but it should be functional. This also applies to query output sent to the big text box. The result of making updates or queries before a file has been loaded is undefined (i.e. it's not your fault, for now, if it makes the program crash).

The back end will use the SQLiteJDBC driver for Java. This interoperates with the `sqlite3` executable, which may make testing somewhat easier. To get started, first you'll need to add the file

```
/home/courses/cs262/sqlitejdbc-v037-nested.jar
```

to your CLASSPATH. The code that handles the top-level interactions with the database will need to import `org.sqlite.JDBC` and `java.sql.*`, and will have the following structure:

Header

```
    Class.forName("org.sqlite.JDBC");
    Connection session = DriverManager.getConnection("jdbc:sqlite:" + filename);
    Statement stmt = session.createStatement();
```

The first statement forces this particular driver to load itself. The second creates a `Connection` object from the given URL; this object is essentially equivalent to a single session at the terminal with `sqlite3`, into which you would type your updates and queries and get responses. You could theoretically have many such objects open on different database files, or even on the same database file (remember ACID?). The third statement creates an extra layer of indirection between the program and the `Connection`; this only becomes relevant if you plan to be scrolling through results of multiple queries at once. (See the documentation for `Statement` for more details.)

Updates

To process an update, you literally send exactly what you would type into the `sqlite3` session to a `Statement` object:

```
    stmt.executeUpdate("create table person (name text, age integer);");
    stmt.executeUpdate("insert into person values ('Chris', 25);");
    stmt.executeUpdate("insert into person values ('Alex', 18);");
    stmt.executeUpdate("insert into person values ('Sam', 27);");
```

That string needn't be a literal, of course, and in a real program certainly wouldn't be, at least not in general.

Queries

Queries are similar, in that you are feeding precisely what you would type at the `sqlite3` session into the `Statement` object. The difference is that queries have return values, aka results, that need to be stored and processed. One might expect this to be done via some sort of `Iterator`, but one would be wrong.

```
    ResultSet tab = stmt.executeQuery("select * from person;");
    while (tab.next()) {
      System.out.printf ("Name: %s   Age: %d\n",
          tab.getString("name"), tab.getInt("age"));
    }
    tab.close();
```

Quite frankly, this is a little gross, but you do what you gotta do.

Footer

```
    session.close();
```

Clean up after yourself! This will commit any updates if you aren't already in autocommit mode.

There are also a few exceptions that Java will make you handle or declare, but I'll let you discover those on your own. The docs for all the various SQL-related classes are in the `java.sql` package in the Javadocs.

A preview of the rest of the term

Hwk 10

A few problems on E-R diagrams and relational algebra. It will go out Wednesday and be due Friday.

Hwk 11

You will improve on whatever you wrote for this homework. There will be two "tracks" here, from which you'll select one:

- Track 1: Do a good UI for your database.
- Track 2: Beef up the back end of your database to make use of more than simple SELECT . . . WHERE . . . queries.

A more detailed spec will go out next Monday, and the project will be due the following Monday, 12 November.