

<http://www.cs.arizona.edu/classes/cs460/fall08/>

### Program #3: Embedded SQL and JDBC

*Due Date: December 9<sup>th</sup>, 2008, at the beginning of class*

**Overview:** Embedded SQL is nice for applications that require more complex calculations than plain SQL can handle, or can handle conveniently. Many DBMSes have 4GLs that can be used for developing nice windowed applications, but even they may not be flexible enough to create the application you have in mind. Having a way to tie a general-purpose programming language (such as Java) into a DBMS gives more application development flexibility.

The United States Census Bureau conducts a census of the US population every 10 years, as required by law. As you might imagine, they gather a huge amount of data in the process, which they summarize and make available over the Internet. Their Gazetteer section has ASCII files that contain some basic demographic information about U.S. states that was gathered during the 1990 census and the 2000 census. The data isn't in a form that can be easily imported into a DBMS, but that's also what makes it particularly suitable for an assignment. That is, it's realistic — it is the fortunate DBA who gets clean, well-formatted data to work with!

**Assignment:** As mentioned above, the Census Bureau supplies data, but its format leaves a lot to be desired from a data importation point of view. If you visit the Gazetteer page (you'll find the URL below) and look at the files, you'll see that they are ASCII files that have lines that look something like this:

```
AZ04019Pima County ...
```

As you can see, these aren't CSV files, nor are they in a format that a DBMS can easily import into a relation.

Your first task is to create tables in your personal Oracle space that contain subsets of the 1990 place and county data for four states of the SW US (Arizona, Nevada, New Mexico, and Utah) and subsets of the 2000 place and county data for those states as well. In particular, here are the subsets that you need in your tables:

Field	Type	Comment
FIPS code	integer	FIPS = Federal Information Processing Standard
Name	string	Remove the type from the names
Type	string	Found at end of Name in source files
Population	integer	
housing units	integer	
land area	integer	Convert 1990 values to sq. meters
water area	integer	Convert 1990 values to sq. meters
latitude	double	Convert 1990 values to decimal degrees
longitude	double	Convert 1990 values to decimal degrees

The place and county files are available from the Gazetteer page (see below, or on the class web page, for the link). You'll need to extract the lines for these four states (and perhaps reformat them) so that your program can read the attributes.

If you would like to share the data cleaning and formatting chores with one or more of your eager, trustworthy classmates, you may. Just be sure that you include in your documentation the names of those who worked with you on this. Writing the Java application is still a solo project; do not solicit or offer help with that.

(Continued ...)

The second part is to write (again, on your own!) a Java/JDBC program that creates and populates the tables, and then provides a simple menu-based way for users to get summary information about these states from your Oracle database. In particular, your program is to allow a user to learn the answers to each of the following questions:

1. Which places (if any) are new in these states since 1990? Display them in increasing order by population.
2. Which places (if any) moved the furthest in that ten year span? (Sounds like an odd question, doesn't it?) Display them in decreasing order by distance moved.
3. Given a substring and census year, display all of the place names and county names (and their populations) from that year that have that substring (upper or lower case) at the beginning of their names and, in a separate list, those that have the substring in any other location within their names. Thus, a place named "Oogoogle" would be in both lists for the substring OOG, but "Oogly" would be in the first list only.
4. Given a latitude and a longitude, which place **and** which county are closest to that location?
5. For a given county and year, what are the names, land areas, and water areas of the next two larger and next two smaller counties in land area AND in water area? (Thus, your output should list up to eight counties.)
6. A query of your choice! Try to come up with a good one. You'll lose a few points for creating a trivial query.

For some of these, writing queries will be easy. For others, you may have to issue multiple queries and do some reorganizing of the data within the program before you are ready to display it. Also, be aware that your user interface doesn't have to be complex — just a simple text menu of the choices is fine.

When the user asks to leave your program (yes, you need to let them quit!), make certain that your program DROPS all tables it created when it was launched. (This will make it much easier for us to test your programs.)

**Data:** As mentioned above, the data for this program can be found on the U.S. Census Bureau's Gazetteer site: <http://www.census.gov/geo/www/gazetteer/gazette.html> ASCII place and county files are available for both 1990 and 2000, plus field information for both years.

**Hand In:** Turn in your printed program code (fully documented according to the Programming Style guidelines, of course) and the output of the queries as run on a few representative test cases of your own design. In addition, use 'turnin' to submit your source code AND source data files (that is, everything we'll need to compile and run your program). The submission folder is **cs460p3**.

#### Other Requirements and Hints:

- For whatever help it may be, here's the content of my CLASSPATH shell variable:  
`CLASSPATH=./opt/oracle/product/10.2.0/client/jdbc/lib/ojdbc14.jar`
- When dealing with string comparisons in SQL, you soon learn that SQL is case-sensitive. To compare strings, try converting the string to all upper or all lower case. For example:  

```
select * from AZplaces1990 where lower(zipcode_name)='tucson';
```

The corresponding upper-case conversion function is `upper()`.
- Be aware that the data may be missing some field values, some of the rows of some of those files may be summary data, etc. This is what is meant by 'unclean' data. One guess as to who gets to perform any necessary cleaning ...
- Curious as to how to compute distances between locations? Seek out the Haversine formula; it will suffice for this program.
- And the usual reminder: A correct query is a query that produces the correct result *in a logically correct way!* Write queries that will function correctly even if the relations' content changes.