CSc 460 — Database Design
Spring 2023 (McCann)

http://u.arizona.edu/~mccann/classes/460

Program #3: JDBC

Due Date: March 30th, 2023, at the beginning of class

Overview: Embedding SQL within another programming language is nice for applications that require more complex calculations or manipulations than plain SQL can handle. Many DBMSes have add–ons 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.

Since 1971, the Central Sierra Snow Laboratory (CSSL), a research station of UC–Berkeley, has collected data on the snowfall at Donner Pass in the Sierra Nevada mountains. They provide seasonal CSV files that include snowfall measurements ending with September 30th of the year in the file name, and starting with October 1st of the previous year. For this assignment, we have provided lightly–trimmed CSV files for the years 1979, 1989, 1999, 2009, and 2019. It's easy to write a program that reads the CSV files and inserts the data into tables of a DBMS; it's less easy to adjust the file content to be consistent from year to year, but you'll need to do that, too.

Assignment: For this assignment, you will need to do the following, probably in this order:

1. Get the five data CSV files from the class data directory on lectura. The Data section has the location.
2. As necessary, "scrub" the files to make them consistent and suitable for importing into an Oracle database. (Again, see the **Data** section, below.)
3. Within your Oracle database, create five tables, one for each year, with good, meaningful table and attribute names.
4. Import the CSV files' content into the corresponding relations.
5. Write a Java program named `Prog3.java` that offers the user a simple text menu of queries that can be asked of the snowpack data and uses JDBC and SQL to interact with the database to produce the correct answers (with the help of some Java processing, if needed/desired). The queries are:

   (a) For a (table) year provided by the user, which day (or days, in the case of ties) has the greatest difference in max and min air temperatures? (By "(table) year" we mean the year used in the table's CSV filename. Each table covers parts of two years; the filename is based on the second of those years.)

   (b) For a (table) year provided by the user, how many days had no precipitation, how many had only rain, how many had only snow, and how many had a combination of both?

   (c) Considering all tables, what are the 10 days with the largest drop in snowpack depth as compared to the next day, and what was the Air Temp Max the first day? For example, if the depth was 60cm with a high of 20C on April 6 and 15cm on with a high of 12C April 7th, the drop of 45cm would be credited to April 6th's 20C high.

   (d) A query of your own design! The only requirements are that your query must get at least one piece of information used by the query from the user, and the query must use data from multiple tables.

   For each of these queries, you may answer it using one or more SQL SELECT statements, with as much or as little additional Java processing as you wish (or need) to use.

(Continued...)

Data: You can find the CSV files on lectura. This is the location and name of the 1979 file:

```
/home/cs460/spring23/CSSL1979.csv
```

Just change the year to 1989, 1999, etc., to get the names of the others.

The CSV formatting is reasonably consistent from year to year, but there are several issues you will need to address before you can create relational tables from the data. (1) Skip the metadata rows (they aren't real data, they just describe the real data that follows). (2) All air temperatures are to be stored as real numbers. (3) When measurements are recorded as "T" (for 'trace') or "NA" ('not available'), they need to be stored as zeroes in your tables. (4) Values of '- -' (two adjacent hyphens) and missing values are to be represented with NULL. (5) All of the dates need to be converted to SQL date values. (You're right; we didn't talk about SQL date types in class. You'll have to educate yourselves.) You may well find additional issues. If so, ask us about them; we'll let you know what to do.

The menu format, structure of the requests for user info needed by the queries, etc., is up to you.

Output: Your application is to display the answer to a query in a clear, easy to read format of your choice (remember, you're writing your own program; you are not restricted to SQL's output format).

Hand In: You are required to submit all of your well–documented application program file(s) — including any code written to automate the database population process, although that code need not be well–documented — via turnin to the folder `cs460p3`. Name your main application program's `main()` class `Prog3`, so that we don't have to guess which file to compile, but feel free to split up your code over additional files as appropriate for good code modularity. Please do not 'package' your files; just 'turnin' them as–is.

Want to Learn More?

- The CSSL's web page is `https://cssl.berkeley.edu/`
- The original data files for all of the years from 1971 through 2019 can be found here: `https://doi.org/10.6078/D1941T`

Other Requirements and Hints:

- Because we will be grading your program on lectura using Oracle, it needs to run on lectura and use Oracle.
- If you wish to share any necessary data pre–processing and "scrubbing" chores with your classmates, that's fine. Stop collaborating when you're coding `Prog3.java`; that needs to be your own work. In your documentation, be sure to credit those who helped you with the data organization. DO NOT post scripts, etc., on Piazza; we don't want one generous person doing all of the dirty work for the entire class.
- It is OK to share query *results* on Piazza. Doing so can help you discover that your query isn't finding everything (or is finding more than) it should be finding.
- (IMPORTANT!) Make certain that your database tables are accessible to us (by `GRANT`ing us `SELECT` privileges) and that your relations are prefixed with "`yourNetID.`" in your application so that we can execute your program against your database, just as my tables for Homework #3 were accessible with the "`mccann.`" prefix. The form of the `GRANT` command is:

  ```
  GRANT SELECT ON tablename TO PUBLIC;
  ```

  If you need to delete and re–create your tables, you'll need to re–issue the `GRANT` commands on the new tables.
- Avoid the temptation to wait to start writing the JDBC code and your application program until you have all of the data loaded into tables. You can (and should!) create and populate small tables for testing purposes early in the development process.