CSCI 521 Database Architecture
Fall 2018
Homework Assignment (Option 1)

JDBC & Spatial Database

Due: December 12, 2018 @ 23:59 PM PST

This assignment will involve implementation of a simple spatial database, accessing a database programmatically using Java Database Connectivity (JDBC). Please pay extra attention to the respective due dates and submission requirements. The due dates is not intended as an indication of the amount of time and effort needed for doing it; you are advised to start as soon as possible so as not to lose any time.

Preliminaries: MySQL Spatial Extensions

MySQL supports the storage, indexing and querying of spatial data through spatial extensions based on the OpenGIS geometry model. No additional download or installation is required.

Read the documentation! Supported types, syntax and behavior vary between different DBMS's.

It should be noted that MySQL support for spatial analysis is not complete. A few key functions (such as `DISTANCE()`) are unimplemented, or are only approximated. In particular, functions for testing spatial relations between geometric objects are approximated using minimum bounding rectangles (MBRs) instead of the actual geometry object itself. For the purposes of this assignment, **you should use the OpenGIS-standard functions (**see the Reference Manual); you will not be penalized for approximation errors due to the MySQL implementation.

Preliminaries: Java

For this assignment, we will use **Java Standard Edition JDK** . Note that we will not be able to accommodate any compatibility issues if you use a different version.

You are free to use any Integrated Development Environment (IDE) such as Eclipse, NetBeans, Visual Studio, etc. if you wish. However, you **must be able to compile and execute your code from the Windows command line (or Linux/Mac shell) environment**, outside of the IDE.

Preliminaries: JDBC

Java Database Connectivity (JDBC) is a standard database-independent API that could be used to interact with virtually any database from within your application. In this part of the assignment, you will gain practical experience by writing a Java program that will use JDBC to access and query a spatial database.

To connect to your MySQL database from within your Java program, you will need to use JDBC. You are advised to use the most recent available release of the official MySQL JDBC driver.

You may NOT use any other third-party libraries apart from MySQL Connector/J.

Connecting to the Database

To connect to your MySQL database from within your Java program, you will need to supply it with the necessary connection parameters. For this assignment, these parameters **must** be stored in a separate configuration file to be passed as an argument to your Java program during execution. Your program should read the configuration file and use the parameters contained therein. **Do not hardcode any of the connection parameters in your Java source code.**

A sample `db.properties` file has been provided for your reference. The five lines of this text file correspond to the host, port, database name, username and password for the database. Default values for host and port values have been used; you may need to adjust them accordingly if you had changed your settings when installing MySQL. Obviously, you will need to change the username and password according to your individual system setup in order to connect to your MySQL instance.

For grading, we will of course use a different `db.properties` configuration based on the setup of the machine used for grading. Make sure you **follow the configuration file format. We will not be able to grade your assignment if your program fails to parse our database configuration file.**

Scenario Description

The CSULB Police Department (https://www.csulb.edu/university-police) seeks to provide a safe and secure environment in which the social and academic endeavors of the University community may be fully realized.

CSULB-PD organizes its officers into squads, and divides its coverage area into smaller areas, called "zones". Each zone is has a squad assigned to provide coverage. CSULD-PD also has pre-designated patrol routes for officers to follow when patrolling. Lastly, CSULD-PD needs to track crimes and incidents that happen within its coverage area.

In order for CSULD-PD to better keep track of its patrol resources so that it can manage and deploy them more efficiently and serve the community better, you have been tasked to build a database that will help monitor and track the location of CSULD-PD officers, patrol routes, and zones, etc.

Input Files

You will be given the following input files:

- `zone.txt`. Each zone is represented by a 2D polygon, with the following data.:
  - Col 1: Unique Zone ID
  - Col 2: Zone name
  - Col 3: Currently assigned squad number
  - Col 4: Number of vertices on the polygon
  - The numbers after column 4 are the coordinates of the vertices. They are ordered as
    $long_1, lat_1, long_2, lat_2, \ldots, long_n, lat_n$.

  For example, a row:
  15, "Zone X", 3, 4, -118.3, 34.03, -118.3, 34.02, -118.2, 34.02, -118.2, 34.03

  Represents a zone with ID 15, the name "Zone X", has squad number 3 assigned to it, and having 4 vertices with geo-coordinates:
  34.03°N 118.3°W, 34.02°N 118.3°W, 34.02°N 118.2°W, 34.03°N 118.2°W

- `officer.txt`. Each officer has:
  - Col 1: Unique Badge number
  - Col 2: Name
  - Col 3: Squad number
  - Col 4 & 5: Current location longitude and latitude.

- `route.txt`. Each patrol route has:
  - Col 1: Unique Route number
  - Col 2: Number of vertices on the route
  - The numbers after column 2 are the coordinates of the vertices:
    $long_1, lat_1, long_2, lat_2, \ldots, long_n, lat_n$.

  Patrol routes do not loop back to the first point.

- `incident.txt`. Each incident has:
  - Col 1: Unique Incident ID
  - Col 2: Type of incident
  - Col 3 & 4: Incident location (longitude and latitude)

The data files provided to you are samples demonstrating file format only. When grading, we will test your programs using a different set of unseen data. As such, you may wish to create additional data for your own testing purposes.

Spatial Database Implementation (15 points)

Prepare and submit a `createdb.sql` file containing SQL statements that will create a database `PublicSafety` containing all required tables, using appropriate types for all data. You **must** use appropriate spatial data types where applicable, and implement spatial indexes for spatial data. This will require you to use MyISAM tables – note that MyISAM tables do not support foreign keys. In addition, it should include applicable primary keys, indexes, and any other DDL statements you think you might need for the application.

Also prepare and submit a `dropdb.sql` file that drops the database, along with all tables, types, and all other objects that were created by your `createdb.sql` file.

As before, we will process your SQL files by running them using the `mysql` command line client.

Populating the Database (15 points)

You are required to implement a Java program `Populate.java` that gets the names of the input data files as command line parameters and populate the data contained within them into your database by executing individual insert statements for each row using JDBC. Note that you should use the JDBC `PreparedStatement` construct when executing the same statement repeatedly.

Your program should be compiled and executed from the Windows command line or Linux/Mac shell. It should be compiled like this:

```
> javac -classpath .;mysql-connector-java-5.1.18-bin.jar Populate.java
```

And executed like this:

```
> java -classpath .;mysql-connector-java-5.1.18-bin.jar ↵
      Populate db.properties zone.txt officer.txt route.txt incident.txt
```

Sample command line batch files / shell scripts for both compilation and execution are provided.

Note that every time you run this program, it should start by first removing the previous data in your tables; otherwise the tables will have redundant or incorrect data.

Querying the Database (70 points)

You are required to implement a Java program `Hw.java` that provides the capability to run queries on the system from the Windows command line or Linux/Mac shell environment. It should be compiled by running the provided `compile_hw.bat` (Windows) or `compile_hw.sh` (Linux/Mac):

```
> javac -classpath .; mysql-connector-java-5.1.18-bin.jar Hw.java
```

And executed like this:

```
> java -classpath .;mysql-connector-java-5.1.18-bin.jar ↵
      Hw db.properties <query_number> <arguments>
```

Depending on the query number, your program should take the arguments provided and process the corresponding query, then terminate. (Query details are given below.)

Sample command line batch files / shell scripts are also provided:

- `p1q1.bat` thru `p1q4.bat` (Windows)
- `p1q1.sh` thru `p1q4.sh` (Linux/Mac)

You should test your program to ensure that they can be executed using the provided `*.bat` or `*.sh` files, depending on your OS. You are encouraged to come up with your own test cases and modify the query arguments in the provided `*.bat` or `*.sh` files to test your program. However, **you should not modify the classpath – you must use it as-is.**

Q1: Range Query

Given a polygon, list all incidents that occurred within the polygon. For instance, given:

```
> java -classpath .;mysql-connector-java-5.1.18-bin.jar Hw db.properties↵
  q1 4 -118.3 34.03 -118.3 34.02 -118.2 34.02 -118.2 34.03
```

Your program should find all incidents that occurred in the polygon bounded by 4 vertices:
34.03°N 118.3°W, 34.02°N 118.3°W, 34.02°N 118.2°W, 34.03°N 118.2°W

And display the ID, location and type of each incident on screen, one line per incident, ordered by incident ID:

```
15  34.0225323,-118.2507421  Assault
32  34.0267473,-118.2313245  Robbery
47  34.0244292,-118.2892469  Petty Theft
```

Q2: Point Query

Given an incident ID and a distance in meters, find all officers that are within the given distance of the incident. For instance, given:

```
> java -classpath .;mysql-connector-java-5.1.18-bin.jar Hw db.properties↵
  q2 15 250
```

Your program should find all officers that are within 250 meters of the location of incident #15 and display the badge number, distance (rounded to nearest meter) and name of each officer, ordered with the nearest officer first:

```
657  103m  Letitia Mendoza
321  167m  Samuel Getz
759  243m  Bobby Farley
```

IMPORTANT: to calculate distance in meters, you must use the provided `gcdist()` function[1]. You will need to run the provided `gcdist.sql` file to create the function and add this functionality to your database.

Q3: Find Squad

Given a squad number, return the zone that it is currently assigned to, and list all officers assigned to that squad. For each officer, also indicate if their current location is inside or outside of the assigned zone. For instance, given:

```
> java -classpath .;mysql-connector-java-5.1.18-bin.jar Hw db.properties q3 9
```

Show the name of the squad's assigned zone and list the officers in the squad in ascending order of their badge number. Your program output should look like:

```
Sqaud 9 is now patrolling: University Village
143  IN   Rebecca McDonald
597  OUT  Tom  Anderson
654  IN  Mike  Hendley
816 IN Charlie Fox
```

[1] For anyone interested, this function computes the Great Circle Distance using the Haversine Formula.

Q4: Route Coverage

Given the route number of a patrol route, list the zones that the patrol route passes through (intersects). For instance, given:

```
> java -classpath .;mysql-connector-java-5.1.18-bin.jar Hw db.properties q4 7
```

List zones in ascending order by their zone ID. Your program output should look like:

```
3 Cardinal Gardens / Century Apartments
9 University Village
22 University Ave
```

Notes

You **must** follow the sample output formats for each query as shown for each query.

You are expected to **incorporate sensible logic** into your programs. For example, if the user tries to run a query involving an inexistent incident, you should return an informative error message. Likewise, you should print an informative message if a query returns no results.

Your programs are also expected to **handle database errors gracefully**, and return informative error messages before exiting rather than terminating abruptly. For instance, the connection to the database may fail, or the database user (in `db.properties`) may not have the necessary permissions to perform the required operations.

You may assume, however, that your program will always be run according to the given command format – i.e., arguments will be given in the specified order, and there will never be missing arguments.

FAQ

Q: Can I use Eclipse / NetBeans / Visual Studio / (insert IDE here)

A: Yes, but, you **must ensure that you are able to compile and execute your code from the Windows command line (or Linux/Mac shell) environment**, outside of the IDE. You should check using the provided `*.bat` or `*.sh` files before submission to ensure conformance.

Q: Can I use other java libraries or packages?

A: No. You may **NOT** use any other third-party libraries apart from MySQL Connector/J.

Q: What package should I use for my Java program?

A: The default package will suffice – no package declaration is needed.

Q: Where will `db.properties` be located? Where will the input files be located?

A: The location of `db.properties` and input files will be specified using relative path with respect to the working folder, as demonstrated in the given `*.bat` and `*.sh` files.

Q: How should I set up my classpath variable?

A: Do **not** use the CLASSPATH system environment variable. Your environment variables only work on your system, and it is not guaranteed that another system (the one used for grading!) will have its environment variables defined exactly the same as yours. Furthermore, the program source files (`*.java`) and compiled class files (`*.class`) may end up in different file system locations than you have on your computer. For these reasons, you should not rely on environment variables for the compilation/execution of your program.

The recommended way to ensure that your programs will compile and run correctly anywhere is by using the -classpath switch in the compiler (`javac`) and interpreter (`java`) to specify the classpath on compilation or execution, as demonstrated in the given `*.bat` and `*.sh` files.

During grading:

- Your source (`*.java`) files will be placed in the working directory
- After compilation, the class files (`*.class`) will end up in the same directory
- We will place the needed MySQL connector `.jar` file there as well
- And then execute from there.

Q: Can I use file names that are different than the ones specified?

A: You **must not** alter the name, case, spacing or extension of any of the specified files. The grading process uses these exact names and altering them will cause some files not to be graded.

Submission Instructions

- This assignment is to be submitted ***electronically*** on beachboard , using dropbox option.

  - You should have 2 SQL files: `createdb.sql` and `dropdb.sql`, and at least two Java source files, `Populate.java` and `Hw.java`. Failure to adhere to filenames will incur a 5 point penalty. Make sure the file extensions are correct!
    - You may optionally organize your program into multiple Java classes. If you do so, make sure to include all the necessary source files and indicate accordingly in the compilation and execution instructions in your readme.txt
  - Also prepare a `readme.txt` containing your name, id and email address. You may also include comments and note any assumptions you made in the file (optional).
  - Do not submit *.class files. We will compile your program from submitted source.
  - Compress your `readme.txt`, SQL files and Java source files into a single zip archive and name it `lastname_firstname_hw.zip`. Use only standard zip format. Do **not** use other formats such as zipx, rar, ace, etc. Improper filename or format will incur a 5 point penalty. There is no need to include MySQL Connector/J in your submission.
  - Make sure that you have attached the file the when submitting. Failure to do so will be treated as non-submission. Late policy applies until correctly submitted.
  - Successful submission will be indicated in the assignment's submission history. We advise that you verify the timestamp, download and double check your zip file for good measure.
  - It is ok to modify and re-send your file. We will automatically assume the submission with the latest timestamp as your final answer for grading. **Caution**: if you had submitted earlier but choose to amend your submission *after* the deadline has passed, it will be treated as a late submission, with penalties as specified below.

- This assignment is electronically by the due date and time shown above.
- There is a 20% deduction per day for late submissions. Assignments more than 5 days late will not be accepted.

Student Collaboration Policy

You may discuss general strategies to be used on the assignment, but please refrain from revealing and/or hinting at any answers. In this course we encourage you to study together, but do remember that all work submitted for the class **is to be done individually** and within the realm of the CSULB Academic Integrity Guidelines. Violations of the Student Conduct Code will be filed with the Office of Student Conduct, and appropriate **sanctions will be given**. If you have questions about what is allowed, please discuss it with the instructor.

Your source code will be compared with other students' code with MOSS.
https://theory.stanford.edu/~aiken/moss/

Similar code will be disqualified (regardless of which one is the original)