Sudoku Solver
Intro to SQL

Out: Wednesday 09/26
In: Tuesday 10/9

1 Sudoku Solver

In this assignment, you will familiarize yourself with SQL. You will compose a few SQL queries and some Java code to solve 2 x 2 Sudoku puzzles [1]. For more info on Sudoku puzzles, visit `http://en.wikipedia.org/wiki/Sudoku`.

1.1 Tools

1.1.1 JDBC

We will be using SQLite database[2] through a JDBC driver (Java Database Connectivity)[3]. The JDBC SQLite driver is included as a part of your stencil code. There will not be an official help session on how to use JDBC, but TAs will be happy to answer questions on hours or via email. Students are highly encouraged to check out `http://web.archive.org/web/20100814175321/http://www.zentus.com/sqlitejdbc/`, which has a wonderful tutorial on working with JDBC and SQLite.

1.1.2 SQLite

SQLite is installed on all Sunlab machines. It can be accessed from the command line using sqlite3. To create a new database and import an existing schema, simply run

```
sqlite3 db-file < /path/to/schema.sql
```

For more information on using SQLite from the command line, see `http://www.sqlite.org/sqlite.html`

1.1.3 Eclipse

The support code (available at `/course/cs127/pub/sudoku-solver/stencil.tgz`) is already bundled in an Eclipse project, so we highly recommend copying the support code to your course directory and using the Eclipse editor to edit your Java code.
To import the code to Eclipse, complete the following instructions:

[1] Traditional Sudoku puzzles are 3 x 3 Sudoku puzzles. We restrict ourselves to 2 x 2 puzzles because of the astronomical number of possible solutions for 3 x 3 puzzles , but the code you write will be easily generalizable to 3 x 3 and 4 x 4 puzzles.

[2] `http://www.sqlite.org`

[3] `http://java.sun.com/javase/technologies/database/index.jsp`

1. Expand the stencil code inside your course directory. That should create a directory named "sudoku-solver"
2. Open Eclipse. From the top menu bar, navigate to File → Import.
3. From there, expand the "General" tab, and select "Existing Projects into Workspace."
4. Click the "Browse" button next to "Select root directory" and browse to the sudoku-solver directory inside your course directory. Click OK.
5. Check the box next to the SudokuSolver project (if it isn't already checked) and click Finish.

1.2 Methodology

Normally people try to use logical inference to solve a Sudoku by applying a set of rules derived from the rules of the game. This solution seems like a lot of work. After all, a Sudoku puzzle is like a finger print, which uniquely identifies exactly one solution[4]. Databases are designed to efficiently store, retrieve, and modify large bodies of data, so a database is perfect for solving Sudoku puzzles.

You will be working, primarily, with two tables. First, we provide you with a large table that holds all valid 2 x 2 Sudoku solutions called `allsolutions`. Each row of `allsolutions` contains a *location* and *number*, indicating that the box at location *location* holds number *number*. A *sol_id*, or solution identifier, used to group these boxes together in whole solutions. Second, you will need to make a table called `constraints`. This second table is used to store all of the boxes that have fixed, known values. The table has two attributes: location and a number (indicating that the box at the given location must hold the given number).

We have provided a GUI for specifying puzzles, and visualizing solution statistics (such as how many possible values can occupy a given square, and the range of those values). This GUI works with a `SudokuSQLClient` object, which is responsible for opening a connection to the database and performing queries to gather the relevant information.

2 Your Assignment

Your job is to write a class which implements all the functionality of the `SudokuSQLClient` interface:

- public void addConstraint(int loc, int num) throws SQLException;

[4]Unfortunately, there are many solutions to choose from: 288 for 2 x 2 Sudoku, and 6,670,903,752,021,072,936,960 for 3 x 3 Sudoku.

Called when the user constrains a previously unconstrained box. This function should execute a SQL query that adds a row to `constraints` whose location is *loc* and whose number is *num*

- public void removeConstraint(int loc) throws SQLException;

  Called when the user removes the constraint from a box. This function should execute a SQL query that removes the appropriate constraint from `constraints`.

- public void updateConstraint(int loc, int newNum) throws SQLException;

  Called when the user changes the constraint on a box from one number to another. This function should execute a SQL query that modifies the constraint in-place to indicate that *loc* holds *newNum*.

- public CellStatistic[][] solve();

  Called when the user tries to solve the puzzle/gather statistics. This function should execute a query (or fixed sequence of queries) that uses the two tables to figure out which solution(s) (if any) satisfy the constraint table.[5]

  This function returns a 4 x 4 array of CellStatistics (the first dimension is the box column, and the second dimension is box row).[6] An individual CellStatistic object keeps track of the range of possible values for a particular box (minimum, maximum, and quantity). The CellStatistic objects should be null if there are no possible solutions.

  Try to use the database and SQL to do as much of the work as possible here. In particular, it is not acceptable to read the table of constraints into your Java program, then construct a query over just the allsolutions table using those constraints.

Restrictions

As mentioned in the comments for `solve` above, you must perform all operations in the SQL queries, not in your program. For example, you cannot read the entire table of solutions into a Java collection and then try to find a matching puzzle by brute-force.

To simplify the application logic, no more than one instance of the application will be operating at a time. Be sure to take into account what happens if your application is quit and then opened again!

[5] Watch out for the case where the constraint table is empty! In this case, either all solutions should be returned or none.

[6] For example, the statistics for the box in column 0, row 3 is stored in the result indexed at [0][3]. See CellStatistic.LocToCol and CellStatistic.LocToRow

3 Handin

You can handin your project by running the following command from the directory containing all your files:

```
/course/cs127/bin/cs127_handin sudoku-solver
```