

Recitation 1

SQL

Out: Monday 09/28

In: Monday 10/4

1 Background

SQL (Structured Query Language) is one of many languages used to programmatically interact with databases. As you learned in class, this language provides useful high level abstractions for retrieving and modifying data. SQL has the advantage of having many efficient and freely available implementations to work with.

2 Before lab

Before coming to lab, try to complete the assignment to the best of your ability. The assignment consists of three unrelated problems:

1. The reconstruction of a SQL query with a HAVING clause
2. The Wikipedia foreign key problem
3. The Sudoku solver

3 SQL Basics/Part 1

The basics of SQL have been covered in lecture, but for a quick refresher, we would like you to solve this puzzle.

Now here is a query we have constructed such that for each unique rating in the table, calculates the average of all the employees that hold that rating and are older than 18 years. **Reconstruct it without the HAVING clause.** *Hint: The slides MUST have mentioned something about this!*

```
SELECT Employees.rating, AVG(Employees.age) AS Average FROM Employees WHERE  
       Employees.age > 18 GROUP BY Rating HAVING COUNT(*) > 1
```

Which version of the query would perform faster? Does actual practice bear out your hypothesis?

4 Foreign keys/Part 2

The following table schemas (see next page) are derived from MediaWiki, the software that runs Wikipedia. There is a 1-to-many relationship between wiki page and wiki

revision; that is, for each unique article on Wikipedia there is one record in the wiki page table. Each revision of that page made by users will create a new record in the wiki revision table. Augment the schema to add the appropriate constraints for each question. Note: that the following SQL is for MySQL (auto increment is not standard SQL). Please write your answers to follow the SQL standard whenever possible. If you do use vendor-specific, non-standard SQL, please indicate which DBMS you are writing for (e.g., Oracle, Postgres, DB2). For each answer, you may either modify the CREATE TABLE schema or provide ALTER TABLE commands.

4.1

Modify the wiki page table so that each page title is guaranteed to be unique. Wikipedia corrects the capitalization of page titles before storing it in the database, so you do not have to worry about case. Modify the wiki revision table so that no page can have more than one revision with the exact same timestamp.

4.2

Wikipedia is obviously a target for spammers. Modify the wiki revision table to add a constraint that prevents a user from adding text that includes a URL containing the address “mister-spammer.com” into the rev text field.

HINT: You will want to use the LIKE binary predicate operator.

4.3

Modify the wiki revision table so that if an article’s corresponding wiki page record is deleted, the database will also delete all of the revisions for that page.

4.4

Wikipedia allows trusted users to write special programs (bots) to correct common mistakes that users make (e.g., formatting errors, spelling). In order to prevent the site from being overloaded with too many changes from these bots, they need to be throttled. Write a global constraint to make sure that no single rev user entry creates a new wiki revision record in less than a 10 second span. Why is a global constraint problematic? Can you think of a better way to do this?

5 MediaWiki Schema

```
A      MediaWiki Schema
--
-- Core of the wiki: each page has an entry here which identifies
-- it by title and contains some essential metadata.
--
CREATE TABLE wiki_page (
  -- Unique identifier number. The page_id will be preserved across
  -- edits and rename operations, but not deletions and recreation
  page_id int unsigned NOT NULL auto_increment,
  -- The rest of the title, as text.
  -- Spaces are transformed into underscores in title storage.
  page_title varchar(255) binary NOT NULL,
  -- Number of times this page has been viewed.
  page_counter bigint unsigned NOT NULL default '0',
  PRIMARY KEY page_id (page_id)
);
--
-- Every edit of a page creates also a revision row.
-- This stores both the actual text and metadata about the revision
--
CREATE TABLE wiki_revision (
  rev_id int unsigned NOT NULL auto_increment,
  -- Key to page_id. This should never be invalid.
  rev_page int unsigned NOT NULL,
  -- Page text
  rev_text text NOT NULL default '',
  -- Key to user.user_id of the user who made this edit.
  -- Stores 0 for anonymous edits and for some mass imports.
  rev_user int unsigned NOT NULL default '0',
  -- Timestamp
  rev_timestamp timestamp NOT NULL,
  -- Records whether the user marked the 'minor edit' checkbox.
  -- Many automated edits are marked as minor.
  rev_minor_edit tinyint unsigned NOT NULL default '0',
  -- Length of this revision in bytes
  rev_len int unsigned,
  PRIMARY KEY rev_page_id (rev_page, rev_id)
);
```

6 Sudoku Solver/Part 3

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>.

6.1 Tools

6.1.1 JDBC

We will be using PostgreSQL database² through a JDBC driver (Java Database Connectivity)³. The JDBC PostgreSQL driver is available in `/course/cs127/lib/postgresql-8.2-504.jdbc3.jar`; you will need to include this driver in your build path in order to get your code to work.

6.1.2 Eclipse

The support code 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. Be sure and copy the entire directory, as it contains all of the support code and some Eclipse configuration files (the `.classpath` and `.project`) files. Once you have copied everything to your course directory, import the project into Eclipse.

6.1.3 Logging into PostgreSQL

To access the PostgreSQL database, type the following in a console on a department computer:

```
psql -h csci1270 -U cs127student sudoku
```

The password is 'cs127student'. Once you authenticate, you have a command line which allows you to explore the database. Type `\h` for help on the various commands.

When you are trying to access the database through JDBC, the URL String you should use is:

```
"jdbc:postgresql://csci1270/sudoku"
```

With the same username/password pair.

¹Traditional 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.postgresql.org>

³<http://java.sun.com/javase/technologies/database/index.jsp>

6.2 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 solution⁴. 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_login`⁵. 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). When you create this table, be sure to use the standard INT4 datatype used for the `allsolutions` table and to add a uniqueness constraint to the location attribute.

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.

7 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;`
Called when the user constrains a previously unconstrained box. This function should execute a SQL query that adds a row to `constraints_login` 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_login`.
- `public void updateConstraint(int loc, int newNum) throws SQLException;`

⁴Unfortunately, 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.

⁵Replace *login* with your login.

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 table⁶. This function returns a 4 x 4 array of CellStatistics (the first dimension is the box column, and the second dimension is box row⁷. For example, the statistics for the box in column 1, row 3 is stored in the result indexed at `[1][3]`. 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. The CellStatistic Objects should be null if there are no possible solutions.

In addition to writing your Sudoku code, you should write a `main` method which initializes your code and constructs a `SudokuGUI` object with your `SudokuSQLClient`.

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.

8 Getting checked off

Ideally, you want to get checked off during your recitation section. If you can't make it, you have until the due date to get checked off at any TA hours.

⁶Watch out for the case where the constraint table is empty! In this case, either all solutions should be returned or none. Document your choice in your README

⁷See `CellStatistic.LocToCol` and `CellStatistic.LocToRow`