HOMEWORK #1 - SQL

OVERVIEW

The first homework is to construct a set of SQL queries for analysing a dataset that will be provided to you. The dataset contains information about every court case on the Maryland Judiciary Court Search website from the past decade. We focus on Maryland because Andy is a Maryland native and the members of the TA team were curious about his criminal past.

This homework is an opportunity to: (1) learn basic and certain advanced SQL features, and (2) get familiar with using the SQLite DBMS. This is the same relational DBMS that you will be hacking on during the rest of the semester.

This is a single-person project that will be completed individually (i.e., no groups).

Release Date: Aug 28, 2017
Due Date: Sep 13, 2017 @ 11:59pm

PROJECT SPECIFICATION

The homework contains 12 questions in total, and will be graded out of 100 points. For each question, you will need to construct a SQL query that fetches the desired data from the SQLite DBMS. Here's a rough time estimate: 1-2 hours for setting up SQLite; approx. 4-6 hours for completing the questions.

We provide the database dump (`md_courts.dump.gz`) on which your queries will be executed. We will also provide a compressed folder (`sql.tar.gz`) containing empty placeholder files (`00.txt` , `01.txt` ,..., `12.txt`). You will need to fill in the **output** of the SQL queries in these placeholder files.

You can decompress this folder by running the following command on the terminal:

```
$ tar -zxvf sql.tar.gz
```

After filling in the queries, you can compress the folder by running the following command:

```
$ tar -czvf sql.tar.gz hw1
```

INSTRUCTIONS

SETTING UP SQLITE

You will first need to install SQLite on your development machine.

```
$ sudo apt-get install sqlite3 libsqlite3-dev
```

INSTALL SQLITE3 ON MAC OS X

On Mac OS Leopard or later, you don't have to! It comes pre-installed. You can upgrade it, if you absolutely need to, with Homebrew.

LOAD THE DATABASE DUMP

Check if `sqlite3` is properly working by following this tutorial.

Download the database dump file:

```
$ wget http://15445.courses.cs.cmu.edu/fall2017/files/md_courts.dump.gz
```

Reconstruct the database from the provided database dump by running the following command on your shell. This page contains more details.

```
$ zcat md_courts.dump.gz | sqlite3 md_courts
```

Check the contents of the database by running the `.tables` command on the `sqlite3` terminal. You should see 9 tables, and the output should look like this:

```
$ sqlite3 md_courts.db
SQLite version 3.11.0
Enter ".help" for usage hints.
sqlite> .tables
attorneys   charges     documents   judgements  rawcases
cases       complaints  events      parties
```

PLACEHOLDER FOLDER

Download the placeholder folder at here:

```
$ wget http://15445.courses.cs.cmu.edu/fall2017/files/sql.tar.gz
$ tar zxfv sql.tar.gz
```

This should contain empty placeholder files (`00.txt` , `01.txt` ,..., `12.txt`).

SANITY CHECKS

Get familiar with the schema (structure) of the tables (what attributes do they contain, what are the primary and foreign keys, etc.). Run the `.schema $TABLE_NAME` command on the `sqlite3` terminal for each table. The output should look like this:

```
sqlite> .schema cases
CREATE TABLE cases (
    case_id character varying NOT NULL PRIMARY KEY,
    "timestamp" timestamp without time zone DEFAULT CURRENT_TIMESTAMP,
    title character varying,
    court_system character varying,
```

```
    status character varying,
    disposition character varying,
    disposition_date date,
    violation_county character varying,
    violation_date date
);
sqlite> .schema attorneys
CREATE TABLE attorneys (
    case_id character varying NOT NULL,
    name character varying,
    type character varying,
    appearance_date date,
    removal_date date,
    practice_name character varying,
    address character varying,
    city character varying,
    state character varying,
    zip character varying,
    FOREIGN KEY (case_id) REFERENCES cases(case_id)
);
CREATE INDEX attorneys_case_id_idx ON attorneys (case_id);
```

Perform a sanity check by running the following query:

Q0 [0 points]: Count the number of cases in the Maryland Judiciary Court System. The output should look like this (only a single number):

```
sqlite> select ...;
12345
```

Details: Make use of the `count` function.

Answer: Here's the correct SQL query and expected output:

```
sqlite> select count(case_id) from cases;
2000728
```

To redirect the output of this SQL query to the appropriate file (`00.txt`) in the submission directory (`sql`), enter the above query string (including ';') into a sql file (`00.sql`) and then run the following command:

```
$ sqlite3 md_courts.db < 00.sql > sql/00.txt
```

Now, you can verify the contents of the output text file (`00.txt`):

```
cat solutions/00.txt
2000728
```

CONSTRUCT THE SQL QUERIES

Now, its time to start constructing the SQL queries and fill in their output into the placeholder files.

```
sqlite> select ...;
12345
```

Details: Ensure that there are no duplicates.

Q2 [5 points]: Repeated phone calls can sometimes land you in jail. Count the number of charges related to phone calls by examining their description. The output should look like this:

```
12345
```

Details: The search string for the appropriate column is `%PHONE%`.

Q3 [5 points]: Reckless endangerment is another reason why one can end up in a courthouse. Count the number of cases related to reckless endangerment in each county. The output should look like this:

```
County A|500
County B|400
County C|300
```

Details: Print the county name and number of cases in that particular county. Sort the counties by the number of cases in descending order, and break ties by ordering them in ascending order with respect to the county name. Report only the top `3` counties with the maximum number of cases. The search string for the appropriate column is `%RECKLESS%`. Ensure that you only fetch the cases whose county name is not empty.

Q4 [5 points]: Let's now go back in time and look at the cases filed in the 1950s. The output should look like this:

```
CASE1|1950-03-12
CASE2|1951-01-01
CASE3|1952-01-01
```

Details: Print the case id and filing date for the cases filed in the 1950s. List the oldest cases first, and report only the earliest 3 cases.

Q5 [10 points]: It looks like a lot of cases got filed in the 1950s. In which decades did the most number of cases get filed? The output should look like this:

```
10000|1970s
5000|2000s
1000|1980s
```

Details: Print the number of cases and the relevant decade. We will print the relevant decade in a fancier format by constructing a string that looks like this `1970s`. Sort the decades in decreasing order with respect to the number of cases. Report only the top `3` decades wherein the most number of cases got filed. Ensure that you only fetch the cases whose filing date is not empty.

```
5.123456789
```

Details: Print the percentage of cases. To compute the percentage, you will need to multiply the numerator of the fraction by `100.0`. While searching the case's status, use the following string: `Case Closed Statistically`. To keep things simple, there is no need to truncate or round numbers. To compute the percentage, simply multiply the numerator by `100.0` and divide by the appropriate denominator.

Q7 [5 points]: Let's look at some prolific defendants. List the top 3 parties who have been charged in the most number of distinct counties. The output should look like this:

```
A|100
B|50
C|10
```

Details: Print the name of the party along with the number of distinct counties. Sort the parties by the number of distinct counties in descending order, and report only the top `3` parties. Ensure that you only fetch the parties who are defendants (`Defendant`) and whose name is not empty.

Q8 [20 points]: How does the average age of guilty criminals vary over time? The output should look like this:

```
2017|50.123456789
2016|55.123456789
2015|60.123456789
2014|55.123456789
2013|50.123456789
```

Details: Print the filing year and the average age of the criminals that were found guilty in cases filed in that particular year. To compute the average age, first determine the age of the criminal by using the case's filing date and the party's date of birth. Use the `strftime('%Y.%m%d',...)` function for this purpose. To determine the filing year from the filing date, again make use of the `strftime('%Y',...)` function. Look at the disposition to pick only `Guilty` parties (`charges.disposition = 'Guilty'`). Ensure that you only fetch the cases whose filing date is not empty. Also, ensure that you only fetch the parties who are defendants (`parties.type = Defendant`), whose name is not empty, whose date of birth is not empty, and whose computed age is greater than `0` and less than `100` years. List the tuples in descending order with respect to the filing year and only display `5` tuples. You might want to leverage common table expressions (CTEs) in this query. Here's more information on using CTEs in SQLite.

Q9 [15 points]: Let's next look at case disposition by race to see if there is any inherent bias in the system. The output should look like this:

```
African American|Guilty|60.000
African American|Not Guilty|40.000
Caucasian|Guilty|50.000
Caucasian|Not Guilty|50.000
```

`Not Guilty`). To compute the percentage, you will need to multiply the numerator of the fraction by `100.0`. Ensure that the race of the party is not empty. You might want to leverage common table expressions (CTEs) in this query. Here's more information on using CTEs in SQLite.

Q10 [5 points]: Certain zip codes might have -- ahem -- more interesting citizens than Squirrel Hill. Retrieve the top 3 zip codes in Maryland where the most number of cases were filed. The output should look like this:

```
21000|500
21001|400
21002|300
```

Details: Print the zip code along with the number of cases filed in that particular zip code. List them in decreasing order with respect to the number of cases. Display only the top `3` zip codes. Ensure that the zip code is not empty.

Q11 [15 points]: Some attorneys are awesome at their job. List the top 5 attorneys in Maryland by examining the number of cases that an attorney handles and the percentage of cases wherein they were successful. The output should look like this:

```
A|100|60.123
B|200|45.123
C|100|30.123
D|500|20.123
E|100|10.123
```

Details: Print the attorney's name, number of cases handled, and the percentage of cases won (i.e. the disposition was `Not Guilty`). Examine only attorneys who have handled more than `100` cases. List the attorneys in decreasing order with respect to their success percentage and number of cases handled, respectively. Display only the top `5` attorneys. Ensure that the attorney's name is not empty. You might want to leverage common table expressions (CTEs) in this query. Here's more information on using CTEs in SQLite.

Q12 [5 points]: Find the attorney with the seventh highest success percentage (by extending the previous query). The output should look like this:

```
G|50|5.123
```

Details: Print the attorney's name, number of cases handled, and the percentage of cases won (i.e. the disposition was `Not Guilty`). Examine only attorneys who have handled more than `100` cases. Ensure that the attorney's name is not empty.

GRADING RUBRIC

Each submission will be graded based on whether the SQL queries fetch the expected sets of tuples from the database. Note that your SQL queries will be auto-graded by comparing their outputs (i.e. tuple sets) to the correct outputs. For your queries, the **order** of the output columns is important; their names are not.

LATE POLICY

SUBMISSION

We use Autolab for grading in order to provide you with immediate feedback. After completing the homework, you can submit your compressed folder `sql.tar.gz` (only one file) to Autolab:

https://autolab.andrew.cmu.edu/courses/15445-f17

We will be comparing the output files using a function similar to `diff`. Ensure that your output matches the expected output in `00.txt`. You can submit your answers as many times as you like. Your score will be sent via email to your andrew account within a few minutes after your submission.

Additional Rule: In addition to placing the text files containing the query output in `sql.tar.gz`, you will also need to submit all the SQL queries in files named `00.sql`, `01.sql`, ..., `13.sql` in the same folder. If your handin folder `sql.tar.gz` does not contain these files with SQL queries, your final homework score will unfortunately be reduced.

We have downloaded the submission order and scoreboard before this rule was added. We will align your rankings with your prior submission order. Adding the SQL queries will not affect you earlier rankings.

COLLABORATION POLICY

Every student has to work individually on this assignment.
Students are allowed to discuss high-level details about the project with others.
Students are **not** allowed to copy the contents of a white-board after a group meeting with other students.
Students are **not** allowed to copy the solutions from another colleague.

⚠ **WARNING:** All of the code for this project must be your own. You may not copy source code from other students or other sources that you find on the web. Plagiarism **will not** be tolerated. See CMU's Policy on Academic Integrity for additional information.

Last Updated: Sep 02, 2017