Lab 13 Solutions **lab13.zip (lab13.zip)**

Solution Files

Usage

First, check that a file named `sqlite_shell.py` exists alongside the assignment files. If you don't see it, or if you encounter problems with it, scroll down to the Troubleshooting section to see how to download an official precompiled SQLite binary before proceeding.

You can start an interactive SQLite session in your Terminal or Git Bash with the following command:

```
python3 sqlite_shell.py
```

While the interpreter is running, you can type `.help` to see some of the commands you can run.

To exit out of the SQLite interpreter, type `.exit` or `.quit` or press `Ctrl-C`. Remember that if you see `...>` after pressing enter, you probably forgot a `;`.

You can also run all the statements in a `.sql` file by doing the following: (Here we're using the `lab13.sql` file as an example.)

1. Runs your code and then exits SQLite immediately afterwards.
   ```
   python3 sqlite_shell.py < lab13.sql
   ```
2. Runs your code and then opens an interactive SQLite session, which is similar to running Python code with the interactive `-i` flag.
   ```
   python3 sqlite_shell.py --init lab13.sql
   ```

Survey Data

Survey Data

Last week, we asked you and your fellow students to complete a brief online survey through Google Forms, which involved relatively random but fun questions. In this lab, we will interact with the results of the survey by using SQL queries to see if we can find interesting things in the data.

First, take a look at `data.sql` and examine the table defined in it. Note its structure. You will be working with the two tables in this file.

The first is the table `students`, which is the main results of the survey. Each column represents a different question from the survey, except for the first column, which is the time of when the result was submitted. This time is a unique identifier for each of the rows in the table. The last several columns all correspond to the last question on the survey (more details below.)

Column Name	**Question**
`time`	The unique timestamp that identifies the submission
`number`	What's your favorite number between 1 and 100?
`color`	What is your favorite color?
`seven`	Choose the number 7 below. Options: • 7 • Choose this option instead • seven • the number 7 below. • I find this question condescending

`song`	If you could listen to only one of these songs for the rest of your life, which would it be? Options: • "Smells Like Teen Spirit" by Nirvana • "Shelter" by Porter Robinson • "Clair de Lune" by Claude Debussy • "Dancing Queen" by ABBA • "Down With The Sickness" by Disturbed • "Everytime We Touch" by Cascada • "All I want for Christmas is you" by Mariah Carey • "STAY" by The Kid LAROI, Justin Bieber • "Old Town Road" by Lil Nas X • "Turntables" by Janelle Monáe • "Shake It Off" by Taylor Swift
`date`	Pick a day of the year!
`pet`	If you could have any animal in the world as a pet, what would it be?
`instructor`	Choose your favorite photo of John DeNero
`smallest`	Try to guess the smallest unique positive INTEGER that anyone will put!

The second table is `numbers`, which is the results from the survey in which students could select more than one option from the numbers listed, which ranged from 0 to 10 and included 2021, 2022, 9000, and 9001. Each row has a time (which is again a unique identifier) and has the value `'True'` if the student selected the column or `'False'` if the student did not. The column names in this table are the following strings, referring to each possible number: `'0'`, `'1'`, `'2'`, `'4'`, `'5'`, `'6'`, `'7'`, `'8'`, `'9'`, `'10'`, `'2021'`, `'2022'`, `'9000'`, `'9001'`

Since the survey was anonymous, we used the timestamp that a survey was submitted as a unique identifier. A time in `students` matches up with a time in `numbers`. For example, a row in `students` whose `time` value is `"11/17/2021 10:52:40"` matches up with the row in `numbers` whose `time` value is `"11/17/2021 10:52:40"`. These entries come from the same Google form submission and thus belong to the same student.

> *Note*: If you are looking for your personal response within the data, you may have noticed that some of your answers are slightly different from what you had inputted. In order to make SQLite accept our data, and to optimize for as many matches as possible during our joins, we did the following things to clean up the data:
>
> - `color` and `pet`: We converted all the strings to be completely lowercase.
> - For some of the more "free-spirited" responses, we escaped the special characters so that they could be properly parsed.

You will write all of your solutions in the starter file `lab13.sql` provided. As with other labs, you can test your solutions with OK. In addition, you can use either of the following commands:

```
python3 sqlite_shell.py < lab13.sql
python3 sqlite_shell.py --init lab13.sql
```

Questions

Q1: What Would SQL print?

> Note: there is no submission for this question

First, load the tables into sqlite3.

```
$ python3 sqlite_shell.py --init lab13.sql
```

Before we start, inspect the schema of the tables that we've created for you:

```
sqlite> .schema
```

This tells you the name of each of our tables and their attributes.

Let's also take a look at some of the entries in our table. There are a lot of entries though, so let's just output the first 20:

```
sqlite> SELECT * FROM students LIMIT 20;
```

If you're curious about some of the answers students put into the Google form, open up `data.sql` in your favorite text editor and take a look!

For each of the SQL queries below, think about what the query is looking for, then try running the query yourself and see!

```
sqlite> SELECT * FROM students LIMIT 30; -- This is a comment. * is shorthand for all colu
------

sqlite> SELECT color FROM students WHERE number = 7;
------

sqlite> SELECT song, pet FROM students WHERE color = "blue" AND date = "12/25";
------
```

Remember to end each statement with a `;` ! To exit out of SQLite, type `.exit` or `.quit` or hit `Ctrl-C`.

Q2: Go Bears! (And Dogs?)

Now that we have learned how to select columns from a SQL table, let's filter the results to see some more interesting results!

It turns out that 61A students have a lot of school spirit: the most popular favorite color was `'blue'`. You would think that this school spirit would carry over to the pet answer, and everyone would want a pet bear! Unfortunately, this was not the case, and the majority of students opted to have a pet `'dog'` instead. That is the more sensible choice, I suppose...

Write a SQL query to create a table that contains both the column `color` and the column `pet`, using the keyword `WHERE` to restrict the answers to the most popular results of color being `'blue'` and pet being `'dog'`.

You should get the following output:

```
sqlite> SELECT * FROM bluedog;
blue|dog
blue|dog
blue|dog
blue|dog
blue|dog
blue|dog
blue|dog
blue|dog
blue|dog
blue|dog
blue|dog
blue|dog
blue|dog
blue|dog
blue|dog
blue|dog
```

```
CREATE TABLE bluedog AS
  SELECT color, pet FROM students WHERE color = 'blue' AND pet = 'dog';
```

This isn't a very exciting table, though. Each of these rows represents a different student, but all this table can really tell us is how many students both like the color blue and want a dog as a pet, because we didn't select for any other identifying characteristics. Let's create another table, `bluedog_songs`, that looks just like `bluedog` but also tells us how each student answered the `song` question.

You should get the following output:

```
sqlite> SELECT * FROM bluedog_songs;
blue|dog|Clair De Lune
blue|dog|Shake It Off
blue|dog|Old Town Road
blue|dog|Dancing Queen
blue|dog|Old Town Road
blue|dog|Clair De Lune
blue|dog|Dancing Queen
blue|dog|Clair De Lune
blue|dog|STAY
blue|dog|Old Town Road
blue|dog|Shake It Off
blue|dog|STAY
blue|dog|Clair De Lune
blue|dog|Clair De Lune
blue|dog|STAY
blue|dog|Clair De Lune
```

```
CREATE TABLE bluedog_songs AS
  SELECT color, pet, song FROM students WHERE color = 'blue' AND pet = 'dog';
```

This distribution of songs actually largely represents the distribution of song choices that the total group of students made, so perhaps all we've learned here is that there isn't a correlation between a student's favorite color and desired pet, and what song they could spend the rest of their life listening to. Even demonstrating that there is no correlation still reveals facts about our data though!

Use Ok to test your code:

```
python3 ok -q bluedog
```

Q3: The Smallest Unique Positive Integer

Who successfully managed to guess the smallest unique positive integer value? Let's find out!

Write an SQL query to create a table with the columns `time` and `smallest` which contains the timestamp for each submission that made a unique guess for the smallest unique positive integer - that is, only one person put that number for their guess of the smallest unique integer. Also include their guess in the output.

> *Hint:* Think about what attribute you need to `GROUP BY`. Which groups do we want to keep after this? We can filter this out using a `HAVING` clause. If you need a refresher on aggregation, see the topics section.

The submission with the timestamp corresponding to the minimum value of this table is the timestamp of the submission with the smallest unique positive integer!

```
CREATE TABLE smallest_int_having AS
  SELECT time, smallest FROM students GROUP BY smallest HAVING COUNT(*) = 1;
```

Use Ok to test your code:

```
python3 ok -q smallest-int-having
```

Q4: Matchmaker, Matchmaker

Did you take 61A with the hope of finding a new group of friends? Well you're in luck! With all this data in hand, it's easy for us to find your perfect match. If two students want the same pet and have the same taste in music, they are clearly meant to be friends! In order to provide some more information for the potential pair to converse about, let's include the favorite colors of the two individuals as well!

In order to match up students, you will have to do a join on the `students` table with itself. When you do a join, SQLite will match every single row with every single other row, so make sure you do not match anyone with themselves, or match any given pair twice!

> **Important Note:** When pairing the first and second person, make sure that the first person responded first (i.e. they have an earlier `time`). This is to ensure your output matches our tests.
>
> *Hint:* When joining table names where column names are the same, use dot notation to distinguish which columns are from which table: `[table_name].[column name]`. This sometimes may get verbose, so it's stylistically better to give tables an alias using the `AS` keyword. The syntax for this is as follows:
>
> ```
> SELECT <[alias1].[column name1], [alias2].[columnname2]...>
>     FROM <[table_name1] AS [alias1],[table_name2] AS [alias2]...> ...
> ```
>
> The query in the football example from earlier uses this syntax.

Write a SQL query to create a table that has 4 columns:

- The shared preferred `pet` of the pair

- The shared favorite `song` of the pair
- The favorite `color` of the first person
- The favorite `color` of the second person

```
CREATE TABLE matchmaker AS
  SELECT a.pet, a.song, a.color, b.color FROM students AS a, students AS b
    WHERE a.time < b.time AND a.pet = b.pet AND a.song = b.song;
```

Use Ok to test your code:

```
python3 ok -q matchmaker
```

Q5: Sevens

Let's take a look at data from both of our tables, `students` and `numbers`, to find out if students that really like the number 7 also chose `'7'` for the obedience question. Specifically, we want to look at the students that fulfill the below conditions:

Conditions:

- reported that their favorite number (column `number` in `students`) was 7
- have `'True'` in column `'7'` in `numbers`, meaning they checked the number `7` during the survey

In order to examine rows from both the `students` and the `numbers` table, we will need to perform a join.

How would you specify the `WHERE` clause to make the `SELECT` statement only consider rows in the joined table whose values all correspond to the same student? If you find that your output is massive and overwhelming, then you are probably missing the necessary condition in your `WHERE` clause to ensure this.

> *Note:* The columns in the `numbers` table are strings with the associated number, so you must put quotes around the column name to refer to it. For example if you alias the table as `a`, to get the column to see if a student checked 9001, you must write `a.'9001'`.

Write a SQL query to create a table with just the column `seven` from `students`, filtering first for students who said their favorite number (column `number`) was 7 in the `students` table and who checked the box for seven (column `'7'`) in the `numbers` table.

The first 10 lines of this table should look like this:

```
sqlite> SELECT * FROM sevens LIMIT 10;
seven
7
7
7
7
the number 7 below.
the number 7 below.
7
the number 7 below.
7
```

```
CREATE TABLE sevens AS
  SELECT s.seven FROM students AS s, numbers AS c WHERE s.number = 7 AND c.'7' = 'True' AN
```

Use Ok to test your code:

```
python3 ok -q sevens
```

Q6: Average Difference

Let's try to find the average absolute difference between every person's favorite number `number` and their guess at the smallest number anyone will put, `smallest`, rounded to the closest number.

For example, suppose two students put the following:

```
| number | smallest |
| ------ | -------- |
| 10     | 1        |
| 2      | 3        |
```

Then, average absolute difference is (abs(10-1) + abs(2-3)) / 2 = 5.0.

Create a table that contains one row with one value, the rounded value of the average absolute difference between every person's favorite number and their guess at the smallest value.

Hints:

- `abs` is a `sqlite3` function that returns the absolute value of the argument.
- `round` is a function that rounds the value of the argument.
- `avg` is a function that returns the average value in a set.

```
CREATE TABLE avg_difference AS
  SELECT round(avg(abs(number - smallest))) as avg_difference FROM students;
```

Use Ok to test your code:

```
python3 ok -q avg-difference
```