Lab 13: SQL **lab13.zip (lab13.zip)**

Due by 11:59pm on Tuesday, December 5.

Starter Files

Download lab13.zip (lab13.zip). Inside the archive, you will find starter files for the questions in this lab, along with a copy of the Ok (ok) autograder.

Required Questions

Getting Started Videos

SQL

Consult the drop-down if you need a refresher on SQL. It's okay to skip directly to the questions and refer back here should you get stuck.

SQL

Survey Data

Over Thanksgiving break, 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.

Column Name	**Question**
`time`	The unique timestamp that identifies the submission
`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: • "Clair de Lune" by Claude Debussy • "Dancing Queen" by ABBA • "Bohemian Rhapsody" by Queen • "Shake It Off" by Taylor Swift • "Thinking Out Loud" by Ed Sheeran • "good 4 u" by Olivia Rodrigo • "The Feels" by TWICE • "Hopes and Dreams" by Toby Fox • "Shelter" by Porter Robinson • "All I want for Christmas is you" by Mariah Carey
`date`	Pick a day of the year!
`pet`	If you could have any animal in the world as a pet, what would it be?
`smallest`	Try to guess the smallest unique positive INTEGER that anyone will put!

The second table is `numbers`, which contains the results from the question: "What's your favorite number between 1 and 100?" Each row has a time (which is again a unique identifier) and one column `'1'`, `'2'` ... `'99'`, `'100'` for every possible number between 1 and 100. The

column has the value `'True'` if the student selected that number or `'False'` if the student did not.

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 `"11/23/2023 14:48:14"` matches up with the row in `numbers` whose `time` value is `"11/23/2023 14:48:14"`. These entries come from the same Google form submission and thus belong to the same student.

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 date = "2/14";
------

sqlite> SELECT song, pet FROM students WHERE color = "powder 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
```

```
CREATE TABLE bluedog AS
  SELECT "REPLACE THIS LINE WITH YOUR SOLUTION";
```

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|Dancing Queen
blue|dog|Shake It Off
blue|dog|The Feels
blue|dog|Bohemian Rhapsody
blue|dog|Dancing Queen
blue|dog|Shake It Off
blue|dog|All I want for Christmas is you
```

```
CREATE TABLE bluedog_songs AS
  SELECT "REPLACE THIS LINE WITH YOUR SOLUTION";
```

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 "REPLACE THIS LINE WITH YOUR SOLUTION";
```

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 "REPLACE THIS LINE WITH YOUR SOLUTION";
```

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 (where students were asked "Choose 7 from the following options below"). Specifically, we want to look at the students that fulfill the below conditions:

Conditions:

- reported that their favorite number was 7 (`'True'` in column `'7'` from the `numbers` table)
- chose a number *less than or equal to* 7 when asked to predict the smallest unique positive integer (column `smallest` from the `students` table)

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's favorite number is 5, you must write `a.'5'`.

Write a SQL query to create a table with just the column `seven` from `students`, filtering first for students who said their favorite number was 7 (column `'7'`) in the `numbers` table and who predicted the smallest unique positive integer to be less than or equal to 7 (column `smallest`) in the `students` table.

The final output should look like this:

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

```
CREATE TABLE sevens AS
  SELECT "REPLACE THIS LINE WITH YOUR SOLUTION";
```

Use Ok to test your code:

```
python3 ok -q sevens
```

Check Your Score Locally

You can locally check your score on each question of this assignment by running

```
python3 ok --score
```

This does NOT submit the assignment! When you are satisfied with your score, submit the assignment to Gradescope to receive credit for it.

Submit

Make sure to submit this assignment by uploading any files you've edited **to the appropriate Gradescope assignment.** For a refresher on how to do this, refer to Lab 00 (https://cs61a.org/lab/lab00/#submit-with-gradescope).