Homework 11: **hw11.zip (hw11.zip)**

Due by 11:59pm on Thursday, 5/2

Instructions

Download hw11.zip (hw11.zip).

Submission: When you are done, submit with `python3 ok --submit`. You may submit more than once before the deadline; only the final submission will be scored. Check that you have successfully submitted your code on okpy.org (https://okpy.org/). See Lab 0 (/lab/lab00#submitting-the-assignment) for more instructions on submitting assignments.

Using Ok: If you have any questions about using Ok, please refer to this guide. (/articles/using-ok.html)

Readings: You might find the following references useful:

- Section 4.3 - Declarative Programming (http://composingprograms.com/pages/43-declarative-programming.html)

Grading: Homework is graded based on effort, not correctness. However, there is no partial credit; you must show substantial effort on every problem to receive any points.

To complete this homework assignment, you will need to use SQLite version 3.8.3 or greater. See Lab 12 (/lab/lab12) for setup and usage instructions.

To check your progress, you can run `sqlite3` directly by running:

```
./sqlite3 --init hw11.sql
```

You should also check your work using `ok`:

```
python3 ok
```

Dog Data

In each question below, you will define a new table based on the following tables.

```
CREATE TABLE parents AS
  SELECT "abraham" AS parent, "barack" AS child UNION
  SELECT "abraham"          , "clinton"         UNION
  SELECT "delano"           , "herbert"         UNION
  SELECT "fillmore"         , "abraham"         UNION
  SELECT "fillmore"         , "delano"          UNION
  SELECT "fillmore"         , "grover"          UNION
  SELECT "eisenhower"       , "fillmore";

CREATE TABLE dogs AS
  SELECT "abraham" AS name, "long" AS fur, 26 AS height UNION
  SELECT "barack"         , "short"      , 52           UNION
  SELECT "clinton"        , "long"       , 47           UNION
  SELECT "delano"         , "long"       , 46           UNION
  SELECT "eisenhower"     , "short"      , 35           UNION
  SELECT "fillmore"       , "curly"      , 32           UNION
  SELECT "grover"         , "short"      , 28           UNION
  SELECT "herbert"        , "curly"      , 31;

CREATE TABLE sizes AS
  SELECT "toy" AS size, 24 AS min, 28 AS max UNION
  SELECT "mini"       , 28       , 35        UNION
  SELECT "medium"     , 35       , 45        UNION
  SELECT "standard"   , 45       , 60;
```

Your tables should still perform correctly even if the values in these tables change. For example, if you are asked to list all dogs with a name that starts with h, you should write:

```
SELECT name FROM dogs WHERE "h" <= name AND name < "i";
```

Instead of assuming that the `dogs` table has only the data above and writing

```
SELECT "herbert";
```

The former query would still be correct if the name `grover` were changed to `hoover` or a row was added with the name `harry`.

Q1: Size of Dogs

The Fédération Cynologique Internationale classifies a standard poodle as over 45 cm and up to 60 cm. The `sizes` table describes this and other such classifications, where a dog must be over the `min` and less than or equal to the `max` in height to qualify as a `size`. Create a `size_of_dogs` table with two columns, one for each dog's `name` and another for its `size`.

```
-- The size of each dog
CREATE TABLE size_of_dogs AS
  SELECT "REPLACE THIS LINE WITH YOUR SOLUTION";
```

The output should look like the following:

```
sqlite> select * from size_of_dogs;
abraham|toy
barack|standard
clinton|standard
delano|standard
eisenhower|mini
fillmore|mini
grover|toy
herbert|mini
```

Use Ok to test your code:

```
python3 ok -q size_of_dogs
```

Q2: By Parent Height

Create a table `by_parent_height` that has a column of the names of all dogs that have a `parent`, ordered by the height of the parent from tallest parent to shortest parent.

```
-- All dogs with parents ordered by decreasing height of their parent
CREATE TABLE by_parent_height AS
  SELECT "REPLACE THIS LINE WITH YOUR SOLUTION";
```

For example, `fillmore` has a parent (`eisenhower`) with height 35, and so should appear before `grover` who has a parent (`fillmore`) with height 32. The names of dogs with parents of the same height should appear together in any order. For example, `barack` and `clinton` should both appear at the end, but either one can come before the other.

```
sqlite> select * from by_parent_height;
herbert
fillmore
abraham
delano
grover
barack
clinton
```

Use Ok to test your code:

```
python3 ok -q by_parent_height
```

Q3: Sentences

There are two pairs of siblings that have the same size. Create a table that contains a row with a string for each of these pairs. Each string should be a sentence describing the siblings by their size.

```
-- Filling out this helper table is optional
CREATE TABLE siblings AS
  SELECT "REPLACE THIS LINE WITH YOUR SOLUTION";

-- Sentences about siblings that are the same size
CREATE TABLE sentences AS
  SELECT "REPLACE THIS LINE WITH YOUR SOLUTION";
```

Each sibling pair should appear only once in the output, and siblings should be listed in alphabetical order (e.g. `"barack and clinton..."` instead of `"clinton and barack..."`), as follows:

```
sqlite> select * from sentences;
abraham and grover are toy siblings
barack and clinton are standard siblings
```

> *Hint*: First, create a helper table containing each pair of siblings. This will make comparing the sizes of siblings when constructing the main table easier.

Use Ok to test your code:

```
python3 ok -q sentences
```

Q4: Stacks

Sufficiently sure-footed dogs can stand on either other's backs to form a stack (up to a point). We'll say that the total height of such a stack is the sum of the heights of the dogs. Create a two-column table describing all stacks of up to four dogs at least 170 cm high. The first column should contain a comma-separated list of dogs in the stack, and the second column should contain the total height of the stack. Order the stacks in increasing order of total height.

```
-- Ways to stack 4 dogs to a height of at least 170, ordered by total height
CREATE TABLE stacks_helper(dogs, stack_height, last_height);

-- Add your INSERT INTOs here

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

A valid stack of dogs includes each dog only once, and the dogs should be listed in increasing order of height within the stack. You may assume that no two dogs have the same height.

```
sqlite> select * from stacks;
abraham, delano, clinton, barack|171
grover, delano, clinton, barack|173
herbert, delano, clinton, barack|176
fillmore, delano, clinton, barack|177
eisenhower, delano, clinton, barack|180
```

You should use the provided helper table `stacks_helper`. It has 3 columns: (1) `dogs` - a stack of dogs as a comma separated list of dog names, (2) `stack_height` - the height of the stack, and (3) `last_height` - the height of the last dog added to the stack (in order to ensure we have the right order in the stack).

First, fill this table up by doing the following:

1. Use an `INSERT INTO` to add stacks of just one dog into `stacks_helper`. You can use this syntax to insert rows from a table called `t1` into a table called `t2`:

   ```
   INSERT INTO t2 SELECT [expression] FROM t1 ...;
   ```

   For example:

   ```
   sqlite> CREATE TABLE t1 AS
   ...>        SELECT 1 as a, 2 as b;
   sqlite> CREATE TABLE t2(c, d);
   sqlite> INSERT INTO t2 SELECT a, b FROM t1;
   sqlite> SELECT * FROM t2;
   1|2
   ```

2. Now, use the stacks of one dog to insert stacks of two dogs. It's possible to `INSERT INTO` a table rows selected from that same table. For example,

   ```
   sqlite> CREATE TABLE ints AS
      ...>   SELECT 1 AS n UNION
      ...>   SELECT 2      UNION
      ...>   SELECT 3;
   sqlite> INSERT INTO ints(n) SELECT n+3 FROM ints;
   sqlite> SELECT * FROM ints;
   1
   2
   3
   4
   5
   6
   ```

3. Repeat step 3 to create stacks of three dogs, then of four dogs.

Once you've built up to stacks of four dogs in your `stacks_helper` table, use it to fill in the `stacks` table!

Use Ok to test your code:

```
python3 ok -q stacks
```

CS 61A (/)

Weekly Schedule (/weekly.html)

Office Hours (/office-hours.html)

Staff (/staff.html)

Resources (/resources.html)

Studying Guide (/articles/studying.html)

Debugging Guide (/articles/debugging.html)

Composition Guide (/articles/composition.html)

Policies (/articles/about.html)

Assignments (/articles/about.html#assignments)

Exams (/articles/about.html#exams)

Grading (/articles/about.html#grading)