Homework 2 | Basic SQL Queries

If any updates are made to the assignment spec after release they are highlighted in red.

Objectives: To create / import databases and to practice simple SQL queries, both using SQLite.

Due date: Thursday, April 17 @ 9:00pm

Median completion time (23sp): **4 hours**

Resources

- Flights dataset (zipped CSV files)
- Full SQLite documentation
- A SQL style guide in case you are interested (FYI only)

Before You Begin

Dataset Details

The data in the provided dataset (zipped CSV files) is abridged from the Bureau of Transportation Statistics, and describes a subset of flights that took place in July 2015. The dataset consists of a single .zip file which unzips to contain several .csv files.

The first step will be to create a database of four tables. The schema you should use is as follows:

```
FLIGHTS (fid int,
         month_id int,         -- 1-indexed (ie, 1-12)
         day_of_month int,     -- 1-indexed (ie, 1-31)
         day_of_week_id int,   -- 1-indexed (ie, 1-7). 1 = Monday,
                               -- 2 = Tuesday, etc
         carrier_id varchar(7),
         flight_num int,
         origin_city varchar(34),
         origin_state varchar(47),
         dest_city varchar(34),
         dest_state varchar(46),
         departure_delay int, -- in mins
```

```
        taxi_out int,          -- in mins
        arrival_delay int,     -- in mins
        canceled int,          -- boolean; 1 means canceled
        actual_time int,       -- flight duration, in mins
        distance int,          -- in miles
        capacity int,
        price int              -- in USD (ie, dollars)
        )
```

```
CARRIERS (cid varchar(7), name varchar(83))
```

```
MONTHS (mid int, month varchar(9))
```

```
WEEKDAYS (did int, day_of_week varchar(9))
```

In addition, you must impose the following constraints to the tables above:

- The primary key for the `FLIGHTS` table is `fid`.
- The primary keys for the other tables are `cid`, `mid`, and `did` respectively.
    - Other than these four primary keys, you MAY NOT ASSUME any other attribute is a key and/or unique across tuples.
- `FLIGHTS.carrier_id` references `CARRIERS.cid`
- `FLIGHTS.month_id` references `MONTHS.mid`
- `FLIGHTS.day_of_week_id` references `WEEKDAYS.did`

Prework

To import the flights dataset into a SQLite database, you will first need to run sqlite3 with a new database file (eg, `sqlite3 hw2.db`). This will create an empty, file-backed database that you can load in future SQLite sessions.

Once you have an empty database, you can run `CREATE TABLE` statements to create the tables while specifying all key constraints as described above.

```
CREATE TABLE table_name ( ... );
```

Currently, SQLite does not enforce foreign keys by default. To enable foreign keys, use the following command.

```
PRAGMA foreign_keys=ON;
```

Next, use SQLite's `.import` command to read data from each downloaded text file. You will use one file per table, and you will need to set the input format to CSV (comma-separated value). If you need additional help, check SQLite's built-in documentation or sqlite3's website.

```
.mode csv
.import filename tablename
```

Lastly, put all the code for *creating* your tables and enforcing foreign keys into a file called `create-tables.sql`, and all the code for *importing* the data into these tables into a separate file called `import-tables.sql` . If done correctly, you should be able to open up a new, empty `.db` file in sqlite and setup the database using these two commands:

```
.read create-tables.sql
.read import-tables.sql
```

Problem Set

Instructions:

- You should answer all the questions below using SQL queries which do NOT contain any subqueries!
- The predicates in your queries should correspond to the exact English descriptions in the question.
    - For example, if a question asks you to find flights by "Alaska Airlines Inc.", your query should check for that specific string instead of its matching `cid` (eg, do not use **`cid='AS'`**).
    - For example, if a question asks you to find flights on "Tuesday", your query should check for that specific string instead of its corresponding `did` (eg, do not use **`did=2`**).
- Return the output columns exactly as indicated. Do not change the output column names, the output order, or return more/fewer columns.
- Do not assume that carrier names will be unique. Your queries should account for the case where two different carriers have the same name. (i.e. for GROUP BY, make 2 separate groups for them)
- Unless otherwise noted, include canceled flights in your output.

- When asked to output specific durations, report them in minutes instead of doing the minute-to-hour conversion.
- If a query uses a GROUP BY clause, ensure that all attributes in your SELECT clause are either grouping keys or aggregate values.
    - Recall that SQLite will let you select non-grouping-and-non-aggregate attributes. However, this is not correct SQL and other DBMSs would reject that query
- Although the provided dataset consists entirely of flights made in July, ***your queries should NOT assume this***. We may test your queries on other datasets when grading.

Please review the above instructions before starting the problem and also before submitting your responses.

0. (20 points)
   Submit your pre-work as `create-tables.sql` and `import-tables.sql`.

1. (10 points) (Output relation cardinality: **3 rows**)
   List the distinct flight numbers for all flights from Seattle to Boston by Alaska Airlines Inc. on Mondays. Notice that, in the dataset, city names include the state (eg, Seattle appears as "`Seattle WA`"); you should hardcode both "`Seattle WA`" and "`Boston MA`" in your query.

   Your output should return the value of the `flight_num` column (ie, not the `fid`), and the output column should be named `flight_num`.

2. (10 points) (Output relation cardinality: **1472 rows**)
   Find all itineraries from Seattle to Boston on July 15th that have exactly one stop (i.e., flight 1: Seattle -> [somewhere], flight2: [somewhere] -> Boston).

   Both flights must *depart* on the same day and must be with the same carrier; it's fine if the landing date is different from the departing date (i.e., an overnight flight). You don't need to verify whether the first flight overlaps with the second, since the departing and arriving time of the flights are not provided in the dataset. The total duration of the entire itinerary should be fewer than 420 minutes (ie, 7 hours)

   For each itinerary, your query should return the name of the carrier, the first flight number, the origin and destination of that first flight, the flight duration, the second flight number, the origin and destination of the second flight, the second flight duration, and finally the total flight duration. Only count flight durations here; do not include any layover time.

   Name the output columns as follows: `name` (ie, the name of the carrier), `f1_flight_num`, `f1_origin_city`, `f1_dest_city`, `f1_actual_time`, `f2_flight_num`, `f2_origin_city`, `f2_dest_city`, `f2_actual_time`, and

`actual_time` (ie, total flight duration). Your output columns must be in this order.

3. (10 points) (Output relation cardinality: **1 row**)
   Calculate the day of the week with the longest average arrival delay. Hint: consider using `LIMIT` (look up what it does!).

   Name the output columns `day_of_week` and `delay`, in that order.

4. (10 points) (Output relation cardinality: **12 rows**)
   Find the names of all airlines that ever flew more than 1000 flights in one day; we define "day" as "a specific day/month", not as an arbitrary 24-hour period. Return only the names of the airlines, and do not return any duplicate names.

   Name the output column `name`.

5. (10 points) (Output relation cardinality: **3 rows**)
   Find the names of all airlines that had more than 1% (= 0.01) of their flights out of Seattle canceled. Percentages should be outputted in percent format (ie, represent 3.5% as 3.5, not 0.035). Order the results by the percentage of canceled flights, in ascending order.

   Name the output columns `name` and `percentage`, in that order.

6. (10 points) (Output relation cardinality: **3 rows**)
   For each carrier, output their name and the maximum ticket price for a direct flight between Seattle and New York, NY. This includes flights from Seattle to New York as well as those from New York to Seattle.

   Name the output columns `carrier` and `max_price`, in that order.

7. (10 points) (Output relation cardinality: **31 rows**)
   Calculate the total capacity of all direct flights between Seattle and San Francisco, CA (ie, flights from Seattle to SF and also flights from SF to Seattle) for each day in the month of July.

   Name the output columns `day_of_month` and `capacity`, in that order.

8. (10 points) (Output relation cardinality: **22 rows**)
   Calculate the total departure delay for each airline, summed across all its flights. Some departure delays may be negative (indicating an early departure); these delays should reduce the total.

   Name the output columns `name` and `delay`, in that order.

9. (3 points)
   Please reflect on your own personal experience completing this homework. Specifically:
   a. What is one thing that you learned while doing this assignment?
   b. What is one thing that surprised you while doing this assignment?
   c. What is one question that you still have after doing this assignment?

   The following questions are completely optional, but help us understand if we are creating appropriately-challenging questions:

   - How many hours did it take you to finish this assignment, including time to set up your computer (if necessary)?
   - How many of those hours did you feel were valuable and/or contributed to your learning?
   - Did you collaborate with other students on this assignment? If so, approximately how many people did you collaborate with? Do not include yourself in the count.

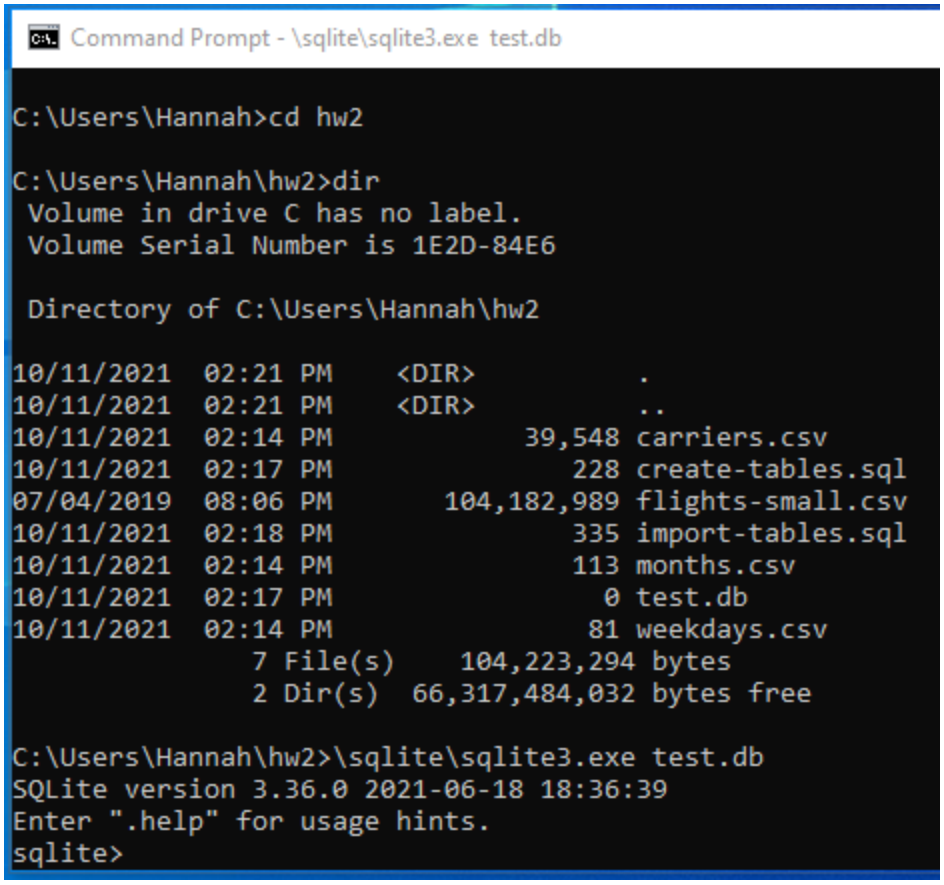
Save your answers to the reflection and feedback questions in a file called hw2-writeup.txt in the submission directory.

Submission Instructions

You should submit your files on Gradescope under HW2. You can resubmit however many times until the due date, which will save your progress without submitting all answers. If you don't have access to Gradescope or are having issues with submitting, contact course staff as soon as possible.

For each question in the problem set, write a **single SQL query**. Each query should be in a separate file, named `hw2-q#.sql` (eg, `hw2-q1.sql`, `hw2-q2.sql`, etc). Your feedback file should be named `hw2-writeup.txt`. Do not submit your actual database (your .db file) nor the .csv's you used to build the database.

Our autograders assume that your `.sql` files all reside in the same directory. To avoid hardcoding absolute paths into your Gradescope submission, all your files must be in the same directory in which you invoke sqlite. As an example, our directory looks like this:



This is so important we're going to say again: your **filenames must match** the expected names (eg, `create-tables.sql`, `hw2-q1.sql`, `hw2-writeup.txt`, etc) and your **output must match** the expected format (eg, `f1_flight_num`,`f1_origin_city`,`f1_dest_city` in that order). Our autograders hardcode your filenames, your output column names, and the number of output columns; if you use a different convention your submission will not be graded. You will be very sad if that happens, so please triple-check that this is all correct. Also, if you have any "dot commands" in your query files (q1-q8), such as `.headers on` or `.mode columns`, you **must** remove these prior to submission; again, you will be very very sad if you do not do this.

Please do not be sad.