Homework 2 | Basic SQL Queries

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

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

Due date: Monday, October 13th @ 11:59pm

Resources
Part 1: Create Database & Import Dataset
    Creating the Flights Database
    Importing the Flights Dataset
Part 2: SQL Queries
Submission Instructions

Resources

- Flights dataset (zipped CSV files)
- Full SQLite documentation
- A SQL style guide in case you are interested (FYI only, we don't grade on style or require any particular rules.)
- Information about the Flights Schema

Part 1: Create Database & Import Dataset

As you go through these steps, you will need to save your create and import statements into two separate `.sql` files that you will submit: one for creating tables, `create-tables.sql`, and one for importing data, `import-tables.sql`. Do not just type the commands into the SQLite shell without saving them!

Creating the Flights Database

The first step will be to create a database of five tables in SQLite. The schema you should use can be found in the Flights Schema document.

To create 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 in the Flights Schema document**.

```
CREATE TABLE Table_Name ( ... );
```

Currently, SQLite does not enforce foreign keys by default. To enable foreign key constraints, **use the following command at the end of your file**.

```
PRAGMA foreign_keys=ON;
```

▶️ ***Submission Item:*** Put all the SQLite commands/statements for creating your tables and enforcing foreign keys into a file called `create-tables.sql`

Importing the Flights Dataset

To import data into your tables, use SQLite's `.import` command to read data into each table from its corresponding .csv file. 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.

Since our .csv files have headers, you will need to add `--skip 1` to ignore them.

```
.mode csv
.import --skip 1 filename tablename
```

▶️ ***Submission Item:*** Put all the SQLite commands/statements for importing your tables into a file called `import-tables.sql`

If your database create and import statements are 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
```

Part 2: SQL Queries

Below are some notes to keep in mind for this part:

- For each question, **write a single SQL query to answer that question**.

- **You should only use the parts of SQL we have covered in class**
    - With the exception that you may use any operators/functions from the documentation of SQLite)
    - Do not use windowing, compound queries, subqueries, or WITH.
- 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 attributes that aren't primary keys will be unique.**
- Unless otherwise noted, include canceled flights in your output.
- 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 standard SQL and other DBMSs would reject that query
- Although the provided dataset consists entirely of flights made in September 2024, *your queries should NOT assume this*. **We WILL test your queries on other datasets.**
- Please write queries which can be understood by humans; we use a mix of autograders and manual review to determine your grade.

Please review the above instructions before starting the problem and also before submitting your responses. The below problems are NOT in order of difficulty.

▶️ ***Submission Items***: Put each query in a separate numbered .sql file, corresponding to the question number (`hw2-q1.sql`, `hw2-q2.sql`, etc).

1. *(Output relation cardinality: **5 rows**)*
   List the distinct carrier names of all carriers that offer a flight from the Seattle-area airport `SEA` to any airport in Chicago. (Note that the full string for Seattle is 'Seattle, WA', and Chicago is 'Chicago, IL')

   Name the output column `name`.

   (Hint: there is more than one airport in Chicago. Your query should work for all such airports, even if they build yet another airport in Chicago.)

2. *(Output relation cardinality: **113 rows**)*
   List all itineraries on September 7, 2024, from `SEA` to any airport in Chicago; these itineraries must have exactly one intermediate stop (i.e. `SEA` -> some airport -> some Chicago airport).

   Both flights must depart on the same day and must be with the same carrier, and the first flight must depart before the second flight departs; do not worry about whether the first flight arrives before the second flight departs, because that is a bit tricky with this dataset, due to overnight flights. The total flight duration of the entire itinerary should be fewer than 360 minutes (*i.e.*, 6 hours).

For each itinerary, your query should return the name of the carrier (not its code), the first flight number, the first flight duration in minutes, the name of the intermediate city (not the airport code), the second flight number, the second flight time, and finally the total flight time in minutes. Only count the time spent for the flights here; do not include any layover time.

Name the output columns as follows: `carrier_name`, `f1_flight_num`, `f1_duration_mins`, `intermediate_city`, `f2_flight_num`, `f2_duration_mins`, and `total_duration_mins` in that order.

3. *(Output relation cardinality: **25 rows**)*
   Write a query to count how many non-cancelled flights that each aircraft has made. Sort the output by this count from highest to lowest, and return only the top 25 rows; break ties using the aircraft's `tail_num`. Please return the manufacturer and model for each tail number, as well as the name of the aircraft's carrier.

   Name the output columns `tail_num`, `carrier`, `mfr`, `model`, and `cnt`, in that order.

4. *(Output relation cardinality: **42 rows**)*
   Find all aircraft that departed the same airport (identified by its code, not its departure city) twice on the same day to different destination airports, where the two flights totaled at least 5200 miles. **Do not include duplicate rows.**

   Output columns `tail_number` and `airport` (the origin airport), in that order.

5. *(Output relation cardinality: **5 rows**)*
   Find all registered aircraft (in the `N_Numbers` table) owned by `'UNIVERSITY OF WASHINGTON'` along with the number of seats and its type, as specified in the `Aircraft_Types` table.

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

6. *(Output relation cardinality: **88 rows**)*
   Find all aircraft that were manufactured in 2024 and took a flight from the location in which they are registered. Do not include duplicates.

   Name the output columns `n_number`, `name`, and `city`, in that order. All of these columns correspond directly to their analogs in the `N_Numbers` table.

   Hint 1: How many cities in the United States are named "Portland"? How can we distinguish between these locations?

Hint 2: You may find the SQL concatenation operator `||` useful; for example `'hello' || 'world'` evaluates to `'helloworld'`. You'll also potentially need to account for uppercase versus lowercase; check the SQLite documentation to find useful functions to help you.

7. *(Output relation cardinality: **29 rows**)*
   Find all flights from `'Atlanta, GA'` on a 737 that were canceled due to weather.

   For the purposes of this problem, consider "a 737" to be any aircraft that has `'737'` as a substring of its aircraft type's `model` column; you may want to look at the `LIKE` operator to find substrings. Ensure your query explicitly searches for weather-related cancellations; do not hardcode a specific set of cancellation code IDs.

   Name the output columns `year`, `month`, `day_of_month`, `cname` (the carrier name), and `flight_num` (the carrier's flight number, not the `fid`), in that order.

8. *(Output relation cardinality: **4 rows**)*
   For each possible cancellation reason (`description` in `Cancellation_Codes`), find the number of flights that were canceled for that reason. Include cancellation reasons that are never used, which you should output with 0 flights.

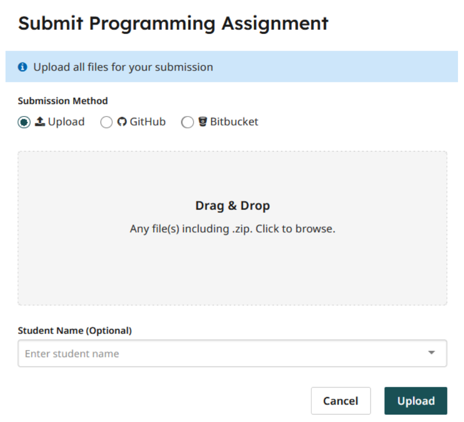
   Name the output columns `cancellation_reason` and `num_flights`, in that order.

Submission Instructions

What to turn in: `create-tables.sql`, `import-tables.sql`,, and `hw2-q1.sql`, `hw2-q2.sql`, …, `hw2-q8.sql`

Please ensure that

- If you have any “dot commands” in your **query files** (q1-q8), such as `.headers on` or `.mode columns`, please remove these prior to submission. You should keep your `PRAGMA` declarations, however.
- You are submitting the .sql files directly to Gradescope, no need to zip up the files. It's easiest to drag the files directly onto this screen:


Submit Programming Assignment

Upload all files for your submission

Submission Method

[✓ Upload] GitHub Bitbucket

Drag & Drop
Any file(s) including .zip. Click to browse.

Student Name (Optional)

Enter student name

- Your file names match the expected file names (see above). **Do not submit your actual database (your .db file) nor the .csv's you used to build the database**
    - Double check you have named all your files correctly!
- Our autograders assume that your files all reside in the same directory. **Do not hardcode absolute paths into your import statements**; they will fail to autograde. All your files must be in the same directory in which you invoke sqlite.