DATA 514 Homework 2: Basic SQL Queries

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

Assignment tools: SQLite 3, the flights dataset. (Reminder: To extract the content of a tar file, try the following command on a linux machine: `tar zxvf flights-small.tar.gz`)

Assigned date: Tuesday, Jan. 15, 2019.

Due date: Monday, Jan 21. You have 1 week for this assignment.

Questions: Make sure your post them on the discussion board.

What to turn in:

`create-tables.sql` and `hw2-q1.sql`, `hw2-q2.sql`, etc (see below).

Assignment Details

In this homework, you will write several SQL queries on a relational flights database. The data in this database is abridged from the Bureau of Transportation Statistics The database consists of four tables:

```
FLIGHTS (fid int, year int, month_id int, day_of_month int,
         day_of_week_id int, carrier_id varchar(3), flight_num int,
         origin_city varchar(34), origin_state varchar(47),
         dest_city varchar(34), dest_state varchar(46),
         departure_delay double, taxi_out double, arrival_delay double,
         canceled int, actual_time double, distance double, capacity int,
         price double)
CARRIERS (cid varchar(7), name varchar(83))
MONTHS (mid int, month varchar(9))
WEEKDAYS (did int, day_of_week varchar(9))
```

(FYI All data except for the capacity and price columns are real.) We leave it up to you to decide how to declare these tables and translate their types to sqlite. But make sure that your relations include all the attributes listed above.

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

```
  - The primary key of the `FLIGHTS` table is `fid`.
  - The primary keys for the other tables are `cid`, `mid`, and `did`
respectively.
  - `Flights.carrier_id` references `Carrier.cid`
  - `Flights.month_id` references `Months.mid`
  - `Flights.day_of_week_id` references `Weekdays.did`
```

We provide the flights database as a set of plain-text data files in the linked `.tar.gz` archive. Each file in this archive contains all the rows for the named table, one row per line.

In this homework, you need to do two things:

1. import the flights dataset into SQLite
2. run SQL queries to answer a set of questions about the data.

IMPORTING THE FLIGHTS DATABASE (20 points)

To import the flights database into SQLite, you will need to run sqlite3 with a new database file. for example `sqlite3 hw2.db`. Then you can run `CREATE TABLE` statement to create the tables, choosing appropriate types for each column and 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. The command will have no effect if your version of SQLite was not compiled with foreign keys enabled. In that case do not worry about it (i.e., you will need to enforce foreign key constraints yourself as you insert data into the table).

```
PRAGMA foreign_keys=ON;
```

Then, you can use the SQLite `.import` command to read data from each text file into its table:

```
.import filename tablename
```

See examples of `.import` statements in the lecture and section notes, and also look at the SQLite documentation or sqlite3's help online for details.

Put all the code for this part (four `create table` statements and four `.import` statements) into a file called `create-tables.sql` inside the `hw2/submission` directory.

Writing SQL QUERIES (80 points, 10 points each)

HINT: You should be able to answer all the questions below with SQL queries that do NOT contain any subqueries!

For each question below, write a single SQL query to answer that question. Put each of your queries in a separate `.sql` file as in HW1, i.e., `hw2-q1.sql`, `hw2-q2.sql`, etc. Add a comment in each file indicating the number of rows in the query result.

**Important: The predicates in your queries should correspond to the English descriptions. For example, if a question asks you to find flights by Alaska Airlines Inc., the query should include a predicate that checks for that

specific name as opposed to checking for the matching carrier ID. Same for predicates over months, weekdays, etc.

Also, make sure you name the output columns as indicated as we will be grading your assignment using scripts.**

In the following questions below flights **include canceled flights as well, unless otherwise noted.** Also, when asked to output times you can report them in minutes and don't need to do minute-hour conversion.

If a query uses a `GROUP BY` clause, make sure that all attributes in your `SELECT` clause for that query are either grouping keys or aggregate values. SQLite will let you select other attributes but that is wrong as we discussed in lectures. Other database systems would reject the query in that case.

1. (10 points) List the distinct flight numbers of all flights from Seattle to Boston by Alaska Airlines Inc. on Mondays. Also notice that, in the database, the city names include the state. So Seattle appears as Seattle WA.
   Name the output column `flight_num`. [Hint: Output relation cardinality: 3 rows]

2. (10 points) Find all flights from Seattle to Boston on July 15th 2015. Search only for itineraries that have one stop. Both legs of the flight must have occurred on the same day (same day here means the date of the flight). It's fine if the landing date is different from the flight date, say in the case of an overnight flight and must be with the same carrier. The total flight time (`actual_time`) of the entire itinerary should be fewer than 7 hours (but notice that actual_time is in minutes). For each itinerary, the query should return the name of the carrier, the first flight number, the origin and destination of that first flight, the flight time, the second flight number, the origin and destination of the second flight, the second flight time, and finally the total flight time. Only count flight times here; do not include any layover time. Name the output columns `name` as 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` as the total flight time. List the output columns in this order. Put the first 20 rows of your result right after your query as a comment. [Output relation cardinality: 488 rows]

3. (10 points) Find the day of the week with the longest average arrival delay. Return the name of the day and the average delay. Name the output columns `day_of_week` and `delay`, in that order. (Hint: consider using `LIMIT`. Look up what it does!) [Output relation cardinality: 1 row]

4. (10 points) Find the names of all airlines that ever flew more than 1000 flights in one day (i.e., a specific day/month/year, but not any 24-hour period). Return only the names of the airlines. Do not return any duplicates (i.e., airlines with the exact same name). Name the output column `name`. [Output relation cardinality: 11 rows]

5. (10 points) Find all airlines that had more than 0.5 percent of their flights out of Seattle be canceled. Return the name of the airline and the percentage of canceled flight out of Seattle. Order the results by the percentage of canceled flights in ascending order. Name the output columns `name` and `percent`, in that order. [Output relation cardinality: 6 rows]

6. (10 points) Find the average price of tickets between Seattle and New York, NY in the entire month. Show the average price for each airline separately. Name the output columns `carrier` and `average_price`, in that order. [Output relation cardinality: 3]

7. (10 points) Find the total capacity of all direct flights that fly between Seattle and San Francisco, CA on July 10th, 2015. Name the output column `capacity`. [Output relation cardinality: 1]

8. (10 points) Compute the total departure delay of each airline across all flights in the entire month. Name the output columns `name` and `delay`, in that order. [Output relation cardinality: 22]

Programming style

To encourage good SQL programming style please follow these two simple style rules:

- Give explicit names to all tables referenced in the `FROM` clause. For instance, instead of writing:

  ```
  select * from flights, carriers where carrier_id = cid
  ```

  write

  ```
  select * from flights as F, carriers as C where F.carrier_id = C.cid
  ```

  So that it is clear which table you are referring to.

- Similarly, reference to all attributes must be qualified by the table name. Instead of writing:

  ```
  select * from flights where fid = 1
  ```

  write

  ```
  select * from flights as F where F.fid = 1
  ```

  This will be useful when you write queries involving self joins.

Submission Instructions

Answer each of the queries above and put your SQL query in a separate file. Call them `hw2-q1.sql`, `hw2-q2.sql`, etc. Make sure you name the files exactly as is. Put your `.sql` files inside `hw2/submission` (along with your `create-tables.sql` from the first part of this assignment).

Your directory structure should look like this after you have completed the assignment:

```
\-- hw2
    \-- hw2.pdf      # this is the file that you are currently reading
    \-- submission
        \-- hw2-q1.sql  # your solution to question 1
        \-- hw2-q2.sql  # your solution to question 2
```

```
        \-- hw2-q3.sql  # your solution to question 3
        ...
```

Make a zip/tarball of the hw2 directory and submit the zipped/tarball file.