Homework 2 | Basic SQL Queries

CSE 414 - Data Management

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

Assignment tools: SQLite 3, the flights dataset hosted here.

What to turn in: create-tables.sql, import-tables.sql, hw2-q1.sql, hw2-q2.sql, etc (see below).

Where to turn in: Gradescope

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 regarding a subset of flights that took place in 2015. The schema you should use is as follows:

```
FLIGHTS (fid int,
        month_id int,        -- 1-12
        day_of_month int,    -- 1-31
        day_of_week_id int,  -- 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,        -- 1 means canceled
        actual_time int,     -- in mins
        distance int,        -- in miles
        capacity int,
        price int            -- in $
        )

CARRIERS (cid varchar(7), name varchar(83))
MONTHS (mid int, month varchar(9))
WEEKDAYS (did int, day_of_week varchar(9))
```

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. Other than these, *do not assume any other attribute(s) is a key / unique across tuples.*
- Flights.carrier_id references Carriers.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 .zip 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 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;

Then, you can use the SQLite .import command to read data from each text file into its table after setting the input data to be in CSV (comma separated value) form:

.mode csv
.import filename tablename

See examples of .import statements in the SQLite documentation or sqlite3's help online for details.

Put all the code for *creating* your tables 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 db file in sqlite and setup the database using these two commands:

.read create-tables.sql

.read import-tables.sql

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! Do not change the output column names / return more or fewer columns!

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. Please use the flight_num column instead of fid. Name the output column **flight_num**.
   [Hint: Output relation cardinality: 3 rows]

2. (10 points) Find all itineraries from Seattle to Boston on July 15th. Search only for itineraries that have one stop (i.e., flight 1: Seattle -> [somewhere], flight2: [somewhere] -> Boston). Both flights must depart on the same day (same day here means the date of flight) and must be with the same carrier. It's fine if the landing date is different from the departing date (i.e., in the case of an overnight flight). You don't need to check whether the first flight overlaps with the second one since the departing and arriving time of the flights are not provided.

   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 in 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. [Output relation cardinality: 1472 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, 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: 12 rows]

5. (10 points) Find all airlines that had more than 0.5% (= 0.005) of their flights out of Seattle canceled. Return the name of the airline and the percentage of canceled flights out of Seattle. Percentages should be outputted in percent format (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.
   [Output relation cardinality: 6 rows]

6. (10 points) Find the maximum price of tickets between Seattle and New York, NY (i.e. Seattle to NY or NY to Seattle). Show the maximum price for each airline separately.
   Name the output columns **carrier** and **max_price**, in that order.
   [Output relation cardinality: 3 rows]

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

8. (10 points) Compute the total departure delay of each airline across all flights. Some departure delays may be negative (indicating an early departure); they should reduce the total, so you don't need to handle them specially. Name the output columns **name** and **delay**, in that order. [Output relation cardinality: 22 rows]

Submission Instructions

Please make sure that

- You are submitting the script files directly to Gradescope
- Your file names match the expected file names (create-tables.sql, import-tables.sql, and hw2-q1.sql, hw2-q2.sql, ..., hw2-q8.sql)
    - **Please make sure to double check you have named all your files correctly.**