Homework 2

(20 points)

Overview

Objectives: To create and import databases and to connect to existing SQLite databases and to practice simple SQL queries using SQLite. In this homework, you will write queries for two databases, constructed from the following files: chinook.db and flight-small.csv.
Database Files: chinook.db and flights.zip
Submission File: hw2.yaml
What to Submit: Submit the .yaml file you downloaded above, with your answers to Gradescope.
Due Date: Check Gradescope

> **Warning**
>
> To encourage local testing of commands, and to reduce the load on the Gradescope autograder, you are allowed a **MAXIMUM of 10 submissions**. If you exceed the number of submissions, your submission will not be graded. The submission with the highest score will be your final grade.

Part 1: Chinook Dataset

The chinook database is an open source SQLite database consisting of information about various elements in a fictional digital music store, such as `artists`, `albums`, `employees`, and `customers`. This information is contained in eleven tables.

Connect to the db and view all the relations

```
sqlite3  chinook.db
sqlite> .tables

albums          employees       invoices        playlists
artists         genres          media_types     tracks
customers       invoice_items   playlist_track
```

Info on each relation:

- **artists** stores artists data. It is a simple table that contains only the artist id and name.

- **albums** stores data about a list of tracks. Each album belongs to one artist. However, one artist may have multiple albums.
- **employees** stores employees data such as employee id, last name, first name, etc. It also has a field named ReportsTo to specify who reports to whom.
- **customers** stores customers data.
- **invoices & invoice_items**: these two tables store invoice data. The invoices table stores invoice header data and the invoice_items table stores the invoice line items data
- **media_types** stores media types such as MPEG audio and AAC audio files.
- **genres** stores music types such as rock, jazz, metal, etc.
- **tracks** stores the data of songs. Each track belongs to one album.
- **playlists and playlist_track**: playlists table store data about playlists. Each playlist contains a list of tracks. Each track may belong to multiple playlists. The relationship between the playlists table and tracks table is many-to-many. The playlist_track table is used to reflect this relationship.**

You can use the following command to find out the schema of a particular table in chinook database via the SQLite command-line shell program. For example try:

```
.schema artists
```

Here is the output

```
CREATE TABLE IF NOT EXISTS "artists"
(
  [ArtistId] INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
  [Name] NVARCHAR(120)
);
```

SQL Queries (7 points)

*Hint -- You should be able to write each query without any subqueries

> **Note**
>
> 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. Other database systems would reject the query in that case.

1. (1 Point) Find all the tracks that have a length of 1,000,000 milliseconds or less. Return only the `TrackId` column. *Output relation cardinality: 3288 row*

2. (1 Point) Find all the invoices from the billing country USA, and Canada and sort in descending order by invoice ID. Return two attributes - `InvoiceID` and `Total`. *Output relation cardinality: 147 rows*
3. (1 Point) Find the albums with 25 or more tracks. Return `AlbumId` and count of tracks as `TrackCount` for each `AlbumId`. *Output relation cardinality: 6 rows*
4. (1 Point) Write a query that returns a table consisting of the billing countries and the number of invoices for each country sorted by the country name in ascending order. Your output should include `BillingCountry` attribute and a count column for the number of invoices as `InvoiceCount`. *Output relation cardinality: 24 rows*
5. (1 Point) Write a query that returns a table consisting of the customers and the total amount of money spent by each customer. Output sorted `CustomerID` attribute and total money spent per customer as `TotalSpent`. *Output relation cardinality: 59 rows*
6. (1 Point) Write a query that returns the `CustumerId` for customers that listen to the `Blues` genre. Sort by `CustomerId`. *Output relation cardinality: 23 rows*
7. (1 point) For the genre `Blues`, write a query that returns artist names and total number of tracks as `NumberOfTracks`. *Output relation cardinality: 5 rows*

Part 2: Flight Dataset

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. Here is the schema:

```
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, ...
        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))
```

> **Note**
>
> The `capacity` and `price` columns in the dataset are randomly generated, All other data is 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. 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

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

In this section, you need to do two things:

- Import the flights dataset into SQLite
- Run SQL queries to answer a set of questions about the data.

IMPORTING THE FLIGHTS DATABASE (5 points)

> **Warning**
>
> You should exit out of the chinook database, and use the `sqlite3` command to create a new database, e.g. call it "flights.db". If you create tables and insert data without doing this you will add the data to the chinook database.

8. (2 points) Create all the tables necessary for the Flights database.

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 you installed your own 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;
```

9. (3 points) Insert all of the flights data from the 4 `.csv` files into the new database.

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:

> ⚠ **Warning**
>
> Loading the `.csv` files, especially `flights-small.csv` can take a while (3-10 minutes), so please be patient

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

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

SQL QUERIES (8 points)

*Hint -- You should be able to write each query without any subqueries

> 🖉 **Note**

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.

10. (2 points) Compute the total departure delay of each airline across all flights. Name the output columns `name` and `delay`, in that order. Order by airline name. *Output relation cardinality: 22 rows*
11. (2 points) Find the total capacity of all direct flights between San Diego and San Francisco on July 1st (i.e. SD to SF or SF to SD). Name the output column `totalcapacity`. *Output relation cardinality: 1 row*
12. (2 points) Write a query that returns the airline name and the percentage of canceled flights out of San Diego for the airlines that have more than 1% of their flights out of San Diego cancelled. 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: 5 rows

    Hint
    Multiplying by `1.0` is a trick used to implicitly cast an integer to a float to get a decimal result during division.

13. (2 points) Find the names of airlines that flew more than 5000 flights in at least one month originating from California. Return the names of the airlines, month name and the number of flights. Name the output columns `name`, `month` and `flightcount`. Sort the result in descending order of `flightcount`. *Output relation cardinality: 6 rows*