DSC 208R: Data Management for Analytics

Programming Assignment 1: Installing and Using SQLite

Introduction

The purpose of this assignment is to create and import databases to connect to existing SQLite databases and to practice simple SQL queries using SQLite.

We will use SQLite for this assignment. SQLite is a software library that implements an SQL database engine. We will use SQLite in this assignment because it offers an extremely lightweight method to create and analyze structured datasets (by structured, we mean datasets in the form of tables rather than, say, free text). Using SQLite is a minimal hassle approach to realizing the benefits of a relational database management system.

Of course, SQLite does not do everything, but we will get to that point in later assignments. In the meantime, you can also learn when to use SQLite and when not to use it.

This assignment contains 4 parts, all involving writing queries for two databases: chinook and flight. To turn in your assignment, submit one zipped folder containing all the .sql files from **Part I** and **Part II** to this assignment. Follow the naming convention: LastName_FirstName_PID.zip.

Note: See the assignment page in Canvas for more resources to help you complete your assignment.

Part I: CHINOOK Dataset

What you will turn in: `hw2-1.1.sql`, `hw2-1.2.sql` etc., see below

Instructions

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.

Use the following command to connect to the `chinook` database:

```
sqlite3  .../chinook.db
```

You can use the .tables to view all the tables available in the `chinook` database:

```
sqlite> .tables
albums          employees       invoices        playlists
artists         genres          media_types     tracks
customers       invoice_items   playlist_track
```

- `artists` table stores artists' data.
    - It is a simple table that contains only the artist ID and name.
- `albums` table stores data about a list of tracks.
    - Each album belongs to one artist. However, one artist may have multiple albums.
- `employees` table 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` table stores customers' data.
- `invoices` & `invoice_items` tables: 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` table stores media types such as MPEG audio and AAC audio files.
- `genres` table stores music types such as rock, jazz, metal, etc.
- `tracks` table stores the data of songs.
    - Each track belongs to one album.
- `playlists` and `playlist_track` tables: `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:

```
sqlite3  .schema table_name
```

For example, to show the statement that created the `artists` table, you use the following command:

```
sqlite3  .schema artists
```

Here is the output:

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

Questions: Writing SQL Queries (50 Points total)

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

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. **(5 points)** Find all the tracks that have a length of 1,000,000 milliseconds or less.
    a. Return only the `TrackId column`.

    [Output relation cardinality: 3288 row]

2. **(5 points)** Find all the invoices from the billing country USA, and Canada and sort in descending order by invoice ID.
    a. Return two attributes: `invoiceID` and `Total`.

    [Output relation cardinality: 147 rows]

3. **(5 points)** Find the albums with 25 or more tracks.
    a. Return `albumId` and `count` of tracks for each albumId.

    [Output relation cardinality: 6 rows]

4. **(5 points)** 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.
    a. Your output should include `BillingCountry` attribute and a `count` column for the number of invoices.

    [Output relation cardinality: 24 rows]

5. **(10 points)** Write a query that returns a table consisting of the customers and the total amount of money spent by each customer.

a. Output `customerID` attribute and `total` money spent.

   [Output relation cardinality: 59 rows]

6. **(10 points)** Write a query that returns the `customerID` for customers that are Blues listeners. The answer should not contain duplicates.

   [Output relation cardinality: 23 rows]

7. **(10 points)** Write a query that returns the artist name and total number of tracks of the Blues bands.

   [Output relation cardinality: 5 rows]

Part II: Flight Dataset

What you will turn in: `create-tables.sql` and `hw2-2.1.sql`, `hw2-2.2.sql`, etc., see below

Instructions

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:

```
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))
```

Note: 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. Other than these, *do not assume any other attribute(s) is a key / unique across tuples.*
- `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.

Question 1: Importing the Flights Database (10 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 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;
```

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 section notes, and also look at the SQLite documentation or sqlite3's help online for details.

Questions 2-5: Writing SQL Queries (40 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 (`hw2-2.1.sql`, `hw2-2.2.sql`, etc) and add a comment in each file indicating the number of rows in the query result.

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.

1. **(10 points)** Compute the total departure delay of each airline across all flights.
    a. Name the output columns `name` and `delay`, in that order.

    [Output relation cardinality: 22 rows]

2. **(10 points)** Find the total capacity of all direct flights between San Diego and San Francisco on July 1th (i.e., SD to SF or SF to SD).
    a. Name the output column `totalcapacity`.

    [Output relation cardinality: 1 row]

3. **(10 points)** Write a query that returns the name and the percentage of canceled flights out of San Diego for all the airlines that more than 1% of their flights out of San Diego were canceled. Order the results by the percentage of canceled flights in ascending order.
    a. Name the output columns `name` and `percent`, in that order.

[Output relation cardinality: 5 rows]

4. **(10 points)** Find the names of all airlines that ever flew more than 5000 flights in one month from California. Return the names of the airlines and the number of flights. Do not return any duplicates.
    a. Name the output columns `name` and `flightcount`.

    [Output relation cardinality: 6 rows]

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
```

(notice the `as`), 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 in later assignments.