Lab 13: More SQL

> Adapted from cs61a of UC Berkeley.

Starter Files

Get your starter file by cloning the repository: https://github.com/JacyCui/sicp-lab13.git

```
git clone https://github.com/JacyCui/sicp-lab13.git
```

`lab13.zip` is the starter file you need, you might need to unzip the file to get the skeleton code.

```
unzip lab13.zip
```

`README.md` is the handout for this homework. `solution` is a probrab solution of the lab. However, I might not give my solution exactly when the lab is posted. You need to finish the task on your own first. If any problem occurs, please make use of the comment section.

Topics

SQLite

Usage

First, check that a file named `sqlite_shell.py` exists alongside the assignment files. If you don't see it, or if you encounter problems with it, scroll down to the Troubleshooting section to see how to download an official precompiled SQLite binary before proceeding.

> **Note**: The default `sqlite_shell.py` is empty, which is to encourage you to manually config a sqlite environment. However, to make the ok autograder work, you need a version of `sqlite_shell.py` as support. I have provided it in the `solution` folder, you may need to copy and paste the content of `solution/sqlite_shell.py` to get your ok autograder to work.

You can start an interactive SQLite session in your Terminal or Git Bash with the following command:

```
python3 sqlite_shell.py
```

While the interpreter is running, you can type `.help` to see some of the commands you can run.

To exit out of the SQLite interpreter, type `.exit` or `.quit` or press `Ctrl-C`. Remember that if you see `...>` after pressing enter, you probably forgot a `;`.

You can also run all the statements in a `.sql` file by doing the following:

1. Runs your code and then exits SQLite immediately afterwards.

```
python3 sqlite_shell.py < lab13.sql
```

2. Runs your code and then opens an interactive SQLite session, which is similar to running Python code with the interactive `-i` flag.

```
python3 sqlite_shell.py --init lab13.sql
```

SQL Aggregation

Previously, we have been dealing with queries that process one row at a time. When we join, we make pairwise combinations of all of the rows. When we use `WHERE`, we filter out certain rows based on the condition. Alternatively, applying an aggregate function such as `MAX(column)` combines the values in multiple rows.

By default, we combine the values of the *entire* table. For example, if we wanted to count the number of flights from our `flights` table, we could use:

```
sqlite> SELECT COUNT(*) from FLIGHTS;
13
```

What if we wanted to group together the values in similar rows and perform the aggregation operations within those groups? We use a `GROUP BY` clause.

Here's another example. For each unique departure, collect all the rows having the same departure airport into a group. Then, select the `price` column and apply the `MIN` aggregation to recover the price of the cheapest departure from that group. The end result is a table of departure airports and the cheapest departing flight.

```
sqlite> SELECT departure, MIN(price) FROM flights GROUP BY departure;
AUH|932
LAS|50
LAX|89
SEA|32
SFO|40
SLC|42
```

Just like how we can filter out rows with `WHERE`, we can also filter out groups with `HAVING`. Typically, a `HAVING` clause should use an aggregation function. Suppose we want to see all airports with at least two departures:

```
sqlite> SELECT departure FROM flights GROUP BY departure HAVING COUNT(*) >= 2;
LAX
SFO
SLC
```

Note that the `COUNT(*)` aggregate just counts the number of rows in each group. Say we want to count the number of *distinct* airports instead. Then, we could use the following query:

```
sqlite> SELECT COUNT(DISTINCT departure) FROM flights;
6
```

This enumerates all the different departure airports available in our `flights` table (in this case: SFO, LAX, AUH, SLC, SEA, and LAS).

Suggested Questions

Cyber Monday

Q1: Price Check

After you are full from your Thanksgiving dinner, you realize that you still need to buy gifts for all your loved ones over the holidays. However, you also want to spend as little money as possible (you're not cheap, just looking for a great sale!).

Let's start off by surveying our options. Using the `products` table, write a query that creates a table `average_prices` that lists categories and the average price of items in the category (using MSRP as the price).

You should get the following output:

```
sqlite> SELECT category, ROUND(average_price) FROM average_prices;
computer|109.0
games|350.0
phone|90.0
```

```
CREATE TABLE average_prices AS
  SELECT "REPLACE THIS LINE WITH YOUR SOLUTION";
```

Use Ok to test your code:

```
python3 ok -q cyber-monday-part1 --local
```

Q2: The Price is Right

Now, you want to figure out which stores sell each item in products for the lowest price. Write a SQL query that uses the `inventory` table to create a table `lowest_prices` that lists items, the stores that sells that item for the lowest price, and the price that the store sells that item for.

You should expect the following output:

```
sqlite> SELECT * FROM lowest_prices;
Hallmart|GameStation|298.98
Targive|QBox|390.98
Targive|iBook|110.99
RestBuy|kBook|94.99
Hallmart|qPhone|85.99
Hallmart|rPhone|69.99
RestBuy|uPhone|89.99
RestBuy|wBook|114.29
```

```
CREATE TABLE lowest_prices AS
  SELECT "REPLACE THIS LINE WITH YOUR SOLUTION";
```

Use Ok to test your code:

```
python3 ok -q cyber-monday-part2 --local
```

Q3: Bang for your Buck

You want to make a shopping list by choosing the item that is the best deal possible for every category. For example, for the "phone" category, the uPhone is the best deal because the MSRP price of a uPhone divided by its ratings yields the lowest cost. That means that uPhones cost the lowest money per rating point out of all of the phones. Note that the item with the lowest MSRP price may not necessarily be the best deal.

Write a query to create a table `shopping_list` that lists the items that you want to buy from each category.

After you've figured out which item you want to buy for each category, add another column that lists the store that sells that item for the lowest price.

You should expect the following output:

```
sqlite> SELECT * FROM shopping_list;
GameStation|Hallmart
uPhone|RestBuy
wBook|RestBuy
```

```
CREATE TABLE shopping_list AS
  SELECT "REPLACE THIS LINE WITH YOUR SOLUTION";
```

> **Hint**: You should use the `lowest_prices` table you created in the previous question.
>
> **Hint 2**: One approach to this problem is to create another table, then create shopping_list by selecting from that table.

Use Ok to test your code:

```
python3 ok -q cyber-monday-part3 --local
```

Q4: Driving the Cyber Highways

Using the Mb (megabits) column from the `stores` table, write a query to calculate the total amount of bandwidth needed to get everything in your shopping list.

```
CREATE TABLE total_bandwidth AS
  SELECT "REPLACE THIS LINE WITH YOUR SOLUTION";
```

> **Hint**: You should use the `shopping_list` table you created in the previous question.

Use Ok to test your code:

```
python3 ok -q cyber-monday-part4 --local
```

Troubleshooting/Advanced SQLite

Troubleshooting

Python already comes with a built-in SQLite database engine to process SQL. However, it doesn't come with a "shell" to let you interact with it from the terminal. Because of this, until now, you have been using a simplified SQLite shell written by us. However, you may find the shell is old, buggy, or lacking in features. In that case, you may want to download and use the official SQLite executable.

If running `python3 sqlite_shell.py` didn't work, you can download a precompiled sqlite directly by following the following instructions and then use `sqlite3` and `./sqlite3` instead of `python3 sqlite_shell.py` based on which is specified for your platform.

Another way to start using SQLite is to download a precompiled binary from the SQLite website. The latest version of SQLite at the time of writing is 3.28.0, but you can check for additional updates on the website.

However, before proceeding, please remove (or rename) any SQLite executables (`sqlite3`, `sqlite_shell.py`, and the like) from the current folder, or they may conflict with the official one you download below. Similarly, if you wish to switch back later, please remove or rename the one you downloaded and restore the files you removed.

Windows

1. Visit the download page linked above and navigate to the section Precompiled Binaries for Windows. Click on the link **sqlite-tools-win32-x86-*.zip** to download the binary.
2. Unzip the file. There should be a `sqlite3.exe` file in the directory after extraction.
3. Navigate to the folder containing the `sqlite3.exe` file and check that the version is at least 3.8.3:

```
$ cd path/to/sqlite
$ ./sqlite3 --version
3.12.1 2016-04-08 15:09:49 fe7d3b75fe1bde41511b323925af8ae1b910bc4d
```

macOS Yosemite (10.10) or newer

SQLite comes pre-installed. Check that you have a version that's greater than 3.8.3:

```
$ sqlite3
SQLite version 3.8.10.2
```

Mac OS X Mavericks (10.9) or older

SQLite comes pre-installed, but it is the wrong version.

1. Visit the download page linked above and navigate to the section **Precompiled Binaries for Mac OS X (x86)**. Click on the link **sqlite-tools-osx-x86-*.zip** to download the binary.
2. Unzip the file. There should be a `sqlite3` file in the directory after extraction.
3. Navigate to the folder containing the `sqlite3` file and check that the version is at least 3.8.3:

```
$ cd path/to/sqlite
$ ./sqlite3 --version
3.12.1 2016-04-08 15:09:49 fe7d3b75fe1bde41511b323925af8ae1b910bc4d
```

Ubuntu

The easiest way to use SQLite on Ubuntu is to install it straight from the native repositories (the version will be slightly behind the most recent release):

```
$ sudo apt install sqlite3
$ sqlite3 --version
3.8.6 2014-08-15 11:46:33 9491ba7d738528f168657adb43a198238abde19e
```