CS 564 Project Stage 2

Invoking SQLite

SQLite is a simple relational database management system (RDBMS). In this project stage you will load a set of tables into SQLite, then write SQL queries over these tables.

First you need to log into a CS machine where SQLite has been installed. SQLite should be preinstalled on most CS machines by default. For example, it should have been installed on CS machines rockhopper-09, rockhopper-07, and royal-01.

If you use any other CS machines, check if SQLite has been installed by running the command *'sqlite3'* from the command line terminal. You should see a screen similar to the one below, which indicates that SQLite has been installed on that machine:

$sqlite3
SQLite version 3.37.2 2022-01-06 13:25:41
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite>

Once you have found a CS machine where SQLite has been installed, log into that machine and execute the command *'sqlite3'*, to bring up the SQLite command prompt (which is "sqlite>"). In the next step you will create tables and load data into them using this command prompt.

IMPORTANT:

- You do not have to use CS machines. Feel free to install SQLite on your laptop or anywhere else. However, if you do so and run into SQLite problems, we will NOT be able to help you debug problems. Due to limited TA resource, we can only provide support to debug SQLite problems if you use CS machines.
- You can learn more about SQLite and commands you can execute in SQLite at https://www.sqlitetutorial.net/. If you are unsure how to do something in SQLite, just google for it.

Creating Tables

To create the tables, execute the following steps:

1. Go to the course homepage on Canvas. Click on the "Files" link in the navigation bar on the left. Download the file CS564_stage2.zip. Verify that this file contains 7 files with csv extension. Each csv file represents a table.

2. Make sure to have the 7 csv files available on the machine that you are running SQLite. For example, if you are remote tunneling to a CS lab system via *ssh*, you will need to *scp* the files from your computer to the CS lab system. For reference about ssh and scp, see https://pages.cs.wisc.edu/~loris/cs536/ssh.html

3. Create the schemas for all tables by doing the following steps:
    a. Invoke SQLite by running the following command in the terminal (if you haven't done so already): *sqlite3*
    b. Execute the following commands at the SQLite prompt to create the schemas for all 7 tables:

```
CREATE TABLE regions (
        region_id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
        region_name text NOT NULL
);

CREATE TABLE countries (
        country_id text NOT NULL,
        country_name text NOT NULL,
        region_id INTEGER NOT NULL,
        PRIMARY KEY (country_id ASC),
        FOREIGN KEY (region_id) REFERENCES regions (region_id) ON DELETE CASCADE
ON UPDATE CASCADE
);

CREATE TABLE locations (
        location_id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
        street_address text,
        postal_code text,
        city text NOT NULL,
        state_province text,
        country_id INTEGER NOT NULL,
        FOREIGN KEY (country_id) REFERENCES countries (country_id) ON DELETE
CASCADE ON UPDATE CASCADE
);

CREATE TABLE departments (
        department_id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
        department_name text NOT NULL,
        location_id INTEGER NOT NULL,
        FOREIGN KEY (location_id) REFERENCES locations (location_id) ON DELETE
CASCADE ON UPDATE CASCADE
);

CREATE TABLE jobs (
```

```
        job_id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
        job_title text NOT NULL,
        min_salary double NOT NULL,
        max_salary double NOT NULL
);

CREATE TABLE employees (
        employee_id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
        first_name text,
        last_name text NOT NULL,
        email text NOT NULL,
        phone_number text,
        hire_date text NOT NULL,
        job_id INTEGER NOT NULL,
        salary double NOT NULL,
        manager_id INTEGER,
        department_id INTEGER NOT NULL,
        FOREIGN KEY (job_id) REFERENCES jobs (job_id) ON DELETE CASCADE ON
UPDATE CASCADE,
        FOREIGN KEY (department_id) REFERENCES departments (department_id) ON
DELETE CASCADE ON UPDATE CASCADE,
        FOREIGN KEY (manager_id) REFERENCES employees (employee_id) ON DELETE
CASCADE ON UPDATE CASCADE
);

CREATE TABLE dependents (
        dependent_id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
        first_name text NOT NULL,
        last_name text NOT NULL,
        relationship text NOT NULL,
        employee_id INTEGER NOT NULL,
        FOREIGN KEY (employee_id) REFERENCES employees (employee_id) ON DELETE
CASCADE ON UPDATE CASCADE
);
```

The above commands will have created empty tables with the correct schema.

4. Now load the 7 csv files into the above empty tables. To do so, execute the following steps:
    a. Run the following command in SQLite
        sqlite> .mode csv
    b. Now run the following command 7 times to load data into the 7 tables. Here 'filename' is the name of the csv file.
        sqlite> .import '| tail -n +2 {filepath}/{filename}.csv' {filename}

        For example, suppose the 7 csv files are stored in directory /users/user1/cs564 on the machine that you are using, then *sqlite> .import '| tail -n +2 /Users/user1/CS564_stage2/countries.csv' countries* will load data from countries.csv into table 'countries' in SQLite.

        The above (complicated) command removes the first row of each csv table

(because this row contains meta data such as the column names), then loads the remaining rows into the corresponding table.

NOTE 1: Notice that commands '.mode' and '.import' start with a dot.
NOTE 2: Make sure when you execute the above .import commands, you specify the CORRECT table names, which must be the same as the csv file names. If you misspell a table name, SQLite will silently create a table with that mispelt name for you, and later you will have problems executing SQL queries. For example, if you execute .import cs564/regions.csv regggions, then a table with the name "regggions" will be created and data will be loaded into that table, not the correct table with the name "regions".

Verifying that the Tables Have Been Created Properly

The 7 tables that you have just created contain the following information:

1. Employees table - Stores the data of employees.
2. Jobs table - Stores the job data including job title and salary range.
3. Departments table – Stores department data.
4. Dependents table - Stores the employee's dependents.
5. Locations table - Stores the location of the departments of the company.
6. Countries table - Stores the data of countries where the company is doing business.
7. Regions table - Stores the data of regions Asia, Europe, America, the Middle East and Africa. The countries are grouped into regions.

You should verify that these tables have been created properly in SQLite, by doing the following actions. If you fail to verify that the tables have been created properly, your SQL queries may return different outputs than what we are expecting.

- Validate that all tables with correct schema have been created by listing all tables using the *.tables* command. You should see 7 tables, such as shown below.

```
sqlite> .tables
countries     dependents    jobs          regions
departments   employees     locations
```

- Validate that the tables have been created with the correct schema by running *.schema* command. You should see an output similar to the one below.

```
sqlite> .schema
CREATE TABLE regions (
        region_id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
        region_name text NOT NULL
);
CREATE TABLE sqlite_sequence(name,seq);
CREATE TABLE countries (
        country_id text NOT NULL,
```

```
        country_name text NOT NULL,
        region_id INTEGER NOT NULL,
        PRIMARY KEY (country_id ASC),
        FOREIGN KEY (region_id) REFERENCES regions (region_id) ON DELETE CASCADE
ON UPDATE CASCADE
);
CREATE TABLE locations (
        location_id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
        street_address text,
        postal_code text,
        city text NOT NULL,
        state_province text,
        country_id INTEGER NOT NULL,
        FOREIGN KEY (country_id) REFERENCES countries (country_id) ON DELETE
CASCADE ON UPDATE CASCADE
);
CREATE TABLE departments (
        department_id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
        department_name text NOT NULL,
        location_id INTEGER NOT NULL,
        FOREIGN KEY (location_id) REFERENCES locations (location_id) ON DELETE
CASCADE ON UPDATE CASCADE
);
CREATE TABLE jobs (
        job_id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
        job_title text NOT NULL,
        min_salary double NOT NULL,
        max_salary double NOT NULL
);
CREATE TABLE employees (
        employee_id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
        first_name text,
        last_name text NOT NULL,
        email text NOT NULL,
        phone_number text,
        hire_date text NOT NULL,
        job_id INTEGER NOT NULL,
        salary double NOT NULL,
        manager_id INTEGER,
        department_id INTEGER NOT NULL,
        FOREIGN KEY (job_id) REFERENCES jobs (job_id) ON DELETE CASCADE ON
UPDATE CASCADE,
        FOREIGN KEY (department_id) REFERENCES departments (department_id) ON
DELETE CASCADE ON UPDATE CASCADE,
        FOREIGN KEY (manager_id) REFERENCES employees (employee_id) ON DELETE
CASCADE ON UPDATE CASCADE
);
CREATE TABLE dependents (
        dependent_id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
        first_name text NOT NULL,
        last_name text NOT NULL,
        relationship text NOT NULL,
        employee_id INTEGER NOT NULL,
        FOREIGN KEY (employee_id) REFERENCES employees (employee_id) ON DELETE
CASCADE ON UPDATE CASCADE
```

If you see the appropriate schemas for all 7 tables, the creation of tables was successful.

- Find the number of records in each table using the sql query
  SELECT COUNT(*) FROM <table name>;

  Example:

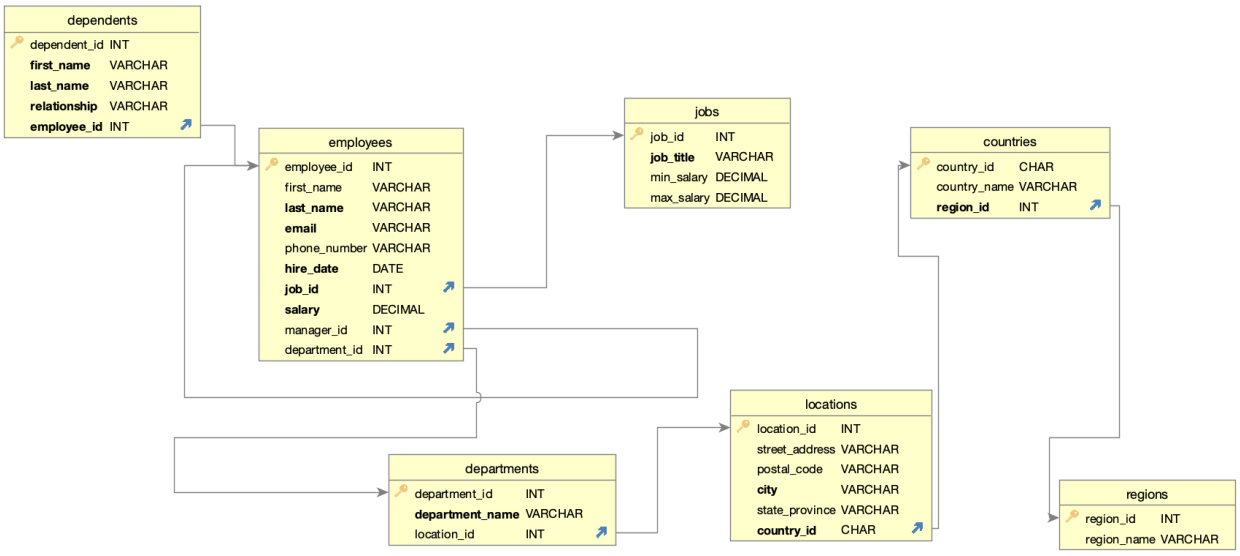
```
sqlite> SELECT COUNT(*) FROM countries;
 25
```

IMPORTANT: Make sure you put ";" at the end of the SQL query, otherwise SQLite will wait for it and won't execute your query.

To validate that loading data into tables was successful, verify that each table has the following number of records:

Table	Rows
employees	40
dependents	30
departments	11
jobs	19
locations	7
countries	25
regions	4

In case you are interested, the schemas of the 7 tables look as below. Here an arrow indicates a foreign-key constraint. For example, the arrow from dependents.employee_id into employees.employee_id means that any value in the employee_id column of table dependents must appear in the employee_id column of table employees.
.import '| tail -n +2 /Users/ruthvikareddyloka/Downloads/CS564_stage2/countries.csv' countries



Write SQL Queries

Now you need to write SQL queries for the following ten requests.

1) Return all job titles with max salary greater than or equal to 5000 and min salary less than or equal to 4000. The set that you return should not contain duplicate job titles.

2) Return all pairs <department name, number of employees in that department>, such that the department with the most employees is listed first, then the department with the second most employees is listed second, and so on (thus, the departments are listed in descending order of the number of employees). For example, you may list
Shipping, 7
Sales, 6
...

Hint: use 'order by'. You also will have to look up the operator JOIN in SQLite and use that operator.

3) For each department, compute the average of max salaries of employees in that department. Then return the names and the average max salaries of departments where the average max salary is greater than 8000.

4) Find the number of employees in the "Shipping" department.

5) Return the names of all countries in the region “Europe”.

6) Return the number of employees working in the “Europe” region.

7) Return all pairs of employees (that is, their names) who work in the same department and report to the same manager and have a salary exceeding 10000. Return only those pairs where the salary of the first employee is greater or equal the salary of the second employee.

8) Find the ID of the manager and salary of the employee with the lowest salary reports.

9) Find the name of the department that has the employee with the highest salary.

10) Return the IDs of those employees who have no dependents.

What and How to Submit

Each project group has a name, such as G1, G2, etc. You should find out what is the name of your group: click on "People" in the Canvas page for 564 and you should find the group name. Suppose your group name is G6.

Create a directory called G6. Place in this directory the files q1.sql, q2.sql, ..., q10.sql, where a file such as q1.sql contains the SQL query that is your answer to request 1. This file should just contain the SQL query and nothing else.

Also place in this directory a file G6.txt, which list the full names of all group members and their email addresses.

Zip the directory into a file G6.zip, then upload to Canvas. Each group should have just ONE member uploading this zip file. To upload, click on "Assignments", then on Project Stage 2. You should see a button that allows you to upload the zip file.

The above instruction is for a fictional group named G6. You should modify it accordingly, using your true group name.

IMPORTANT: Please follow the above instruction precisely. We will grade using a script. So if you deviate from the instruction, the script may not run and you may get points deducted.