Problem Set 1: SQL

Release Date: Feb 22, 2021

Due Date: 11:59 pm ET March 2, 2021

(Note due date moved 1 day later)

1 Introduction

The purpose of this assignment is to give you hands-on experience with the SQL programming language. SQL is a declarative language in which you specify the data you want in terms of its properties. This assignment focuses on the `SELECT` subset of SQL, which is all about querying data rather than modifying it.

We will be using SQLite, which provides a standards-compliant SQL implementation. In reality, there are slight variations between the SQL dialects of different vendors (PostgreSQL, MySQL, SQLite, Oracle, Microsoft, etc.) – especially with respect to built-in functions. The SQL tutorial at `http://sqlzoo.net/`, provides a good introduction to the basic features of SQL. After following this tutorial you should be able to answer most of the problems in this problem set. A more detailed tutorial is available at `https://www.sqlitetutorial.net/`.

To install SQLite, you can simply use the command `apt install sqlite` on Debian-based Linux distributions like Ubuntu, or `brew install sqlite` on Mac. Downloads for the pre-compiled binaries can be found at `https://sqlite.org/download.html` for Windows (as well as Linux and Mac, if you'd prefer to install with the binaries). If you use a pre-compiled binary, you might have to make sure that the path to your installed directory is in the PATH environment variable.

The SQLite SELECT documentation at `https://sqlite.org/lang_select.html` will be helpful to you, and you can access all the other SQLite documentation on that site as well. You may also wish to refer to Chapter 5 of "Database Management Systems."

Please note that the version of SQLite we are using for the autograder (3.22) does not support window functions (PARTITION BY etc). None of the queries below require window functions in their solution.

2 Dataset

For this assignment we use a dataset on modern Olympic Games adapted from `https://www.kaggle.com/heesoo37/120-years-of-olympic-history-athletes-and-results?select=athlete_events.csv`. This dataset includes details about all athletes who competed in an Olympic event from Athens 1896 to Rio 2016. Everything you need to understand the dataset is contained in your SQLite database and this pset.

The database tables include:

`athletes`: contains the id (unique to each athlete), name, sex, age when they competed in the Olympics for the first time, height (in centimeters), and weight (in kilograms) for each athlete who has participated in the Olympics.

`host_cities`: contains the games (year and season, eg. "2016 Summer"), year, season ("Summer" or "Winter"), and city (host city) for each Olympic Games.

`regions`: contains the noc (National Olympic Committee 3-letter code), region (country name), and notes for each country that has participated in the Olympics.

`athlete_events`: contains id (athlete's id), team (team name), noc (code of the country that the athlete represented), games (year and season of Olympics), sport, event, and medal ("Gold", "Silver", "Bronze", or "NA"). Each row corresponds to an individual athlete competing in an Olympic event.

Note that it is possible for any value in a non-primary key column to be `NULL`, so pay attention to whether a query may require

you to filter out null values.

3 Using the Database

Download the handout tarball from the Lecture Notes and Assignments page on the class website and untar it. To access the SQLite shell for the database, `cd` to the handout directory and run:

```
sqlite3 olympics.db
```

Once in the SQLite shell, there are two kinds of commands useful to a database user. The first kind are the client meta-commands. The most important one, of course, is `.help`, which gives you help on meta-commands. There are two others that are useful:

We can list all the table schemas in the database with `.tables`:

```
sqlite> .tables
athlete_events  athletes        host_cities     regions
```

We can check the schema (recall, that the "schema" of a database is like a class definition in an object oriented language) of a given table with `.schema <table_name>`:

```
sqlite> .schema athletes
CREATE TABLE athletes(
    id INTEGER PRIMARY KEY,
    name TEXT,
    sex TEXT,
    age INTEGER,
    height INTEGER,
    weight INTEGER
);
```

The second class of useful commands are SQL commands. All SQL queries in SQLite must be terminated with a semi-colon. For example, to get a list of all records in the `page` table, you would type:

```
SELECT * FROM athlete_events LIMIT 10;
```

This query requests a max 10 rows from the table. Using `LIMIT` in this manner one can explore the data small bits at a time. If you really wanted to produce all the records, though, the query is:

```
SELECT * FROM athlete_events;
```

You can use `Ctrl+C` to end a query that is taking too long (it is very possible to write such bad queries even unintentionally). Note that using the `LIMIT` keyword, when used by itself, offers no guarantee on which 10 rows from the result are returned, so do not assume an ordering.

You can change the way the SQLite shell displays the result sets to suit you better. In particular, you may find the commands `.header on` and `.mode column` useful.

Finally, to exit the SQLite shell, you can use `.exit`

4 Questions

Q1. Find all athletes who participated in an Olympic Games before they were 12. Print the id, name, and age of the athlete in the increasing order of id. Display the output like '`1234|Miraitowa|11`'. **[5 points]**

Q2. Print the medals table for "2016 Summer" Olympic Games. Print the NOC of the country, number of gold, silver, bronze, and total medals won by each country. Sort them in the decreasing order of gold, silver, and then bronze medals. Use NOC of the country in the increasing order as tie-breaker. Print rows only for countries that participated in the 2016 Summer games. You might find `CASE` expression useful. Display like '`USA|10|20|30|60`'. **[5 points]**

Q3. Find the number of athletes who participated in each Olympic Games. Print the games, host city, number of athletes in the increasing order of games. Display like '`1896 Summer|Athina|100`'. **[10 points]**

Q4. Find the athlete(s) with the most number of medals. Print the id, name and number of medals for each athlete. If there are multiple athletes, sort them in the increasing order of id. Display like '`1234|Vinicius|20`'. **[10 points]**

Q5. Find the oldest athlete(s) to win a gold medal in a Summer Olympic Games. Print the id, name and age of the athlete when they won their last gold medal. Note that the age column in the `athletes` table is the age when the athlete participated in their first Olympic Games. Use the age when they won a gold medal for finding the oldest athletes. If there are multiple athletes, sort them in the increasing order of id. Display like '`1234|Wenlock|64`'. **[10 points]**

Q6. Find the number of Summer Olympic Games that athletes whose name starts with 'x' have played in Asia. Only 3 Summer Olympic Games were hosted in Asia - 1964, 1988 and 2008. Print the id, name, and number of Summer Olympic Games in the increasing order of id. Display like '`1234|x Fuwa|1`'. **[10 points]**

Q7. Find all countries that have won at least one gold medal at every Summer Olympic Games. Print the name of the countries in the increasing order. Display like '`United States of America`'. **[15 points]**

Q8. Find the first Summer Olympic Games with a female participant. Print a single row with the games column. Display like '`1896 Summer`'. **[5 points]**

Q9. Find the first Summer Olympic Games in which every participating country had sent a female athlete to it or a previous Summer Olympic Games. Note that all countries in the `regions` table have not participated in every Olympic Games. Print a single row with the games column. Display like '`1896 Summer`'. **[15 points]**

Q10. Find the number of Olympic Games in which all participating countries sent female athletes. Print a single integer with the count. Display like '`10`'. **[15 points]**

5 Submission

You may work in pairs on this problem set. Only one of you needs to submit on Gradescope, but the other member must be added as a group member on the submission.

Do not hard code answers into your SQL queries. We will execute your queries using different data with the same schema.

Answer each question in the corresponding file under the `submission` directory in the handout. After filling in the queries, compress the folder by running the following command:

```
zip -j submission.zip submission/*.sql
```

The `-j` flag lets you compress all the SQL queries in the zip file without path information. The grading scripts will not work correctly unless you do this.

Each submission will be graded based on whether the SQL queries fetch the expected sets of tuples from the database. Note that your SQL queries will be auto-graded by comparing their outputs (i.e. tuple sets) to the correct outputs. For your queries,

the order of the output columns is important; their names are not. We will be comparing the output files using a function similar to diff. You can submit your answers as many times as you like.

We use the Autograder from Gradescope for grading in order to provide you with immediate feedback. The autograder will timeout after 10 mins. If a query is taking too long then try changing it. All questions have solutions that run within seconds.

After completing the homework, you can submit your compressed folder `submission.zip` (only one file) to Gradescope: `https://www.gradescope.com/courses/229885`. Use the entry code "ZRWEYN".