This is CS50x

Problem Set 7

What to Do

1. Submit [Movies](#)
2. Submit [Houses](#)

When to Do It

By 11:59pm on 31 December 2020.

Advice

- Head to [w3schools.com/sql (https://www.w3schools.com/sql/)](https://www.w3schools.com/sql/) for a handy reference!

This is CS50x

OpenCourseWare

David J. Malan (https://cs.harvard.edu/malan/)
malan@harvard.edu

f (https://www.facebook.com/dmalan) ⬡ (https://github.com/dmalan) ⓘ (https://www.instagram.com/davidjmalan/) **in** (https://www.linkedin.com/in/malan/) **Q** (https://www.quora.com/profile/David-J-Malan) 🐦 (https://twitter.com/davidjmalan)

Movies

Write SQL queries to answer questions about a database of movies.

Getting Started

Here's how to download this problem into your own CS50 IDE. Log into CS50 IDE (https://ide.cs50.io/) and then, in a terminal window, execute each of the below.

- Execute `cd` to ensure that you're in `~/` (i.e., your home directory, aka `~`).
- If you haven't already, execute `mkdir pset7` to make (i.e., create) a directory called `pset7` in your home directory.
- Execute `cd pset7` to change into (i.e., open) that directory.
- Execute `wget https://cdn.cs50.net/2019/fall/psets/7/movies/movies.zip` to download a (compressed) ZIP file with this problem's distribution.
- Execute `unzip movies.zip` to uncompress that file.
- Execute `rm movies.zip` followed by `yes` or `y` to delete that ZIP file.
- Execute `ls` . You should see a directory called `movies` , which was inside of that ZIP file.
- Execute `cd movies` to change into that directory.
- Execute `ls` . You should see a `movies.db` file, and some empty `.sql` files as well.

Alternatively, you're welcome to download and unzip cdn.cs50.net/2019/fall/psets/7/movies/movies.zip (https://cdn.cs50.net/2019/fall/psets/7/movies/movies.zip) on your own computer and then open it in DB Browser for SQLite (https://sqlitebrowser.org/dl/). But be sure to upload your `.sql` files to CS50 IDE ultimately so that you can submit them via `submit50` .

Understanding

Provided to you is a file called `movies.db` , a SQLite database that stores data from IMDb (https://www.imdb.com/) about movies, the people who directed and starred in them, and their ratings. In a terminal window, run `sqlite3 movies.db` so that you can begin executing queries on the database.

First, when `sqlite3` prompts you to provide a query, type `.schema` and press enter. This will output the `CREATE TABLE` statements that were used to generate each of the tables in the database. By examining those statements, you can identify the columns present in each table.

Notice that the `movies` table has an `id` column that uniquely identifies each movie, as well as columns for the `title` of a movie and the `year` in which the movie was released. The `people` table also has an `id` column, and also has columns for each person's `name` and `birth` year.

Movie ratings, meanwhile, are stored in the `ratings` table. The first column in the table is `movie_id` : a foreign key that references the `id` of the `movies` table. The rest of the row contains data about the `rating` for each movie and the number of `votes` the movie has received on IMDb.

Finally, the `stars` and `directors` tables match people to the movies in which they acted or directed. (Only principal (https://www.imdb.com/interfaces/) stars and directors are included.) Each table has just two columns: `movie_id` and `person_id` , which reference a specific movie and person, respectively.

The challenge ahead of you is to write SQL queries to answer a variety of different questions by selecting data from one or more of these tables.
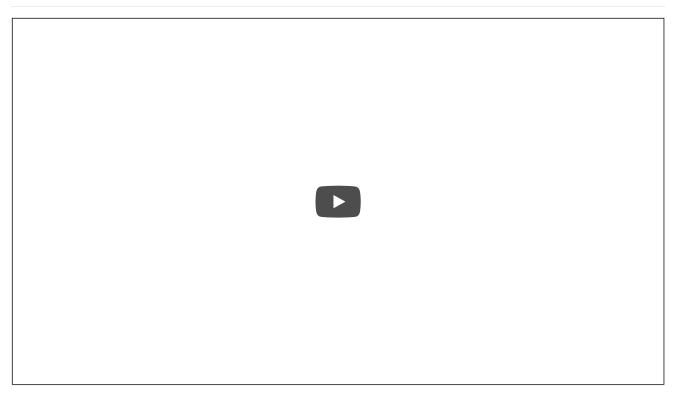
Specification

For each of the following problems, you should write a single SQL query that outputs the results specified by each problem. Your response must take the form of a single SQL query, though you may nest other queries inside of your query. You **should not** assume anything about the `id` s of any particular movies or people: your queries should be accurate even if the `id` of any particular movie or person were different. Finally, each query should return only the data necessary to answer the question: if the problem only asks you to output the names of movies, for example, then your query should not also output the each movie's release year.

You're welcome to check your queries' results against [IMDb (https://www.imdb.com/)](https://www.imdb.com/) itself, but realize that ratings on the website might differ from those in `movies.db` , as more votes might have been cast since we downloaded the data!

1. In `1.sql` , write a SQL query to list the titles of all movies released in 2008.
   - Your query should output a table with a single column for the title of each movie.
2. In `2.sql` , write a SQL query to determine the birth year of Emma Stone.
   - Your query should output a table with a single column and a single row (plus optional header) containing Emma Stone's birth year.
   - You may assume that there is only one person in the database with the name Emma Stone.
3. In `3.sql` , write a SQL query to list the titles of all movies with a release date on or after 2018, in alphabetical order.
   - Your query should output a table with a single column for the title of each movie.
   - Movies released in 2018 should be included, as should movies with release dates in the future.
4. In `4.sql` , write a SQL query to determine the number of movies with an IMDb rating of 10.0.
   - Your query should output a table with a single column and a single row (plus optional header) containing the number of movies with a 10.0 rating.
5. In `5.sql` , write a SQL query to list the titles and release years of all Harry Potter movies, in chronological order.
   - Your query should output a table with two columns, one for the title of each movie and one for the release year of each movie.
   - You may assume that the title of all Harry Potter movies will begin with the words "Harry Potter", and that if a movie title begins with the words "Harry Potter", it is a Harry Potter movie.
6. In `6.sql` , write a SQL query to determine the average rating of all movies released in 2012.
   - Your query should output a table with a single column and a single row (plus optional header) containing the average rating.
7. In `7.sql` , write a SQL query to list all movies released in 2010 and their ratings, in descending order by rating. For movies with the same rating, order them alphabetically by title.
   - Your query should output a table with two columns, one for the title of each movie and one for the rating of each movie.
   - Movies that do not have ratings should not be included in the result.
8. In `8.sql` , write a SQL query to list the names of all people who starred in Toy Story.
   - Your query should output a table with a single column for the name of each person.
   - You may assume that there is only one movie in the database with the title Toy Story.
9. In `9.sql` , write a SQL query to list the names of all people who starred in a movie released in 2004, ordered by birth year.
   - Your query should output a table with a single column for the name of each person.
   - People with the same birth year may be listed in any order.
   - No need to worry about people who have no birth year listed, so long as those who do have a birth year are listed in order.
   - If a person appeared in more than one movie in 2004, they should only appear in your results once.
10. In `10.sql` , write a SQL query to list the names of all people who have directed a movie that received a rating of at least 9.0.
    - Your query should output a table with a single column for the name of each person.
11. In `11.sql` , write a SQL query to list the titles of the five highest rated movies (in order) that Chadwick Boseman starred in, starting with the highest rated.
    - Your query should output a table with a single column for the title of each movie.
    - You may assume that there is only one person in the database with the name Chadwick Boseman.
12. In `12.sql` , write a SQL query to list the titles of all movies in which both Johnny Depp and Helena Bonham Carter starred.
    - Your query should output a table with a single column for the title of each movie.
    - You may assume that there is only one person in the database with the name Johnny Depp.
    - You may assume that there is only one person in the database with the name Helena Bonham Carter.
13. In `13.sql` , write a SQL query to list the names of all people who starred in a movie in which Kevin Bacon also starred.
    - Your query should output a table with a single column for the name of each person.
    - There may be multiple people named Kevin Bacon in the database. Be sure to only select the Kevin Bacon born in 1958.
    - Kevin Bacon himself should not be included in the resulting list.

Walkthrough

Usage

To test your queries on CS50 IDE, you can query the database by running

```
$ cat filename.sql | sqlite3 movies.db
```

where `filename.sql` is the file containing your SQL query.

Or you can paste them into DB Browser for SQLite's **Execute SQL** tab and click ▶.

Hints

- See [this SQL keywords reference (https://www.w3schools.com/sql/sql_ref_keywords.asp)](https://www.w3schools.com/sql/sql_ref_keywords.asp) for some SQL syntax that may be helpful!

Testing

No `check50` for this problem! But be sure to test each query and ensure that the output is what you expect. You can run `sqlite3 movies.db` to run additional queries on the database to ensure that your result is correct.

If you're using the `movies.db` database provided in this problem set's distribution, you should find that

- Executing `1.sql` results in a table with 1 column and 9,480 rows.
- Executing `2.sql` results in a table with 1 column and 1 row.
- Executing `3.sql` results in a table with 1 column and 35,755 rows.
- Executing `4.sql` results in a table with 1 column and 1 row.
- Executing `5.sql` results in a table with 2 columns and 10 rows.
- Executing `6.sql` results in a table with 1 column and 1 row.
- Executing `7.sql` results in a table with 2 columns and 6,835 rows.
- Executing `8.sql` results in a table with 1 column and 4 rows.
- Executing `9.sql` results in a table with 1 column and 18,013 rows.
- Executing `10.sql` results in a table with 1 column and 1,841 rows.
- Executing `11.sql` results in a table with 1 column and 5 rows.
- Executing `12.sql` results in a table with 1 column and 6 rows.

- Executing `13.sql` results in a table with 1 column and 176 rows.

How to Submit

Execute the below, logging in with your GitHub username and password when prompted. For security, you'll see asterisks (*) instead of the actual characters in your password.

```
submit50 cs50/problems/2020/x/movies
```

Acknowledgements

Information courtesy of IMDb (imdb.com (http://www.imdb.com)). Used with permission.

Houses

Implement a program to import student data into a database, and then produce class rosters.

```
$ python import.py characters.csv
$ python roster.py Gryffindor

Lavender Brown, born 1979
Colin Creevey, born 1981
Seamus Finnigan, born 1979
Hermione Jean Granger, born 1979
Neville Longbottom, born 1980
Parvati Patil, born 1979
Harry James Potter, born 1980
Dean Thomas, born 1980
Romilda Vane, born 1981
Ginevra Molly Weasley, born 1981
Ronald Bilius Weasley, born 1980
```

Getting Started

Here's how to download this problem into your own CS50 IDE. Log into CS50 IDE (https://ide.cs50.io/) and then, in a terminal window, execute each of the below.

- Execute `cd` to ensure that you're in `~/` (i.e., your home directory, aka `~`).
- If you haven't already, execute `mkdir pset7` to make (i.e., create) a directory called `pset7` in your home directory.
- Execute `cd pset7` to change into (i.e., open) that directory.
- Execute `wget https://cdn.cs50.net/2019/fall/psets/7/houses/houses.zip` to download a (compressed) ZIP file with this problem's distribution.
- Execute `unzip houses.zip` to uncompress that file.
- Execute `rm houses.zip` followed by `yes` or `y` to delete that ZIP file.
- Execute `ls` . You should see a directory called `houses` , which was inside of that ZIP file.
- Execute `cd houses` to change into that directory.
- Execute `ls` . You should see a `characters.csv` file and a `students.db` database.

Background

Hogwarts is in need of a student database. For years, the professors have been mantaing a CSV file containing all of the students' names and houses and years. But that file didn't make it particularly easy to get access to certain data, such as a roster of all the Ravenclaw students, or an alphabetical listing of the students enrolled at the school.

The challenge ahead of you is to import all of the school's data into a SQLite database, and write a Python program to query that database to get house rosters for each of the houses of Hogwarts.
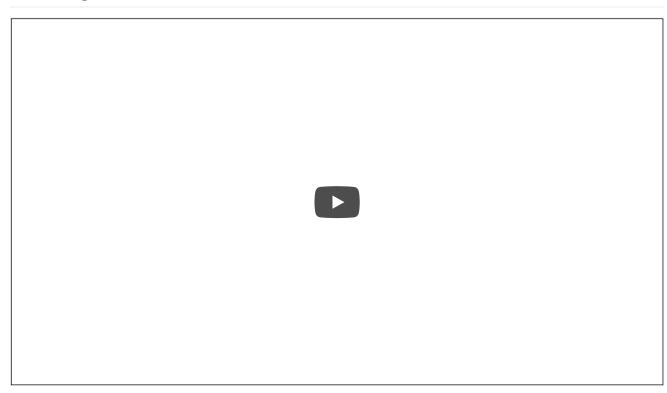
Specification

In `import.py` , write a program that imports data from a CSV spreadsheet.

- Your program should accept the name of a CSV file as a command-line argument.
  - If the incorrect number of command-line arguments are provided, your program should print an error and exit.
  - You may assume that the CSV file will exist, and will have columns `name`, `house`, and `birth`.
- For each student in the CSV file, insert the student into the `students` table in the `students.db` database.
  - While the CSV file provided to you has just a `name` column, the database has separate columns for `first`, `middle`, and `last` names. You'll thus want to first parse each name and separate it into first, middle, and last names. You may assume that each person's name field will contain either two space-separated names (a first and last name) or three space-separated names (a first, middle, and last name). For students without a middle name, you should leave their `middle` name field as `NULL` in the table.

In `roster.py`, write a program that prints a list of students for a given house in alphabetical order.

- Your program should accept the name of a house as a command-line argument.
  - If the incorrect number of command-line arguments are provided, your program should print an error and exit.
- Your program should query the `students` table in the `students.db` database for all of the students in the specified house.
- Your program should then print out each student's full name and birth year (formatted as, e.g., `Harry James Potter, born 1980` or `Luna Lovegood, born 1981`).
  - Each student should be printed on their own line.
  - Students should be ordered by last name. For students with the same last name, they should be ordered by first name.

Walkthrough

Usage

Your program should behave per the example below:

```
$ python import.py characters.csv
```

```
$ python roster.py Gryffindor
Hermione Jean Granger, born 1979
Harry James Potter, born 1980
Ginevra Molly Weasley, born 1981
Ronald Bilius Weasley, born 1980
...
```

Hints

- Recall that after you've imported `SQL` from `cs50` , you can set up a database connection using syntax like

```
db = SQL("sqlite:///students.db")
```

Then, you can use `db.execute` to execute SQL queries from inside of your Python script.

- Recall that when you call `db.execute` and perform a `SELECT` query, the return value will be a `list` of rows that are returned, where each row is represented by a Python `dict` .

Testing

No `check50` for this problem, but be sure to test your code for each of the following.

```
$ python import.py characters.csv
$ python roster.py Gryffindor
Lavender Brown, born 1979
Colin Creevey, born 1981
Seamus Finnigan, born 1979
Hermione Jean Granger, born 1979
Neville Longbottom, born 1980
Parvati Patil, born 1979
Harry James Potter, born 1980
Dean Thomas, born 1980
Romilda Vane, born 1981
Ginevra Molly Weasley, born 1981
Ronald Bilius Weasley, born 1980

$ python roster.py Hufflepuff
Hannah Abbott, born 1980
Susan Bones, born 1979
Cedric Diggory, born 1977
Justin Finch-Fletchley, born 1979
Ernest Macmillan, born 1980

$ python roster.py Ravenclaw
Terry Boot, born 1980
Mandy Brocklehurst, born 1979
Cho Chang, born 1979
Penelope Clearwater, born 1976
Michael Corner, born 1979
Roger Davies, born 1978
Marietta Edgecombe, born 1978
Anthony Goldstein, born 1980
Robert Hilliard, born 1974
Luna Lovegood, born 1981
Isobel MacDougal, born 1980
Padma Patil, born 1979
Lisa Turpin, born 1979

$ python roster.py Slytherin
Millicent Bulstrode, born 1979
Vincent Crabbe, born 1979
Tracey Davis, born 1980
Marcus Flint, born 1975
Gregory Goyle, born 1980
Terence Higgs, born 1979
Draco Lucius Malfoy, born 1980
Adelaide Murton, born 1982
Pansy Parkinson, born 1979
Adrian Pucey, born 1977
Blaise Zabini, born 1979
```

How to Submit

Execute the below, logging in with your GitHub username and password when prompted. For security, you'll see asterisks (*) instead of the actual characters in your password.

```
submit50 cs50/problems/2020/x/houses
```