This is CS50x

OpenCourseWare

Donate (https://cs50.harvard.edu/donate)

David J. Malan (https://cs.harvard.edu/malan/)
malan@harvard.edu
(https://www.facebook.com/dmalan) (https://github.com/dmalan) (https://www.instagram.com/davidjmalan/) (https://www.linkedin.com/in/malan/) (https://orcid.org/0000-0001-5338-2522) (https://www.quora.com/profile/David-J-Malan) (https://www.reddit.com/user/davidjmalan) (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 ~` (or simply `cd` with no arguments) to ensure that you're in your home directory.
- Execute `mkdir pset7` to make (i.e., create) a directory called `pset7`.
- Execute `cd pset7` to change into (i.e., open) that directory.
- Execute `wget https://cdn.cs50.net/2020/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.

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

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 each movie's release year.

You're welcome to check your queries' results against IMDb (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 (not including the 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 (not including the 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 (not including the 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.
    - If a person directed more than one movie that received a rating of at least 9.0, they should only appear in your results once.
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.

You can also run

```
$ cat filename.sql | sqlite3 movies.db > output.txt
```

to redirect the output of the query to a text file called `output.txt`. (This can be useful for checking how many rows are returned by your query!)

Hints

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

Testing

While `check50` is available for this problem, you're encouraged to instead test your code on your own for each of the following. 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 9,545 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 50,863 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 6,864 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 18,237 rows.
- Executing `10.sql` results in a table with 1 column and 1,887 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.

Note that row counts do not include header rows that only show column names.

If your query returns a number of rows that is slightly different from the expected output, be sure that you're properly handling duplicates! For queries that ask for a list of names, no one person should be listed twice, but two different people who have the same name should each be listed.

Execute the below to evaluate the correctness of your code using `check50`.

```
check50 cs50/problems/2021/x/movies
```

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/2021/x/movies
```

Acknowledgements

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