


# This is CS50

## CS50's Introduction to Computer Science

OpenCourseWare

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

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

[malan@harvard.edu](mailto:malan@harvard.edu)

 (<https://www.facebook.com/dmalan>)  (<https://github.com/dmalan>) 

(<https://www.instagram.com/davidjmalan/>)  (<https://www.linkedin.com>

[in/malan/](https://www.linkedin.com/in/malan/))  (<https://www.reddit.com/user/davidjmalan>) 

(<https://www.threads.net/@davidjmalan>)  (<https://twitter.com/davidjmalan>)

## Movies

The IMDb logo is displayed on a yellow rectangular background. The letters "IMDb" are in a bold, black, sans-serif font. The "i" is lowercase and smaller than the other letters.

## Problem to Solve

Provided to you is a file called `movies.db`, a SQLite database that stores data from [IMDb](https://www.imdb.com/) (<https://www.imdb.com/>) about movies, the people who directed and starred in them, and their ratings. Write SQL queries to answer questions about this database of movies.

## Demo

```
+-----+
sqlite> SELECT * FROM stars LIMIT 5;
+-----+
| movie_id | person_id |
+-----+
| 11801    | 459029    |
| 11801    | 681726    |
| 11801    | 692612    |
| 11801    | 726256    |
| 11801    | 776458    |
+-----+
sqlite> .quit
```

Recorded with [asciinema](#)

## Getting Started

For this problem, you'll use a database provided to you by CS50's staff.

### ▼ Download the distribution code

Log into [cs50.dev \(https://cs50.dev/\)](https://cs50.dev/), click on your terminal window, and execute `cd` by itself. You should find that your terminal window's prompt resembles the below:

```
$
```

Next execute

```
wget https://cdn.cs50.net/2023/fall/psets/7/movies.zip
```

in order to download a ZIP called `movies.zip` into your codespace.

Then execute

```
unzip movies.zip
```

to create a folder called `movies`. You no longer need the ZIP file, so you can execute

```
rm movies.zip
```

and respond with “y” followed by Enter at the prompt to remove the ZIP file you downloaded.

Now type

```
cd movies
```

followed by Enter to move yourself into (i.e., open) that directory. Your prompt should now resemble the below.

```
movies/ $
```

Execute `ls` by itself, and you should see 13 `.sql` files, as well as `movies.db`.

If you run into any trouble, follow these same steps again and see if you can determine where you went wrong!

## 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 \(https://www.imdb.com/\)](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 counting 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 counting 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 counting 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 Bradley Cooper and Jennifer Lawrence 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 Bradley Cooper.
  - You may assume that there is only one person in the database with the name Jennifer Lawrence.
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.

## Hints

### ▼ Understand the schema of `movies.db`

Whenever you engage with a new database, it's best to first understand its *schema*. 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 \(https://www.imdb.com/interfaces/\)](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.

### ▼ Consistently style your queries

See [sqlstyle.guide \(https://www.sqlstyle.guide/\)](https://www.sqlstyle.guide/) for pointers on good style in SQL, especially as your queries get more complex!

### ▼ List the titles of all movies released in 2008

Recall that you can select one (or more) columns from a database using `SELECT`, per the example below,

```
SELECT column0, column1 FROM table;
```

where `column0` is the title of one column, and `column1` is the title of another.

And recall that you can filter the rows returned in a query with the `WHERE` keyword, followed by a condition. You can use `=`, `>`, `<`, and [other operators \(https://www.w3schools.com/sql/sql\\_operators.asp\)](https://www.w3schools.com/sql/sql_operators.asp) too.

```
SELECT column FROM table
WHERE condition;
```

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!

### ▼ Determine the birth year of Emma Stone

Recall that a `WHERE` clause can evaluate conditions not just with numbers, but with strings.

### ▼ List the titles of all movies with a release date on or after 2018, in alphabetical order

Try breaking this query into two steps. First, find the movies with a release date on or after 2018. Then, put those movies' titles in alphabetical order.

To find the movies with a release date on or after 2018, recall that a condition in SQL

supports the use of many common [comparison operators \(https://www.w3schools.com/sql/sql\\_operators.asp\)](https://www.w3schools.com/sql/sql_operators.asp), including `>=` for “greater than or equal to.” Check to see if your query returns the correct number of movies, per [How to Test](#).

Finally, sort the query’s results alphabetically by title. Recall that `ORDER BY` can sort data by a column in your results, per the example below.

```
...  
ORDER BY column;
```

#### ▼ Determine the number of movies with an IMDb rating of 10.0

Notice this question asks you not for *individual* movies with a rating of 10.0, but for the *number* of movies with such a rating. In other words, you should collect (“aggregate”) the results of your query into a single number (the number of rows). Recall that SQL supports an “aggregation function” called `COUNT`, which you can use on a column per the example below.

```
SELECT COUNT(column)  
FROM table;
```

#### ▼ List the titles and release years of all Harry Potter movies, in chronological order

For this query, you’ll likely want to make use of SQL’s `LIKE` keyword. Recall that `LIKE` can make use of so-called “wildcard characters”, such as `%`, that will match any character (or sequence thereof).

```
SELECT column0, column1  
FROM table  
WHERE column1 LIKE pattern;
```

#### ▼ Determine the average rating of all movies released in 2012

Here’s another example of a query in which you’ll need to aggregate data. Consider SQL’s `AVG` aggregation function, to compute an average.

Consider, too, that this query makes use of data stored in two separate tables: `ratings` and `movies`. Recall that—so long as one table has a foreign key that matches a column in another table—you can combine two tables using SQL’s `JOIN` keyword. To use the `JOIN` keyword, you should specify the table you’d like to join and the column by which to do so.

```
SELECT column0  
FROM table0  
JOIN table1 ON table0.column1 = table1.column2
```

#### ▼ List all movies released in 2010 and their ratings, in descending order by rating

Recall that `ORDER BY` need not always sort in ascending order. You can specify that your results be sorted in *descending* order by appending `DESC`.

```
...  
ORDER BY column DESC;
```

### ▼ List the names of all people who starred in Toy Story

When you see a more complex query such as this one, it's best to break it down into smaller pieces. Ultimately, your query should arrive at a list of names, per the below.

```
-- Select names  
SELECT name  
FROM people  
WHERE ...
```

But how's best to arrive at the names of those who starred in Toy Story? Consider that the `people` table alone doesn't have this information (but the `stars` table might!). Indeed, the `stars` table combines two columns, `person_id` and `movie_id`: any person with a `person_id` that is associated with Toy Story's `movie_id` starred in Toy Story.

```
-- Select names  
SELECT name  
FROM people  
WHERE ...  
  
-- Select person IDs  
SELECT person_id  
FROM stars  
WHERE movie_id = ...
```

A natural next step, then, is to find Toy Story's movie ID.

```
-- Select names  
SELECT name  
FROM people  
WHERE ...  
  
-- Select person IDs  
SELECT person_id  
FROM stars  
WHERE movie_id = ...  
  
-- Find Toy Story's ID  
SELECT id  
FROM movies  
WHERE title = 'Toy Story';
```

Of course, you've presently written three *separate* queries. But notice that some queries (the first two) would be complete by including results of the query directly below them. The



process of making a query that depends on the results of a “subquery” is called “nesting” queries. It’s quite the hint, but here’s one way to nest the above queries!

```
-- Select names
SELECT name
FROM people
WHERE id IN
(
    -- Select person IDs
    SELECT person_id
    FROM stars
    WHERE movie_id = (
        -- Select Toy Story's ID
        SELECT id
        FROM movies
        WHERE title = 'Toy Story'
    )
);
```

▼ **List the names of all people who starred in a movie released in 2004, ordered by birth year**

Notice that this query, like the previous, requires you to use data from multiple tables. Recall that you can “nest” queries in SQL, which allows you to break a larger query into smaller ones. Perhaps you could write queries to...

1. Find the IDs of movies released in 2004
2. Find the IDs of people who starred in those movies
3. Find the names of people with those people IDs

Then, try nesting those queries to arrive at a single query that returns all people who starred in a movie released in 2004. Consider how you might then order the results of your query.

▼ **List the names of all people who have directed a movie that received a rating of at least 9.0**

Notice that this query, like the previous, requires you to use data from multiple tables. Recall that you can “nest” queries in SQL, which allows you to break a larger query into smaller ones. Perhaps you could write queries to...

1. Find the IDs of movies with at least a 9.0 rating
2. Find the IDs of people who directed those movies
3. Find the names of people with those people IDs

Then, try nesting those queries to arrive at a single query that returns the names of all people who have directed a movie that received a rating of at least 9.0.

▼ **List the titles of the five highest rated movies (in order) that Chadwick Boseman starred in,**

### starting with the highest rated

Notice that this query, like the previous, requires you to use data from multiple tables. Recall that you can “nest” queries in SQL, which allows you to break a larger query into smaller ones. Perhaps you could write queries to...

1. Find the ID of Chadwick Boseman
2. Find the IDs of movies associated with Chadwick Boseman's ID
3. Find the movie titles with those movie IDs

Then, try nesting those queries to arrive at a single query that returns the titles of Chadwick Boseman's movies.

From there, you'll need to determine the ratings of those titles and sort those titles by rating, in descending order. Consider how you could combine a relevant table (likely `ratings`!) and order the results by a relevant column.

Finally, read up on SQL's `LIMIT` (<https://www.sqlitetutorial.net/sqlite-limit/>) keyword, which will return the top  $n$  rows in a query.

### ▼ List the titles of all movies in which both Bradley Cooper and Jennifer Lawrence starred

Notice that this query, like the previous, requires you to use data from multiple tables. Recall that you can “nest” queries in SQL, which allows you to break a larger query into smaller ones. Perhaps you could write queries to...

1. Find the ID of Bradley Cooper
2. Find the ID of Jennifer Lawrence
3. Find the IDs of movies associated with Bradley Cooper's ID
4. Find the IDs of movies associated with Jennifer Lawrence's ID
5. Find movie titles from the movie IDs associated with *both* Bradley Cooper and Jennifer Lawrence

Then, try nesting those queries to arrive at a single query that returns the movies in which both Bradley Cooper and Jennifer Lawrence starred.

Recall that you can build compound conditions in SQL using `AND` or `OR`.

### ▼ List the names of all people who starred in a movie in which Kevin Bacon also starred

Notice that this query, like the previous, requires you to use data from multiple tables. Recall that you can “nest” queries in SQL, which allows you to break a larger query into smaller ones. Perhaps you could write queries to...

1. Find the ID of Kevin Bacon (the one born in 1958!)

2. Find the IDs of movies associated with Kevin Bacon's ID
3. Find the IDs of people associated with those movie IDs
4. Find the names of people with those people IDs

Then, try nesting those queries to arrive at a single query that returns the names of all people who starred in a movie in which Kevin Bacon also starred. **Keep in mind that you'll want to exclude Kevin Bacon himself from the results!**

## Walkthrough



## Usage

To test your queries in VS Code, 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!)

## How to Test

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 10,276 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 110,014 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 11 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 7,192 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 19,325 rows.
- Executing `10.sql` results in a table with 1 column and 3,854 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 4 rows.
- Executing `13.sql` results in a table with 1 column and 182 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.

## Correctness

```
check50 cs50/problems/2024/x/movies
```

## How to Submit

```
submit50 cs50/problems/2024/x/movies
```

## Acknowledgements

---

Information courtesy of IMDb ([imdb.com \(https://www.imdb.com\)](https://www.imdb.com)). Used with permission.