

# CITS1402 Week 9 Lab

Gordon Royle

2022 Semester Two

There are five queries to submit this week to the usual location at

<https://secure.csse.uwa.edu.au/run/cssubmit>

in files names `A12.sql`, `A13.sql`, `A14.sql`, `A15.sql` and `A16.sql`.

Your lab mark will be the sum total of your 16 lab marks, except that it is capped at a total of 15 marks. So anyone who has lost a mark during the previous labs has an opportunity to “earn it back” again.

As usual, double-check that that *contents* of your file are in the correct format for testing by typing

```
cat A12.sql
```

at the Terminal / Powershell prompt and making sure that the file is a plain text file containing a single query, starting with `SELECT` and terminating with a semicolon.

In addition, check that the *operation* of your query is correct by testing it against the sample database `imdb_top_250.db` by starting `sqlite3` from the Terminal/Powershell and then

```
.open imdb_top_250.db
.read A12.sql
```

and checking that the output *meets the specifications* of the question.

It is vital that you perform these tests using the *actual file* that you have submitted. Do not use another file that you have created by cutting and pasting or retyping your query.

Double check that you have the correct number of columns, that the columns are *in the specified order*, and that your query is consistent with any self-check hints.

I will be testing your queries against a larger database with the Top-1000 movies, not the Top-250. Make sure that your queries do not rely on features that “just happen” to be true for the Top-250. In particular, while the Top-250 movies happen to be uniquely identified by their title, this is not guaranteed to be true for the Top-1000 movies. However, you can always be certain that `title_id` uniquely identifies a movie.

## Learning Aims

This lab consolidates some more advanced queries, and also introduces the use of `NULL` and `LEFT OUTER JOIN`.

premiered	COUNT(*)
1995	8
2019	6
2014	6
2009	6
2006	6
2004	6
2003	6
2001	6

Figure 1: Output expected for Question 3

## Questions

1. Write a single SQL query that, *for each category* of crew member, lists the *category* and the *number of occurrences* of that category in the `crewmembers` table.

SELF CHECK HINT There are 133 *cinematographer* listings in the database, so one of the rows of your table should indicate this.

2. Write a single SQL query that, *for each category* of crew member, lists the *category* and the *number of people* who have crewed in that category.

This is different from the previous question because this counts a person *once only*, even if they have been a cinematographer on several movies.

SELF CHECK HINT There are 103 different people who have been cinematographers in the sample database.

3. **Submission: A12.sql** Write a single SQL query that lists, for each year represented in the database, the *year* and *number of movies* from that year in the database, listing them first by number of movies in descending order, and then by descending year within each group of years with the same number of movies.

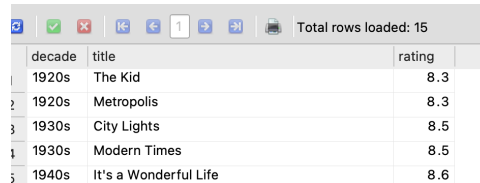
SELF-CHECK HINT: The most prolific year for Top-250 movies is 1995, with no less than 8 movies from that year in the Top-250, as shown in Figure 1.

4. **Submission: A13.sql** The column `person_id` is used as the primary key for the table `people`, which is necessary because it is possible for different people to have the same name. Test whether or not this *actually occurs* by writing a single SQL query that will list the personal details (i.e., all the columns of `people`) of everyone who shares a name with someone else.

If it happens to be the case that there are four people named “John Smith”, then the output should contain four rows, one for each of the different John Smiths.

SELF CHECK HINT In the Top-250 database, there are two pairs of people with the same name, and so the output should contain four rows.

5. Write a single SQL query that lists the *person id*, *name* and *category* for each person who has crewed in *more than one category* (not necessarily in the same film). There should be one row for each person / category combination.
6. Write a single SQL query that counts the *number of living people* in `people`.
7. Write a single SQL query, *not using a subquery* that lists, for each movie in the database, its *title* and the number of *still-living* cast members for that movie. This is easy to do with subqueries, so the task is to do it without.



decade	title	rating
1920s	The Kid	8.3
1920s	Metropolis	8.3
1930s	City Lights	8.5
1930s	Modern Times	8.5
1940s	It's a Wonderful Life	8.6

Figure 2: Sample output for “movies of the decade”

8. Write a single SQL query that will list, for each *combination* of movie and category of crew member, the *movie title*, the *category* and the *number of crew members* in that category for that movie. (If a movie has, say, no credited **writer**, then you *do not* have to create a row explicitly showing 0 writers.)

SELF CHECK HINT According to the database, the movie *City Lights* has 2 cinematographers and 2 writers, but no other crew.

9. **Submission: A14.sql** Write a single SQL query that will list, for each movie, its *title* and the *number of crew members* listed for that movie. If a movie has no crew listed for it in **crewmembers**, it should still appear in the list, with 0 as the corresponding number of crew members.

SELF-CHECK HINT The movie *The Kid* has no credited crew members, while *Metropolis* has 6.

10. **Submission: A15.sql** Write a single SQL query that lists the *title* of every movie for which *all of* the associated cast members are still alive.

SELF-CHECK HINT: Even though *The Godfather II* is nearly 50 years old, its main stars are all still alive, and so this should be on your list.

11. **Submission: A16.sql** Write a single SQL query that lists, for each decade, the *name of the decade* (in the format detailed below) and the *title* and *rating* of the highest rated movie (or movies, if there is more than one equally best rated) that premiered in that decade, listed in increasing order of decade.

For this purpose, a *decade* refers to a 10-year time-span, such as 1920–1929, 1930–1939, and continuing until 2020–2029. Refer to each decade using the strings 1920s, 1930s, and so on.

SELF CHECK HINT See Figure 2 for the first few rows of the table.

12. Which actor(s) are in the most Top-250 movies? Write a single SQL query that lists, *for each person* in the **people** table (not just the castmembers table), their *person id*, *name* and the *number* of Top-250 movies in which they have appeared, ordered so that the actors in the most Top-250 movies occur first.

It was a surprise to me that there is one actor who is far ahead of the others at the top of this table, having appeared in 50% more Top-250 movies than the next-best.