CITS1402 Week 4 Lab

Gordon Royle

2023 Semester Two

Learning Aims

In this lab, we will use a program that provides a *graphical user interface* (GUI) to a SQLite database to explore the SQLite database `imdb_top_250.db` containing information about the 250 highest-rated movies as listed by the Internet Movie Database (`https://imdb.com`).

The questions will focus on joins of tables, and will introduce a few more SQL features such as the ability to sort the rows of the output table according to user-defined criteria.

Install SQLiteStudio

We will use a GUI called SQLiteStudio, so first you should download the version for your operating system from `https://sqlitestudio.pl` and place the executable file somewhere convenient.

When you open SQLiteStudio, you will see a graphical interface with a number of windows. The different windows allow you to open a database, and then browse through the tables and their contents by clicking, and examine the database in various ways.

Please remember that this is a convenient "front-end" to SQLite and *is not part of SQLite* itself. It is just a different way to interact with SQLite than the command line based interface provided by the `sqlite3`.

However we can still use this interface to learn SQL, because SQLiteStudio allows you to type a SQL query which is then run (by SQLite) and whose output is then displayed in a user-friendly fashion.

Explore

In order to connect to a database from SQLiteStudio, you must do the following:

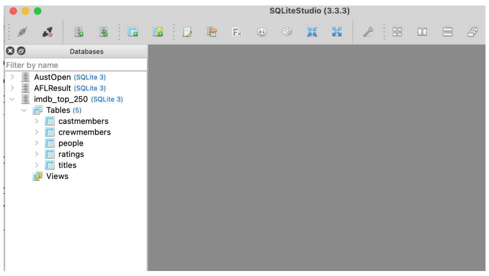
- Start SQLiteStudio running by double-clicking the SQLiteStudio icon.

This should work immediately on Windows, but can be a little more complicated for Mac Users, because recent versions of Mac OS X have enhanced, perhaps over-enhanced, security features that prevent an unknown application (i.e., one downloaded directly from the internet, rather than via the Mac App Store) running when double-clicked. To install the program you may need to open Apple-Menu / System Settings and then select the Privacy and Security tab, then somewhere at the bottom right, it will give you the option to install and/or run the software.

Then you will get another warning about untrusted applications, but this time you will be able to choose "Open". You only need to do this once (to demonstrate that you *genuinely* want to run the application). Eventually, you should get a mostly empty graphical user interface waiting for further commands.

Although the windows will look slightly different, the remainder of the process should be essentially the same for both Mac and Windows users.

- Use the menu item `Database > Add Database` which presents an interface where you can navigate to the file `imdb_top_250.db`; it will then appear in the left-most window pane.



- The left-hand panel allows you to interactively click on the database to see the tables, or on the tables to see the columns and so on. If you double click on a table, the middle area will contain a tabbed panel with lots of information about the table.

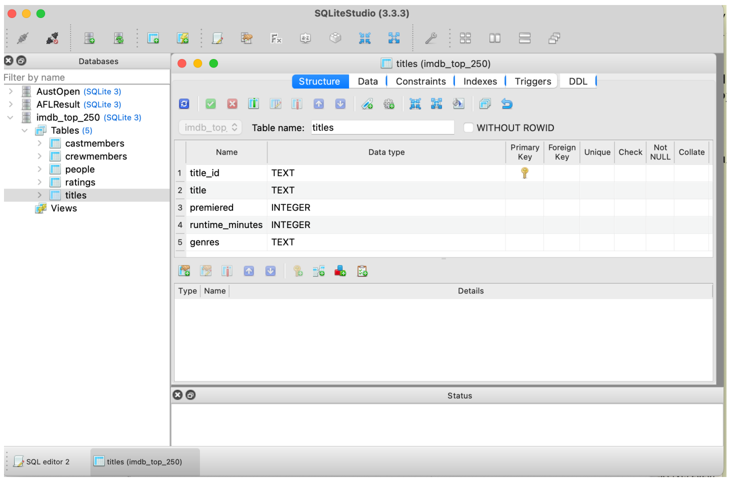
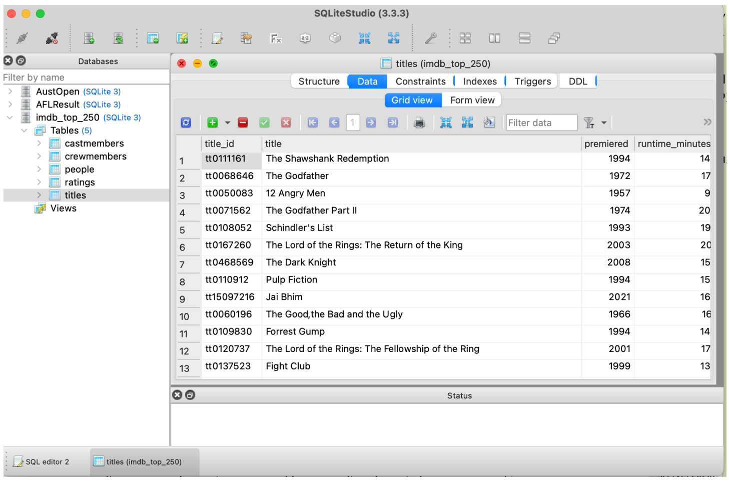


Table name: titles

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate
1	title_id	TEXT	🔑					
2	title	TEXT						
3	premiered	INTEGER						
4	runtime_minutes	INTEGER						
5	genres	TEXT						

- The default tab that is selected is "Structure", but if you choose "Data" then it will display the actual data from that table in a form that you can easily scroll through and examine by hand.

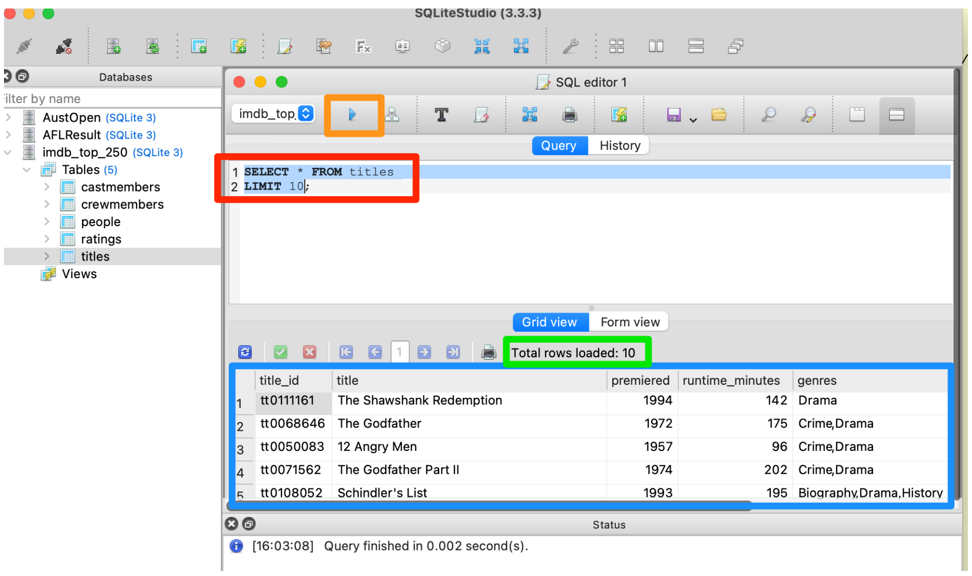


	title_id	title	premiered	runtime_minutes
1	tt0111161	The Shawshank Redemption	1994	14
2	tt0068646	The Godfather	1972	17
3	tt0050083	12 Angry Men	1957	9
4	tt0071562	The Godfather Part II	1974	20
5	tt0108052	Schindler's List	1993	19
6	tt0167260	The Lord of the Rings: The Return of the King	2003	20
7	tt0468569	The Dark Knight	2008	15
8	tt0110912	Pulp Fiction	1994	15
9	tt15097216	Jai Bhim	2021	16
10	tt0060196	The Good,the Bad and the Ugly	1966	16
11	tt0109830	Forrest Gump	1994	14
12	tt0120737	The Lord of the Rings: The Fellowship of the Ring	2001	17
13	tt0137523	Fight Club	1999	13

- Finally, close the window showing the data, and then choose the menu item `Tools > Open SQL editor`. An area will appear where you can type SQL commands, and then press the blue arrow button to run them.

NOTE: You may need to drag and/or resize these windows.

This image shows the SQL editor pane, including the SQL query (in red box), the blue arrow button (in orange box), the output (in blue box) and the number of rows returned (in green box).



```
SELECT * FROM titles
LIMIT 10;
```

Total rows loaded: 10

	title_id	title	premiered	runtime_minutes	genres
1	tt0111161	The Shawshank Redemption	1994	142	Drama
2	tt0068646	The Godfather	1972	175	Crime,Drama
3	tt0050083	12 Angry Men	1957	96	Crime,Drama
4	tt0071562	The Godfather Part II	1974	202	Crime,Drama
5	tt0108052	Schindler's List	1993	195	Biography,Drama,History

```
[16:03:08]  Query finished in 0.002 second(s).
```

Before answering the following questions, you should use SQLiteStudio to examine all of the tables in the database and work out what all the columns refer to.

For example, the table `titles` has five columns,

- `title_id`

  The primary key for the table is an artificial identifier used to uniquely identify a movie title. For example, the identifier for the 2000 movie titled "In the Mood For Love" is `tt0118694`.

- `title`

  The actual name of the movie — notice that this is *not a key* because there are lots of movies that have the same name as other movies. (As it happens, there are no duplicated names in this small set of 250 movies, but in the entire database there are many duplicates.)

  I will be using a larger subset of the IMDB to test your code and this *will contain* some movies with the same names. Be warned that you *cannot rely on* the values in this column being unique.

- `premiered`

  The year in which the movie was released.

- `runtime_minutes`

  The length of the movie in minutes.

- `genres`

  This is a text string that listing the genres of this movie. For example, "In the Mood for Love" is a dramatic and romantic movie, so its list of genres appears as `Drama,Romance`.

  Notice that this is really a *multi-valued* variable, and therefore this database is not in first normal form (1NF).

There is a table called `people` which lists all the people involved in any of these movies — the primary key for this table is another artificial identifier. People can be involved in a movie either as a cast member (actor) or as part of the crew (director etc). This information is contained in the tables `castmembers` and `crewmembers`. For example, the first few rows of `castmembers` are

```
title_id  person_id category
tt0015864 nm0675356 composer
tt0015864 nm0005906 cinematographer
```

which tells us things like "The person identified by `nm0675356` was the *composer* for the movie identified by `tt0015864`." If we want to find the actual name of the movie or person, we will have to *join* those tables.

Lab Questions

This lab should be completed by the end of Teaching Week 4, where "end of Week X" means "by 5pm on Sunday at the end of Week X". This week's submission comprises *four files* from this lab.
You should submit four files via `cssubmit` named `A4.sql`, `A5.sql`, `A6.sql` and `A7.sql` with each file containing a single query (starting with `SELECT` and ending with a semi-colon).
When run, the query should produce *exactly* the output specified in the question, nothing more and nothing less. Try to complete the questions *no later than* the end of your weekly lab session, possibly with the assistance of the lab facilitator.

All unanswered questions on `help1402` are cleared (by me) on Friday evening, approximately 5pm, but questions posted after that time may not be viewed until Monday. (Maira sometimes answers a few questions over the weekend, but you should not rely on that.)

As usual, you should submit *plain text* files only with the correct names, and each file should contain just the SQL query starting with `SELECT` and ending with a semicolon. No RTF, no dot commands, no SQLite output, no student numbers, and no terminal prompts from SQLite or the shell (Terminal/Powershell).

Your commands will be tested against a database with the same schema as `imdb_top_250` but not necessarily the same data. (For example, I might create a database `imdb_top_1000` just to make sure that your commands work in general when there are multiple movies with the same title.)

You *may not rely on* any features of the data that are not guaranteed by the schema. For example, you cannot rely on movies having unique *names*, but you can rely on them having a unique `title_id` (because the entire purpose of the column `title_id` is to act as a unique identifier).

Some questions will indicate *precisely which columns* the question is asking for, for example "Give the *name* and *date of birth* of ...". In this case, the output should have two columns, one for the name and one for the date of birth, and these two columns *should occur in the same order* as they are named in the question.

Other questions might just say "all the details" or "all the data" or something like that, in which case you should just use `SELECT *`.

Sometimes the question gives some partial output as an illustration. Your code should produce these sample rows *somewhere* in the table that it outputs, but not necessarily in the same order.

Question 1. Write a SQL query to find all the details (as stored in `people`) of the person identified by `nm0675356`.

```sql
SELECT *
FROM people
WHERE person_id = 'nm0675356';
```

Question 2. Write a SQL query to list all the details of the castmembers in the movie with `title_id` equal to `tt0172495`.

(Your list should have four rows.)

```
SELECT *
FROM castmembers
WHERE title_id = 'tt0172495';
```

Question 3. **Submission: `A4.sql`** Write a single SQL query to list, for each cast member of the movie with `title_id` equal to `tt0172495`, the cast member's *real name* and the *character(s)* they played in that movie. You will need to join the tables `people` and `castmembers` in order to do this. (The character or characters played by a cast member are listed in a single text string in the `characters` column of the `castmembers` table. You should just use the values in this column unaltered.)

Make sure your query produces *just two columns*, with the real name first and character(s) second.

If your query is correct, then one of the rows will be

```
Russell Crowe ['Maximus']
```

indicating that the famous Australian actor played the character *Maximus* in the movie. (He won the 2001 Best Actor Oscar for this performance, although this information is not in the database.)

```
SELECT name, characters
FROM people JOIN castmembers
  USING (person_id)
WHERE title_id = 'tt0172495';
```

Question 4. Write a SQL query to list the *title* and *rating* of all the movies with an IMDB rating strictly greater than 8.5. (Here "strictly greater than" means that the list should *not contain* movies with rating of exactly 8.5.)

(There are 33 such movies.)

```
SELECT title, rating
FROM titles JOIN ratings
USING (title_id)
WHERE rating > 8.5;
```

Learning Interlude

In principle, a table is just a *set* of rows in no particular order and so, depending on the query optimizer, SQL can return the output rows in any order it wants. However it is possible to force SQL to sort the rows into any desired order using `ORDER BY` and then the name of the field.

For example, if we wanted to the list the movies in chronological order then we could do

```sql
SELECT *
FROM titles
ORDER BY premiered;
```

The default ordering is *ascending order*, but you can reverse this if desired using `DESC` for "descending order". If you like your code to be very readable, you can also specify `ASC` for "ascending order", but this makes no difference to SQL.

```sql
SELECT
FROM titles
ORDER BY premiered DESC;
```

Question 5. Submission: `A5.sql` Write a single SQL query that lists the *title*, *rating* and *number of votes* for all movies with more than 1000000 votes ordered from highest-rated to lowest-rated

```sql
SELECT title, rating, votes
FROM titles JOIN ratings
USING (title_id)
WHERE votes > 1000000
ORDER BY rating DESC;
```

Your answer should have 63 rows, with "The Shawshank Redemption" at the top of the list and four movies including "Gone Girl" at the bottom.

Question 6. Write a single SQL query to list the *title* and the *name(s) of the director(s)* for each of the movies. (A movie may have more than one director, in which case there will be multiple rows for that movie, one per director.) The column `category` in the table `crewmembers` indicates the role of each of the listed crew.

Start designing your query by first working out *which tables* contain the data that you require, and how these tables should be joined. Getting the *initial join* correct is fundamental to writing a correct query.

If you need to join *three* tables, then you will need *two* `JOIN` statements.

```
SELECT title, name
FROM titles JOIN crewmembers USING (title_id)
  JOIN people USING (person_id)
  WHERE category = 'director';
```

You may be surprised that "The Wizard of Oz" has so many directors, but this is just what is contained in the IMDB database.

Learning Interlude

SQL has a number of *summary functions* that produce a single value from a *number of rows*. Some of the simplest examples are `COUNT`, `MAX` and `MIN`.

For today we will just learn the very simplest use of these commands, but soon we will see much more sophisticated ways to use them.

The SQL expression `COUNT(*)` can be used to *count* the number of rows returned by any query. The output is just the *number of rows* of the table created by the query (and not the rows themselves).

```
SELECT COUNT(*)
FROM titles
WHERE premiered = 2000;
```

produces the output `6`.

The SQL expression `MAX` gives the maximum value of the named column over all of the rows that are processed, and `MIN` gives the minimum value.

For example,

```
SELECT MIN(premiered) AS earliest,
       MAX(premiered) AS latest
       FROM titles;
```

creates a table with one row and two columns called `earliest` and `latest`.

Although SQLite does not remove duplicate rows from tables by default (because it is computationally expensive to check every pair for rows to see if two rows are the same), you can force the `SELECT` statement to remove duplicates by using `SELECT DISTINCT`.

Question 7. Write a single SQL query that lists the values, *once each*, that occur as the *rating* of a movie. For example, the output should contain the value `8.2` because there are some movies in the IMDB Top 250 that have a rating of `8.2`, but it should not contain `7.8` because this value does not occur.

```
SELECT DISTINCT rating
FROM ratings;
```

Question 8. Write a single SQL query that returns the *number of actors* in the movie with `title_id` `tt0167260`. (Of course, this means the number of actors that are listed in the database, which only lists the main actors.)

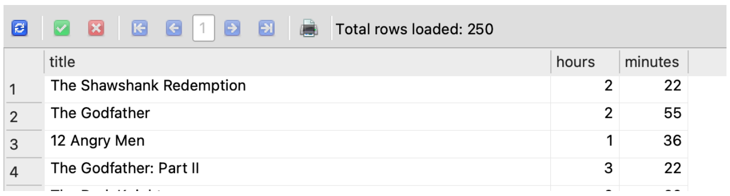
```
SELECT COUNT(*)
FROM castmembers
WHERE title_id = 'tt0167260';
```

Question 9. Submission: `A6.sql` Write a single query to find the *lengths* of the shortest movie, longest movie and the *average length* of a movie, outputting the result in a table with three columns called `shortest`, `longest`, `average`.

I haven't told you how to find the average of the values in a column, so you will have to head over to `sqlite.org` and *search* for the relevant function. The phrase "built-in aggregate functions" might help you find the right thing. Once you find it, the function behaves just like `MAX`, `MIN` etc.

```
SELECT MIN(runtime_minutes) as shortest,
         MAX(runtime_minutes) as longest,
         AVG(runtime_minutes) as average
FROM titles;
```

Question 10. Write a single SQL query to list the *title* of each movie, together with its running time split into *hours* and *minutes*. For example, *The Shawshank Redemption* is 142 minutes long, which is 2 hours and 22 minutes. The output table should have columns named `title`, `hours` and `minutes` and the first few rows should look as follows:


Total rows loaded: 250

	title	hours	minutes
1	The Shawshank Redemption	2	22
2	The Godfather	2	55
3	12 Angry Men	1	36
4	The Godfather: Part II	3	22

You will need to investigate how the *division* / and *modulus* % operators work in SQLite. Unfortunately the documentation in SQLite for all of the operators is very compressed or even non-existent.

However, these two operators are the same in SQLite as in Python, and there is a nice short tutorial on this located at

`https://www.pythontutorial.net/advanced-python/python-floor-division/`

```sql
SELECT title,
       runtime_minutes / 60 as hours,
       runtime_minutes % 60 as minutes
FROM titles;
```

QUESTION 11. Write a single SQL query to list the *title*, *runtime* and *rating* for every movie whose rating is above 8 and whose runtime is less than two hours. The output table should have just three columns called MovieName, RuntimeMinutes and Rating. (Note that "above 8" implies values that are greater than 8, but it does not include the value 8 itself. In other words, values like 8.1, 8.2, 9, and so on.)

```sql
SELECT title AS MovieName, runtime_minutes AS RuntimeMinutes, rating AS Rating
FROM titles JOIN ratings USING (title_id)
WHERE rating > 8 AND runtime_minutes < 120;
```

QUESTION 12. Submission: `A7.sql` Write a single SQL query to produce a list of the *movie titles* and *cast names* for every movie that is rated *at least* 9 in the database. (Note that "at least 9" means to include 9.) The output table should have *just two columns* called `title` and `name`. For every highly-rated movie, there should be one row per cast member, with the row listing the title of the movie and the name of the cast member.

Remember that if you need to join N tables, then you will need $N - 1$ `JOIN` statements.

The famous actor *Henry Fonda* was a cast member in the movie *12 Angry Men*, which has a rating of 9.0, and so your query should at least produce the row

```
12 Angry Men    Henry Fonda
```

amongst a number of other rows.

```sql
SELECT title, name
FROM ratings JOIN titles USING (title_id)
          JOIN castmembers USING (title_id)
          JOIN people USING (person_id)
WHERE rating >= 9;
```