Database Technologies for Business Analytics
BEM2040
Practice – Week 3

The following instructions can be followed by using university computers, a virtual desktop or your personal computer, if the software has been installed.

1. Download the file Week3.zip. We will be using:
   - films_subset.db
     The tables in the database are as follows:

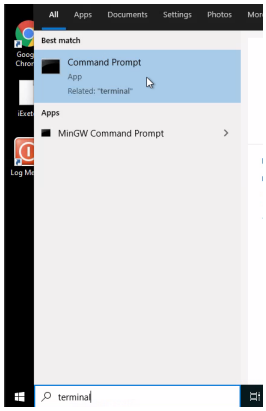
```
ACTOR:
 actor_id,
 actor_name,
 actor_year_born,
 actor_year_dead

FILM:
 film_id,
 film_title,
 film_year_start,
 film_year_end,
 film_major_genre

GENRE:
 genre_id,
 genre_name

RATING:
 rating_film_id,
 rating_average,
 rating_num_votes
```

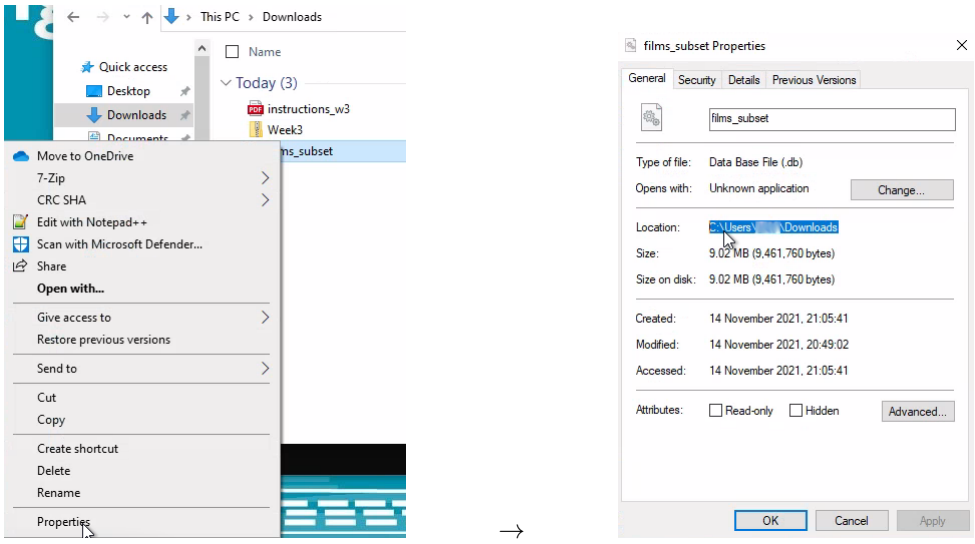
2. Decompress the files in your downloads folder.
3. We will be working with the command line tool for SQLite. To open it, type "terminal" in the windows starting search box:



The terminal would look like this, from a university computer (instead of ab000 you should see you user name):

```
Microsoft Windows [Version 10.0.19042.1288]
(c) Microsoft Corporation. All rights reserved.
d:\ab000.ISAD>
```

4. Get the location of the database file: right-click the file in the explorer, select *Properties* and copy the text in *Location*:



5. Open the database with the sqlite3 command. Type *sqlite3* a blank space and then right click. The folder you copied should appear. Then complete it so that the entire line looks similar to this:

```
d:\ab000.ISAD>sqlite3 C:\Users\ab000\Downloads\films_subset.db
```

Press enter and you should be working on the database:

```
SQLite version 3.27.2 2019-02-25 16:06:06
Enter ".help" for usage hints.
sqlite>
```

6. Let's see the content of table *film*. Write the instruction

```
select * from film;
```

in the prompt:

```
sqlite> select * from film;
```

The instruction has to end in semicolon (;). The result would be as follows (only the last rows shown here):

```
tt1362551|Look to Lockheed for Leadership|1940||Documentary
tt13640566|Goodbye Yesterday|1941||Drama
tt13642790|Radio City Matinee|1941|1946|Family
tt13649610|Television Premiere|1941||Family
tt13649618|Final show on the Dumont Network|1941||Family
```

7. Let's now obtain films for a specific year (1940). The instruction is:

```
select * from film where film_year_start=1940;
```

In the command line:

```
sqlite> select * from film where film_year_start=1940;
```

```
tt13376548|The Green Archer Exposed|1940||Action
tt13386366|La caza del puma|1940||Animation
tt1341199|Kedok ketawa|1940||Drama
tt13545878|Goebbels-Geburtstagsfilm - 29.10.1940|1940||Short
tt1362551|Look to Lockheed for Leadership|1940||Documentary
```

8. Now, let's look at table *actor*. The instruction

```
select * from actor;
```

gives an empty output, as there are no rows.

```
sqlite> select * from actor;
```

9. Let's add some rows:

```
insert into actor(actor_id, actor_name, actor_year_born, actor_year_dead)
values ("a1", "Timothée Chalamet", 1994, null);
```

```
sqlite> INSERT into actor(actor_id,actor_name,actor_year_born,actor_year_dead)
VALUES ("a1", "Timothée Chalamet", 1994, null);
```

```
INSERT into actor(actor_id,actor_name,actor_year_born,actor_year_dead)
VALUES ("a2", "Rebecca Ferguson", 1983, null);
```

```
sqlite> INSERT into actor(actor_id,actor_name,actor_year_born,actor_year_dead)
VALUES ("a2", "Rebecca Ferguson", 1983, null);
```

```
INSERT into actor(actor_id,actor_name,actor_year_born,actor_year_dead)
VALUES ("a3", "Greta Garbo", 1905, 1990);
```

```
sqlite> insert into actor(actor_id,actor_name,actor_year_born,actor_year_dead)
values ("a3", "Greta Garbo", 1905, 1990);
```

10. Check the table *actor* after the inserts:

```
sqlite>  select * from actor;
a1|Timothée Chalamet|1995|
a2|Rebecca Ferguson|1983|
a3|Greta Garbo|1905|1990
```

11. Now lets add headers to the output and activate the column mode to make the output more readable. Use the commands:

```
.header on
.mode column
```

```
sqlite> .header on
sqlite> .mode column
```

We can now see the column names (actor_id, actor_name, etc.) in the output when querying:

```
select * from actor;
```

```
sqlite> select * from actor;
actor_id  actor_name         actor_year_born  actor_year_dead
--------  -----------------  ---------------  ---------------
a1        Timothée Chalamet  1994
a2        Rebecca Ferguson   1983
a3        Greta Garbo        1905             1990
```

12. We can update one row, based on the id:

```
update actor set actor_year_born = 1995 where actor_id = "a1";
```

```
sqlite> update actor set actor_year_born = 1995 where actor_id = "a1";
```

And check the result:

```
sqlite> select * from actor;
actor_id  actor_name         actor_year_born  actor_year_dead
--------  -----------------  ---------------  ---------------
a1        Timothée Chalamet  1995
a2        Rebecca Ferguson   1983
a3        Greta Garbo        1905             1990
```

13. We can order the result of a query in ascending or descending directions.

For example, by name:

```
select * from actor order by actor_name;
```

```
sqlite> select * from actor order by actor_name;
actor_id  actor_name         actor_year_born  actor_year_dead
--------  -----------------  ---------------  ---------------
a3        Greta Garbo        1905             1990
a2        Rebecca Ferguson   1983
a1        Timothée Chalamet  1995
```

Or by year born (this time let's try in descending order):

```
select * from actor order by actor_year_born desc;
```

```
sqlite> select * from actor order by actor_year_born desc;
actor_id  actor_name         actor_year_born  actor_year_dead
--------  -----------------  ---------------  ---------------
a1        Timothée Chalamet  1995
a2        Rebecca Ferguson   1983
a3        Greta Garbo        1905             1990
```

We can substitute * in a *select* query for specific columns:

```
select actor_name, actor_year_born from actor order by actor_year_born desc;
```

```
sqlite> select actor_name, actor_year_born from actor order by actor_year_born desc;
actor_name         actor_year_born
-----------------  ---------------
Timothée Chalamet  1995
Rebecca Ferguson   1983
Greta Garbo        1905
```

14. We can use the following operators:
<,
>,
<=,
>=,
=,
<>,
IN,
LIKE,
IS NULL

Try the following queries:

- Actors born in 1985 or after:

```
select * from actor where actor_year_born >= 1985;
```

- Films where the title as the sequence of letters "sea":

```
select * from film where film_title like '%sea%';
```

- Films where the title ends in "country":

```
select * from film where film_title like '%country';
```

15. Can you find the records in table *rating*, with an average greater than 8?

16. We can delete a single row if we can identify it uniquely. Let's suppose we want to eliminate the movie, "Perils of the Jungle".that has the id "tt1199512". We can look at the row before deleting:

```
select * from film where film_id = 'tt1199512';
```

We then issue the following instruction to delete it:

```
delete from film where film_id = 'tt1199512';
```

```
sqlite> delete from film where film_id = 'tt1199512';
```

17. We can delete several rows. The following deletes all movies from 1941:

```
delete from film where film_year_start = 1941;
```

```
sqlite> delete from film where film_year_start = 1941;
```

18. We can update rows much in the same whay than we delete. We can update a single row as we have seen before. Or we can update several rows.

It is common when migrating data to a database that some rows contain unwanted characters that might cause problems. If we look at the IMDB database, sometimes, the major gengre for a film is \N, which might mean there is no information bout the genre for a specific movie.

Let's count the number of movies that have this problem:

```
select count(*) from film where film_major_genre = '\N';
```

A much better formating for cases of missing data is to use the *null* value. Let's change all those movies and set *film_major_genre* to *null*

```
update film set film_major_genre = null where film_major_genre = '\N';
```