**A (short) tutorial on SQLite + DB Browser**

**CMPUT 291 Fall/2021, Mario Nascimento ©**

The main goal of this (short) tutorial is to introduce students to a GUI to use SQLite in their own computers. There are a few such GUIs but we will focus on DB Browser. Note that the lab computers do not have such a GUI. In the lab you will need to use SQLite’s command line interface ([here is a lab-based tutorial for that](https://drive.google.com/file/d/1OKQzUpi15v2_c4o-oy2Y3i4Vuzk6H3lt/view?usp=sharing)).

*Please, do keep in mind that all assignments and the like will be marked on lab computers. While you are welcome to use your own computers to develop your submissions, you are encouraged to test them in the lab computers before final submission.*

Before starting make sure you have both [SQLite](https://www.sqlite.org/) and [DB Browser](https://sqlitebrowser.org/) installed on your computers.[[1]](#footnote-0)

**TABLES[[2]](#footnote-1):**

Let us first create some test tables using DB Browser GUI.

We are now going to use the following tables, though you can certainly create and populate your own tables. Using the CSV files provided you can import them onto a SQLite database.

Students(sid, name) ([csv file](https://drive.google.com/file/d/18T7qwRyDK7XOP_r64JDoe9_x5mrt-Zfp/view?usp=sharing))

Profs(pid, name) ([csv file](https://drive.google.com/file/d/1W5bBW7tZl-rf4FRVLBGmaArq9x_JHecJ/view?usp=sharing))

Courses(cid, dept) ([csv file](https://drive.google.com/file/d/1VNA2P7iZucvrIccTUMLoyOcCwV_aSkgv/view?usp=sharing))

Takes(sid, cid, grade) ([csv file](https://drive.google.com/file/d/1TMy9sL6BL0Wg4kMlCbMtSWdNNsRPGV3a/view?usp=sharing))

Teaches(pid, cid) ([csv file](https://drive.google.com/file/d/1NiqOgf7DjY2g0oWDZGBkgNB9ZGdRtQ4B/view?usp=sharing))

**QUERIES:**

The CSV files above were created using randomly generated data, without any concern regarding primary keys. Once the csv files are imported on a SQLite database, we can (probably should) define them.

Tables Teaches has a problem wrt its primary key. How to find which “key” is duplicated?

SELECT pid, cid

FROM Teaches

GROUP BY pid, cid

HAVING COUNT(\*) > 1

(Changing pid from 7 to 6 in row 5 fixes it and allows a foreign key to be properly defined.)

Now let’s define foreign keys. The way we generated this dataset guarantees that the (implicit) foreign keys in all relations are enforced, but … how could we double-check that otherwise, e.g., that Takes.sid is a valid foreign key referring to Students.sid?

SELECT T.sid

FROM Takes T

WHERE NOT EXISTS (

SELECT \*

FROM Students S

WHERE S.id = T.sid )

The S.id typo is there on purpose in order to trigger an error message.

If we switch the inner and outer queries in the query above, i.e.:

SELECT S.sid

FROM Students S

WHERE NOT EXISTS (

SELECT \*

FROM Takes T

WHERE S.sid = T.sid )

What do we learn from the query’s result? Can we write the same query not using EXISTS?

How do we find the attendance of all courses, from most to least attended?

SELECT cid, COUNT(\*)

FROM Takes

GROUP BY cid

ORDER BY COUNT(\*) DESC

Can you find the courses that no student has attended?

What does the following query return?

SELECT P.name, T.cid

FROM Profs P LEFT OUTER JOIN Teaches T ON (P.pid = T.pid)

What if I wanted to know the names of all professors and number of courses they teach (using null as a proxy for 0 courses)?

SELECT P.name, Load.nCourses

FROM Profs P LEFT OUTER JOIN (

SELECT pid, COUNT (\*) as nCourses

FROM Teaches

GROUP BY pid ) Load ON (P.pid = Load.pid)

Find the name and grade of the student(s) who has(have) the highest grade over all courses.

SELECT S.name, MAX (grade) as maxGrade

FROM Takes T, Students S

WHERE T.sid = S.sid

GROUP BY T.sid, S.name

ORDER BY maxGrade DESC

LIMIT 1

The query above is misleading, as the LIMIT 1 clause “hides” any ties (and one cannot know which of the ties is actually output).

SELECT sid

FROM Takes

WHERE grade >= (SELECT MAX(grade)

FROM Takes)

**EXERCISE:**

Create your own database, say named PropertiesYEG, and using the CSV files freely available as [open data made by the City of Edmonton](https://data.edmonton.ca/), create two tables using data for [Property Information](https://data.edmonton.ca/City-Administration/Property-Information-Current-Calendar-Year-/dkk9-cj3x) and [Property Assessment](https://data.edmonton.ca/City-Administration/Property-Assessment-Data-Current-Calendar-Year-/q7d6-ambg), respectively. (Note: they will produce reasonably sized tables with missing information and some attribute names might be “adjusted,” e.g., spaces removed, etc.)

Think of a few queries that could be interesting and try them out. For instance:

* In each neighbourhood, what is the most expensive property?
* What is the average price of properties built in the last, say, 20 years?
* What is the average price of the properties for each neighbourhood in a given ward?
* Which street is the one with more and less properties?
* Considering that the “size cost” of property is the price of a property divided by its lot size, in which neighbourhoods are the properties with the largest and the smallest size cost?
* Given a location (latitude and longitude) what is the closest property to it? (You will likely need to look into more information in order to compute distance using latitude and longitude.
* Be creative!

1. Note that given the nature of open source applications, the University of Alberta cannot be responsible for any direct or indirect consequences of their use. [↑](#footnote-ref-0)
2. The names of people were generated using [this website](https://www.randomlists.com/middle-name-generator). [↑](#footnote-ref-1)