SQL on Your Personal Computer

- While learning the basics of SQL, we will make use of two tools on your personal computer: SQLiteStudio and IPython SQL Magic.

SQLite:

- SQLite is a small, fast, self-contained, high-reliability, full-featured, SQL database engine. It is the most used database engine in the world—it is built into all mobile phones and most computers. It comes bundled inside countless other applications that people use every day.

- Unlike most other SQL databases, SQLite does not have a separate server process. SQLite reads and writes directly to ordinary disk files. A complete SQL database is contained in a single file.

- SQLite is not directly comparable to client/server SQL database engines such as MySQL, Oracle, or PostgreSQL. You can read more about SQLite here.

SQLiteStudio:

- We will make use of SQLiteStudio in this course, which is a desktop application for managing SQLite databases. It is free, cross-platform, and provides some of the most important features to work with SQLite databases, including importing and exporting data in various formats.

- SQLiteStudio is not the same as SQLite. SQLite is a database, while SQLiteStudio is an application to manage such databases.

- You can download SQLiteStudio by clicking on the **Download** button of the SQLiteStudio homepage:

  https://sqlitestudio.pl/

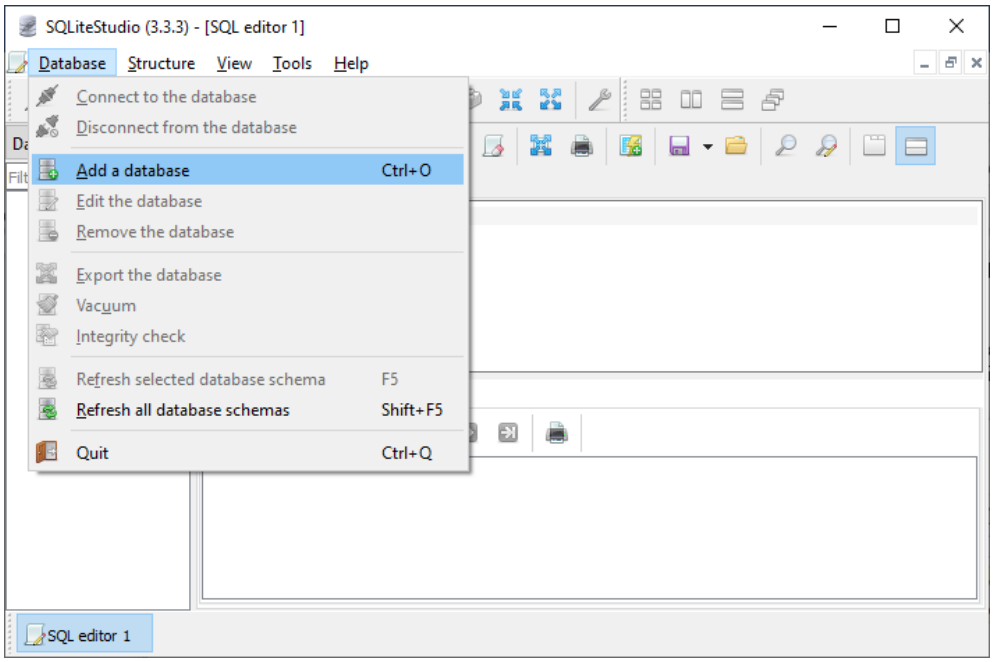
- **On MacOS**, the downloaded file should be **sqlitestudio-3.3.3.dmg**. This is a *container* for the app SQLiteStudio. There is no installation necessary—you just need to double-click the DMG file to open and mount it to your Mac. Note that if you get a "Cannot be opened because the developer cannot be verified" error, then please try the solutions listed on this web page:

  Lifewire Fix for Developer Cannot Be Verified

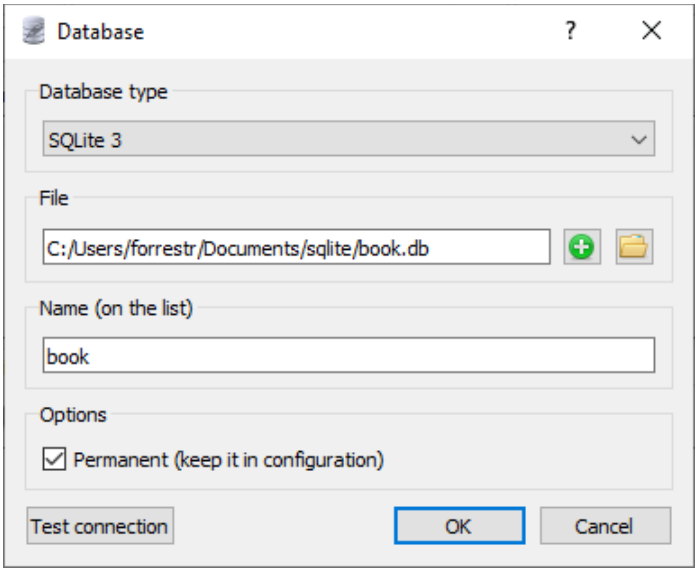
- **On Windows**, the downloaded file should be **sqlitestudio-3.3.3.zip**. Note that there is no installer for this application. Simply unzip the file to an appropriate folder and then double click on the **SQLiteStudio.exe** file to run the application.

- To get started, download the `book.db` SQLite database file from Teams and save it to an appropriate directory.

- Locate and run the SQLiteStudio application. From the **Database** menu, select **Add a database**.

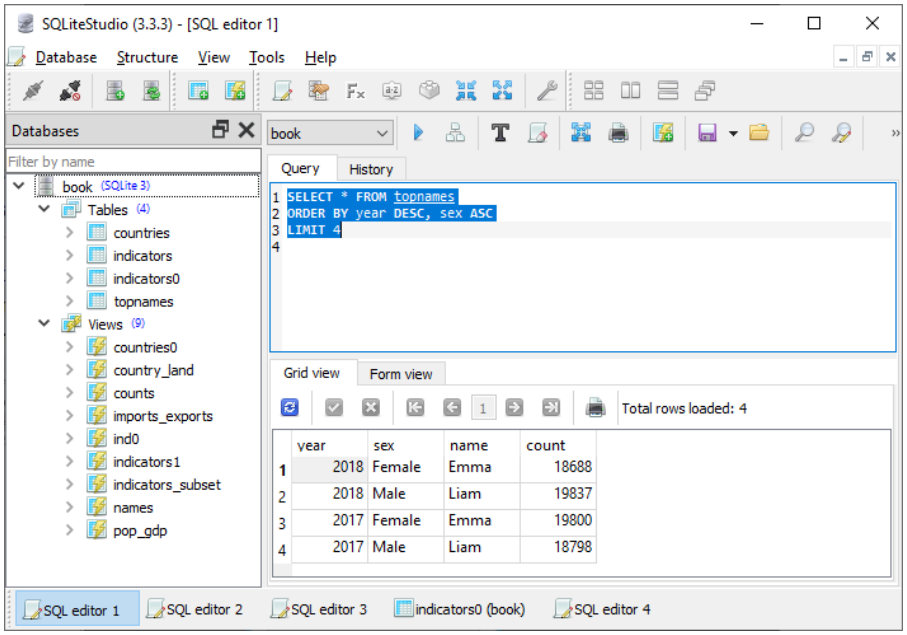


- This will open a dialog box. Click on the folder icon and navigate to the location of the `book.db` database file and select it. Then click **OK**.

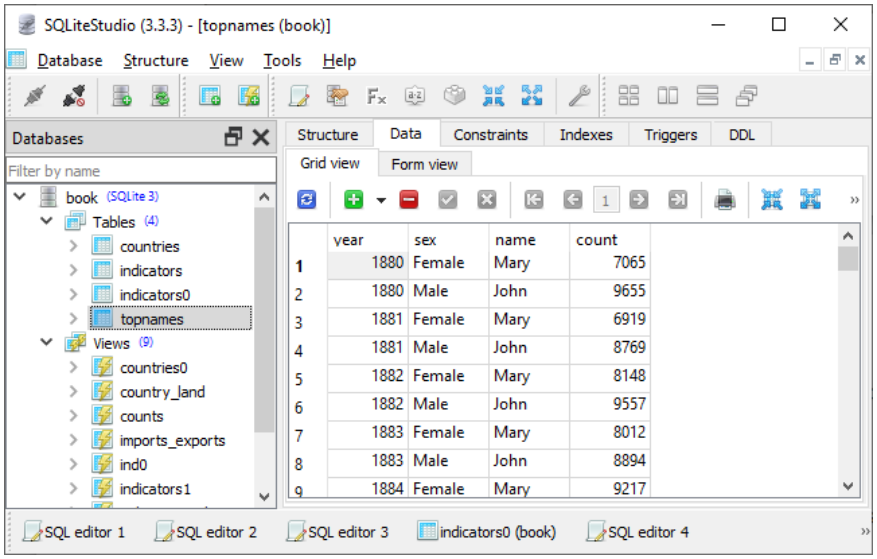


- If the SQL editor is not visible, then do the following: from the **Tools** menu, select **Open SQL editor**. From the editor, you can enter SQL commands, and when you are ready you can click the run button (standard "play" icon).

- For example, here is the output from the SQL code on page 324 of out textbook:



- To view the data contained in a particular table, click on the table name, such as **topnames** from the **book.db** database, and then click the **Data** tab:



- Below are a couple of nice tutorials on how to use SQLiteStudio:

  https://executecommands.com/how-touse-sqlite-studio-manage-sqlite-database/

  http://cswire.blogspot.com/2017/11/sqlitestudio-tutorial-basic-database.html

IPython SQL Magic:

- The IPython SQL magic extension makes it possible to write SQL queries directly into code cells of Jupyter notebooks, as well as read the results straight into a Pandas data frame.

- *Magic commands* are a set of convenient functions in Jupyter notebooks that are designed to solve some of the common problems in standard data analysis.

- To use Python magic with SQL, we need to install the `ipython-sql` library first. You can install it by running this code from **Anaconda Command Prompt** on Windows or **Terminal** on macOS:

```
conda install -c conda-forge ipython-sql
```

- Jupyter notebooks use "`%`" sign before the command to indicate that the function will be a "magic function." This "magic" is a group of pre-defined functions contained in the IDE's kernel that allows us to execute provided commands.

Loading the External SQL Module

- To load the SQL module using Python magic, run the following command in a code cell of a Jupyter notebook:

```
%load_ext sql
```

Connecting to an Existing Database

- To open an existing SQLite database, such as the `book.db` database, we could use the absolute path to the database like this (note that there are three backward slashes):

```
%sql sqlite:///C:\Users\forrestr\Documents\sqlite\book.db
```

- Alternately, we could give the relative path. For example, if the `book.db` file is in the same directory as the Jupyter notebook we could load it as follows:

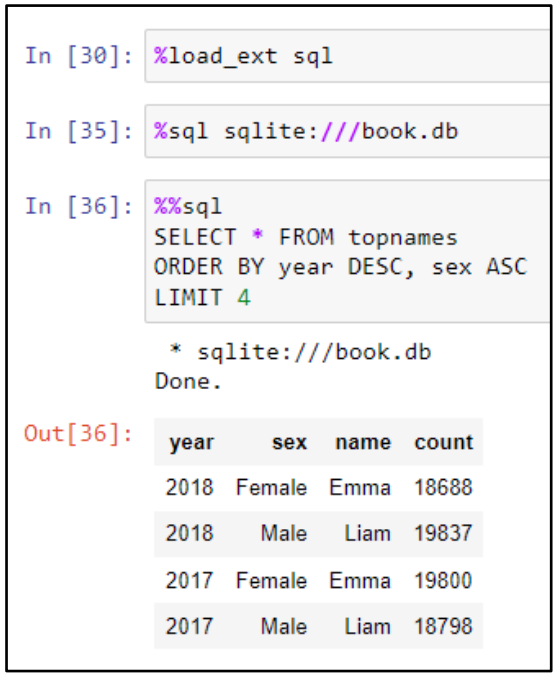
```
%sql sqlite:///book.db
```

- Note that if SQL magic cannot find the database, then it will silently create an empty database (without throwing an error).

SQL Cell Magic

- *Cell magic* applies to the whole cell. It is utilized by double percentage signs (%). That is, all you have to type is `%%sql` and the rest of the cell will be treated as an SQL script.

- We demonstrate in a Jupyter notebook below with some code from page 322 of the textbook:



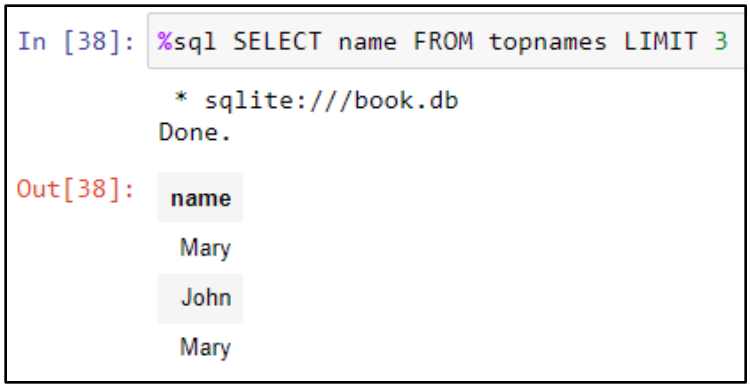
```
In [30]: %load_ext sql

In [35]: %sql sqlite:///book.db

In [36]: %%sql
         SELECT * FROM topnames
         ORDER BY year DESC, sex ASC
         LIMIT 4
          * sqlite:///book.db
         Done.
```

Out[36]:

year	sex	name	count
2018	Female	Emma	18688
2018	Male	Liam	19837
2017	Female	Emma	19800
2017	Male	Liam	18798

SQL Line Magic

- *Line magic* will turn only one line of your code cell to an SQL script. All you have to do is type `%sql` before your SQL query and the rest of the line will be interpreted as an SQL script. We demonstrate in a Jupyter notebook below:



```
In [38]: %sql SELECT name FROM topnames LIMIT 3
          * sqlite:///book.db
         Done.
```

Out[38]:

name
Mary
John
Mary

Assigning Results of an SQL Query to a Pandas Data Frame

- We can assign the results of an SQL query to a Python variable by using an assignment operator before the SQL magic line operation. For example, consider the following session:

```
In [47]: result = %sql SELECT year, sex, name FROM topnames LIMIT 6

         print(type(result))
         print()
         print(result)
         resultDF = pd.DataFrame(result)
         print()
         print(resultDF)
```

```
 * sqlite:///book.db
Done.
<class 'sql.run.ResultSet'>

+------+--------+------+
| year |  sex   | name |
+------+--------+------+
| 1880 | Female | Mary |
| 1880 |  Male  | John |
| 1881 | Female | Mary |
| 1881 |  Male  | John |
| 1882 | Female | Mary |
| 1882 |  Male  | John |
+------+--------+------+

      0       1     2
0  1880  Female  Mary
1  1880    Male  John
2  1881  Female  Mary
3  1881    Male  John
4  1882  Female  Mary
5  1882    Male  John
```

The result is a `sql.run.ResultSet`. You can easily convert it to a Pandas data frame by passing it into the `pd.DataFrame()` function.

- We can use the << operator to assign the results of a multi-line SQL query to a Python variable as in the example below.

```
In [48]: %%sql
         result << SELECT * FROM topnames
         ORDER BY sex, count DESC
         LIMIT 8
```

```
 * sqlite:///book.db
Done.
Returning data to local variable result
```

```
In [50]: resultDF = pd.DataFrame(result)
         print(resultDF)
```

```
      0       1      2      3
0  1947  Female  Linda  99689
1  1948  Female  Linda  96211
2  1949  Female  Linda  91016
3  1950  Female  Linda  80431
4  1921  Female   Mary  73985
5  1951  Female  Linda  73978
6  1924  Female   Mary  73534
7  1922  Female   Mary  72173
```