Lab 1: Getting Started with SQLite

# 1. Overview

This lab introduces how to use SQLite and operate a file-based database, which will be used later in other labs. After finishing this lab, you should:

* Understand how to use SQLite
* Understand basic database operations (such as creating and querying a table) with SQLite

# 2. Introduction

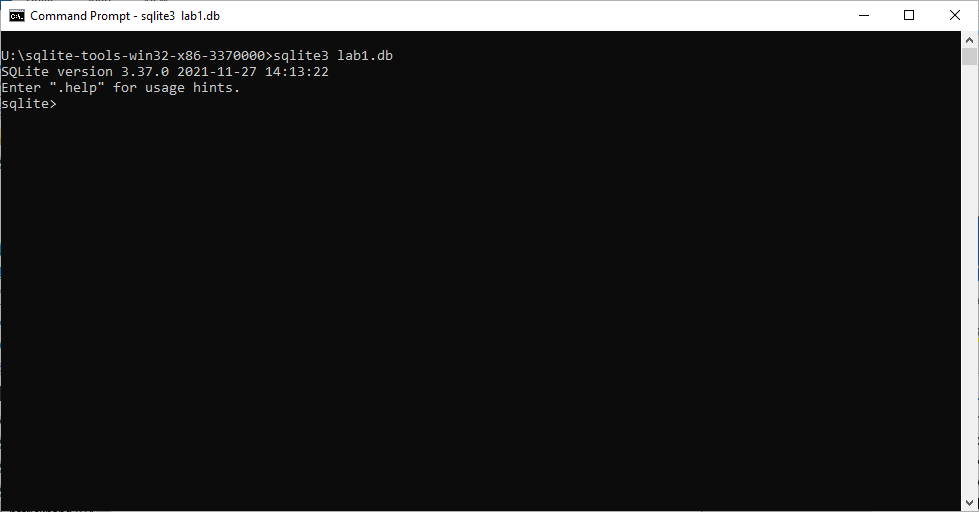
There are a wide range of database systems that we can use, including MySQL, Oracle, Microsoft SQL Server and PostgreSQL. Most of these databases are designed to be running all the time and run as a server process. What if we don’t want or need the database to be running all the time? Or what if we are only storing data for a single user in a single application?

SQLite is a very lightweight database library written in C that allows SQL queries to run with a local data file. In programming, SQLite is accessible from most programming languages (e.g., Python, PHP, Java, etc.). The source code for SQLite is in the public domain and so it is viewable and modifiable.

You can learn more about SQLite at their website: <https://sqlite.org/index.html>

# 3. Setup

Perform the steps below using the lab computer:

1. Download the sqlite3.zip file provided on Canvas
2. Extract the zip file to your own U: drive. Note that the lab files of this module **MUST** be stored in **U:** to avoid data loss.
3. Open Command Prompt, change the current directory to the new directory extracted from the zip file (using the cd command)
4. Type sqlite3 lab1.db, SQLite will run and a new database file called lab1.db will be created in the current directory. The screen will look similar to the one below.  
   

If you are using your own Windows PC, you can also follow the above steps to setup a copy of SQLite on your computer.

## 3.1 For macOS Users

If you are using a MacBook, you can install SQLite with two steps:

1. Install Homebrew if you haven’t done so. Visit <https://brew.sh> and follow the instructions to install.
2. Open Terminal, run brew install sqlite3

# 4. Basic Usage of SQLite

The program we just ran to work with a database file is called the **SQLite Shell**. It accepts both operational commands and normal SQL statements. These inputs can be placed after the sqlite> prompt in the command line window.

There are several commands available from within the shell including:

* .databases – lists all the currently loaded databases
* .schema – lists the schema for the database(s)
* .tables – lists the names of all tables
* .read –executes SQL statements in the specified file
* .echo – repeats the commands inputted on the display
* .quit –quits the shell
* .help –lists all supported commands

Note that these commands are starting with a period.

# 5. Operating the Database

Before running the abovementioned commands, we need to create a table. Run the following statement with the Shell.

CREATE TABLE employee (

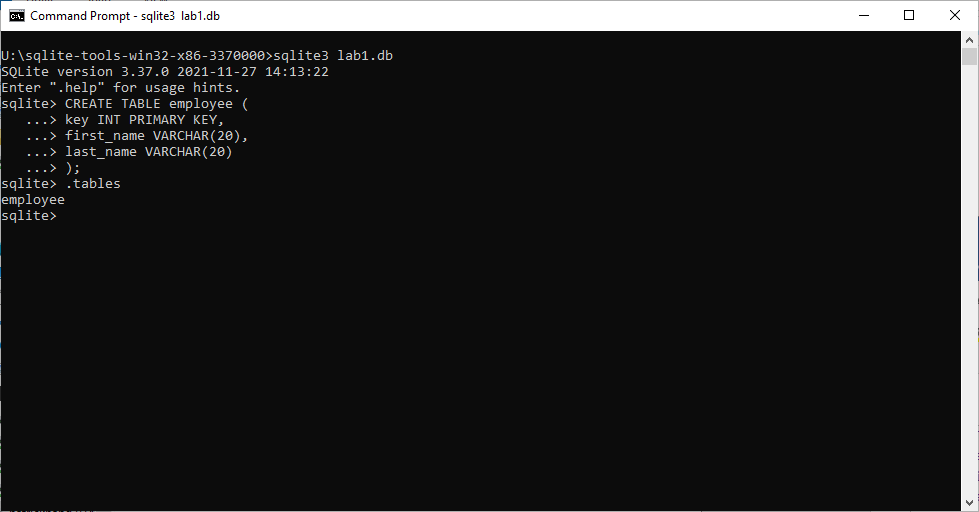
key INT PRIMARY KEY,

first\_name VARCHAR(20),

last\_name VARCHAR(20)

);

Now the table employee is created. Type .tables and you will see that SQLite display this new table on the screen. Run .schema employee to show the CREATE statement again, and you can use this command to show the structure of other tables in the future.



Next, copy and paste the following INSERT statements to add some data records into the database.

INSERT INTO employee VALUES (3,'Alice','Munro');

INSERT INTO employee VALUES (4,'Ned','Kelly');

INSERT INTO employee VALUES (5,'Andrew','Jackson');

INSERT INTO employee VALUES (6,'Clare','Underwood');

INSERT INTO employee VALUES (7,'Todd','Beamer');

INSERT INTO employee VALUES (8,'Nancy','Cartwright');

INSERT INTO employee VALUES (9,'Brier','Patch');

INSERT INTO employee VALUES (10,'Sarah','Fergusson');

INSERT INTO employee VALUES (11,'Sophie','Monk');

INSERT INTO employee VALUES (12,'Sanjay','Patel');

INSERT INTO employee VALUES (13,'Rita','Skeeter');

INSERT INTO employee VALUES (14,'Gigi','Montez');

INSERT INTO employee VALUES (15,'Maggie','Smith');

INSERT INTO employee VALUES (16,'Paul','Innit');

INSERT INTO employee VALUES (17,'James','Mason');

INSERT INTO employee VALUES (18,'Pat','Clarkson');

INSERT INTO employee VALUES (19,'Mark','Zhang');

Then, let’s try to retrieve some data from the database by running SELECT queries. Run SELECT \* FROM employee; (don’t forget the semi-colon) and you will see an output like:

3|Alice|Munro

4|Ned|Kelly

5|Andrew|Jackson

6|Clare|Underwood

7|Todd|Beamer

…

However, this output is not easy to read. To fix this issue, we need to change the output mode to **table** by running .mode table

After setting the mode, re-run the SELECT query again and it is now displayed as a table form, which is much easier for reading.

+-----+------------+------------+

| key | first\_name | last\_name |

+-----+------------+------------+

| 3 | Alice | Munro |

| 4 | Ned | Kelly |

| 5 | Andrew | Jackson |

| 6 | Clare | Underwood |

| 7 | Todd | Beamer |

…

It is advised that you should set the table mode before starting to work on queries in SQLite. **It is important to use the table mode for your assignments in this module.**

When you are finished, type .quit to exit SQLite.

# 6. Exercises

Try to write SELECT queries for the requirements below, run them with SQLite, and observe the results:

* Find out the names of employees with their keys greater than 10
* Find out the first name of the employee who has the last name Smith

# 7. Further Readings

* <https://sqlite.org/docs.html> – A comprehensive documentation of SQLite
* <https://www.youtube.com/watch?v=HQKwgk6XkIA> – SQLite Tutorial For Beginners - Make A Database In No Time
* <https://www.youtube.com/watch?v=girsuXz0yA8> – SQLite 3 Python Tutorial in 5 minutes - Creating Database, Tables and Querying