Εργαστήριο 4
Getting started with SQLite

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

SQLite is an embedded SQL database engine. 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 with multiple tables, indices, triggers, and views, is contained in a single disk file. The database file format is cross-platform - you can freely copy a database between 32-bit and 64-bit systems or between big-endian and little-endian architectures.

SQLite is a compact library. With all features enabled, the library size can be less than 750KiB, depending on the target platform and compiler optimization settings. There is a tradeoff between memory usage and speed. SQLite generally runs faster the more memory you give it. Nevertheless, performance is usually quite good even in low-memory environments. Depending on how it is used, SQLite can be faster than direct filesystem I/O.

In this tutorial you will install SQLite and SQLite Browser (GUI) on your Ubuntu VM. You will then create a database, read data from it, insert items and delete items.

Step 1 – Installing SQLite on Ubuntu VM

To install the SQLite command-line interface on Ubuntu, first update your package list:

```
sudo apt update
```

Now install SQLite:

```
sudo apt install sqlite3
```

To verify the installation, check the software's version:

```
sqlite3 --version
```

You will receive an output like this:

```
3.33.1 2020-08-14 13:23:32
3bfa9cc97da10598521b342961df8f5f68c7388fa117345eeb516eaa837balt
1
```

With SQLite installed, you are now ready to create a new database.

Step 2 — Creating a SQLite Database

In this step you will create a database containing different sharks and their attributes. To create the database, open your terminal and run this `sqlite3` command:

```
sqlite3 sharks.db
```

This will create a new database named `sharks` and open a command-line shell for interacting with the database. If the file `sharks.db` already exists, SQLite will open a connection to it; if it does not exist, SQLite will create it.

You will receive an output like this:

```
SQLite version 3.33.0 2020-08-14 13:23:32
Enter ".help" for usage hints.
```

Following this, your prompt will change. A new prefix, `sqlite>`, now appears:

```
sqlite>
```

If the file `sharks.db` does not already exist and if you exit the `sqlite` promote without running any queries the file `sharks.db` will not be created. To make sure that the file gets created, you could run an empty query by typing `;` and then pressing "Enter". That way you will make sure that the database file was actually created.

Special commands to sqlite3 (dot-commands)

Most of the time, sqlite3 just reads lines of input and passes them on to the SQLite library for execution. But input lines that begin with a dot (`"."`) are intercepted and interpreted by the sqlite3 program itself. These "dot commands" are typically used to change the output format of queries, or to execute certain prepackaged query statements. There were originally just a few dot commands, but over the years many new features have accumulated so that today there are over 60.

For a listing of the available dot commands, you can enter `".help"` with no arguments. Or enter `".help TOPIC"` for detailed information about TOPIC. The list of available dot-commands follows:

```
sqlite> .help
.archive ...             Manage SQL archives
.auth ON|OFF             Show authorizer callbacks
.backup ?DB? FILE        Backup DB (default "main") to FILE
.bail on|off             Stop after hitting an error.  Default OFF
.binary on|off           Turn binary output on or off.  Default OFF
.cd DIRECTORY            Change the working directory to DIRECTORY
.changes on|off          Show number of rows changed by SQL
.check GLOB              Fail if output since .testcase does not match
.clone NEWDB             Clone data into NEWDB from the existing database
.connection [close] [#]  Open or close an auxiliary database connection
.databases               List names and files of attached databases
.dbconfig ?op? ?val?     List or change sqlite3_db_config() options
.dbinfo ?DB?             Show status information about the database
.dump ?OBJECTS?          Render database content as SQL
.echo on|off             Turn command echo on or off
.eqp on|off|full|...     Enable or disable automatic EXPLAIN QUERY PLAN
.excel                   Display the output of next command in spreadsheet
.exit ?CODE?             Exit this program with return-code CODE
.expert                  EXPERIMENTAL. Suggest indexes for queries
.explain ?on|off|auto?   Change the EXPLAIN formatting mode.  Default: auto
.filectrl CMD ...        Run various sqlite3_file_control() operations
```

```
.fullschema ?--indent?   Show schema and the content of sqlite_stat tables
.headers on|off          Turn display of headers on or off
.help ?-all? ?PATTERN?   Show help text for PATTERN
.import FILE TABLE       Import data from FILE into TABLE
.imposter INDEX TABLE    Create imposter table TABLE on index INDEX
.indexes ?TABLE?         Show names of indexes
.limit ?LIMIT? ?VAL?     Display or change the value of an SQLITE_LIMIT
.lint OPTIONS            Report potential schema issues.
.load FILE ?ENTRY?       Load an extension library
.log FILE|off            Turn logging on or off.  FILE can be stderr/stdout
.mode MODE ?TABLE?       Set output mode
.nonce STRING           Disable safe mode for one command if the nonce matches
.nullvalue STRING        Use STRING in place of NULL values
.once ?OPTIONS? ?FILE?   Output for the next SQL command only to FILE
.open ?OPTIONS? ?FILE?   Close existing database and reopen FILE
.output ?FILE?           Send output to FILE or stdout if FILE is omitted
.parameter CMD ...       Manage SQL parameter bindings
.print STRING...         Print literal STRING
.progress N              Invoke progress handler after every N opcodes
.prompt MAIN CONTINUE    Replace the standard prompts
.quit                    Exit this program
.read FILE               Read input from FILE
.recover                 Recover as much data as possible from corrupt db.
.restore ?DB? FILE       Restore content of DB (default "main") from FILE
.save FILE               Write in-memory database into FILE
.scanstats on|off        Turn sqlite3_stmt_scanstatus() metrics on or off
.schema ?PATTERN?        Show the CREATE statements matching PATTERN
.selftest ?OPTIONS?      Run tests defined in the SELFTEST table
.separator COL ?ROW?     Change the column and row separators
.session ?NAME? CMD ...  Create or control sessions
.sha3sum ...             Compute a SHA3 hash of database content
.shell CMD ARGS...       Run CMD ARGS... in a system shell
.show                    Show the current values for various settings
.stats ?ARG?             Show stats or turn stats on or off
.system CMD ARGS...      Run CMD ARGS... in a system shell
.tables ?TABLE?          List names of tables matching LIKE pattern TABLE
.testcase NAME           Begin redirecting output to 'testcase-out.txt'
.testctrl CMD ...        Run various sqlite3_test_control() operations
.timeout MS              Try opening locked tables for MS milliseconds
.timer on|off            Turn SQL timer on or off
.trace ?OPTIONS?         Output each SQL statement as it is run
.vfsinfo ?AUX?           Information about the top-level VFS
.vfslist                 List all available VFSes
.vfsname ?AUX?           Print the name of the VFS stack
.width NUM1 NUM2 ...     Set minimum column widths for columnar output
sqlite>
```

With your Shark database created, you will now create a new table and populate it with data.

Step 3 — Creating a SQLite Table

SQLite databases are organized into tables. Tables store information. To better visualize a table, one can imagine rows and columns.
The rest of this tutorial will follow a common convention for entering SQLite commands. SQLite commands are uppercase and user information is lowercase. Lines **must** end with a semicolon.

Some of the SQLite commands that will be studied in this lab are shown below:

- CREATE TABLE creates a new table.

- INSERT INTO adds a new row to a table.
- SELECT queries data from a table.
- UPDATE edits a row in a table.
- ALTER TABLE changes an existing table.
- DELETE FROM deletes rows from a table.

Now let's create a table and some columns for various data:
- An ID
- The shark's name
- The shark's type
- The shark's average length (in centimeters)

Within the interactive shell, use the following command to create the table:

```
sqlite> CREATE TABLE sharks(id integer NOT NULL, name text NOT NULL, sharktype text NOT NULL, length integer NOT NULL);
```

Make sure you type a semicolon at the end of each SQL command! The sqlite3 program looks for a semicolon to know when your SQL command is complete. If you omit the semicolon, sqlite3 will give you a continuation prompt and wait for you to enter more text to complete the SQL command. This feature allows you to enter SQL commands that span multiple lines. For example, the previous command would be:

```
sqlite> CREATE TABLE sharks (
   ...>   id integer NOT NULL,
   ...>   name text NOT NULL,
   ...>   sharktype text NOT NULL,
   ...>   length integer NOT NULL
   ...> );
sqlite>
```

Using `NOT NULL` makes that field required. We will discuss `NOT NULL` in greater detail in the next section.
After creating the table, an empty prompt will return. To see a list of the tables in the database, you can enter `.tables`.

```
sqlite> .tables
shark
```

Now let's insert some values into it.

Important! – Execute SQL commands outside of the interactive shell

You can execute the aforementioned command from terminal without using the interactive shell as shown below:

```
sqlite3 sharks.db <<< "CREATE TABLE sharks(id integer NOT NULL, name text NOT NULL, sharktype text NOT NULL, length integer NOT NULL);";
```

Inserting Values into Tables

In SQLite, the command for inserting values into a table follows this general form:

```
sqlite> INSERT INTO tablename VALUES(values go here);
```

where tablename is the name of your table, and values go inside parentheses.
Now insert three rows of VALUES into your sharks table:

```
sqlite> INSERT INTO sharks VALUES (1, "Sammy", "Greenland Shark",
427);
sqlite> INSERT INTO sharks VALUES (2, "Alyoshka", "Great White
Shark", 600);
sqlite> INSERT INTO sharks VALUES (3, "Himari", "Megaladon",
1800);
```

Because you earlier specified NOT NULL for each of the variables in your table, you **must** enter a value for each.
For example, try adding another shark without setting its length:

```
sqlite> INSERT INTO sharks VALUES (4, "Faiza", "Hammerhead
Shark");
```

You will receive this error:

```
Error: table sharks has 4 columns but 3 values were supplied
```

In this step you created a table and inserted values into it. In the next step you will read from your database table.

Step 4 – Reading Tables in SQLite

In this step, we will focus on the most basic methods of reading data from a table. Recognize that SQLite provides more specific methods for viewing data in tables.

To view your table with all of the inserted values, use SELECT:

```
sqlite> SELECT * FROM sharks;
```

You will see the previously inserted entries:

```
1|Sammy|Greenland Shark|427
2|Alyoshka|Great White Shark|600
3|Himari|Megaladon|1800
```

To view an entry based on its id (the values we set manually), add the WHERE command to your query:

```
sqlite> SELECT * FROM sharks WHERE id IS 1;
```

This will return the shark whose id equals 1:

```
1|Sammy|Greenland Shark|427
```

Let's take a closer look at this command.

1. First, we `SELECT` all (`*`) values from our database, `sharks`.
2. Then we look at all `id` values.
3. Then we return all table entries where `id` is equal to `1`.

So far you have created a table, inserted data into it, and queried that saved data. Now you will update the existing table.

Step 5 — Updating Tables in SQLite

In the following two sections you will first add a new column into your existing table and then update existing values in the table.

Adding Columns to SQLite Tables

SQLite allows you to change your table using the `ALTER TABLE` command. This means that you can create new rows and columns, or modify existing rows and columns.

Use `ALTER TABLE` to create a new column. This new column will track each shark's age in years:

```
sqlite> ALTER TABLE sharks ADD COLUMN age integer;
```

You now have a fifth column, age.

Updating Values in SQLite Tables

Using the `UPDATE` command, add new `age` values for each of your sharks:

```
sqlite> UPDATE sharks SET age = 272 WHERE id=1;
sqlite> UPDATE sharks SET age = 70 WHERE id=2;
sqlite> UPDATE sharks SET age = 40 WHERE id=3;

1|Sammy|Greenland Shark|427|272
2|Alyoshka|Great White Shark|600|70
3|Himari|Megaladon|1800|40
```

In this step you altered your table's composition and then updated values inside the table. In the next step you will delete information from a table.

Step 6 – Deleting Information in SQLite

In this step you will delete entries in your table based on the evaluation of an argument.

In the following command you are querying your database and requesting that it delete all sharks in your `sharks` table whose age is less than 200:

```
sqlite> DELETE FROM sharks WHERE age <= 200;
```

Typing `SELECT * FROM sharks;` will verify that `Alyoshka` and `Himari`, who were each less than 200 years old, were deleted. Only `Sammy` the Greenland Shark remains:

```
1|Sammy|Greenland Shark|427|272
```

Step 7 – Installing SQLite Browser on Ubuntu VM

After successfully installing SQLite 3, you are now ready to install and get started with the SQLite Browser application. The SQLite Browser package can be installed using either of two methods:

- Install SQLite Browser Using Apt Repository
- Install SQLite Browser Using Snap

To install the SQLite Browser using the apt repository, first, update your system's apt-cache repository.

```
sudo apt update
```

Next, install the SQLite Browser via the following command:

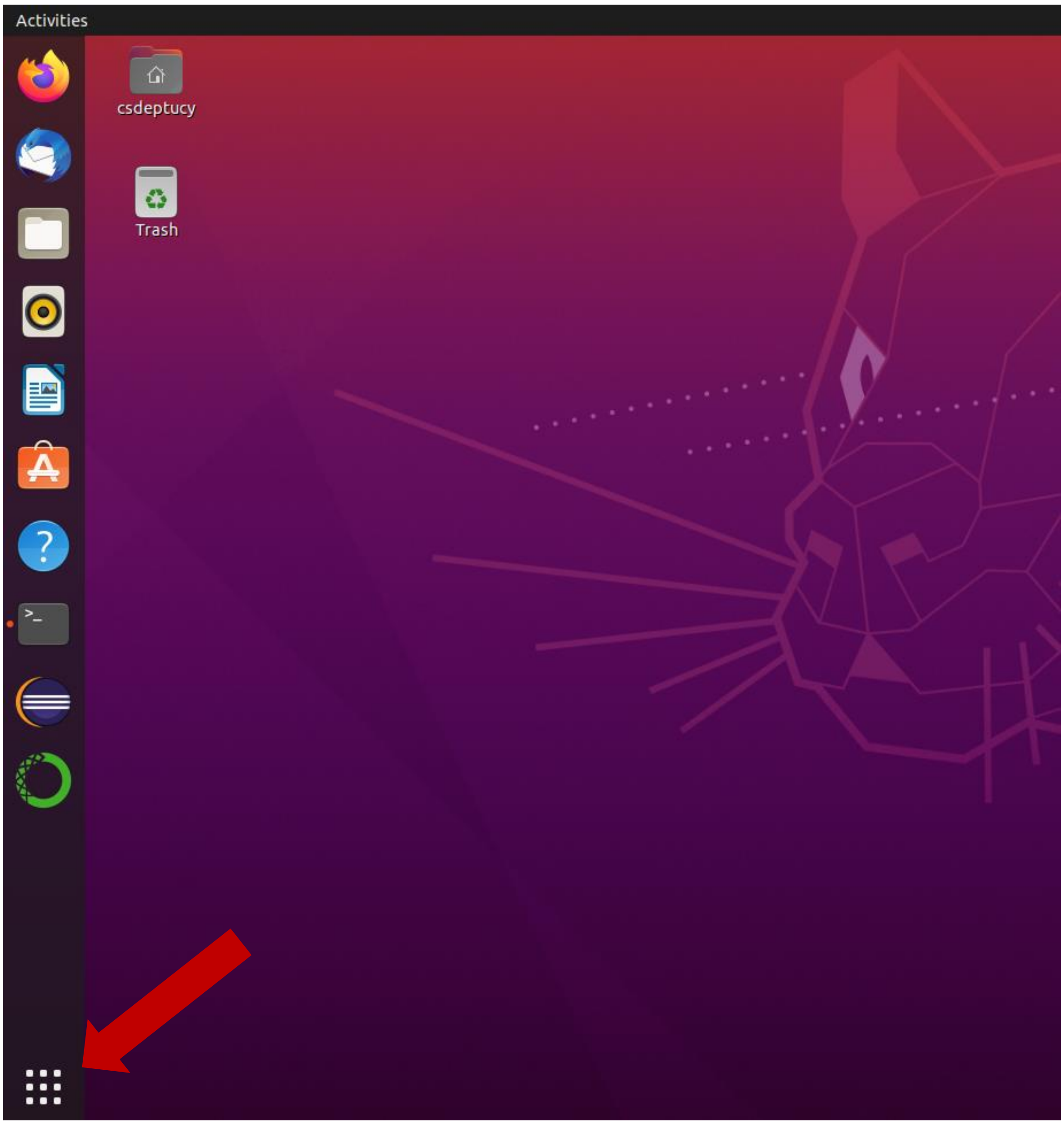
```
sudo apt install sqlitebrowser
```

You will then be asked whether you want to continue to take additional disk space or quit the installation process. Press 'y' to continue the installation. Once the installation of SQLite Browser has completed, you are now ready to launch and use the SQLite browser in your Ubuntu system.

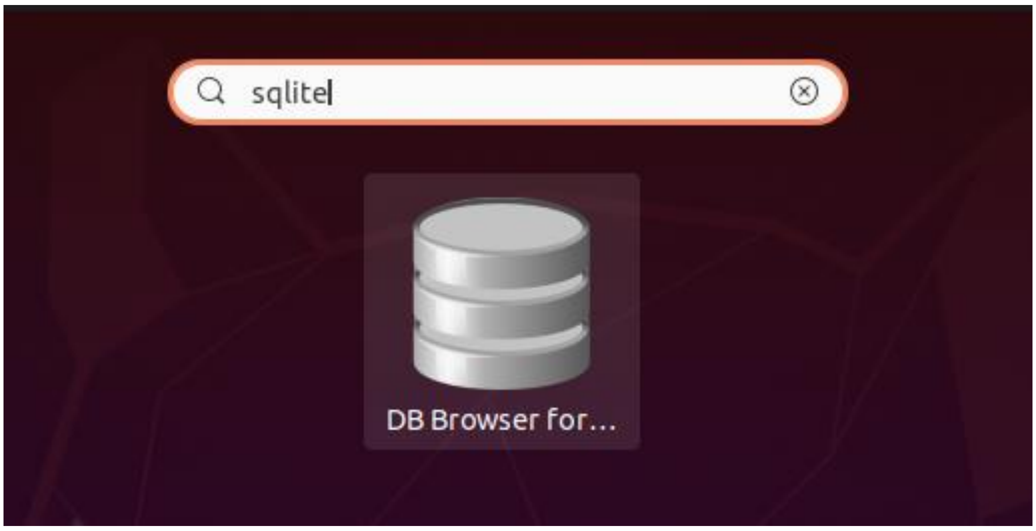
To start the SQLite Browser, simply execute from terminal the command:

```
sqlitebrowser
```

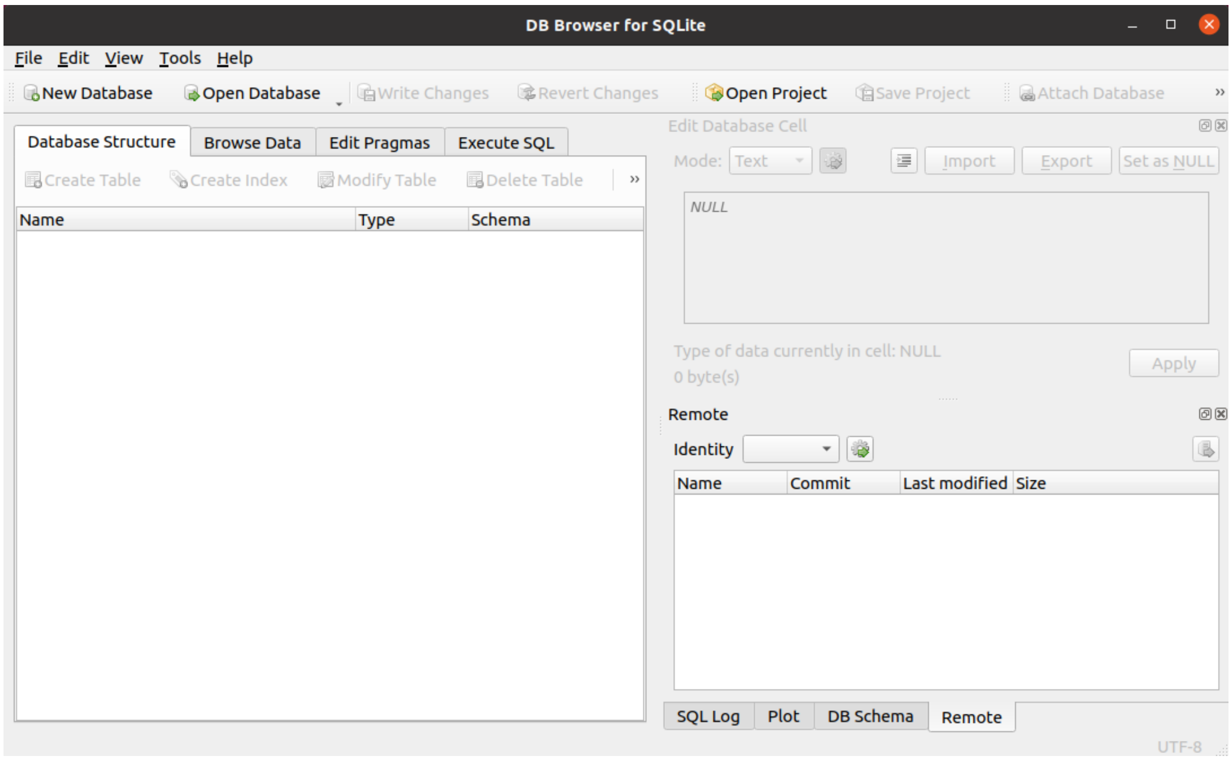
or search for the SQLite Browser in the Application Menu.



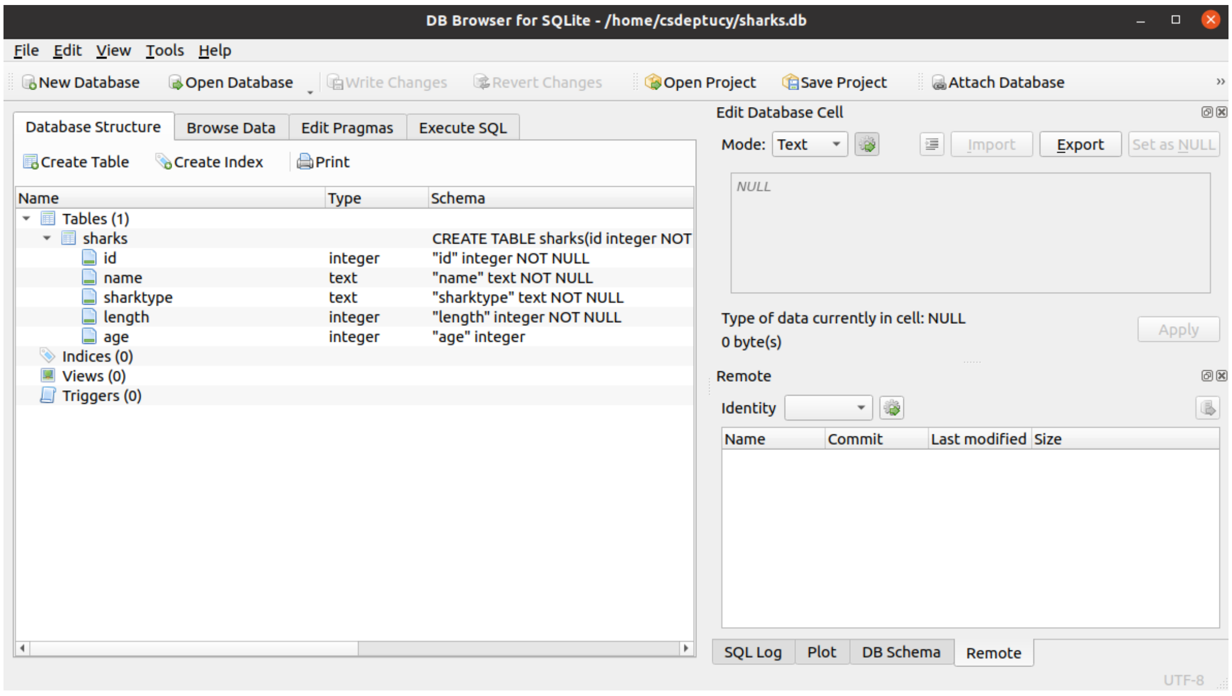
Search for sqlite browser and then, click the SQLite Browser Icon to open it.



As shown below, this is the Welcome screen of the SQLite Browser.



Open Database → Select shark.db



Name	Type	Schema
Tables (1)		
sharks		CREATE TABLE sharks(id integer NOT
id	integer	"id" integer NOT NULL
name	text	"name" text NOT NULL
sharktype	text	"sharktype" text NOT NULL
length	integer	"length" integer NOT NULL
age	integer	"age" integer
Indices (0)		
Views (0)		
Triggers (0)		

Browse data

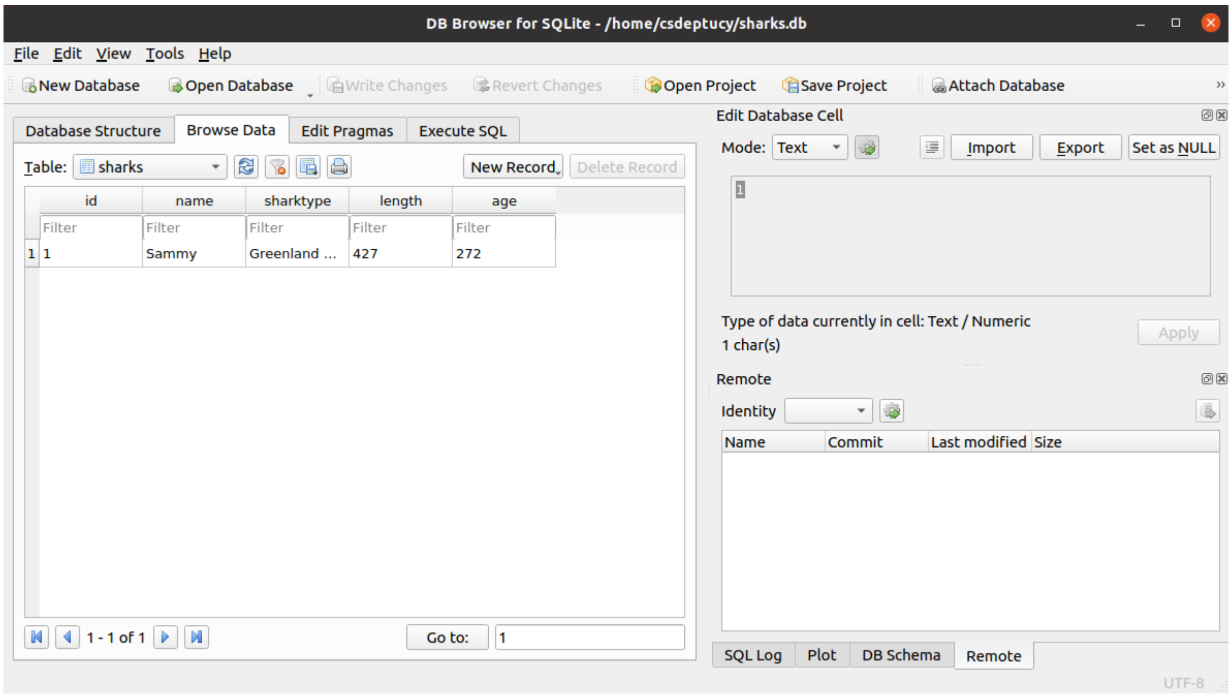


Table: sharks

	id	name	sharktype	length	age
	Filter	Filter	Filter	Filter	Filter
1	1	Sammy	Greenland ...	427	272

1 - 1 of 1   Go to: 1

Mode: Text

Type of data currently in cell: Text / Numeric
1 char(s)

Name	Commit	Last modified	Size

Exercises

Without using the interactive shell, write the commands to accomplish the following:

1. Create a new database called connections.db with a table called hosts with the following information: hostip as text, protocol as text, state as text.
2. Open firefox and navigate to https://www.cs.ucy.ac.cy
3. Using the command netstat discussed in Lab2, get the host IP address, the protocol (e.g. http, https) and the connection state (e.g. ESTABLISHED, TIME_WAIT) for the first active connection and insert it to the hosts table
4. Using the command netstat discussed in Lab2, get the host IP address, the protocol (e.g. http, https) and the connection state (e.g. ESTABLISHED, TIME_WAIT) for the second active connection and insert it to the hosts table
5. Select and print the https connections
6. Select and print the ESTABLISHED connections