Introduction to using SQLite

SQLite is an open source software library that implements a Relational Database engine. In its most basic form it provides a command line shell for creating and modifying Relational Databases. This is the tool we will be using for the module, and will allow you to develop standard SQL programming skills. The guide is based on the Windows version of the software – but Linux and Mac OS X versions are also available. The SQLite homepage is at http://www.sqlite.org/index.html

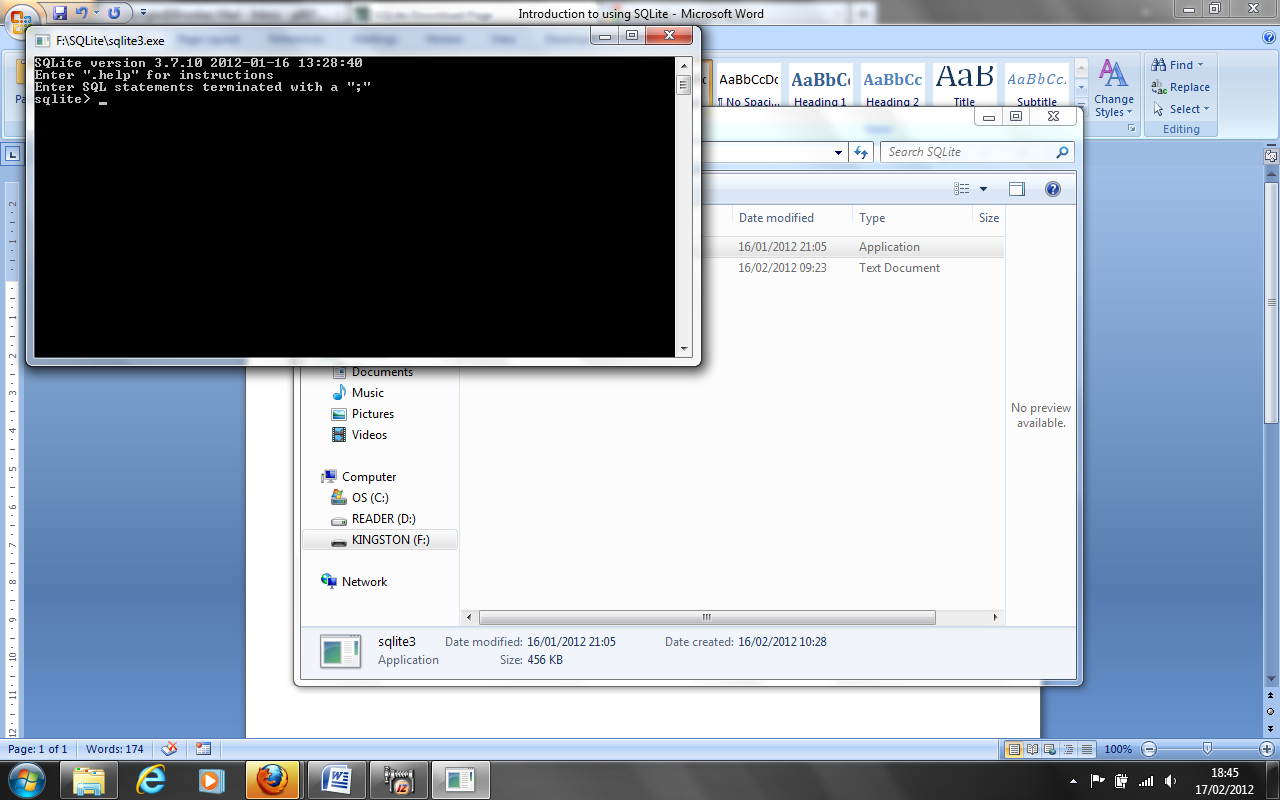
The following guide provides an introduction to downloading and installing the engine, and creating and modifying your first database. The SQLite database engine can be stored on, and run from the H: drive, or a USB memory stick.

Downloading and installing the database engine.

The download page is at <http://www.sqlite.org/download.html>

Select the **Precompiled Binaries For Windows,** and then thesqlite-shell-win32-x86-3071000.zip

Extract the file ‘sqlite3.exe’, and store this application on your H: drive. This is the command line shell, which is all you will need for the programming exercises and the coursework implementation. Double-click the application icon to run the engine, and you will open the shell in a new window, with the sqlite prompt, as shown below:



Creating, modifying and saving a database.

Most of the time you will be typing SQL commands at the sqlite prompt, in order to define a database or to manipulate its contents. There are some special commands which you will need to use in order to manage the database and its interface. A full list of these commands is available at: <http://www.sqlite.org/sqlite.html>

Some of the more important are described in the sections below.

At the end of this guide, you will have created a database with one table, inserted data into the table, and saved the entire database to disc. You will also have used notepad as an editor for SQL commands.

Type the commands, as shown in bold. Note that all complete SQL commands are terminated with a semicolon, but may be split over several lines; special commands do not require the semicolon. Explanations are given in italics.

sqlite> **create table MODULE (**

- - -> **CODE text primary key,**  *CODE is the primary key*

- - -> **TITLE text,**

- - -> **STATUS text constraint mod\_val check (STATUS in (‘B’, ‘A’)),** *values of STATUS restricted*

- - -> **CREDIT numeric constraint mod\_cred check (CREDIT in (1, 2)));** *values of CREDIT restricted*

sqlite> *semicolon ends command-*

*any error messages would appear here (of the form”Error: near syntax error”)*

The effect of this SQL command is to create a table, called ‘MODULE’, with columns (attributes) CODE, TITLE, STATUS and CREDIT. If you want to see the structure of a table, at any stage, use the .schema (special) command at the sqlite prompt:

sqlite> **.schema MODULE** *no need for semicolon*

You can insert data into this table, using the following SQL command:

sqlite> **insert into MODULE values (‘U08049’, ‘Database Design’, ‘A’, 1);**

sqlite> *no confirmation is given*

You can see all of the data currently stored in a table:

sqlite> **select \* from MODULE;** *\* for all attributes*

U08049|database Design|A|1

sqlite>

**IMPORTANT CHECKPOINT**

At this stage you have NOT permanently stored your database. While the sqlite application is running, all changes to the database are stored in TEMPORARY files. To save the database, you must use the .backup (special) command:

sqlite> **.backup university** *creates an ascii file, called ‘university’, storing all tables and data currently in the sqlite temporary files*.

sqlite> **.exit** *terminates the application (closing the window).*

The next time you want to work with the ‘university’ database, run sqlite.exe, and then:

sqlite> **.restore university** *restores the tables and data into the sqlite temporary files.*

**Using notepad as an SQL command editor**

Rather than typing SQL commands at the command line prompt, with all the dangers associated with typos, it is more efficient to use a text editor, and then execute directly from the text file.

Open the file uni\_tables (on the module website, practical session for week 4. In it, you will see the SQL commands for a collection of tables: MODULE, PREQ, STUDENT and REGISTER. Save this file in the same directory as you have sqlite.exe.

Start a new SQLite session, and restore the university database. Now execute the SQL commands in uni\_tables.txt:

sqlite> **.read uni\_tables.txt** *executes the contents of uni\_tables.txt as SQL commands*

Error near line 1: table MODULE already exists *we knew this – but the other tables have been created, and MODULE still exists*

sqlite> **.schema student** *see the expected student table structure*

sqlite> **.backup university *DON’T FORGET!***

sqlite> **.exit**

As a last exercise, sketch the EAR diagram for the university database you have created.

Footnote: Data ‘affinities’ in SQLite.

Use the following data types for attributes. SQLite provides dynamic typing which map other types onto one of these: INTEGER, REAL, TEXT. NUMERIC (for boolean, date, datetime, etc.)