# CS 3380 Lab Assignment 11

**Directions** : This assignment is due on **Sunday, November 23 at 11:59 PM**. (But remember, you may want to start before the weekend because of Thanksgiving Break.) You must submit a ZIP or TAR file containing all of your source code through Blackboard prior to the deadline in order to receive credit for this assignment. Your ZIP/TAR file should contain your two source code files for this assignment and one Makefile. It should not contain any binary files; run `make clean` before creating your ZIP/TAR file.

**Goals:**

1. Learn to use the SQLite C/C++ API
2. Learning to read the API documentation for a software library
3. Implement an application that dumps the contents of a SQLite database table and writes it to a CSV file
4. Implement an application that loads to contents of a CSV file into a SQLite database table

**Template Download:** DO NOT download this lab template into your public_html directory. If you do so, your code will be visible to all students. If someone steals your code and submits it, I'll have no way of knowing if you voluntarily gave it to them or not. You will both receive a zero on the assignment if caught.

If you don't already have a cs3380 directory in your home directory, create the directory and navigate to it with the following command.

```
mkdir ~/cs3380
cd ~/cs3380
```

Decide whether you want to work in C or C++ for this assignment. After deciding execute one of the following commands:

```
wget http://babbage.cs.missouri.edu/~klaricm/fs14/cs3380/lab11/c/lab11-c.tgz
wget http://babbage.cs.missouri.edu/~klaricm/fs14/cs3380/lab11/cpp/lab11-cpp.tgz
```

You can unpack the source code with the following command, where `filename.tgz` is replaced with the appropriate name.

```
tar xzfv filename.tgz
```

**Database Creation:** Create an example database by executing the command `sqlite3 mydatabase.db` at the prompt on babbage. This opens the SQLite application using the newly created database file. Then, execute the following SQL command to create a database table:

```
CREATE TABLE mytable (id integer PRIMARY KEY, foo varchar (25), bar varchar(25));
```

Add some data to your table by executing something like the following. You don't have to add these exact records. In fact the data that you test with isn't terribly important.

```
INSERT INTO mytable VALUES(1,‘some’,‘value’);
INSERT INTO mytable VALUES(2,‘another’,‘value’);
INSERT INTO mytable VALUES(3,‘yet another’,‘one’);
```

**The dump application:** Using the template that you previously downloaded, create a C or C++ application that utilizes the SQLite API that is capable of dumping the contents of **any** SQLite database table to a text file in CSV format. CSV format is a simple format in which each line of the text file corresponds to one database record. Within each line, the fields are separated by commas. Text fields that use the SQL data type VARCHAR (as opposed to numeric fields such as integer) should be surrounded by single quotes. Using the three inserts statements listed above, the corresponding CSV file looks like this:

```
1,‘some’,‘value’
2,‘another’,‘value’
3,‘yet another’,‘one’
```

Your dump application should take arguments in the following order:

```
Usage: ./dump <database file> <table name> <CSV file>
```

An example execution of this application follows:

```
./dump  mydatabase.db mytable mydatabase.csv
```

Your dump application should us the `sqlite3_open` function to open the SQLite database file. Then, it should create a prepared statement using the `sqlite3_prepare_v2` function to prepare a statement that queries for all data from the specified table. Your application should then iterate through the records and inspect them one by one using the `sqlite3_step` function. Note that you can use the `sqlite3_column_count` function to get the number of fields in the record. The `sqlite3_column_type` function returns an indication of the datatype of each field. Your application only needs to be able able to handle text (`SQLITE_TEXT`) and integer (`SQLITE_INTEGER`) field types. Your application can exit with an error message if any other field type is encountered.

While iterating through the records, your dump application should write out a line in the specified CSV file for each record. The fields represented by each line, should be separated by commas.

Lastly, your application should call the `sqlite3_close` function and close the text field that you have open to ensure that writing has been completed by the OS.

**The load application:** Create a C or C++ application that utilizes the SQLite API to load the contents of a CSV file into an existing SQLite database table in an existing SQLite database file. Your load application should take arguments in the following order:

```
Usage: ./load <database file> <table name> <CSV file>
```

An example execution of this application follows:

```
./load mydatabase.db mytable mydatabase.csv
```

The load application should first open the SQLite database file and then delete all records in the specified table. SQLite does not have the `TRUNCATE` command. Thus, to delete the records you must use a statement such as:

```
DELETE FROM sometable;
```

Next, read the contents of the CSV file and build SQL `INSERT` statements. Execute each insert statement to load the data from the CSV file to the SQLite database. Each execute statement should be executed by calling the following set of commands: `sqlite3_prepare_v2`, `sqlite3_step` and `sqlite3_finalize`. Finally, the `sqlite3_close` command should be executed to close the database and release the associated resources.

**Notes:**

- Part of the work for this assignment is reading the SQLite API documentation. I have provided you some of the most important links in the next section.

- I strongly encourage you to write small, simple test programs first, that help you be sure that you have a grasp of the SQLite API. Once you understand how the individual functions work, then write the programs for this assignment.
- You are **not allowed** to use the `sqlite3_exec` and `sqlite3_get_table` functions in this lab assignment. You are also not allowed to use the C `system` function or any of the `exec` family of functions. Ask if you have any questions regarding which functions are allowed on this assignment.
- Your assignment submission should contain **exactly** two (2) source code files and one (1) Makefile. If you are using C, the source code files should be named `dump.c` and `load.c`. If you are using C++, they should be named `dump.cpp` and `load.cpp`. I will provide you a set of files for this assignment. You can download them from `http://babbage.cs.missouri.edu/~klaricm/fs14/cs3380/lab11/index.php`
- The order of the arguments accepted by your application must be **exactly** as shown in my examples. No other ordering will be accepted and no other arguments should be required/allowed.
- Your applications should work with **any database tables** that have any combination of integer and varchar fields. **No assumption** should be made regarding the order and number of fields.

**API Documentation:**

- An Introduction To The SQLite C/C++ Interface : http://www.sqlite.org/cintro.html
- `sqlite3_open` : http://www.sqlite.org/c3ref/open.html
- `sqlite3_prepare_v2` : http://www.sqlite.org/c3ref/prepare.html
- `sqlite3_step` : http://www.sqlite.org/c3ref/step.html
- Column functions : http://www.sqlite.org/c3ref/column_blob.html
- `sqlite3_finalize` : http://www.sqlite.org/c3ref/finalize.html
- `sqlite3_close` : http://www.sqlite.org/c3ref/close.html