This assignment is split into two parts: RESEARCH and TODO. I'm going to place the research at the end of this document, because I'll probably refer back to the TODO section more frequently.

TODO

The first item on our todo list for this assignment is to Install Sqlite3 on my machine. I already completed this, and detailed my setup, in the previous assignment.

Next, we are given a CSV file, titled `nameWithHeightWeight.csv`. Our goal is to "*load it into a sqlite table,*
perform some queries and save the table in a file-based database".

First, I need to copy the CSV file into my Ubuntu container with SQLite3 installed.

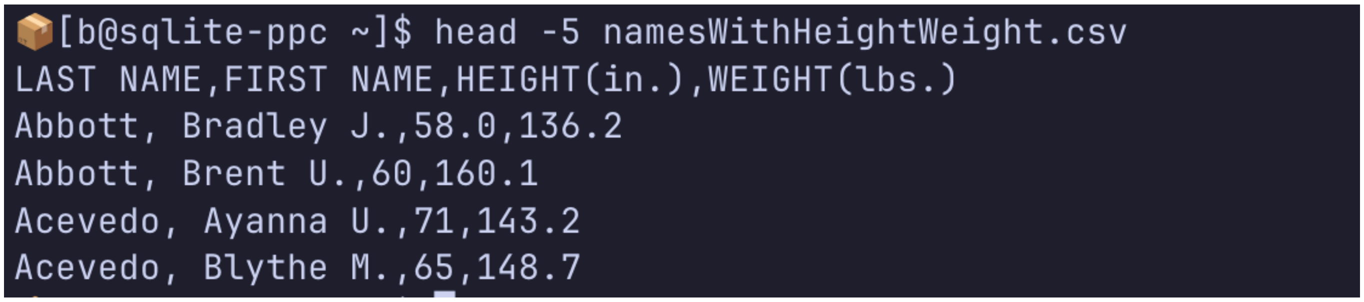
```
~/Downloads
❯ podman cp namesWithHeightWeight.csv sqlite-ppc:/home/b
```

Next, I'll enter my container, and verify that the file copied in properly.

```
❯ distrobox enter sqlite-ppc
Starting container...                            [ OK ]
Installing basic packages...                     [ OK ]
Setting up devpts mounts...                      [ OK ]
Setting up read-only mounts...                   [ OK ]
Setting up read-write mounts...                  [ OK ]
Setting up host's sockets integration...         [ OK ]
Integrating host's themes, icons, fonts...       [ OK ]
Setting up distrobox profile...                  [ OK ]
Setting up sudo...                               [ OK ]
Setting up user's group list...                  [ OK ]

Container Setup Complete!
📦[b@sqlite-ppc ~]$ ls | grep names
namesWithHeightWeight.csv
```

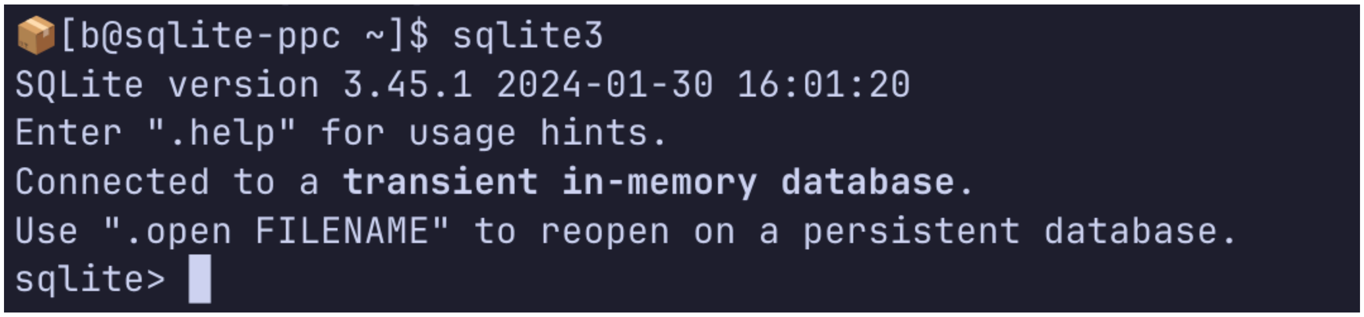
Awesome, the CSV is there, and we can continue. Let's take a look at the contents.



```
📦[b@sqlite-ppc ~]$ head -5 namesWithHeightWeight.csv
LAST NAME,FIRST NAME,HEIGHT(in.),WEIGHT(lbs.)
Abbott, Bradley J.,58.0,136.2
Abbott, Brent U.,60,160.1
Acevedo, Ayanna U.,71,143.2
Acevedo, Blythe M.,65,148.7
```

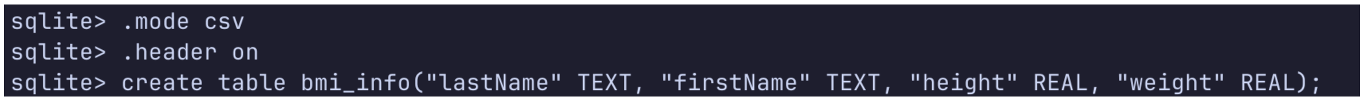
We can see the CSV has 4 Columns, `LAST NAME`, `FIRST NAME`, `HEIGHT(in.)`, and `WEIGHT`(lbs.). We also see the CSV contains a header, which we'll have to remove once we migrate it to a table.

The next step is to open an sqlite3 session.



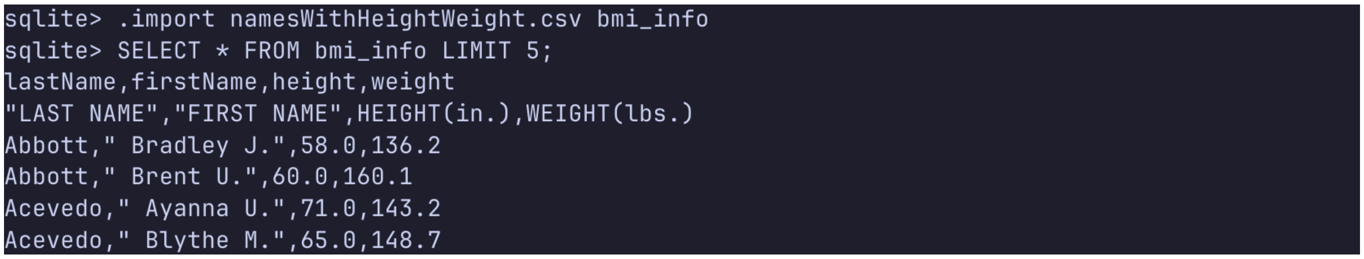
```
📦[b@sqlite-ppc ~]$ sqlite3
SQLite version 3.45.1 2024-01-30 16:01:20
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> 
```

Next, we need to turn on CSV mode and say we want the header displayed while outputting data. After that, we'll create a table called `bmi_info` matching the structure of the csv.



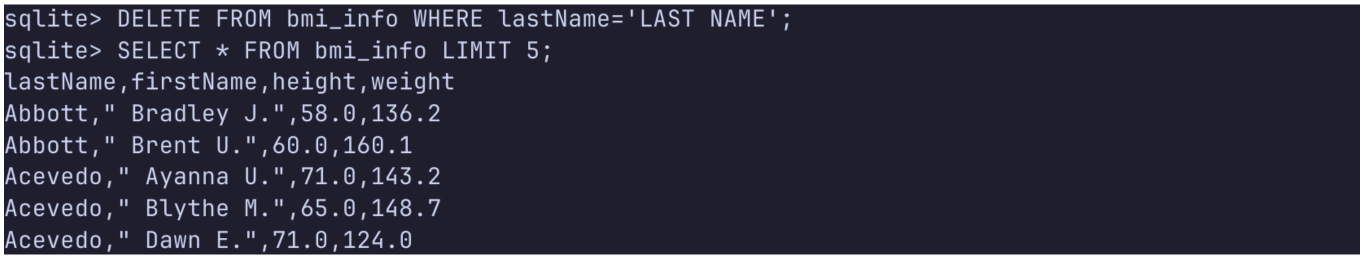
```
sqlite> .mode csv
sqlite> .header on
sqlite> create table bmi_info("lastName" TEXT, "firstName" TEXT, "height" REAL, "weight" REAL);
```

Now, we'll import our CSV file into our new table, and verify it properly imported by displaying the first few rows.



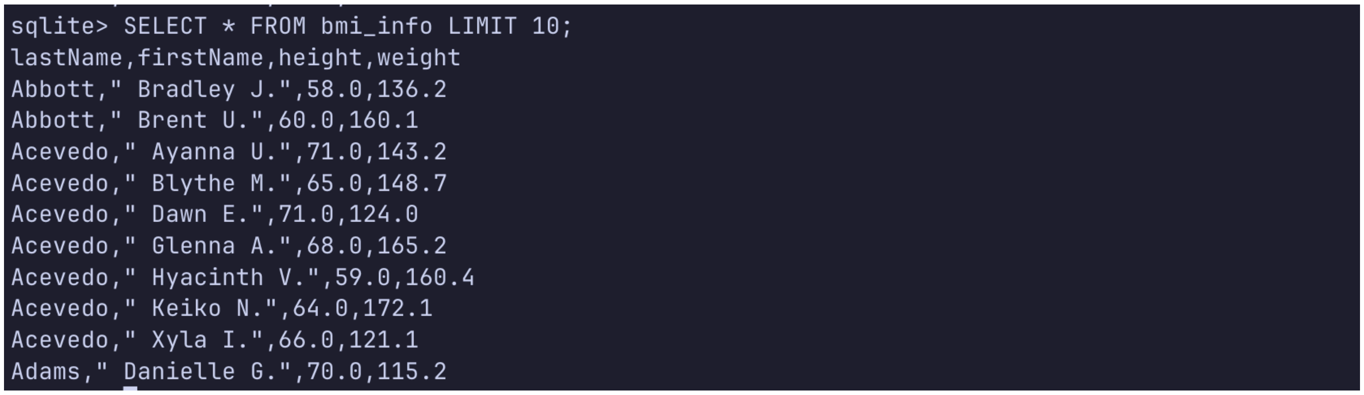
```
sqlite> .import namesWithHeightWeight.csv bmi_info
sqlite> SELECT * FROM bmi_info LIMIT 5;
lastName,firstName,height,weight
"LAST NAME","FIRST NAME",HEIGHT(in.),WEIGHT(lbs.)
Abbott," Bradley J.",58.0,136.2
Abbott," Brent U.",60.0,160.1
Acevedo," Ayanna U.",71.0,143.2
Acevedo," Blythe M.",65.0,148.7
```

Perfect, we properly read the CSV file into the table. Let's remove that pesky header. I'll run a simple `DELETE` statement to remove any rows where the lastName field is LAST NAME (which is only the header). After that, I'll rerun the `SELECT` statement to show the first five rows, and we can see the header is gone.



```
sqlite> DELETE FROM bmi_info WHERE lastName='LAST NAME';
sqlite> SELECT * FROM bmi_info LIMIT 5;
lastName,firstName,height,weight
Abbott," Bradley J.",58.0,136.2
Abbott," Brent U.",60.0,160.1
Acevedo," Ayanna U.",71.0,143.2
Acevedo," Blythe M.",65.0,148.7
Acevedo," Dawn E.",71.0,124.0
```

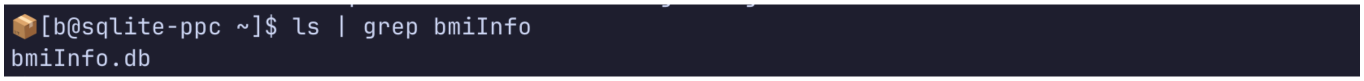
The next step in the TODO is to simply list the first 10 table entries. I'll just slighty alter the `SELECT` statement from before to limit 10 rows instead of 5.



```
sqlite> SELECT * FROM bmi_info LIMIT 10;
lastName,firstName,height,weight
Abbott," Bradley J.",58.0,136.2
Abbott," Brent U.",60.0,160.1
Acevedo," Ayanna U.",71.0,143.2
Acevedo," Blythe M.",65.0,148.7
Acevedo," Dawn E.",71.0,124.0
Acevedo," Glenna A.",68.0,165.2
Acevedo," Hyacinth V.",59.0,160.4
Acevedo," Keiko N.",64.0,172.1
Acevedo," Xyla I.",66.0,121.1
Adams," Danielle G.",70.0,115.2
```

As expected, the first 10 rows are displayed.

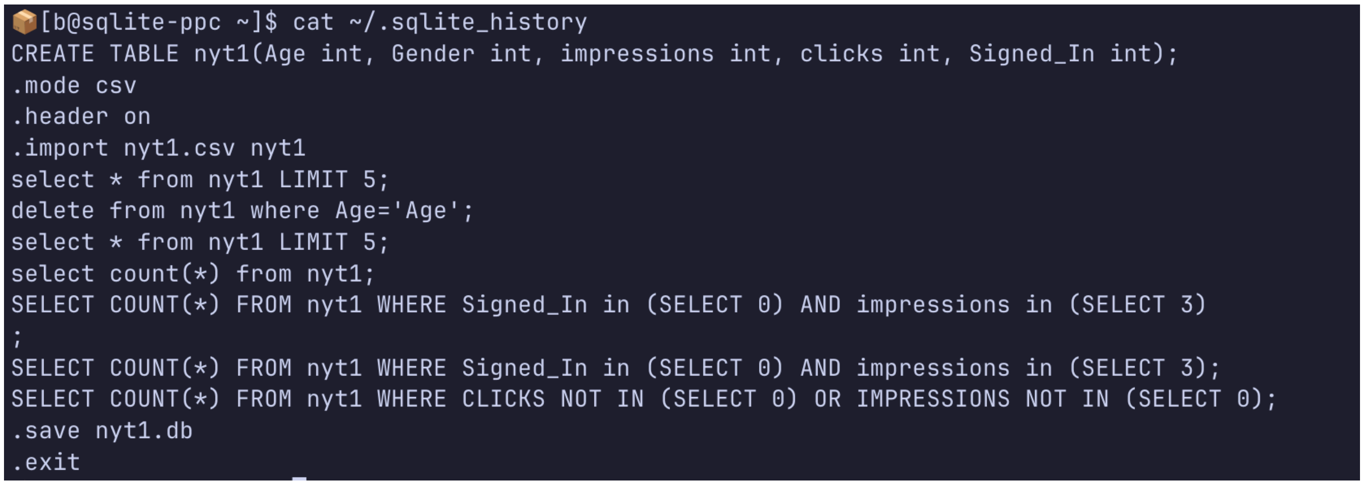
Next, we are instructed to save the data to a database file called `bmiInfo.db`, and then exit the sqlite3 session. After exiting, I'll double check that `bmiInfo.db` properly saved:



```
📦[b@sqlite-ppc ~]$ ls | grep bmiInfo
bmiInfo.db
```

The database file is there, as expected.

Next, we are instructed to find and examine the .sqlite_history file.

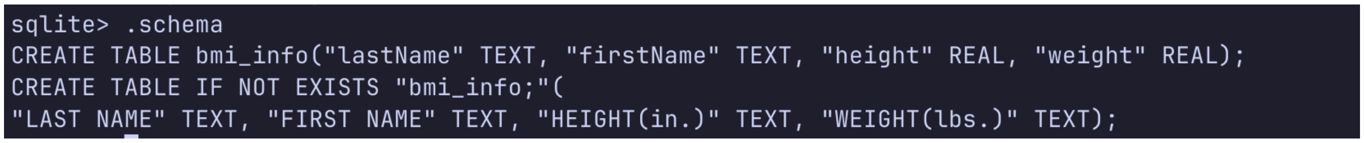


```
📦[b@sqlite-ppc ~]$ cat ~/.sqlite_history
CREATE TABLE nyt1(Age int, Gender int, impressions int, clicks int, Signed_In int);
.mode csv
.header on
.import nyt1.csv nyt1
select * from nyt1 LIMIT 5;
delete from nyt1 where Age='Age';
select * from nyt1 LIMIT 5;
select count(*) from nyt1;
SELECT COUNT(*) FROM nyt1 WHERE Signed_In in (SELECT 0) AND impressions in (SELECT 3)
;
SELECT COUNT(*) FROM nyt1 WHERE Signed_In in (SELECT 0) AND impressions in (SELECT 3);
SELECT COUNT(*) FROM nyt1 WHERE CLICKS NOT IN (SELECT 0) OR IMPRESSIONS NOT IN (SELECT 0);
.save nyt1.db
.exit
```

We can see everything that I entered during the previous assignment, but not anything from the most recent session, which is because I exited the sqlite3 session using ctrl+C instead of .exit. Oops!

It's interesting that we can see all of the statements and commands I inputted, but none of the results of those statements (Like the `SELECT` not showing results).

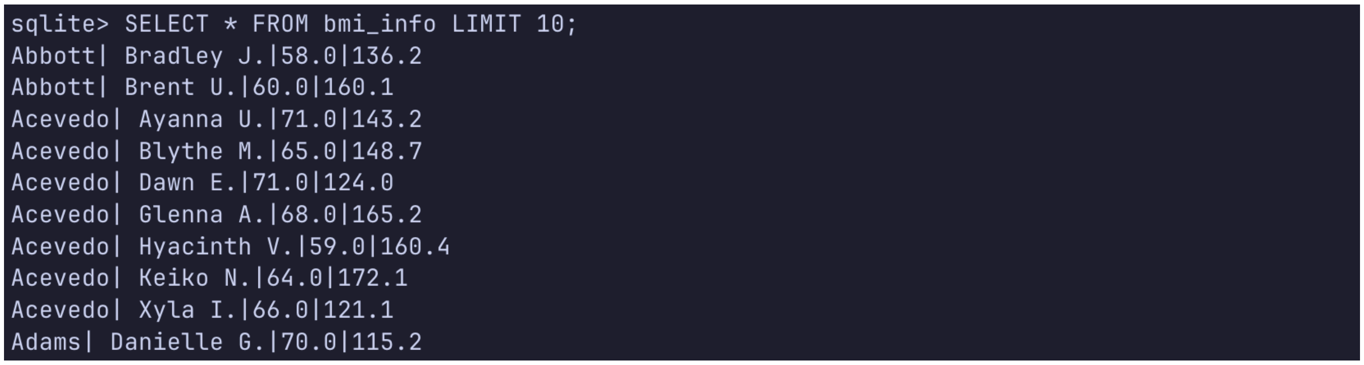
Next, we have to start another sqlite3 session and load the `bmiInfo.db` file that we saved previously. After loading, we run `.schema`.



```
sqlite> .schema
CREATE TABLE bmi_info("lastName" TEXT, "firstName" TEXT, "height" REAL, "weight" REAL);
CREATE TABLE IF NOT EXISTS "bmi_info;"(
"LAST NAME" TEXT, "FIRST NAME" TEXT, "HEIGHT(in.)" TEXT, "WEIGHT(lbs.)" TEXT);
```

`.schema` shows the SQL `CREATE` statements that define the structure of our table.

Finally, we are to list the first 10 table entries again.



```
sqlite> SELECT * FROM bmi_info LIMIT 10;
Abbott| Bradley J.|58.0|136.2
Abbott| Brent U.|60.0|160.1
Acevedo| Ayanna U.|71.0|143.2
Acevedo| Blythe M.|65.0|148.7
Acevedo| Dawn E.|71.0|124.0
Acevedo| Glenna A.|68.0|165.2
Acevedo| Hyacinth V.|59.0|160.4
Acevedo| Keiko N.|64.0|172.1
Acevedo| Xyla I.|66.0|121.1
Adams| Danielle G.|70.0|115.2
```

Because I didn't specify `.mode csv`, we're seeing | separators instead of commas, and no `.header on` means the header isn't displayed when outputting. Our data is the same.

In this assignment, we experimented with creating a sqlite3 database from a CSV file, saving databases to .db files, and retrieving existing databases from .db files.

Research

For research, I've been assigned to read the following:

E. F. Codd, "A Relational Model of Data for Large Shared Data Banks,"
Comm. ACM, vol. 13, no. 6 (June 1970), pages 377-387

And

"SEQUEL: A Structured English Query Language"
Proceedings of the 1974 ACM SIGFIDET Workshop on Data Description, Access and Control.
Association for Computing Machinery:, pages 249-264

From these readings, I'll pull key phrases/words and definitions.

A Relational Model of Data for Large Shared Data Banks

Data Bank: Broader term structured repository of data, term contains Relational Data Base

Data Base: Organized collection of structured data.

Data Structure: Format for organizing and storing data.

Data Organization: The process of collecting, structuring, and storing data.

Hierarchies of Data: Organizational Structure of data elements arranged in a hierarchical order or tree. Parent/child setup.

Networks of Data: More flexible way to organize data, where hierarchical is stricter. Allows for individual data to have more than one parent/child.

Relations: Description of connections/interactions between SQL tables.

Derivability: Certain data, tables, etc. can be derived from other data, rather than being physically stored in the database.

Redundancy: Storing data in multiple places. Can be bad (duplicate entries) or good (backups)

Consistency: Ensuring all changes to a database follow the rules and constraints specified.

Composition: A strong dependency between two tables (such as a parent/child).

JOIN: Used to combine rows from multiple tables based off a related column.

Retrieval Language: Fetch data from a database without modifying data (SELECT)

Predicate Calculus: Defining and filtering data based on logical conditions. Specifies WHAT vs HOW to retrieve.

Security: Protect a database from unauthorized access or destruction.

Data Integrity: Ensuring the data in a database maintains it's intended state.

*SEQUEL: A Structured English Query Language

Query Languages: A language used to fetch/modify data from a database

DBMS (Data Base Management Systems): Software used to create and manipulate databases.

Information Retrieval: Methods to pull data from database columns, such as utilizing regex.

Data Manipulation Languages: The subset of SQL that deals with altering data in a database *without* changing database structure.