1. First Build

- First of all, I watched the instructor's demo on how to build sqlite3 from source code.Then, I followed his instruction and intended to do the project on my Mac environment without any virtual machine.
- I downloaded the source code from the website shown in the demo video and extracted it. Next, in terminal, I ran the command "sudo apt install gcc" but met a problem. The terminal said "Unable to locate an executable at /…", which confused me. I was not sure whether my Mac has installed gcc before or not but I decided not to fix this problem but switch to my virtual machine to do this project instead.
- I have VMware Fusion running Linux on my Mac. I installed and have been using it since the beginning of this semester for my CS 149 (Operating Systems) so I am fairly familiar with it. Plus, I already installed gcc on the Linux so one less step to do. I continued to install tcl and started building the first build. Since my VM is quite slow, I used the native Mac environment to look and modify the source code and build the result code in my VM.
- Just like in the demo, I created the first build in the a folder on Desktop, named "bld0". I ran "configure" from sqlite folder and "make" in the bld0 folder. I did not change any source code and the purpose of this first build is to just run the original version of sqlite3 on my VM.

2. Second Build

- After the original version of sqlite3 has run successfully on my VM, I started to actually do the project. Initially, I intended to research online to find out where to start, but I decided to take a look at the "src" folder of sqlite first for a warmup. In there, I found lots of C files and I just happened to see the insert.c file. I started working on it from that point to the end of the project without looking at any online resource.
- After opening insert.c by Geany, I was overwhelmed by the length of the file and how much stuff in it. Luckily, on the side bar, Geany showed a list of all functions, global variables, structs, and macros in the file. Luckily again, I saw the sqlite3Insert function in that list and checked on it. It is fair to say if I used a different app to open insert.c, I would have had much more difficult doing this project. Geany allowed me to finish the project in one day.
- After finding sqlite3Insert(), I read the first few lines of the function to check out its local variables. Although not understanding all these variables since it was the first time I had contact with sqlite3 source code, several ones caught my attention thanks to the author's comment:

```
SrcList *pTabList,    /* Name of table into which we are inserting */
Table *pTab;          /* The table to insert into.  aka TABLE */
int nColumn;          /* Number of columns in the data */
```

```
int nHidden = 0;      /* Number of hidden columns if TABLE is virtual */
```

I didn't use all these variables at the end but they gave me a sense of what I need to do in this project.

- After the local variables, I did not focus on understanding every line of code but I skimmed through the whole function to see if my eyes could catch anything since it is a very long function! Luckily again, I saw in lines 939 - 941 the error message which is generated when we insert a tuple with different number of columns than the table:

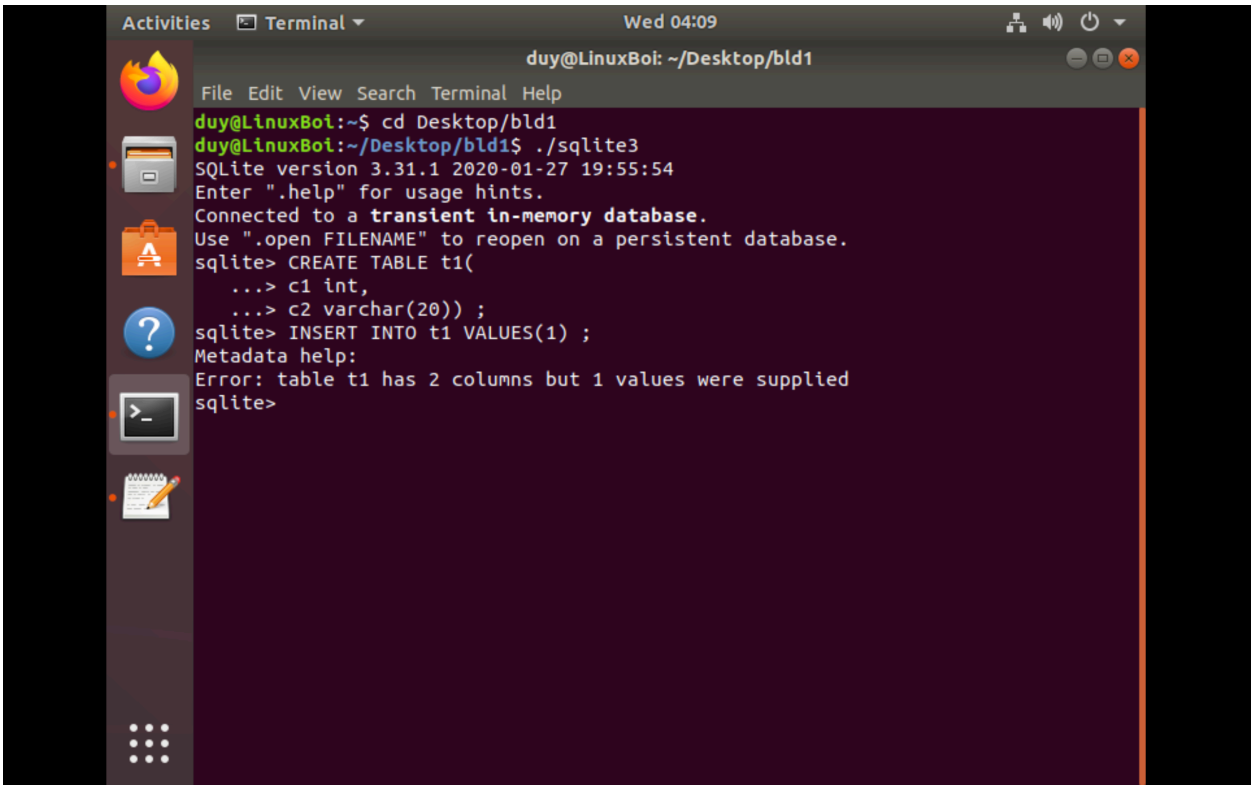
```
sqlite3ErrorMsg(pParse,
    "table %S has %d columns but %d values were supplied",
    pTabList, 0, pTab->nCol-nHidden, nColumn);
```

This is the error that should be generated along with our addition error message, so I knew that I have found the location where I should enter my code.

- I decided to test the first modification by adding a simple additional output: the string "Metadata help: " before the sqlite error message. I added a simple output function as below:

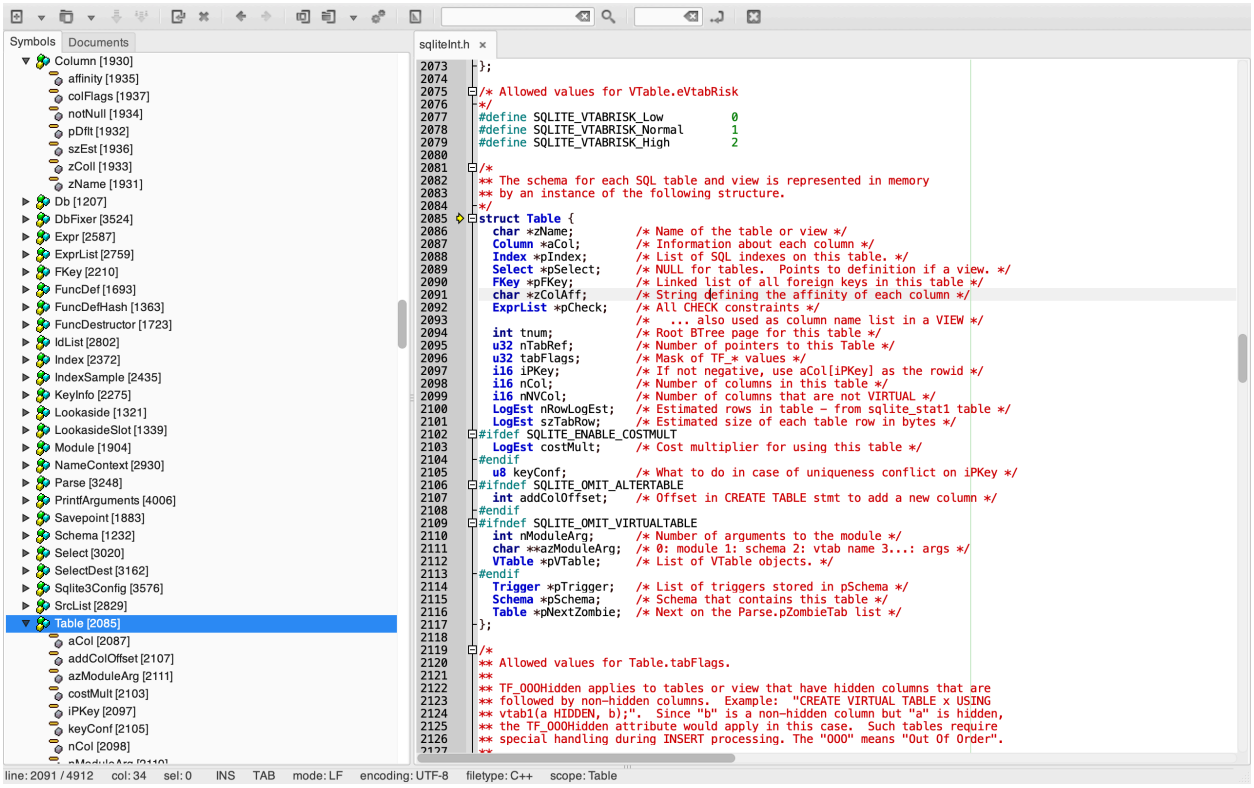
```
puts("Metadata help: ") ;
```

Then I configured and built a new version of sqlite3. Below was the result of my second build, which showed that I was going in the right direction.



3. Third (Final) Build

- With the success of the second build, I determined the remaining tasks is to find out how to print out the name of the table we are inserting into and the name of the columns of that table.
- I came back to the local variables that caught my attention during the process of the second build. In those, I decided to research about “Table pTab” first because it looks the most promising. In the insert situation in this project, we assume our insert command has the correct table name, so pTab should be initialized already before the point of our code modification.
- I tried to search for the type “Table” to understand more about what pTab has that I can use. However, I failed to find additional information about Table in insert.c. But I found out that insert.c includes sqliteInt.h (the internal header file of sqlite), so I guess Table can be in there. And it is.
- At line 2081 of sqliteInt.h, the content of struct Table is represented:



- There were several attributes of struct Table that seem to be useful:

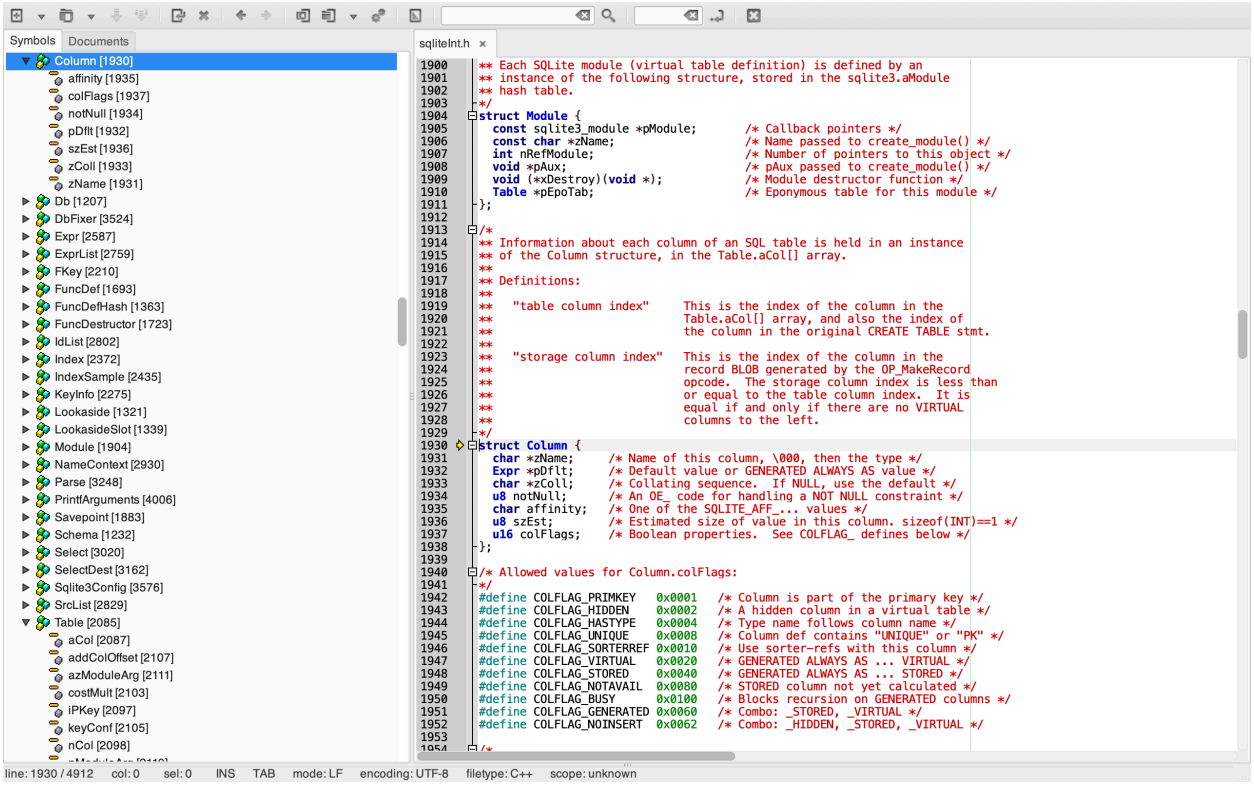
  char *zName; /* Name of the table or view */

  Column *aCol; /* Information about each column */

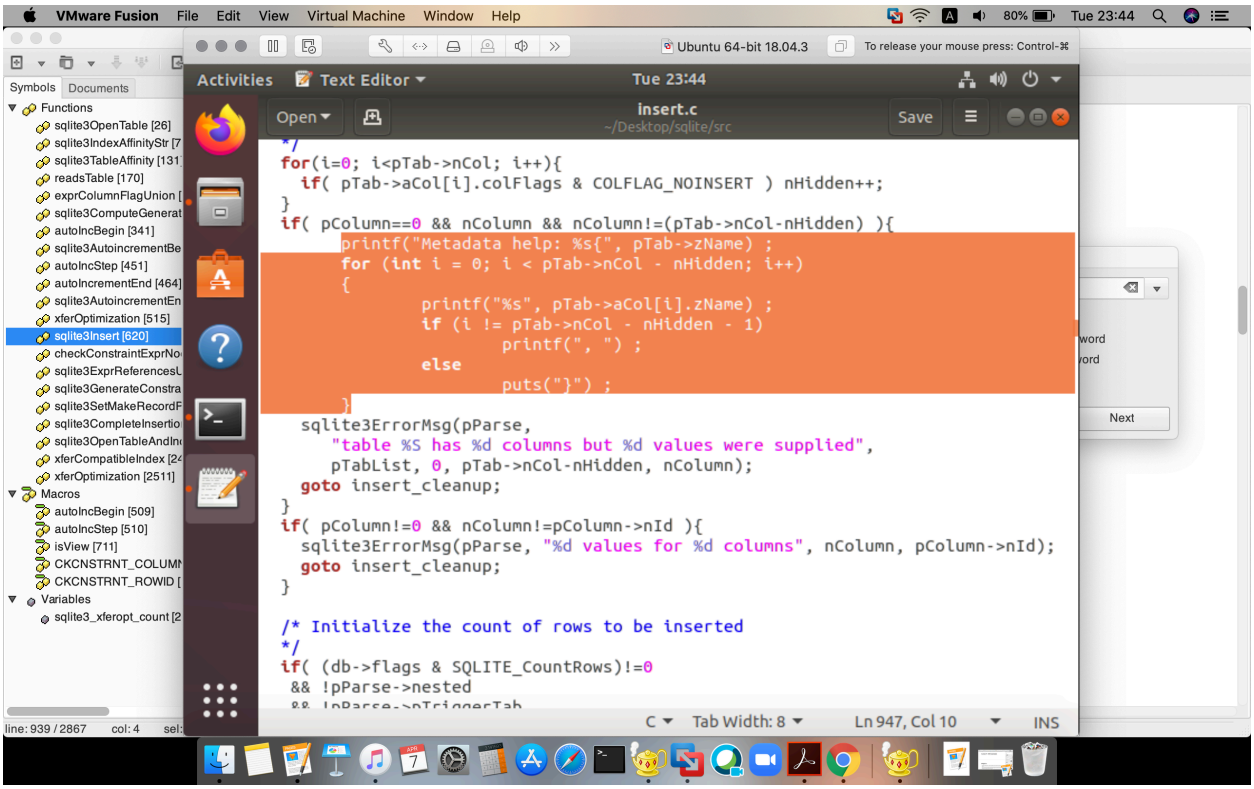
  i16 nCol; /* Number of columns in this table */

- I continued to look for the Column type, which can be found at line 1914. It has:

  char *zName; /* Name of this column, \000, then the type */



- So, I figured out a way to access table name and column name: pTab->zName and pTab->aCol[i].zName, with i as the index of the column
- My algorithm for printing out the schema of the table would be running the for loop n times, n is the number of columns of the table. In each iteration, I will use the index of the loop to access the index of the column. However, in the original error message, I saw that the number of columns the author used was “pTab->nCol-nHidden”. nHidden is “Number of hidden columns if TABLE is virtual” as documented by the author. I did not know what is virtual table, but I followed his notation for the number of columns.
- Finally, my complete code modification is highlighted as below:



- And this is the result of the third (final) build with tested input: