Test Plan

Our program was exhaustively tested using a combination of front end UI testing and OUnit tests while leveraging the bisect tool. The functions in the files *column.ml*, *table.,ml*, and *database.ml* — the largest three implementation files upon which our program relies (excluding the files used to implement the program's ability to read and understand user input) are tested by OUnit. We note that there are some non-tested functions including *col_size* and *elemtype_of_stringparse* from the *column.ml* file, *prt_des* and *str_coltyp* from the *table.ml* file, and *schema* and *select_max_min* from the *database.ml* file. Some functions were challenging to test via OUnit or were more effectively tested using UI testing. In addition to building a full OUnit test suite for the backend, we also thoroughly manually tested the parser and lexer of the program, as well as the program's *main* function. Since the user interface requires human input, manual testing was the most effective option for us. This helped us catch errors that we may have not anticipated without trying to run the program ourselves. This manual testing was also the foundation for many of the parsing error catches that we implemented. We believe that given the application of our project, front end testing with human input was an equally important aspect of our testing development as the backend OUnit test suite.

The *column.ml*, *table.ml*, and *database.ml* were tested by OUnit using primarily black box testing, with many functions being tested on all of their internal pattern matches. Through coding the functions we saw the vulnerabilities and

decided which parts of the program were most important to test. Testing the parts of the system that could break was more important to use than testing every possible line of code. After using black box testing, we also added a few glass box test cases.

We believe that our testing approach demonstrates the correctness of the system because, for our testing, we focused on the parts of the system where we saw vulnerabilities, and because our general testing was done to ensure that the overall application fulfilled its specification — our vision for the product. To us, it's more important to test the parts of the program that could crash than to test every line of code, so we started off with those safety tests first before going on to test other lines of code. We also saw redundancy in cases of glassbox testing for functions that returned types from strings. In general, we believe that through creating the functions and writing the code we saw the vulnerabilities and thought the best course of action was to utilize blackbox testing. Additionally, our testing approach resulted in a bisect of 85% for Column.ml, 72% for Database.ml, and 85% for Table.ml. Our functions that are vulnerable to bugs are fully tested and the majority of our other functions are tested as well. Overall, our team considers this proof that our testing approach is correct for our system.