Abraham Meza

CS 457/657 – 1001

Dongfang Zhao

September 29, 2021

Project 1: Metadata Management

Implementation (Creation, Deletion, Update, Query):

createDatabase(): creates a database by initializing a directory using specified database name (3rd arg.). By passing the root/parent path, the function ensures that users can only create a database while in the root/parent. If they are not, then the function will place them at the root before creating a new database. The function prevents a user from creating an existing database (no overwrite).

deleteDatabase(): deletes a database by recursively deleting a database using system calls, removing the specified database name (3rd arg.). By passing the root/parent path, the function ensures that users can only delete a database while in the root/parent. If they are not, then the function places them at the root before delete an existing database. The function prevents the user from deleting a database that doesn't exist.

useDatabase(): allows a user to access a specific database (2nd arg.) to create, use, modify, and delete tables containing metadata. By passing the root/parent path, the function ensures that users can only access databases while in the root/parent. If they are not, then the function will place them at the root before accessing a new database. The function prevents a user from accessing a database that doesn't exist.

createTable(): creates a table by exporting a .txt file with the table name that contains specified attribute names/types in a tabular format. The function ensures that users are unable to create tables outside a database. The function prevents users from creating an existing table (no overwrite).

deleteTable(): deletes a table by removing the file with the specified table name (deallocated properly). The function prevents a user from deleting a table outside the database. The function prevents users from removing a table/file that doesn't exist.

queryTable(): fetches table information by reading the specified file/table name, using the query to filter out unnecessary information. The function prevents users from fetching information outside a database. The function prevents users from fetching information from a table/file that doesn't exist.

updateTable(): modifies an existing table by appending attribute name/type information to the target file. The function prevents users from modifying an existing table outside a database. The function prevents users from modifying a table/file that doesn't exist.

Compile Instructions:

IMPORTANT NOTES:

- The test script provided behaves weirdly, where the first line read by the program includes the title comment, followed by the first command (CREATE DATABASE db_1;). Since my program ignores all SQL comment lines (any string that begins with "--"), the first command is completely ignored since it's considered a part of the initial comment "CS457 PA1" (same line). As a result, please use the script provided in the submission for proper functionality. The only difference is that the file does not have the initial comment ("CS457 PA1").
    - Please refer to Appendix [A] for a detailed analysis of the issue.

Using the provided Makefile and PA1_test.sql script, type "make"

To run the program, type

- ./db (NO SCRIPT)
- ./db < PA1_test.sql (PIPELINE SCRIPT)

To clean up, type "make clean"

Documentation:

As recommended, the program's design utilizes a file system to manage databases as directories, while using files within those directories to manage table metadata. This method of organization allows there to be clear boundaries between instances, ensuring that there is no collision between databases, and their corresponding tables.

Upon execution, the program takes in a user-input until it reaches the delimiter (';'), then parses the input into a vector to record each argument. Knowing that the base command (i.e., create, drop, use, etc.) will be the first argument, the program fetches the keyword, then compares it to several if-statements representing a tree-like structure. From there, after verifying the command's keyword, the program compares the second argument to determine which structure the command is being used for, allowing the program to choose which algorithm/function is most appropriate. For example, >>CREATE TABLE tbl_1; parses its arguments based on words, resulting in the following words to be pushed into the vector: [CREATE, TABLE, tbl_1]. From there, the program fetches the keyword, and compares it to the available commands (e.g., create, drop, use, alter, and select). After realizing the keyword of the command is "create", the program determines which structure is being used (e.g., database or table). After realizing that the structure is a "table", the program checks to make sure that the user is not outside a database by comparing the root directory to their current directory (getcwd()), then executes createTable(). If the user is outside a database while executing a command, then the program will prompt the user with a warning. Each command undergoes the same process, comparing each argument to a series of if-statements to predict which function is most appropriate, while checking that each parameter from the user-input is valid before continuing. Please refer to Appendix [B] for the list of commands and their formatting.

Based on the structure that the command is being executed on, the program ensures that each structure's command can only be executed in its corresponding environment. In this case, a user cannot execute a "database" command unless they are in root/parent directory. Should the program detect that the user is executing a "database" command within a directory, the program will place the user at the root, then execute the appropriate command. On the other hand, if the program detects that the user is executing a "table" command outside a directory, the program will prompt an error message that the user needs to execute a "use" command first to choose the directory they want to modify.

The program manages multiples databases within a filesystem by assigning a dedicated directory to each database. As a result, if a user has not accessed a database using the "use" command prior to executing a modification, the user will be unable to modify a database (e.g., create, delete, modify table metadata) as the program does not know where to apply the changes. By implementing the system header <sys/stat.h>, the program creates and maneuvers between directories by using built-in file functions, while continuously checking against an if-statement to see if the user is in a root directory before executing directory operations. Using a helper-variable to record the return value of the built-in functions, commands such "create" check to see if a directory currently exists. If the return value represents an unsuccessful execution (-1), then the designated directory already exists and cannot be overwritten. However, in order to delete an existing directory, I utilized the standard library's (iostream) built-in system() to pass the bash command "rm -r [directory]" to the host command processor to recursively delete an existing directory. Though remove() also deletes a designated directory, I utilized system() to account for non-empty databases that require a recursive process to delete database contents before deleting the entire directory. Like "create", "drop" has a helper-variable that records the return value of the built-in function to determine if the system call was successful.

The program manages multiple tables within a database using dedicated .txt files for each table. In other words, using the user-input as guidance, the program implements table changes only to the desired file to avoid potential data collision. Using the fstream library, the program reads, writes, and appends table metadata to the user-designated file. For commands such as "create", "drop", "alter", and "select", each command has a built-in algorithm that uses file.open() to check if a file already exists. For "create", should the file exist, the program prompts an error message that the user cannot create a new file as it would overwrite existing data. For all other commands, if the file doesn't exist, then the program prompts an error message that the user cannot delete/modify/access a file that doesn't exist. For most commands, the fstream library has built-in functions that create/modify/access file contents appropriately without the need of low-level programming. However, in order to delete a table, I utilized the standard library's (iostream) built-in remove() to point to the file's name/path that had to be deleted. By deleting the file, the table metadata was properly deallocated without complication. Using the standard library allowed me to avoid hard-coding system calls such as "rm [filename]" to drop a table. Each function has a helper variable that determines if the command was successful, dictating what was prompted to the user.

Once the program completed the test script successfully, through the use of conditional statements, I integrated a variety of user-input fail-safes to prevent parsing issues (i.e., ignore SQL comments, ignore empty commands, check for sufficient parameters, etc.). After reading the initial user-input, I implemented try/catch statements to ensure that the parsed data was properly stored in the vector instance. If the user provided insufficient parameters, then the program would prompt the user

that they did not include enough information. In addition, each argument check has a default else-statement that prompts an error message if part of the user command is not recognized (i.e., keyword, structure, name, action, etc.), minimizing user-input issues, while promoting a friendly user-interface. Furthermore, I used the standard library's transform() to ignore case sensitivity within the user-input by capitalizing (::toupper) each parameter that didn't represent structure name or data. As a result, case sensitivity does not affect if-statement comparisons, improving the program's functionality.

All in all, the program utilizes a variety of library functions (refer to Appendix [C]) to simulate an SQLite database. Using a tree-like structure, and user-input fail-safes, the program is well-rounded and accurately creates database instances that promote organization and efficiency.

Program Output:

```
didgit_10@Abrahams-MacBook-Pro-2 ameza2_pa1 % make
g++ -Wall -std=c++11  -c db.cpp -o db.o
g++ -Wall -std=c++11  db.o -lm  -o db
g++ db.o -o db -Wall -std=c++11
didgit_10@Abrahams-MacBook-Pro-2 ameza2_pa1 % ./db < PA1_test.sql

////////////////////////////
/////  Database Program /////
////////////////////////////

Database db_1 created.
!Failed to create database db_1 because it already exists.
Database db_2 created.
Database db_2 deleted.
!Failed to delete db_2 because it does not exist.
Database db_2 created.
Using database db_1.
Table tbl_1 created.
!Failed to create table tbl_1 because it already exists.
Table tbl_1 deleted.
!Failed to delete tbl_1 because it does not exist.
Table tbl_1 created.
a1 int    |    a2 varchar(20)
Table tbl_1 modified.
a1 int    |    a2 varchar(20)    |    a3 float
Table tbl_2 created.
a3 float    |    a4 char(20)
Using database db_2.
!Failed to query table tbl_1 because it does not exist.
Table tbl_1 created.
a3 float    |    a4 char(20)
All done.
didgit_10@Abrahams-MacBook-Pro-2 ameza2_pa1 %
```

Appendix A

After implementing an algorithm that ignores SQL comments in a C++ environment, my program began to have issues where the first (CREATE DATABASE db_1;) was ignored by the program's getline(). After excessive testing, I realized that the file was pipelining the title comment "CS 457 PA1" and the "CREATE DATABASE" command on the same line. As a result, once my program detected "--" at the front of the string, it ignored the entire line as it was perceived as a comment. The result of the ignored command is displayed in Fig. 1.

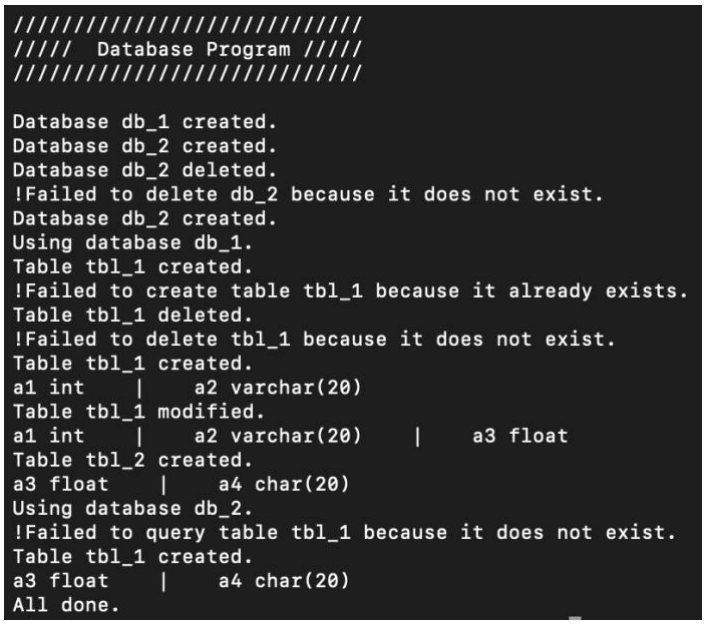

Fig. 1: A screenshot of the program's output using the original PA1_test.sql script. The output is missing the first "create database" command, not showing the database overwrite failsafe.

Unfortunately, I have not found the reason as why the comment and command were grouped together (refer to Fig. 2), leading me to modify the test script by removing all comments above the scripted code.

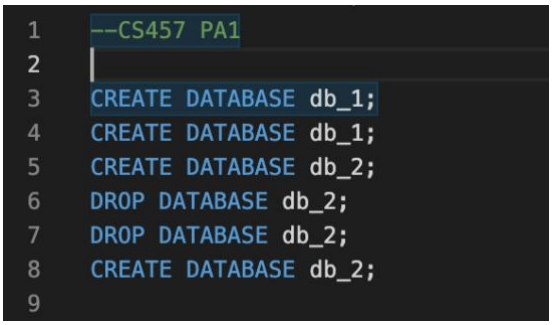

Fig. 2: A screenshot showing the grouping between the title comment and first script command, visually seen using VS Studio Code.

Attached to my submission is a debugging program that reads in the pipelined file input (removing '\n' and '\r' characters) to show how the script handles line-by-line input. Based on the results in Fig. 3, the first line read includes the comment, followed by the first command (CREATE DATABASE db_1;).

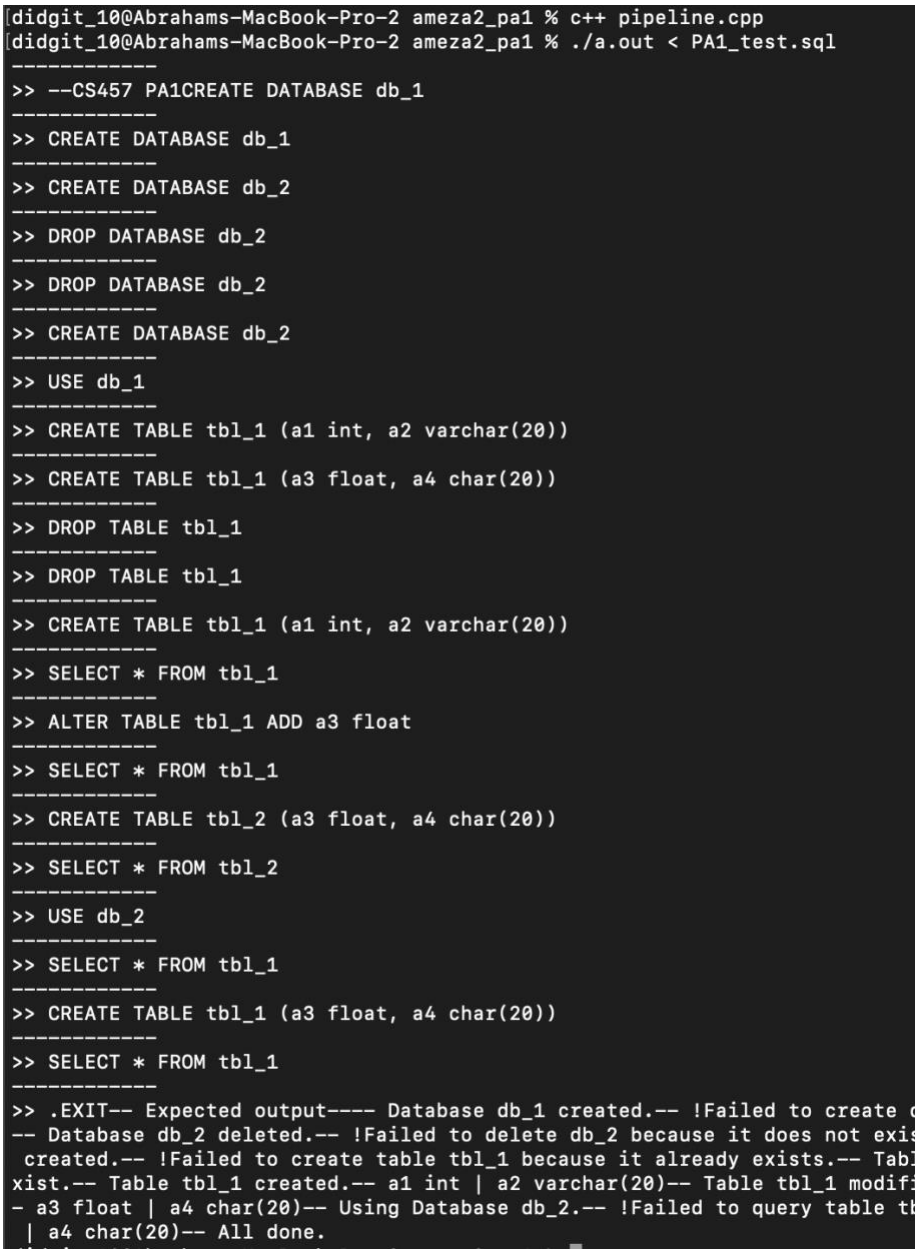

Fig. 3: A screenshot of pipeline.cpp, a debugging program, that shows how the file is pipelined into the program (line-by-line input).

Given the assignment instructions, I was under the impression that if the command has unnecessary information, then the command will output an error. Please refer to the following example for additional clarification.

Example:

>> --This program creates a database CREATE DATABASE db_1;

>> [ERROR]

>> CREATE DATABASE db_1;

>> Database db_1 created.

Compile Instructions:

The following instructions are to be used to verify the findings of Fig. 3:

Access the Debugging Directory

Using pipeline.cpp and the original test script (“PA1_test.sql”), type the following:

- c++ pipeline.cpp

To run the program, type:

- ./a.out < PA1_test.sql

Appendix B

CREATE:

- Database = [keyword, structure, name]
- Table = [keyword, structure, name, attr. name, attr. type, attr. name, …]

DROP:

- Database/Table = [keyword, structure, name]

USE:

- Database = [keyword, name]

SELECT:

- Table = [keyword, query, target, name]

ALTER:

- Table = [keyword, structure, name, action, attr.name, attr. type, attr.name, …]

Appendix C

Program Libraries:

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <iomanip>
#include <cstring>
#include <algorithm>
#include <string>
#include <climits>
```

Program Headers:

```
#include <unistd.h>
#include <sys/stat.h>
```