Abraham Meza

CS 457/657 – 1001

Dongfang Zhao

October 20, 2021

Project 2: Basic Data Manipulation

Implementation (Creation, Deletion, Update, Query):

insertTuple(): inserts table tuple data by appending given information to the next row of the table. The function prevents users from inserting empty records/samples of data. In addition, the function has a built-in failsafe that prevents users from adding multiple tuples in a single line, while also ensuring that the user can only pass tuples that do not exceed the number of current attributes. If the inserted tuple is missing attribute information, the function initializes empty cells to NULL. The function prevents users from inserting information outside a database. The function prevents users from inserting table/file information to a file that doesn’t exist.

deleteTuple(): deletes tuple data based on a given 'where' comparison (limited to a single condition). The function traverses through each record's attribute data and conducts a conditional statement to filter out data. The indexes (line num.) of values that passed the comparison are pushed into a list. The list of indexes is then referenced to omit corresponding tuple data. To update the table, the function creates a new copy of the file, containing relevant metadata information. Once the target data has been omitted, the function deletes the old state and renames the new file to match the previous. The function prevents users from deleting information outside a database. The function prevents users from deleting table/file information from a file that doesn’t exist.

updateTuple(): modifies existing tuple data based on a 'where' comparison (limited to a single condition). The function traverse through each record's attribute data and conducts conditional statement to filter out data. The indexes (line num.) of values that passed the comparison are pushed into a list. The list of indexes is then referenced to modifies corresponding tuple data. After modifying the value, the function generates a new copy of the table containing updated information, deletes the old state, then renames the new file to match the previous. The function prevents users from updating information outside a database. The function prevents users from updating table/file information to a 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 can utilize 'where' as a comparison condition when filtering table data (limited to a single condition). The function creates a list of indexes that correspond to successful comparisons, then uses the indexes to create a temporary file that contains relevant data. After filtering out the data, the function fetches table data from the temp file using getline(). After fetching table information, the temp file is deleted. 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.

Helper Function Implementation:

isNumber(): identifies if the value of a string is a real number by looking for non-numerical characters. Returns a Boolean value.

- If the string contains only the following symbols, then the string is a number: “(0123456789.)”.
    - Return True
- Else the string contains non-numerical symbols.
    - Return False

Compile Instructions:

IMPORTANT NOTES:

- SAME AS PROJECT 1 (Refer to PA1 Documentation, Appendix A)
    - 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 CS457_PA2;). 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 PA2 test script” (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 PA2 test script”).

Using the provided Makefile and PA2_test.sql script, type “make”

To run the program, type

- ./db (NO SCRIPT)
- ./db < PA2_test.sql (PIPELINE SCRIPT)

To clean up, type “make clean”

Documentation:

As discussed in class, the program’s design organizes table data by implementing a row-based and column-based implementation to effectively fetch, insert, modify, and delete tuple data. Using vectors to store row/column values, this method allows there to be clear boundaries between tuple data instances, promoting data integrity by preventing unauthorized access to information (i.e., select, modify, delete, etc.). In other words, information will only be changed should the tuple satisfy the corresponding conditional instruction.

Using the same structure as Project 1, the program takes in a user-input until it reaches the delimiter (‘;’), then parses the input into a vector to record each argument. Using a series of tree-like if-statements, the program compares each argument to identify the most appropriate function, checking that each parameter from the user-input is valid before continuing. Like Project 1, the program ensures that each structure’s command can only be executed in its corresponding environment. In this case, a user cannot execute a “table” command (e.g., select, insert, delete, and update) unless they are in within a database directory. Should the program detect 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. Please refer to Appendix [A] for the list of new table commands and their formatting.

Using insertTuple(), the program takes in a user’s parameterized data and appends the information to the end of an existing table/file to represent a new entry (row-based). By this logic, insertTuple() organizes table data based on their creation order. To append new tuple data, the function looks for the existing table. Should the table not exist, the program prompts an error message indicating that the user cannot insert into a non-existing instance. If the table exists, the function reads the first line of the file (getline()) to determine the current number of attributes. By parsing the first line and identifying each attribute’s name, the program counts the number of existing attributes to determine the max number of arguments the user is allowed to pass when inserting a new tuple. If the user’s attribute values exceed the current number of allowed attributes (numAttributes > maxAttributes), then the program prompts an error message indicating that the user cannot insert the tuple. This failsafe indicates that the “INSERT” command can only add a single tuple at a time. If the user’s input does not violate any of the failsafes, then the program formats the user’s input into a tuple entry and appends the information to the end of an existing file. If the user’s tuple entry is missing attribute data (e.g., insert into product values (3, 'SingleTouch’)), then the program initializes the empty cells to NULL to define proper instance behavior. Using a counter to track successful tuple entries, the program prompts a confirmation statement illustrating the number of successful insertions.

Using deleteTuple(), the program takes in a user’s ‘where’ conditional statement and removes all tuple data that satisfies the condition. To delete tuple data, the function looks for an existing table to read. Should the table not exist, the program prompts an error message indicating that the user cannot delete data from a non-existing instance. If the table exists, the function reads the first line of the file (getline()) to determine the table’s attribute categories (column-based). By parsing the first line and identifying each attribute’s name, the program conducts a comparison of each attribute against the condition’s target attribute to determine which column (attribute index) will undergo the ‘where’ comparison. If the program does not detect the target attribute within the table, then the program will prompt the user that nothing was deleted. However, if the target attribute resides in the table, the program will record the attribute’s index, and use it to fetch each tuple’s corresponding attribute value. After reading and fetching all relevant attribute value(s), the program uses the ‘where’ condition argument to test each value. Based on the condition’s targetOperand, the program uses isNumber() as a helper function to determine if the targetOperand is a number. If targetOperand is a number, then the program tests each attribute value using numerical comparison statements. However, if targetOperand is not a number, this indicates that the attribute item is a string/phrase, prompting the program to test each tuple value using a string conditional statement. After determining the datatype of targetOperand, the program conducts the appropriate comparisons using the condition’s operator (i.e., >, <, >=, =, !=, etc.). If the tuple value passes the comparison, the tuple’s file index is recorded into a list representing pending tuple deletions (targetTuple) (row-based). The programs ‘where’ comparison algorithm also has built-in failsafe that prompts an error message if the user’s conditional operator is invalid. After determining which tuples satisfy the ‘where’ comparison, the program creates a temporary file called “copyTemp” to export the updated table. The program then exports the current state of the table to the temp file. If the program detects the index of the to-be-deleted tuple, the program omits the data and proceeds to export the next tuple in line. As a result, the program exports an updated table by omitting specific tuples to represent a deletion instance. Once the updated table has been exported, the program deletes the previous file/table, and renames the newly updated file/table to that of the previous, maintaining metadata properties (i.e., file name, file directory, etc.). Using a counter to track successful tuple deletions (omits), the program prompts a confirmation statement illustrating the number of successful deletions.

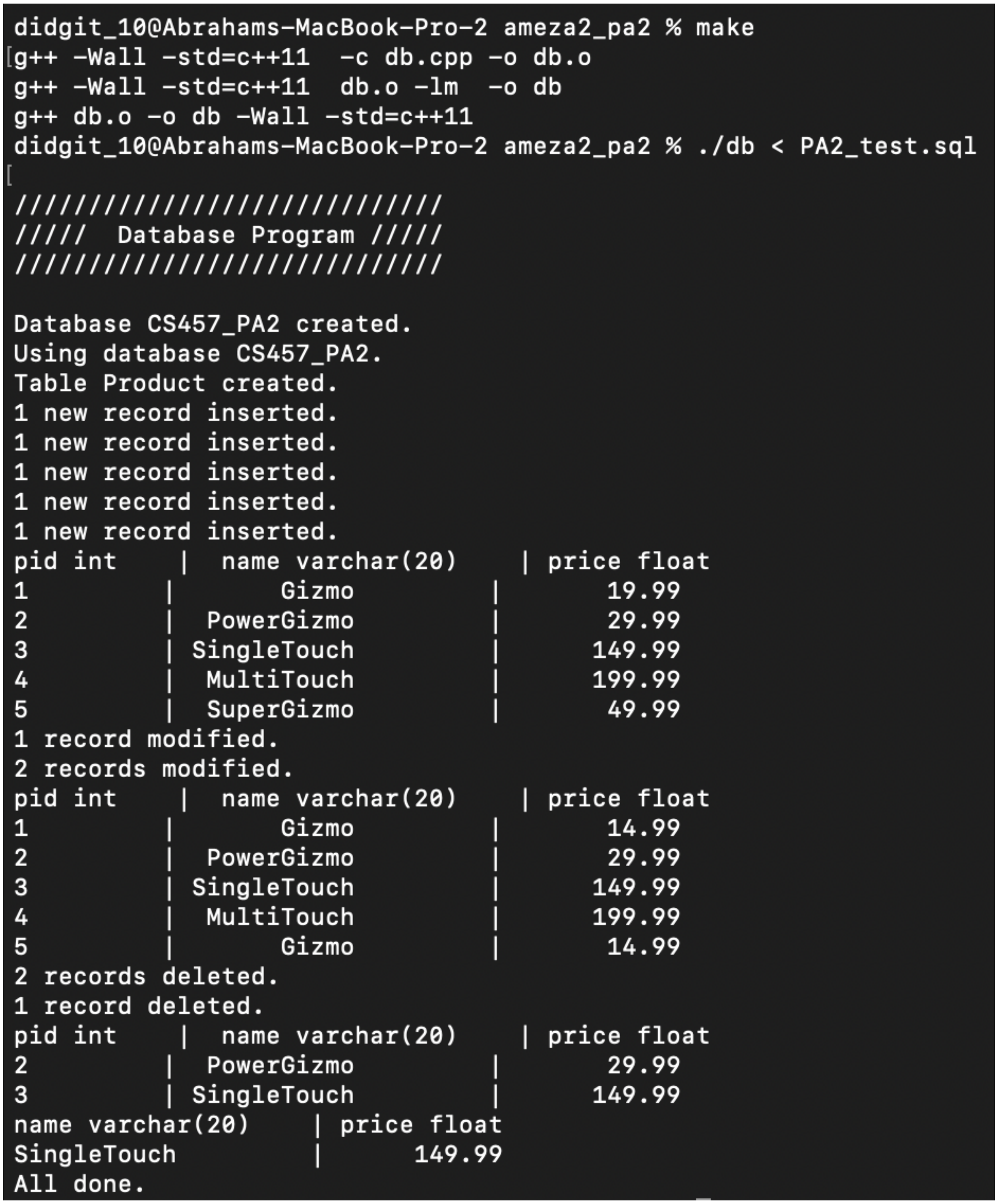
Using the same column-based algorithm as deleteTuple(), updateTuple() takes in a user's 'where' conditional statement and updates each attribute entry that satisfies the condition. Similar to deleteTuple(), the function looks for an existing table to read. Should the table not exist, the program prompts an error message indicating that the user cannot update data from a non-existing instance. If the table exists, the function reads the first line of the file (getline()) to determine the table's attribute categories (column-based). After determining a list of current attributes, the function conducts a series of comparisons against each target attribute (e.g., update attribute and condition attribute) to check to see if both the conditional and update attributes exist. If either attribute is not a valid category, then the program prompts an error message indicating an invalid user-input. If both attributes were found in the file/table, then the program will record each attribute's index, and use it to fetch each tuple's corresponding value. After reading and fetching all conditional attribute value(s), the program uses the 'where' condition argument to test each value. Using the same comparison algorithm as deleteTuple(), updateTuple() generates a list of tuple indexes that satisfy the given 'where' statement, indicating which tuples need to be updated (targetTuple) (row-based). After determining which tuples satisfy the 'where' comparison, the program creates a temporary file called "copyTemp" to export the updated table. The program then exports the current state of the table to the temp file. If the program detects the index of the to-be-updated tuple, the program iterates through the tuple's attributes until it reaches the attribute that needs to be updated (column-based). Once identified, the program updates/changes the value based on the user's input and writes the updated tuple entry to the temp file. Once the updated table has been exported, the program deletes the previous file/table, and renames the newly updated file/table to that of the previous, maintaining metadata properties (i.e., file name, file directory, etc.). Using a counter to track successful tuple modifications, the program prompts a confirmation statement illustrating the number of successful updates.

To conduct a tuple query, the program utilizes queryTable() to fetch relevant attribute information. For this project, queryTable() was restructured to accept multiple queries to return specific attributes. In addition, queryTable() now accepts an optional 'where' statement to account for conditional filtering. Similar to deleteTable(), queryTable() looks for an existing table to read. Should the table not exist, the program prompts an error message indicating that the user cannot fetch data from a non-existing instance. If the table exists, the function reads the first line of the file (getline()) to determine the table's attribute categories (column-based). After determining a list of current attributes, the program checks to see if the queried attributes exist in the table. If any of the attributes do not exist, then the program will prompt an error message indicating one or more invalid attribute inputs. To preserve Project 1's "SELECT *" implementation, the function checks for a * argument within the list of queries. If found, then the program replaces the user's query with all available attributes. Once the query has been verified, if the query command contains an optional comparison statement, the function conducts a series of comparisons against the condition’s target attribute to determine which column (attribute index) will undergo the ‘where’ comparison. If the program does not detect the target attribute within the table, then the program will prompt the user that the query cannot be executed. However, if the target attribute resides within the table, the program will record the attribute’s index, and use it to fetch each tuple’s corresponding attribute value. After reading and fetching all relevant attribute value(s) (e.g., query attribute list, and condition attribute list), the program uses the ‘where’ condition argument to test each value. Using the same comparison algorithm as deleteTuple(), queryTuple() generates a list of tuple indexes that satisfy the given ‘where’ statement, indicating which tuples need to be fetched (targetTuple) (row-based). After determining which tuples satisfy the ‘where’ comparison, the program creates a temporary file called “copyTemp” to export the relevant table data. The program then exports a new table containing all relevant attribute fields, followed by each tuple’s corresponding values. If the command contained an optional ‘where’ statement, then the program only exports tuple data that satisfied the ‘where’ comparison. Once the relevant table has been exported, the program reads the updated temp file/table (getline()), and displays the contents to the user. After the fetching the relevant table information, the program deletes the temp file to promote data integrity (i.e., reading information, not writing information).

After testing the program against the test scripts and a few edge cases, I noticed that there were some minor modifications I had to implement to ensure that all functionalities from Project 1 and Project 2 worked as intended. In Project 1, updateTable() modified an existing table by appending new attributes to the target file. However, since we had not yet implemented tuple data, the function simply appended all new attributes to the end of the file. Given that tuples are continuously added overtime, I had to modify updateTable() such that all new attribute information was appended to the file’s/table’s header, while keeping all other information unchanged. To implement the concept, I simply took the same approach as modifyTuple(), where the function creates a new file representing an updated table. As a result, updateTable() reads the first line of the original file and appends all new attribute data to the end of the header. The new header is then exported to the newly generated file/table, followed by the original tuple data. Once the updated table has been exported, the program deletes the previous file/table, and renames the newly updated file/table to that of the previous, maintaining metadata properties (i.e., file name, file directory, etc.). In addition to updateTable()’s behavior, I also noticed that the test script uses different letter cases to represent file/table names (i.e., Product, product, etc.). Given that Linux uses a case-sensitive file system, I implemented a case modification algorithm that capitalizes the first character of each file name, while ensuring that all following characters are lowercased. As a result, the program works in all file system environments.

All in all, the program utilizes a variety of library functions (refer to Appendix [B]) to simulate an SQLite database. Using a tree-like structure, and user-input fail-safes, the program is well-rounded as it properly executes basic data manipulation, promoting database organization and efficiency.

Program Output:

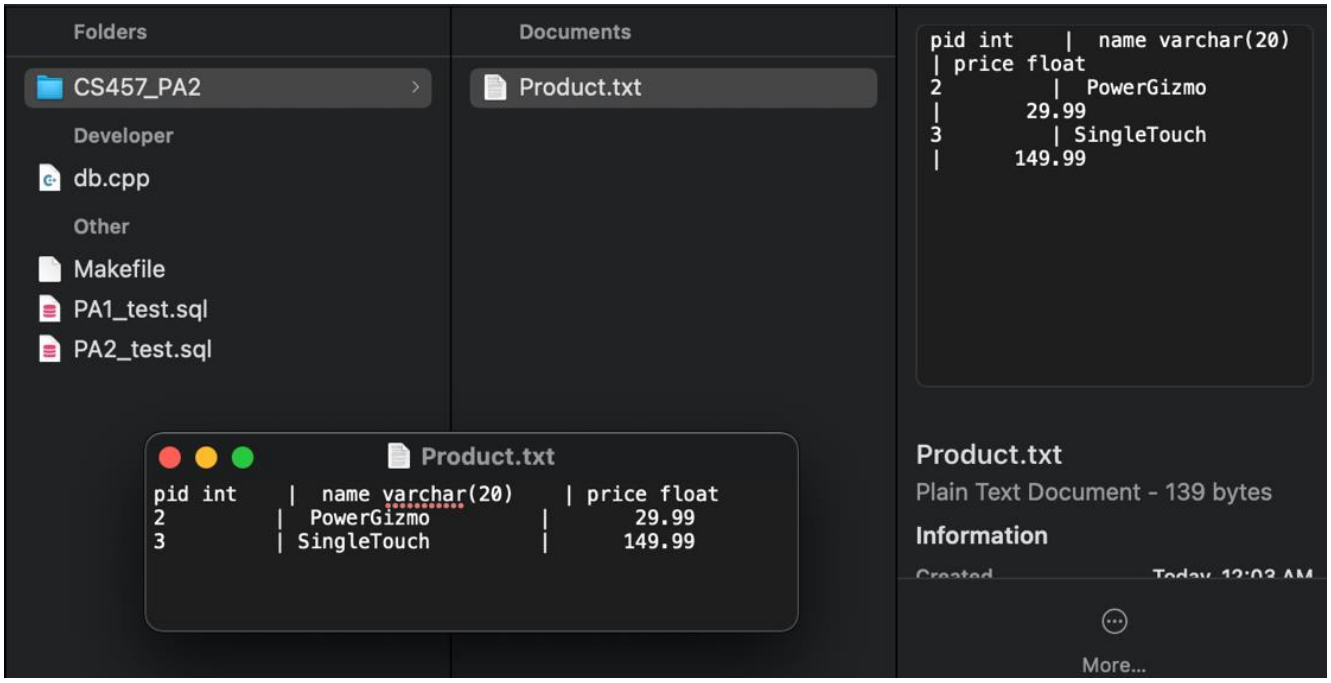


```
didgit_10@Abrahams-MacBook-Pro-2 ameza2_pa2 % make
g++ -Wall -std=c++11  -c db.cpp -o db.o
g++ -Wall -std=c++11  db.o -lm  -o db
g++ db.o -o db -Wall -std=c++11
didgit_10@Abrahams-MacBook-Pro-2 ameza2_pa2 % ./db < PA2_test.sql

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

Database CS457_PA2 created.
Using database CS457_PA2.
Table Product created.
1 new record inserted.
1 new record inserted.
1 new record inserted.
1 new record inserted.
1 new record inserted.
pid int    |  name varchar(20)    | price float
1         |       Gizmo          |      19.99
2         |  PowerGizmo          |      29.99
3         | SingleTouch          |     149.99
4         |  MultiTouch          |     199.99
5         |  SuperGizmo          |      49.99
1 record modified.
2 records modified.
pid int    |  name varchar(20)    | price float
1         |       Gizmo          |      14.99
2         |  PowerGizmo          |      29.99
3         | SingleTouch          |     149.99
4         |  MultiTouch          |     199.99
5         |       Gizmo          |      14.99
2 records deleted.
1 record deleted.
pid int    |  name varchar(20)    | price float
2         |  PowerGizmo          |      29.99
3         | SingleTouch          |     149.99
name varchar(20)     | price float
SingleTouch          |      149.99
All done.
```

File System View:



```
pid int     |  name varchar(20)
| price float
2          |  PowerGizmo
|        29.99
3          | SingleTouch
|       149.99
```

```
pid int     |  name varchar(20)     | price float
2          |  PowerGizmo           |        29.99
3          | SingleTouch           |       149.99
```

Appendix A

DELETE:

- [FORMAT: keyword target name where attribute condition operand]

INSERT:

- [FORMAT: keyword indication name values(attri.values)]

UPDATE:

- [FORMAT: keyword name set attribute value where attribute condition operand]

SELECT:

- [FORMAT: keyword query indication name (where attribute condition operand)]

Appendix B

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>
```