Yeamin Chowdhury
Project 2
CS 457
April 3, 2023

Programming Assignment 2

Design

The program is designed to create databases exclusively within the home directory, where the program is located. Tables can only be created inside a database and not outside it. To create a database and its tables, the SQL USE query must be employed. If a database isn't specified, creating or querying tables is not possible. By using the USE command, the current working directory can be set as the selected database, enabling file creation within it through CREATE commands. This means that although a user can have multiple directories (databases) branching from the home directory, nesting databases is not allowed. However, users can create multiple tables within each database. It's essential to remember that tables cannot be created without first selecting a database. Utilizing the INSERT commands, users can add data according to the defined schema of the table. Users can remove values from a table within a database using the DELETE commands, indicating the specific values to be eliminated in the command line. Additionally, users can modify values by employing the UPDATE commands, specifying the values targeted for updating and the new values to be assigned. Lastly, the SELECT commands offer two types of functionality. Executing 'SELECT * FROM Product;' will display all values in the chosen table, whereas 'SELECT name, price FROM product WHERE pid != 2;' will show only the specified attributes and their associated records. The program will terminate if the user inputs '.EXIT' or if Ctrl+C is pressed.

Imports

To utilize the functionalities of obtaining program location, creating folders as a database, and creating files as tables, the imported libraries were os and shutil. The deletion of folders along with their contained files was carried out through the utilization of shutil. For all other tasks such as creating, deleting, and changing directories, the OS library was used.

Functions

main()

The primary function anticipates user input consisting of commands accompanied by arguments. It can accommodate up to three lines of valid input unless a line ends with a semicolon. The input process terminates if a semicolon appears at the end of a line. Otherwise, it asks until the 3rd line of commands to pass the commands for conditional checks of functionalities. The contents of these three lines are consolidated into a single string, which is then passed to the parsing stage. The string is converted to uppercase and assessed to determine if the input is '.EXIT'. If this condition is met, the program terminates; otherwise, the string is forwarded to the command() function for command execution.

create_database(database_name):

This function takes a database name as its parameter and creates a folder as the database. The database is created in the home directory(where the program is located) and can not be created anywhere else. Nested folders are not allowed.
If the database with the entered database_name already exists, then it will print an error message.

use_database(database_name)

This function takes a database name as its parameter and entered the folder as the database, only from the home directory. If the database does not exit with the specified name, it prints an error message. This function must be used before the creation, deletion, alteration, and selection of the tables.

create_table(table_name, variable_list)

This function takes a table_name as its parameter and creates a table with the specified name, otherwise, prints an error message if it already exists in the database. The USE commands with a database name must be used before creating a table because the creation of files outside of any databases is not allowed.
This function also takes a list of variable that was entered in the command to add schema in the table. The variables from the commands are parsed in the command function and passed to the create_table function. The variables are printed inside the table, joined by a “ | ”.

select_table(table_name)

This function accepts a table_name parameter and accesses a table with the given name in the database; if it cannot find the specified table, it displays an error message and exits. Subsequently, it opens the file associated with the table name and reads its contents. The function then outputs the file contents to the terminal before closing the file.

select_set(table_name, attribute_name, operator_att, operator, operator_value)

This function accepts a table_name parameter and generates a table with the specified name. If the table does not exist in the database, it outputs an error message. Prior to selecting a table, the USE command with a database name must be employed, as multiple tables can reside within a single database. The function identifies the attributes to be displayed and prints records that meet the specified criteria. Providing an incorrect value in place of the proper type (e.g., price > 'GIZMO') will trigger an error and return.

An auxiliary function, "**set_operation(two_d_list, attribute_index, operator, operator_value),**" facilitates the evaluation of the appropriate operator and the display of values that correspond to the given condition.

insert_into(table_name, value_list)

This function accepts a table_name as a parameter and generates a table with the specified name, or displays an error message if the table is not present in the database. The USE command, accompanied by a database name, must be employed prior to selecting a table, as multiple tables may coexist within a single database. The number of parameters provided in the command line must correspond to the existing table attributes. If there is a mismatch, an error is printed and control returns to the main function. If the parameters match, the new set of parameters is appended to the table following the existing records.

delete_from(table_name, attribute_name, operator, operator_value)

This function takes a table_name parameter and creates a table using the provided name. If the table is not found in the database, an error message is displayed. Before selecting a table, the USE command must be applied in conjunction with a database name, as a single database can contain multiple tables.

A supplementary function, **delete_operation(operator, two_d_list, attribute_index, operator_value)**, is employed to evaluate the suitable operator and remove values that match the specified condition. If the conditional part of the commands contains incorrect data types, an error message will be displayed and the process will return to the main function.

update_table(table_name, set_name, attribute_name, operator, operator_value, new_value)

This function, like the insert_into and delete_from functions, necessitates the correct table_name and prior utilization of USE commands. An auxiliary function, "**update_operation(operator, two_d_list, attribute_index, set_index, operator_value, new_value)**", is employed to evaluate the suitable operator and update values corresponding to the specified condition. If the conditional portion of the commands contains incorrect data types, an error message will be displayed and the function will return to the message and return to main.

NOTE

For the select_set, insert_into, delete_from, and update_table functions, the following applies: If the conditional segment of the commands contains invalid data types, an error message will appear, and the function will revert to the message before returning to the main function. When inputting select, delete, insert and update commands, make sure that each line does not have a space at the end. This will raise errors.

Test Script

CREATE DATABASE CS457_PA2;

-> (Database CS457_PA2 created.)

USE CS457_PA2;

-> (Using database CS457_PA2.)

CREATE TABLE Product (pid int, name varchar(20), price float);

```
 -> (Table PRODUCT created.)

insert into Product values(1, 'Gizmo', 19.99);

->  1 new record inserted.

select * from Product;

-> 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

update Product
set name = 'Gizmo'
where name = 'SuperGizmo';

-> 1 record modified.

‘update Product
set price = 14.99
where name = 'Gizmo';

->2 record modified.

delete from product
where name = 'Gizmo';

->2 records deleted.

delete from product
where price > 150;

->1 record deleted.

select name, price
from product
```

where pid != 2;

```
->PID INT | NAME VARCHAR(20) | PRICE FLOAT
2 | 'POWERGIZMO' | 29.99
3 | 'SINGLETOUCH' | 149.99
```