Yeamin Chowdhury
Project 3
CS 457
April 24, 2023

Programming Assignment 3

Design

This program is designed to create databases exclusively in the home directory where the program resides. Tables can only be made within a database, not outside. The SQL USE query must be used to create a database and its tables. Without specifying a database, creating or querying tables is impossible. By employing the USE command, the current working directory is set as the chosen database, allowing file creation within it through CREATE commands. Although users can have multiple directories (databases) from the home directory, nesting databases is prohibited. However, multiple tables can be created within each database. Keep in mind that tables cannot be created without first selecting a database. Users can add data according to the table's defined schema using INSERT commands. The SELECT command offers four different functionalities: displaying everything from the selected table; performing inner joins on two tables with the default join command; performing inner joins on two tables with the inner join command and performing left outer joins. Finally, the program can be exited using the ".exit" command.

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.

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.

select_join(t_dict, a_dict, operator)

This function takes t_dict parameter to take the table names with their alias and a_dict parameter to take the attribute names to take the attributes that will be compared. It also takes a operator, which is "=" for this program. Select_join performs inner joins based on the specified attributes and a condition, which in this case is equality. First, it checks if the current working directory matches the program's directory and verifies if the tables exist. It prints an error message if either of these conditions is not met. After reading the content of the tables, the function stores each line as a list of items. It then creates a cross product of the two lists, using the **cross_product** function. Then, it checks if the given attributes for comparison are valid. If it is not, prints error and returns to main. If it is valid, runs the comparison and find the indices of the rows that matches the condition with help of "**indices_of_records_that_match_condition**". With those indices, this function allows us to print or display the values that should be received after performing a join or inner join on two tables. The current version of this function, operates two commands: **join**(with where keyword**) and inner join.**

select_join_outer_left(t_dict, a_dict, operator)

This function takes t_dict parameter to take the table names with their alias and a_dict parameter to take the attribute names to take the attributes that will be compared. It also takes a operator, which is "=" for this program. Select_join performs inner joins based on the specified attributes and a condition, which in this case is equality. First, it checks if the current working directory matches the program's directory and verifies if the tables exist. It prints an error message if either of these conditions is not met. After reading the content of the tables, the function stores each line as a list of items. It then creates a cross product of the two lists, using the **cross_product** function. Then, it checks if the given attributes for comparison are valid. If it is not, prints error and returns to main. If it is valid, runs the comparison and find the indices of the rows that matches the condition with help of "**indices_of_records_that_match_condition**". With those indices, this function allows us to print or display the values that should be received after performing a join or inner join on two tables. Ater performing the inner join, achieved rows performs union operation with the list that contains the values of the left table. It performs the union operation on just the length of each row of the left table. If the left table has only two elements, it will perform the union operation on with just first two elements of the list the that was returns from "**indices_of_records_that_match_condition**". After performing the union operation, the new result is stored in a new list. And prints them with join " | " and "\n" at the end of each row to show left outer join.

cross_product(list_1, list_2)

This function areturns a list containing the cross products of two lists. It is a helper function for select_join and select_join_outer_left function.

separate_attributes_from_types(attributes)

The first line of the a file contains the attributes with type. A list of variable with types are passed into this function to separate the variable names and return the names in a list to find the indices of the attributes that needed to be compared by the program.

indices_of_records_that_match_condition(cross_product, index_1, index_2, operator)

This function finds the indices of the rows of the cross products list to show the value for default joins or inner joins. It performs the comparison with the operator between the columns given by index_1 and index_2 It is a helper function for select_join and select_join_outer_left function.

NOTE

For joins, inner joins or left outer joins, this program can only take two tables and two attributes from separate tables to be compared with only "+" operator.

Test Script

CREATE DATABASE CS457_PA3;
Database CS457_PA3 created.

USE CS457_PA3;
Using database CS457_PA3.

create table Employee(id int, name varchar(10));
Table Employee created.

create table Sales(employeeID int, productID int);
Table Sales created.

insert into Employee values(1,'Joe');
1 new record inserted.

insert into Employee values(2,'Jack');
1 new record inserted.

insert into Employee values(3,'Gill');
1 new record inserted.

insert into Sales values(1,344);
1 new record inserted.

insert into Sales values(1,355);
1 new record inserted.

insert into Sales values(2,544);
1 new record inserted.

```
select *
from Employee E, Sales S
where E.id = S.employeeID;
```

ID INT | NAME VARCHAR(10) | EMPLOYEEID INT | PRODUCTID INT
1 | 'JOE' | 1 | 344
1 | 'JOE' | 1 | 355
2 | 'JACK' | 2 | 544

```
select *
from Employee E inner join Sales S
on E.id = S.employeeID;
```

ID INT | NAME VARCHAR(10) | EMPLOYEEID INT | PRODUCTID INT
1 | 'JOE' | 1 | 344
1 | 'JOE' | 1 | 355
2 | 'JACK' | 2 | 544

```
select *
from Employee E left outer join Sales S
on E.id = S.employeeID;
```

ID INT | NAME VARCHAR(10) | EMPLOYEEID INT | PRODUCTID INT
1 | 'JOE' | 1 | 344
1 | 'JOE' | 1 | 355
2 | 'JACK' | 2 | 544
3 | 'GILL' | |