School of Data Science

The Chinese University of Hong Kong, Shenzhen

CSC3170 Database System

Prof. Clement Ho Cheung Leung

Program design

1. Purpose:

This project involves writing a miniature relational database management system (DBMS) that stores tables of data, where a table consists of some number of labeled columns of information. This database supports simple query language for extracting information from tables and it will not consider efficiency and speed.

2. This database supports the following queries:

create table <table name> (<column name>$^+$,)

create table <table name> **as** <select clause>

load <name>;

store <table name>;

insert into <table name> values <literal>$^+$

print <table name>;

quit;

exit;

select <column name>$^+$, **from** <table name> <condition clause>

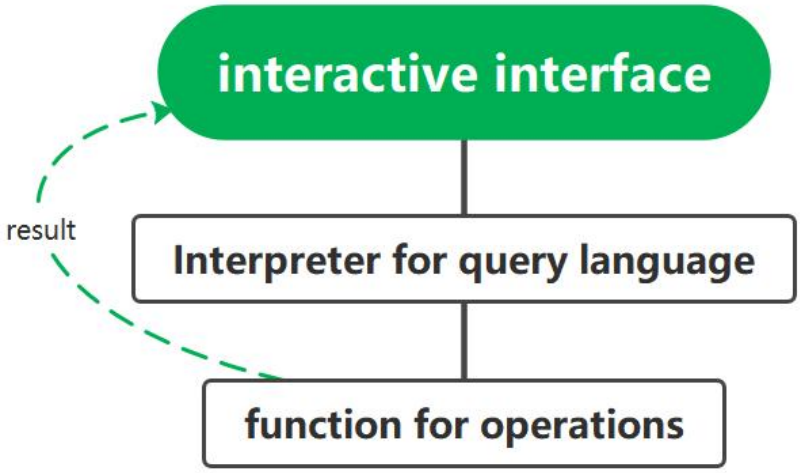
select <column name>$^+$, **from** <table name> , <table name> <condition clause>

3. This database consists of three main parts:

(i) User interface: to read the instruction from users and show the output.

(ii) Interpreter: to interpret the instruction and generate command, it will detect syntax error as well.

(iii) Functions for operation: to execute different type of commands and deal with error in database. It will finally return the result to user interface.



Information of code document

- **Command.py: class Command which is used to pass information from interpreter to functions**
- **Condition.py: class Condition which stores minimum unit of condition clause**
- **Database.py: class Database that includes functions for different queries:**
  - create table
  - create table as
  - load
  - store
  - insert
  - print
  - select
- **Interpreter.py: class Interpreter for interpreting command**
- **Main.py: user interface**
- **Row.py: class Row which is the smallest unit of information**
- **Table.py: class Table which is the structure of a table and it allows operation between tables**

Functionality Implementation

1. Main

The main function is implemented in the ***Main.py*** file. It is design to run our miniature relational database management system, which includes executing input operation as well as using functions in ***Interpreter.py*** and ***Database.py*** to perform the functionality of DBMS. It is used to read in commands from terminal which act as the user interface. The input would be read by interpreter to judge whether it is valid. Only when the command works, the interpreter would return a *"Command"* object which is asigned to *"cmd"* in main. Then, the *"cmd"* would be executed by *"do_command"* operation in database. Command type will also be simply classified into *"quit"*, *"exit"* and other commands. If the command calls the program to *"quit"* or *"exit"*, the *"do_command"* operation would return a value -1. The main function then would receive this signal and bring to the end of the program.

2. Interpreter

The interpreter of queue language is implemented in the ***Interpreter.py*** file. This part aims to convert the SQL commands in form of string to the class *"Command"*, which is implemented in the ***Command.py*** file and stores the information that can be used in the execution of this command. This part also detects the syntax error of the command.

The implementation of this part can be divided into two steps. First, it detects the type of the command. Then, it tries to fetch the information in the command and generates a *"Command"* class. The function that handles type detecting and return the type of the command is *"typeDectect()"*. The functions which deal certain type of commands and return a *"Command"* class have the name in form of *"command type_interpreter"*, such as *"select_interpreter()"*. The main function for this part is *"interpreter(instr)"*, which calls other functions in the file and returns a *"Command"*

class and where *"instr"* is the string that user puts into the terminal. If the command has syntax error, the main function will raise an exception *"error: SQL syntax error"*. The *"Command"* class returned by the *"interpreter()"* function has five attributes.

The *"type"* attribute is a string and is the type of the command. The *"name"* attribute is a list which stores the table names mentioned in the command. The *"row"* attributes is a list which stores the values in the row of the *"insert"* command. The *"column"* attribute is a list which stores the columns' names mentioned in the command. The *"condition"* attribute is a list of *"Condition"* classes, which stores the conditions in the select command. This class is used in the following execution of the command.

3. Table

The property initialization and natural join operation of the Table are implemented in the ***Table.py*** file. This part aims to initialize a Table by *"Table"* class and perform the join operation in the *"join()"* function to facilitate the select operation.

The *"Table"* class contains three attributes: name, column name, and data. The names, column names, and data are strings, lists, and lists within lists, respectively. The *"join()"* function will return a new table by join of two table. First, it merges the two tables and records the index of the same column with *"flag"*. If there is no element in *"flag"*, the tables do not have the same columns, then do outer join. If there is an element in *"flag"*, the tables have the same column, then do natural join. Finally, this function returns the result in *"new_table"*.

4. Database

Database is a class and it has a variable contains a list of *"Table"* object. Each table is stored in a *"Table"* object. When we need to execute the commands, a

function first detects the type of the command, and then calls corresponding function to execute specific operation. If the type of the command is quit or exit, this function will directly return -1 so that the main function will receive the signal and end the program. The other corresponding functions are implemented as follows.

(i) create table: The function will check if there exist a table of same name and delete it. Then it will Creates an empty table with the given name. The names of its columns are given by the column names in order. There must not be any duplicate column names.

(ii) create table as: The function will check if there exist a table of same name and delete it. Then it will use select function to get a table with no name and add the name to it.

(iii) load: The function will check that whether the file exist first. If the file doesn't exist, the function prints out error and terminates. If the file exists, read content in file line by line and separate them by comma. If the data is an integer or float, the function will convert it to integer or float. If the data isn't number, the function will load it as string. Then create a new Table object and store the information of column names and data into it.

(iv) store: The function will search the corresponding table first from the list of Table object. If the table doesn't exist, the function prints out error and terminates. If the table has been loaded or created, create a file and then write the content of column names and data into that file separately.

(v) insert: The function will find the corresponding table first. If the table doesn't exist, the function prints out error and terminates. If the table has been loaded or created, the function will further check the number of columns needed for the insert data. If the data has too much digits, the function will print out error. If the data is valid, the function will first convert numbers to integer or float type and then do duplicate checking. If the data are not in the table, the data will be inserted into table successfully.

(vi) print: The function will find the corresponding table first. If the table doesn't exist, the function prints out error and terminates. If the table has been loaded or created, find the max length of each column. Then get the information of column names and data and print out in certain format.

(vii) select: This function defines a method called *"select"* that is used to select specific columns from a table in the database.

This method has three parameters:

"column": A list containing column names.

"table": A list containing table names.

"condition": A list containing condition classes.

The first part of the method is used to handle single-table queries. If there is only one table name in the table list, loop through all tables in the database and compare the name with the table name in the table list. If a matching table name is found, the table is assigned to the *"ori_table"* variable. If there are two table names in the table list, a natural join operation will be performed. To perform a natural join, all tables in the database are individually iterated and the names are compared to those in the table list. If a matching table name is found, the table is assigned to the table0 and table1 variables. Then, call the join method of the table0 object to join table1 and table0, and assign the result to *"ori_table"*. An error message will be displayed if there are more than two table names in the table list, or if there are no table names.

In the following code, select some rows from a given data table *"ori_table"* according to one or more given conditions (stored in condition variables), and only keep some given columns (stored in column variables) , Build a new data table *"new_table"*.

A data table consists of two parts: a column list and a data list. The column list stores the column names in the data table, the data list stores the data in the data table, and each row of data is stored in the form of a list.

First, the program will initialize a new data table *"new_table"*, and record the column numbers in the original data table that we need to keep in the original data table *"ori_table"*.

Then, according to the conditions in the condition variable, the program will select qualified rows from the original data table *"ori_table"*, and add the data of the specified columns of these rows to the new data table *"new_table"*. If there are multiple conditions in the condition variable, all conditions need to be met before the row is added to the new data table.

If any error is encountered during the query process (for example, the left and right data types of the condition do not match, or the column of the condition does not exist in the data table), the program will give an error.

In the case of double conditions, the program will check whether the left and right data types of the conditions match, and select the qualified rows from the original data table *"ori_table"* during the query process. The difference is that there are two conditions in this code, so when selecting rows, both conditions need to be met at the same time.

Specifically, the program will check whether each row of data satisfies the two conditions at the same time, and if so, add the data of the specified column of this row to the new data table *"new_table"*.

Similarly, if any error is encountered during the query process (for example, the left and right data types of the condition do not match, or the column of the condition does not exist in the data table), the program will give an error message and exit the query process.

Test cases

```
PS C:\Users\anaki\Desktop\project-team10-main\project-team10-main\opt
ion3> & C:/Users/anaki/AppData/Local/Programs/Python/Python311/python
.exe c:/Users/anaki/Desktop/project-team10-main/project-team10-main/o
ption3/Main.py
> create table student (name, ID, totalScore);
table student created
> insert into student values 'Tom', 111222, 300;
> insert into student values "Mary", 122112, 238;
> insert into student values Kitty, 121212, 275;
> print student;
name   ID      totalScore
Tom    111222  300
Mary   122112  238
Kitty  121212  275
> select ID from student;
Select Results:
ID
111222
122112
121212
> select name from student where totalScore > 250;
Select Results:
name
Tom
Kitty
> select name, totalScore from student where ID >= 121212;
Select Results:
name   totalScore
Mary   238
Kitty  275
> select ID, totalScore from student where name = "Tom";
Select Results:
ID      totalScore
111222  300
> create table student (studentName, PE, Geography, Chemistry);
table student created
> insert into student values Tom, 100, 100, 100;
> insert into student values Alice, 78, 94, 100;
> insert into student values Jerry, 100, 65, 84;
> insert into student values Peter, 86, 80, 80;
> insert into student values Helen, 92, 95, 84;
> insert into student values Luo, 86, 89, 90;
> print student;
studentName  PE   Geography  Chemistry
Tom          100  100        100
Alice        78   94         100
Jerry        100  65         84
Peter        86   80         80
Helen        92   95         84
Luo          86   89         90
> store student;
Stored student.db
```

```
> select Geography, Chemistry from student where PE = 100;
Select Results:
Geography  Chemistry
100        100
65         84
> select studentName from student where Geography = 80;
Select Results:
studentName
Peter
> select studentName, Chemistry from student where Chemistry > 90;
Select Results:
studentName  Chemistry
Tom          100
Alice        100
> select studentName, PE from student where PE >= 86;
Select Results:
studentName  PE
Tom          100
Jerry        100
Peter        86
Helen        92
Luo          86
> create table students as select studentName from student;
table students created
> print students;
studentName
Tom
Alice
Jerry
Peter
Helen
Luo
> create table students as select studentName, Chemistry from student
;

table students created
> print students;
studentName  Chemistry
Tom          100
Alice        100
Jerry        84
Peter        80
Helen        84
Luo          90
> create table students as select studentName from student where Chem
istry = 84;
table students created
```

```
> print students;
studentName
Jerry
Helen
> create table students as select studentName from student where PE =
 86 and Geography > 85;
table students created
> print students;
studentName
Luo
> load grades;
Loaded grades.db
> print grades;
studentID  studentName  scoreEnglish  scoreChinese  scoreMath
10001      Tom          100           100           100
10002      Alice        90            80            90
10003      Jerry        70            95            80
10004      Peter        78            94            94
10005      Helen        97            94            94
> insert into grades values 10006, Luo, 87, 90, 94;
> print grades;
studentID  studentName  scoreEnglish  scoreChinese  scoreMath
10001      Tom          100           100           100
10002      Alice        90            80            90
10003      Jerry        70            95            80
10004      Peter        78            94            94
10005      Helen        97            94            94
10006      Luo          87            90            94
> insert into grades values 10006, Luo, 87, 90, 94;
> print grades;
studentID  studentName  scoreEnglish  scoreChinese  scoreMath
10001      Tom          100           100           100
10002      Alice        90            80            90
10003      Jerry        70            95            80
10004      Peter        78            94            94
10005      Helen        97            94            94
10006      Luo          87            90            94
> select PE, Geography from student, grades where Chemistry = 100;
Select Results:
PE   Geography
100  100
78   94
```

```
> select Chemistry, PE from student, grades where PE > 86;
Select Results:
Chemistry  PE
100        100
84         100
84         92
> select Chemistry, Geography, PE from student, grades where PE >= 86
;

Select Results:
Chemistry  Geography  PE
100        100        100
84         65         100
80         80         86
udent, grades;
table ChemistryGrade created
udent, grades;
table ChemistryGrade created
> print ChemistryGrade;
studentName  studentName  Chemistry
Tom          Tom          100
Alice        Alice        100
Jerry        Jerry        84
Peter        Peter        80
Helen        Helen        84
Luo          Luo          90
> create table goodStudent as select studentName from student, grades
 where PE > 90 and Chemistry > 90;
table goodStudent created
> print goodStudent;
studentName  studentName
Tom          Tom
> create table Chemistry as select studentName, Chemistry from studen
t, grades where Chemistry >= 94;
table Chemistry created
> print Chemistry;
studentName  studentName  Chemistry
Tom          Tom          100
Alice        Alice        100
> exit;
PS C:\Users\anaki\Desktop\project-team10-main\project-team10-main\opt
ion3>
```

Difficulty Encountered & Solutions (Optional)

In this group project, we are strangers to each other and lack of experience in cooperation with each other. Therefore, we encountered problems in the function implementation, such as the function variables are not uniform, the code is not clear. The most prominent of these is that inconsistent function variables make them difficult to call. However, through our continuous consultation and concerted efforts, we finally overcame these problems and completed the project satisfactorily.

Task division

Team 10:

李佳齐 120090545 Create table, structure design, report, presentation

万茜 119010289 Main, test case design, report, presentation

张泽萱 120090674 Database operation, report, presentation

徐康裕 120090133 Select operation, report

周泽睿 120090533 Table, report, presentation

樊天宇 120090311 Interpteter, report, presentation

孙鑫昊 120090597 Select operation, report