HW3 --- Simple DBMS feature extend (Advanced)

1 Overview

In this homework, you are asked to implement new features of the simpleDBMS, this project is under Apache 2.0 License, the detail of this project refer to its GitHub Repo**.** The expected result of this homework is to support the below three type of queries:

select { field } from table [where <conditions>] [offset <offset_num>] [limit <limit_num>]

update table set { <field>=<content> } [where <conditions>]

delete from table [where <conditions>]

1.1 Format description

field means the attributes in the table, i.e.: id, name, email, age
offset_num & limit_num are numeric input, they should always be unsigned int
conditions means the search condition of the query
[ ] means optional argument

2 Prerequisite

- Installation of simpleDBMS
  - TA provides a new version of simpleDBMS, which has already implemented all the requirements of hw2
  - To get the newest simpleDBMS, you just need to visit its GitHub page, clone or download it.
- C programming language
  - You can also use C++ to implement this project, but it needs to pass all the system tests and unit test without any modification on test code, as well
- Understanding of simpleDBMS

3 Tasks

- Support where argument
- Aggregation functions
- Update query
- Delete query

3.1 Where Clause

The format of where clause is like

Where <conditions>

The ***conditions*** are composed of one or more conditional statements, there exist three types of condition statements

- Numeric comparison
- String equality comparison
- Logical operator

3.1.1 Numeric comparison

The numeric comparison should support 6 operators and with two data types

Operators:

- = equal to
- != not equal to
- > greater
- < less
- >= greater or equal to
- <= less or equal to

Data types:

- Integer
- Floating point number
    - Please use **double data type in C lang** to do the operation

Example:

Table content:

ID	Name	email	age
1	user1	user1@example.com	20
2	user2	user2@example.com	21

```
db > select * from table where age=20
```

The select result will be the following:

```
(1, "user1", "user1@example.com", 20)
```

3.1.2 String equality comparison

The string comparison should support 2 operators.

Operators:

- = equal to
- != not equal to

Example:

Table content:

ID	Name	email	age
1	user1	user1@example.com	20
2	user2	user2@example.com	21

```
db > select * from table where name=”user1”
```

The select result will be the following:

```
(1, "user1", "user1@example.com", 20)
```

3.1.3 Logical operators

The logical operators are used to extend the conditional statement, and there are two types of operators you need to implement

Operators:

- and --- Just like the general logical and operation
- or --- Just like the general logical or operation

Example:

Table content:

ID	Name	email	age
1	user1	user1@example.com	20
2	user2	user2@example.com	21
3	user3	user3@example.com	22
4	user4	user4@example.com	23

```
db > select * from table where age<21 or age>22
```

The select result will be the following:

```
(1, "user1", "user1@example.com", 20)
(4, "user4", "user4@example.com", 23)
```

3.2 Aggregation Functions

Implement the following aggregation to aggregate data

- avg --- the average of an expression in a SELECT statement and satisfying the criteria specified in the WHERE clause
    - Using double to do the operations, and accurate to only the third digit after the decimal point.
- count --- the number of rows in a table satisfying the criteria specified in the WHERE clause
- sum --- the sum of an expression in a SELECT statement and satisfying the criteria specified in the WHERE clause

Example:
Table content:

ID	Name	email	age
1	user1	user1@example.com	20
2	user2	user2@example.com	21
3	user3	user3@example.com	22
4	user4	user4@example.com	23

If we execute the following command

```
db > select sum(age) from table where age>20
db > select avg(age) from table where age>20
```

The result will be the following because we sum up and average age which rows are satisfied age is larger than 20.

```
(66)
(22.000)
```

Note: Where is optional in the aggregation function. If no where clause is provided, all corresponding data will be included.

3.3 Update query

Implement update statement to modify data in the table

Example:
Table content:

id	name	email	age
1	user1	user1@example.com	20
2	user2	user2@example.com	21
3	user3	user3@example.com	22
4	user4	user4@example.com	23

If we execute the following command

```
        db > update table set email="user4@example.xyz" where
name="user4"
        db > select * from table
```

The select result will be the following because we update the name of user4.

```
(1, "user1", "user1@example.com", 20)
(2, "user2", "user2@example.com", 21)
(3, "user3", "user3@example.com", 22)
(4, "user4", "user4@example.xyz", 23)
```

Note: Where is optional in the update statement. If no where clause is provided, all corresponding data will be updated.

3.4 Delete query

Implement delete statement to remove data from the table

Example:
Table content:

id	name	email	age
1	user1	user1@example.com	20
2	user2	user2@example.com	21
3	user3	user3@example.com	22
4	user4	user4@example.com	23

If we execute the following command

```
db > delete from table where age>22
db > select * from table
```

The result will be as below shows

```
(1, “user1”, “user1@example.com”, 20)
(2, “user2”, “user2@example.com”, 21)
(3, “user3”, “user3@example.com”, 22)
```

Note: Where is optional in the delete statement. If no where clause is provided, all data will be deleted.

4 Testing

System tests (For above requirement)

The system tests focus on system level behavior, and these tests are also used to test the new features you need to implement

Use the following command to execute system tests

- `$ python test/system/system_test.py ./shell [test_case [test_case ...]]`
    - Execute this command in the root folder of this project
    - By default, if no test_cases are given, it will run all test cases, otherwise, it will execute all specified test cases.
    - To see the list of test cases, please check `test/system/testcases` folder
- `$ python test/system/system_test.py [-h]`
    - Show the usage of system test

- You can check the difference of your test output with the expected answer by comparing two files in these two paths
  "test/system/output/<test_case>/<output file>"
  "test/system/answer/<test_case>/<answer_file>"
    - Hint: 1) diff command can show the difference of two files
    - 2) Install vim and using vimdiff provide a better looking

5 Requirements:

- Pass provided system tests
- Upload a zip file to New E3
    - Compress the whole project as a zip file
- Makefile
    - The original project already contains a Makefile, your final submission should also be able to compile the shell using Makefile

Passing system tests only means that your program reaches the basic requirements. There are some hidden tests we do not provide and we will test them on your submission. Therefore, make sure your program works well on query with different combinations of features

6 Submission

6.1 E3

- Compress your whole project into one single zip file named \`<student_ID>.zip\` (like 0756000.zip), and upload to New E3
- Make sure your submission contains a Makefile and the result of make command will generate a binary file named shell
- **If your submission couldn't be compiled by make cmd, or we can't find the shell file as the compiled binary, you will get 0 score**

```
0123456.zip (studentID.zip
`--0123456 (this is your project folder
   |--include
   |   `--*.h (all the header files
   |--src
   |   `--*.c (all the source files
   `--Makefile
```

6.2 Deadline

- You are required to hand in your homework before 2019/05/16 23:59
- Late submission is not allowed

7 Discussion Forum

HW3 discussion <HackMD>

Plagiarism is not allowed, you will get 0 points when we found that happened.