Final project --- Simple DBMS query optimization

1 Overview

In this homework, you are asked to implement **equal join** and optimize query 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 four type of queries:

1.1 Select

```
select <field> from <table1> [join <table2> on <field in
table1> = <field in table2>] [where <conditions>] [offset
<offset_num>] [limit <limit_num>]
```

- field has two types, one is attributes in the table such as **id**, **name**, **email**, **age**, **id1**, **id2** and a special case *, another is aggregation function of an attribute, which can be **sum()**, **avg()** and **count()**. Don't need to specify the table name in attributes
- table1 can be **user** or **like**
- table2 will only be **like**
  - if there is a join query, table1 and table2 will be the **user** table and the **like** table respectively only.
- conditions mean the search condition of the query
  - Only need to accept attributes in user table
  - The conditions will be space splittable string, and **there is no extra bonus for string parsing as HW3**, so you can ignore the string_bonus system test in this project**.**
  - The attribute name will only appear at the left-hand side of the operator, and the right-hand side of the operator will be a scalar or a string, i.e.
    - age > 20
    - name = “user1”
  - There will not exist conditions only contain scalars or strings, such as
    - 3 > 5
    - “123” != “345”
- offset_num & limit_num are numeric input, they should always be unsigned int

1.2 Update

```
update user set <field> = <content> [where <conditions>]
```

- only need to support updating on user table

1.3 Delete

```
delete from user [where <conditions>]
```

- only need to support delete operation on user table

1.4 Insert

```
insert into user <id> <name> <email> <age>
insert into like <id1> <id2>
```

- Must support insert operation on user and like
- The id1 in like table is the primary key, and the number of records in like table will be smaller than user table, the insert queries in test cases to like table will not have duplicated id1 in each test file, so you don't need to handle primary constraint in like table, but you can still do it if you want to.

2 Prerequisite

- Installation of simpleDBMS
  - There is no official version simpleDBMS which is implemented to fit all HW3 tasks this time
  - To get the newest simpleDBMS with latest test cases, 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 without any modification on test code, as well
  - If you rewrite this project using C++, please add some comments in README to let TA know your project needs to be compiled by g++ 7.4.0
- Understanding of simpleDBMS

3 Tasks

- Equal join (40%)
- Query optimization (20%)
- Backward compatibility (40%)

3.1 Equal join (40%)

In this task, you need to implement equal join on '**user**' and '**like**', which means the join condition will only be equality of two fields. And the join conditions will only be id = id1 or id = id2. You can ignore the ordering of the join result, as we will use **count()** to test for your join result only.

Example:

user

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

like

id1	id2
1	4
2	3

If we execute the following query:

```
    db > select count(*) from user join like on id = id1
where age > 20
```

The execution result is:

```
    (1)
```

3.2 Query optimization (20%)

There is a new script(generate_testcase.py) for you to generate test cases for query optimization task.

Usage:

python test/system/generate_testcase.py [-h] [--user_num USER_NUM]

The generated test case file will be different every time, we will also use this script for the final scoring.

Scoring method:

- Correctness: 5%
  - 1% for each test case
- Execution time: 15%
  - We will run your program about 1~3 times and sum up the total execution time

    - Rank by the total execution time, the scores will be given according to the ranking list
        - Assume there are 40 groups
        - Scores = 15 * (41 - Rank) / 40
        - 1st gets 15 points, 3rd gets 14.25 points

(We will use Python time_ns() or time() in the time module to measure the execution time of your program)

3.3 Backward compatibility (40%)

You need to support all the function you have done in HW2 and HW3, except the .load built-in command. Also, the test case in HW2 and HW3 will not use **'table'** anymore, we will use **'user'** instead.

4 Testing

4.1 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

4.2 Environment

We will run your test using the following environment setting:

- **4 GB RAM**

- **1 CPU core**
- **No GPU**
- **No 3rd party packages**

5 Requirements:

- Pass provided system tests
- 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 `<group_ID>.zip` (like 01.zip), and upload to New E3 **by only one of your group member**
- 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**

```
01.zip (group_ID.zip
|--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 2019/06/23 23:59
- Late submission is not allowed

7 Discussion Forum

Final project discussion <HackMD>

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