VE482— Introduction to Operating Systems

Project 2

Manuel — UM-JI (Fall 2017)

Goals of the project

- Understand the basics on databases
- Multi-threaded and efficient programming
- Fix multi-threading specif problems

1 Introduction

You have just been hired by Lemonion Inc.. On your first day you step in the open space, look, and see many people working and moving all over the large room. As you seem disoriented a tall blond man calls out to you saying "You're the new software engineer?". Your answer with a simple "Yes" and follow him as he shows you the way to your cubicle. You quietly seat at your desk, switch on your brand new computer, and realise that you already have a new email from... the CEO. Is that a good news?

Date: Tue. 10 Nov. 2017 05:13:42 +0800
From: xorgates@lemonion.com
To: lemon-staff@lemonion.com
Subject: Call for help on LemonDB

Dear all employees,

I'm Jeff the CEO of Lemonion Inc., the leading online lemon seller. As you know our home-brewed database, "LemonDB" keeps track of all our stock, customer information etc.. Unfortunately due to our ever greater success and the constant increase in the number of transactions it crashed last night.

On the downside (i) the source code of the database system was lost in the breakdown, (ii) the initial developer left the company years ago and we can not contact him for any support, and (iii) besides an old manual containing the list of functionalities that were implemented in the system, no documentation has been found.

After the board members held an emergency meeting in the night, it was decided to take advantage of this sad situation in order to improve our database system. The new system is expected to (i) be very fast (at least twice as fast as the previous version), (ii) take advantage of our git server in order to avoid any future loss of code, and (iii) feature a reasonable amount of documentation explaining the working of the new implementation.

Volunteers who want to join the effort are expected to reply as soon as they read this email. The basic documentation as well as an incomplete and obsolete version of the system source code will be provided to them. We know this is not much but we do not have anything better at the moment.

Thanks a lot for your devotion and cooperation. Long life to Lemonion Inc.!

Best regards,
Jeff, CEO of Lemonion. Inc.

As you have just completed the reading of the email you feel a hand on your shoulder. Turning back, you see a medium sized man with a perfectly adjusted suit standing behind you. His determined and stern look as well as the absence of any trace of consideration in his eyes are very intimidating. Before you can open your mouth and articulate a word he

starts without even greeting you: "Have you read the email?" Looking at your screen and you nodding affirmatively he continues, "I just talked with Jeff, all the guys are busy on other projects. You are not working on anything, so here is your first task: get LemonDB up and running fast, as early as possible." After pausing for a couple of seconds as if he did not know how to conclude, he resumed on a neutral tone contrasting with his previous strict words "I am sure you'll do a good job, good luck."

As soon as he finished his sentence he turns away leaving you alone. While you indistinctly hear a "By the way, welcome to Lemonion!" coming from afar, his words are echoing in your head: "Run FAST [...] as EARLY as possible". No doubt, our future is at stake! You must do it, and do it well...

Feeling your sweat dripping and your heart beating hard, too hard, in your chest, you take a deep breath and try to regain your composure. As you slowly calm down spontaneous thoughts spring in your mind. Soon you can recall the basics on databases from when you were a young, active, and hard working student.

1.1 Basics on databases

A *database* is composed of *tables*. In the following example the database is composed of two tables: `Student` and `Instructor`.

```
Student                                          Instructor
| KEY        | studentID  | class | totalCredit |    | KEY       | departmentID | courseID |
| ---------- | ---------- | ----- | ----------- |    | --------- | ------------ | -------- |
| Bill_Gates | 4008123123 | 2014  | 112         |    | H_Finch   | 42           | 341      |
| Steve_Jobs | 4008517517 | 2014  | 115         |    | S_Groves  | 42           | 343      |
| Jack_Ma    | 4008823823 | 2015  | 123         |    | J_Reese   | 43           | 345      |
```

Each table features some columns and rows, called *fields*, and and *records*, respectively. The `Student` table has four fields: `KEY`, `studentID`, `class`, and `totalCredit`, as well as three records: `Bill_Gates`, `Steve_Jobs`, and `Jack_Ma`. The `KEY` field must be unique, i.e. two rows can never share a same `KEY`. A table is read "per-record". For instance in the table `Instructor`, `H_Finch` has `departmentID` 42, and teaches `courseID` 341. Note that the records (rows) are unordered in a table.

It is possible to specify a string to read, update, and manage a database. Such a string is called a *query*. A query is very much like a human language and always ends with a semicolon. For example the following query finds the total credit of the student named "Jack_Ma":

```
1 SELECT ( totalCredit )
2 FROM Student
3 WHERE ( KEY = Jack_Ma );
```

The `SELECT` clause means we want to look up something. The `FROM` explains which table we are operating on. The `WHERE` clause specifies some criteria on the record to be found. In particular in this example we are interest in the `totalCredit` of the `Student` whose record has `KEY` field `Jack_Ma`. Other fields can also be used to run a search, e.g. `WHERE ( class >= 2015 )`, looks for a student that is after Class 2015.

1.2 The LemonDB project

Comforted by your memories you feel more serene and are about to stand up to go on a quest for the coffee machine when you hear a soft beep emanating from your computer. Grabbing the mouse you click on the pop up that had appeared on the screen and are directed to a new email.

Date: Tue. 10 Nov. 2017 09:36:28 +0800
From: xorgates@lemonion.com
To: n00b@lemonion.com
Subject: The LemonDB project
Attachments: Coding conventions, LemonDB manual, Company directory, Salary conditions

Dear new employee,

As Mr. Frown has already kindly explained you in details, you will be working on the LemonDB rewriting project with three other Junior software developers from a different branch. We expect you to collaborate efficiently and produce a high quality database system. Remember we are a company and our main aim is making profit. As long as LemonDB is down we are loosing money. It is therefore essential that you complete this task within the shortest delay and provide us with the highest quality work.

You are expected to comply with the following requirements:

- The database program should read queries from the standard input, executes them, and return the result on the standard output;
- The programming language must be either C or C++;
- To improve the performance, the new implementation must use multi-threading (other enhancements are also accepted and welcome);
- In order to remain as stable and generic as possible only use `pthread` or the C++ standard `<thread>` library;
- Regularly commit all your work to the company git server and use the master branch to store the final LemonDB product;
- Document your work in a pdf or text file.

To fulfill the previous requirements you can access the following resources in in `/var/lib/lemondb/`:

- The binary that survived the crash;
- The old source code;
- A few test inputs and outputs, along with a some database files. Use these to better understand the input/output format;

Attached to this email you'll find the official Lemonion Inc. database programming convention, the LemonDB manual, and the company directory (such that you can contact the three other members of your group).

Despite his busy schedule and many obligations, Mr. Frown has generously offered to supervise and assist you in this task. Feel free to consult him if you have any major concern regarding the LemonDB project; he assured me he would be very glad to help you.

Best regards,
Jeff, CEO of Lemonion. Inc.

P.S. please also find your salary conditions for the LemondDB project, attached to this email.

At the end of your reading you have a dubious feeling toward this double-face Mr. Frown, "how can he play the nice guy with the boss and be so mean with the employees?" He didn't even provide you with the necessary and useful details. You can definitely not trust him and resolve to only seek his help as a last resort.

As you start to devise a plan of attack your phone loudly rings. By the time you pick it up people around feel exasperated and start whispering, trying to figure out who could be so impolite. Therefore you discreetly exit the open-space and softy answer "Wei" while walking. On the phone you recognise the voice of your mum who kindly inquires how is your first day in this great company. As you keep walking you reassure her and shorten the conversation. Behind a glass door you see coffee machine and decide to step in and relax a bit in order to recollect your ideas and decide what to do next.

The most appropriate seems to get in touch with the other software engineers, then check the salary conditions, and finally go through all the other documents and materials. Feeling much better you walk back to your desk, seat down, and move the mouse. You've got a new email from... Mr. Frown.

Date: Tue. 10 Nov. 2017 10:15:59 +0800
From: mr.frown@lemonion.com
To: n00b@lemonion.com
Subject: Guidance on the LemonDB project

Where are you? I came to your cubicle and it was empty! Deserting your desk on the first day is clearly not a professional attitude. Be assured that from now on I will keep a tight eye on you.

Here are some more details regarding the LemonDB project:

- The old source code is incomplete (many missing functions) and messy in style but structurally well designed. Important features are already implemented (e.g. evaluation of `WHERE`). As it is robust enough to be expanded or modified, as soon as you understand how it works, make good use of it.
- The code makes a heavy use of exceptions. Based on the Lemonion Inc. database programming conventions the users will always input valid query strings so that it is fine if you don't check for any error. However these exceptions forms a safe net to defend from programmer's faults, and as such can be helpful in debugging. Again make good use of these when working on your a multithreaded environment.
- Although many queries should not display anything on the standard output, there is nothing wrong about displaying error messages or debugging information on the standard error output. Once more, make a good use of this strategy.

Mr. Frown.

This Mr. Frown already didn't like you, but now this is clearly hopeless. Just because you forgot to set your phone on silent on arriving and your mum called at the worst time, you failed your first day. Before negative feelings overwhelm you and you decide to resign you remember what your mum has always taught you: "Never give up, never surrender!"

So yes! You will complete this LemonDB project, do an amazing job, and show this pretentious and evil Mr. Frown that Lemonion Inc. needs you. On those thoughts you jump to the company directory and contact your group mates.

2 Back to reality

Although this is an imaginary project scenario take good note of the following points:

- All the technical information provided are correct;
- The company server corresponds to the ve482 server;
- The Salary conditions represent the grading policy;
- The "bug discovery bonus" (salary conditions) will only be awarded to the first team discovering a given bug;
- Instead of contacting Mr. Frown (who is he?) upon discovering a bug, submit your bug report on Canvas;
- As this document inevitably contains errors and imprecisions, please contact us and let us know if you have any question (don't worry Mr. Frown is not that mean in the end);
- In case of major issue, updates will be posted and announced on Canvas;
- We seriously want you do design good quality code, that runs very fast, and is completed early;

3 Salary conditions

The faster the program, the more money

The goal being to improve the performance of the old version by at least two your work will be run against it in a fair setup: (i) both programs will be run on the same set of input (*test suite*), and device; (ii) the real time will be measured (note: not the CPU time). Since the inputs will be similar the output are expected to be identical in order and values;

Submission requirements

The final submission must be in the master branch of the git repository. It should include all the necessary documentation for new employees to fully understand your work, without needing your support. We really want to avoid being in this terrible situation again: have no source code and no good documentation. Remember it represents a massive loss of money for the company. As such providing low quality documentation will lead to a much lower pay.

Therefore the documentation must (i) clearly explain the design, (ii) contain detailed information on the performance improvements, (iii) describe how common problems related to multi-threading were overcome, and (iv) any other information that could ease the work of the future LemonDB developers. Remember that the focus is on the quality of the work, not the length of the document.

Amount calculation

For the special LemonDB project you will be paid according the number of credits you complete, the maximum being 150, without the potential bonuses. They are apportioned as follows:

- Documentation: 30;
- Working version (independently of its performance): 30;
- Performance: 90 or more;

Your program will be timed by running it on every test case from the test suite. Denoting by $t_1, \cdots, t_n$ the time spent by your implementation to accurately run the n tests from the test suite, and $s_1, \cdots, s_n$, the time used by the single threaded version, then the number of credit C you will be awarded for the performance is given by

$$C = \frac{60 \cdot \max\left(0, \mathsf{avg}\left(\log_2 \frac{s_i}{t_i}\right)\right)}{0.6 + 0.2 \cdot \mathsf{stddev}\left(\log_2 \frac{s_i}{t_i}\right)},$$

where stddev is the standard deviation, and avg the mean. This formula should encourage you to achieve a higher and more uniform performance improvement.

As a last note, your program is assumed to be tested on a personal computer with 4 CPUs, each with 2 cores (or in marketing terms "4 cores, 8 threads").

Happy workplace initiative

Following a number of complaints regarding non-cooperative coworkers and workplace disputes, Lemonion In.c introduced the *happy workplace initiative*, a set of cooperation rules that should benefit the well-being of every employee.

This Lemonion Inc. initiative is summarized in the following three major points.

- Software engineers are expected to apply the 2P strategy in their work. Refer to the Appendix for more details on this approach.
- Upon completion of a project each pair must submit a pair-evaluation form in the Quick Unsupervised Input Zone (QUIZ), that can be found in the Company Active Notification and Vision Appreciation System (CANVAS). This simple form focuses on the contribution of the pair as well as their peer's performance.
- Salary condition set with respect to the performance of the pair;

In the special case of the LemonDB project, the number of credits awarded for the performance part will be adjusted with respect to the pair-evaluation, the git commits, and any other work that will be submitted.

For instance if a group of two pairs of software engineers receives 45 credits in the performance part, but one of the pairs did not contribute at all, then the other pair is awarded a total $45 \cdot 2 = 90$ credits.

Bonuses

Speed

Since a day without LemonDB is a loss for Lemonion Inc., we resolved to offer you a special bonus if you complete your work early. If you manage to complete the work before the deadline set by Mr. Frown you will be awarded one extra credit per early day.

Bugs

Although our engineers have noticed long ago that LemonDB had some bugs, nobody took the time to investigate them. If you identify a bug, immediately contact Mr. Frown and properly report it. By doing so you will be awarded 5 extra

credits.

A bug is considered identified and reported only when a brief bug report has been submitted on canvas. The report should contain a *simple* test-case that triggers the bug:

- An input that triggers the bug (database files and commands);
- Differences between the expected and actual behaviors;
- Section of the document that specifies the expected behavior;
- The input must minimal, all inputs combined should be no more than 30 lines (unless absolutely necessary, in which case you need to argue why).

Remember the importance of being clear, precise, and concise when reporting a bug. In particular the input must always be as minimal as possible in order to help understanding what is the exact source of the problem. For instance if a sequence of command leads to a crash, then trim down your input until you are able to precisely spot which instruction fails.

Appendix

Recent studies lead in major companies have highlighted the significant improvement in efficiency and effectiveness resulting from *Pair Programming* (2P, pronounced "double P").

In 2P, a pair of software engineers works as follows.

- Sit in a comfortable environment and work together as a team;
- A software engineer plays the "Driver" and the other one the "Navigator";
- The *driver*'s work is to type on the keyboard while the *navigator* provides suggestions;
- Both the *driver* and the *navigator* should pay attention to common typos and errors;
- Roles can be exchanged after a while;
- Both developers are expected to think of the whole project;

4 Lemonion Inc. database programming conventions

! This document contains commercially sensitive information. For internal use only. !

This document contains the basics on the official coding style in practice at Lemonion Inc.. These database programming conventions result from the internal design of LemonDB. Therefore not respecting them might lead to errors and unexpected results.

Basic queries

Queries can either be written over any number of lines.

```
SELECT ( totalCredit ) FROM Student WHERE ( KEY = Jack_Ma );
```

Spacing

In a query, at least one blank character (space, tab, and newline) separates a `(`,`)`, `=`, etc. from other components.

```
SELECT ( totalCredit )FROM Student WHERE ( KEY = Jack_Ma );
```

In this query a space is missing between `)` and `FROM`.

Alphabet

All the table names, field names, and keys must only be composed of letters, and underscore. Strings are case sensitive.

```
SELECT ( totalCredit ) FROM Student WHERE ( KEY = Jack Ma );
```

In this query a space separates `Jack` and `Ma`, while the character space is not part of the alphabet.

Queries validity

LemonDB always assumes the user inputs a valid query, i.e. the user's input is syntactically correct, while all the arguments are always valid (e.g. the table names or field names exist).

5 LemonDB manual

! This document contains commercially sensitive information. For internal use only. !

5.1 Table Management Queries

Load a new table from a file

```
LOAD tableFilePath;
```

Load a table from a file located at `tableFilePath`. The format of the file is described as follows. The first line contains the name of the table and the number of rows. The second line contains all fields (including the `KEY` field). Starting from the third line, each of them contains a record. All items are space-separated (one or more). The order of fields in the database is assumed to be the order of appearance in the second line.
Nothing should be printed to the standard output.
The program can assume (i) the user always provides a valid file name and a valid file, and (ii) no error will be found during the execution of this query.

Example

The following `student.tbl` file contains the `Student` table used in previous section.

```
Student 4
KEY         studentID   class  totalCredit
Bill_Gates  4008123123  2014   112
Steve_Jobs  4008517517  2014   115
Jack_Ma     4008823823  2015   123
```

More examples are available in the test suite.

Dump existing table to file

```
DUMP table filePath;
```

Dump a table `table` into a file located at `filePath`. The format of the file is as described in the `Load` documentation.
Nothing should be printed to the standard output.
No observable changes should be made to the database even in case of error (e.g. non-existing table name).

Example

The following dumps example table `Student` into the file `student.tbl`.

```
DUMP Student student.tbl;
```

Delete an existing table.

```
DROP table;
```

Delete the table `table` along with all its content. In particular, once this command is executed the table should disappear from the database .
No observable changes should be made to the database on an error.

Clear an existing table.

```
TRUNCATE table;
```

Delete all the content of the table `table`, i.e. removes all the records from this table. This operation does not affect the number, name, or order of the fields. The table becomes empty after this operation.
No observable changes should be applied to the database on an error.

Copy a table

```
COPYTABLE table newtable;
```

Creates a copy of the existing table `table`, and named it `newtable`'. The new copy will contain all the records and fields from the original table.
Nothing should be printed to the standard output.
No observable changes should be applied to the database on an error.

Example

```
COPYTABLE Student NewStudent;
```

This query copies the table `Student` to the new table `NewStudent`.

5.2 Data manipulation

Delete records from a table.

`DELETE ( ) FROM table; DELETE ( ) FROM table WHERE ( cond ) ...;`

Delete rows from the table `table`. This query is a *conditional query*.
Print `Affected <n> rows.` to standard output, where `<n>` is the number of rows that were deleted.

Conditional queries

A Conditional Query is query featuring an **optional** `WHERE` clause. While the `WHERE` clause defines a set of conditions, each record will be tested against this condition, and only affected by the query on a successful test. If the `WHERE` clause is omitted then all rows are considered to match the test.
A `WHERE` clause is formed of the `WHERE` keyword followed by multiple *condition tuple*. Each condition tuple is a 3-tuple in the form of `( field op value)`, where `field` is the name of a field, `op` is a comparison operator (one of >, <, =, >=,<=), and `value` is either a string (`KEY` field) or an integer (other fields).
More specifically the `KEY` field can only compared for equality. For example valid condition tuples on the table `Student` could be `( class > 2014 )`, `( KEY = Steve_Jobs )`.
A test on a record succeeds if only if all conditions defined by the condition tuples are satisfied. For example for the `Student` table:

- `WHERE ( class = 2014 )` affects the first two rows
- `WHERE ( class = 2014 ) ( class > 2014 )` does not affect any row
- `WHERE ( KEY = Jack_Ma )` affects only the student named `Jack_Ma`.
- `WHERE ( KEY = Jack_Ma ) ( class > 2020)` does not affect any row
- `WHERE ( class >= 2014 )` affects all the rows
- `WHERE ( class >= 2014 ) ( totalCredit < 115 )` only affects the first row
- Omitting the `WHERE` clause means that the queries affects all the rows

No observable changes should be applied to the database on an error.

Example

```
DELETE ( ) FROM Student WHERE ( class < 2015 ) ( totalCredit < 115 );
```

The query deletes students who enrolled before 2015 **and** received less than 115 credit. The resulting table is

```
KEY         studentID   class  totalCredit
Steve_Jobs  4008517517  2014   115
Jack_Ma     4008823823  2015   123
```

Insert new record into a table.

```
INSERT ( key value1 value2 ... ) FROM table;
```

Insert the row `( key value1 value2 ... )` into the table `table`. Note that the `values` are inserted according to the order of the fields provided in the `LOAD` instruction: `key` is the value of the `KEY` field, `values` are integers. The length of the tuple must be equal to the number of fields in the table. If `KEY` already exists no changes should be made to the table. Again note that the rows are unordered so we do not care where the new row is inserted.
Nothing should be printed to the standard output.
No observable changes should be applied to the database on an error.

Example

```
INSERT ( luke 666666666 2015 12 ) FROM Students;
```

Result on the `Student` table:

KEY	studentID	class	totalCredit
Bill_Gates	4008123123	2014	112
Steve_Jobs	4008517517	2014	115
Jack_Ma	4008823823	2015	123
luke	666666666	2015	12

Update data in a table

```
UPDATE ( field value ) FROM table WHERE ( cond ) ...;
```

Update the field `field` of the rows satisfying the conditions with new value `value` in `table`. This query is a conditional query.
Print `Affected <n> rows.` to standard output, where `<n>` is the number of updated rows.
No observable changes should be applied to the database on an error.

Example

```
UPDATE ( totalCredit 200 ) FROM Student WHERE ( KEY = Jack_Ma );
```

The query changes the obtained credit of student named `JACK_MA` to 200.

KEY	studentID	class	totalCredit
Bill_Gates	4008123123	2014	112
Steve_Jobs	4008517517	2014	115
Jack_Ma	4008823823	2015	200

Accessing data in a table

```
SELECT ( KEY field ... ) FROM table WHERE ( cond ) ...;
```

For each record satisfying the condition print the fields specified in the SELECT clause. Each field that is not a KEY may appear at most once. The KEY field must always appear at the beginning. This is a conditional query. Print each record in the format (KEY field1 field2 field3 ...). The records must be displayed in ascending lexical order, sorted by KEY.
If no record satisfies the condition nothing should be printed.
No observable changes should be applied to the database on an error.

Example

```
SELECT ( KEY class totalCredit ) FROM Student WHERE ( totalCredit > 100 ) ( class < 2015 );
```

This query prints the KEY, class, and totalCredit of the students who received more than 100 credits and were in a class before 2015. The output of the query is as follows.

```
( Bill_Gates  4008123123  2014  112 )
( Steve_Jobs  4008517517  2014  115 )
```

Duplicating records

U 2017-10-18

```
DUPLICATE ( ) FROM table WHERE ( cond ) ...;
```

This query copies the records satisfying the condition in table table. This query is a conditional query. The affected records are inserted into the table, with key originalKey_copy. If a copy of a record already exists the copy is not overwritten, however the copy can be duplicated into originalKey_copy_copy.
On success print Affected <n> rows. to standard output, where <n> is the number of rows updated.
No observable changes should be applied to the database on an error.

Example

```
DUPLICATE ( ) FROM Student WHERE ( class < 2015 ) ;
```

This query copies the records of students whose class is before 2015 into the Student table. The resulting table is as follows. Note that a second call would create a copy of Bill_gates_copy but no copy of Bill_Gates.

```
KEY              studentID   class  totalCredit
Bill_Gates       4008123123  2014   112
Steve_Jobs       4008517517  2014   115
Jack_Ma          4008823823  2015   123
Bill_Gates_copy  4008123123  2014   112
Steve_Jobs_copy  4008517517  2014   115
```

Swapping values

U 2017-10-18

```
SWAP ( field1 field2 ) FROM table WHERE ( cond ) ...  ;
```

This query swaps the values of `field1` and `field2` for the records of `table` that satisfy the given condition. This is a conditional query. There is no restriction on the fields. When they are the same the query does nothing.
On success print `Affected <n> rows.` to standard output, where `<n>` is the number of rows updated.
No observable changes should be applied to the database on an error.

Example

```
SWAP ( class studentID ) FROM Student WHERE ( class < 2015 ) ;
```

This query swaps the value of the `class` and `studentID` fields for students whose `class` is before 2015 in the `Student` table. The resulting table is as follows.

```
KEY         studentID   class  totalCredit
Bill_Gates  4008123123  112    2014
Steve_Jobs  4008517517  115    2014
Jack_Ma     4008823823  2015   123
```

Counting records

```
COUNT ( ) FROM table WHERE ( cond ) ...  ;
```

This query counts the number of records that satisfies the conditions. This is a conditional query.
On success, the program should print `ANSWER = <numRecords>` to standard output, where `<numRecords>` represents the desired count. If no record is affected, then the count is zero.
No observable changes should be applied to the database on an error.

Examples

```
COUNT ( ) FROM Student ;
```

This queries counts the number of records in the `Student` table. The program should print the following to the standard output.

```
ANSWER = 3
```

```
COUNT ( ) FROM Student WHERE ( class < 2015 ) ;
```

This query counts the number of records which feature a class from before 2015 in the `Student` table. The program should print the following to standard output.

```
ANSWER = 2
```

Basic arithmetics

```
ADD ( fields ...  destField ) FROM table WHERE ( cond ) ...;
SUB ( fieldSrc fields ...  destField ) FROM table WHERE ( cond ) ...;
```

These two queries perform arithmetic operations on records that satisfy the conditions. Both are conditional queries. The `ADD` clause sums up one or more fields given in `fields` and store the result in the `destField`. The `SUB` clause subtracts the **zero or more** values of `fields` field, **from** the `fieldSrc` field, and stores the result in `destField`. Note that the `destField` may be one of the fields used in the computation.
On success print `Affected <n> rows.` to standard output, where `<n>` is the number of rows affected.
No observable changes should be applied to the database on an error.

Examples

```
ADD ( class totalCredit studentID ) FROM Student WHERE ( totalCredit > 100 ) ( class < 2015 );
```

This query determines the sum of `class` and `totalCredit`, and stores the result in the field `studentID`. This is only calculated for students who received more than 100 credits and were in a class before 2015.

```
ADD ( totalCredit studentID ) FROM Student WHERE ( totalCredit > 100 ) ( class < 2015 );
```

This query calculates the sum of `totalCredit` and stores the result in `studentID` for students who received more than 100 credits and were in a class before 2015. Essentially it copies the data from field `totalCredit` into `studentID` for the matching students.

```
ADD ( totalCredit totalCredit totalCredit ) FROM Student WHERE ( totalCredit > 100 ) ( class < 2015 );
```

This query essentially doubles the `totalCredit` of students who received more than 100 credits and were in a class before 2015.

```
SUB ( studentID class studentID ) FROM Student WHERE ( totalCredit > 100 ) ( class < 2015 );
```

This query calculates the value given by subtracting `studentID` from `class` and stores the result in the field `studetnID`, for students who received more than 100 credits and were in a class before 2015.

```
SUB ( class class class class ) FROM Student WHERE ( totalCredit > 100 ) ( class < 2015 );
```

This query negates the `class` students who receive more than 100 credits and were in a class before 2015.

```
SUB ( class class ) FROM Student WHERE ( totalCredit > 100 ) ( class < 2015 );
```

This query subtracts nothing from `class` and stores the result back into the `class` field for students who received more than 100 credits and were in a class before 2015. Essentially this query does nothing.

Summing records

```
SUM ( fields ... ) FROM table WHERE ( cond ) ...;
```

This query aggregates records that satisfies the given conditions. This is a conditional query. The `SUM` clause sums the values of one or more fields given in `fields` over all the affected records. The `KEY` field cannot be summed over.

On success, the program should print `ANSWER = ( <sumFields> ... )` to standard output, where `<sumFields>` represents the sum of the fields, in the order specified in the query. If no record is affected, then the sum is set to zero.

No observable changes should be applied to the database on an error.

Example

```
SUM ( totalCredit class ) FROM Student ;
```

This queries sums the total number of obtained credits and class over all the students in the `Student` table. The program should print the following to the standard output.

```
ANSWER = ( 350 6043 )
```

Finding minima / maxima

```
MIN ( fields ... ) FROM table WHERE ( cond ) ... ;
MAX ( fields ... ) FROM table WHERE ( cond ) ... ;
```

These query aggregates records that satisfy the given conditions. Both are conditional queries. The `MIN` clause finds the minimum value among all affected records for the values of one or more fields given in `fields`. The `KEY` field is considered not comparable thus will not appear in the `MIN` clause.

On success, the program should print `ANSWER = ( <minValues> ... )` to the standard output, where `<minValues>` represents the minimum for each of the fields, in the order specified in the query. If no record is affected, the program should not print anything.

No observable changes should be applied to the database on an error.

The `MAX` query works in similar way, but finds the maximum instead of the.

Example

```
MIN ( totalCredit class ) FROM Student ;
```

This queries finds the minimum credits and the minimum class among all the students. The program should print the following to standard output.

```
ANSWER = ( 112 2014 )
```

5.3 Utilities

Quit database

```
QUIT;
```

Quit from the database. Wait for running queries to complete.

6 Lemonion Inc. company directory

! This document contains private and personal information. For internal use only. !

Lemonion Inc. company directory

Li Guohao & Wu Yichen
Ding Kaili & Xu Dewei

Dong Sijia & Zheng Xuan
Lin Zhi & Liu Yihao

Wei Chen & Zhang Kai
Cheng Junkai & Gao Cunzhi

Hu Yichen & Wu Chenggang
Gong Yuchen & Ji Xingyou

He Ruizhe & Jing Yuanfang
Liu Kaiwen & Qian Xiangru

Xue Leyang & Yao Kaiqi
Song Huanian & Su Hang

Feng Zhengyang & Li Yuying
Chen Zhiqing & Wu Jiachen

Li Yiming & Tao Jingjing
Hu Yunzhe & Liu Xinyi

Liang Tao & Lu Xinsen
Ceng Zhi & Zhu Chen

Evan Bao & David Julius Jacobsson
Ziqi Xia