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

```
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 same table, with a different key. The new key is formed by appending `_copy` to the original key.
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.

```
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

```
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.

Example

```
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