OTTO VON GUERICKE UNIVERSITÄT MAGDEBURG

Architecting and Engineering
Main Memory Database Systems in Modern C
Database and Software Engineering Working Group

Magdeburg, Dec 09th 2017

Marcus Pinnecke, M.Sc.
Prof. Dr. Gunter Saake

Winter Term 2017/2018

Exercise sheet No. 6

String Matching in TPC-H LineItem

submission deadline: Dec. 16th 2017 11:59pm
Note: this is a one-week exercise sheet

This is a *per-student exercise* sheet, i.e., you are not allowed to submit a group solution.

You must submit your solution in time via a pull request to the lectures exercise Git repository[1] (cf. sheet No. 1). For this, branch from the official branch `master`, and commit your solutions in a directory `sheet_06/<your last name>`. For your pull request, use the remote branch `submissions` as target for merging.

Prepare to present details of your solution during the tutorium.

This sheet consists of the following tasks:

Task 1	**A Hard-Coded TPC-H Database Query**	8 Points

Good Luck!

[1] https://github.com/Arcade-Lecture/exercises.git

Task 1 A Hard-Coded TPC-H Database Query 8 Points

TPC-H[2] is a prominent decision support benchmark on structured relational data. The TPC-H benchmark consists of several relations. One of them is the *LINEITEM* relation[3]. In this task you will implement the following query in a hard-coded fashion, and you will provide some user interaction and feedback:

```
SELECT orderkey, partkey, suppkey, comment
FROM lineitem
WHERE comment LIKE '%x%'
LIMIT n;
```

This query returns the first *n* tuples from the *LINEITEM* table whose *COMMENT* field contains the substring *x*.

The requirements for your program are:

1. When starting your program, the *LINEITEM* table (physically organized as a row-wise table) is build and filled with random data.
2. The user is informed that the program will terminate if he/she inputs "`:exit`".
3. Your program displays the database (*LINEITEM* table) size in MiB[4].
4. Your program displays the query to be executed and mention the parameter *x* and *n*.
5. Then your program goes into an endless loop (running *a-d* per iteration) until the user terminate your program by inputting "`:exit`":
    a. The user is prompted to provide a substring (needle) *x*
    b. Then, the user in prompted to provide a limit *n*. In case the user inputs a negative number, your program will ignore the limit *n* and will display all tuples rather than the first *n* tuples.
    c. Your program answers the query with given *x* and *n*, and prints the result to standard output. (Formatted string output is further explained in *Appendix 3*)
    d. After the result is printed, your program prints the number *m* of all tuples that are contained in the result set (since *n* might be less than *m*), and outputs the query response time in ms. One way to measure time distances in C is given in *Appendix 2*.

Tip: In *Appendix 4*, we provide a program skeleton that implements the required data structures and populates random tuples to the *LINEITEM* relation.

Example:

Assume your *LINEITEM* table consists of the following tuple (projected to `orderkey, partkey, suppkey,` and `comment` the for ease of understanding):

```
1026897930d, 1005966874d, 1730022865d, erjgffiybossqawzacpvmduxbmopkjyobhugofgrtylj
1264095060d, 1411549676d, 1843993368d, jmafadrrwsofsbcnuvqhffbsaqxwpqcacehchzvfrkml
2008078427d, 183650190d, 1353939664d, glnvhzfylutbhugoszuhrqzkrakdqpcxdqskrzketdhd
524688209d, 700108581d, 1566288819d, zriicfskpggkbbipzzrzucxamludfykgruowzgiooobp
1583571043d, 559301039d, 1395132002d, mbfjxjcvudjsuyibyebmwsiqyoygyxymzevypzvjegeb
1169030037d, 1333710445d, 250609978d, jviyhugodxdvywwuhfvsgvwnygaavbjgxtfennutnzon
```

Your program outputs the following to standard out:

```
Type ':exit' to exit.
Database size: 6.866455 MiB
SELECT orderkey, partkey, suppkey, comment FROM lineitem WHERE comment LIKE %x% LIMIT n.
Enter x$ hugo
Enter n (-1 for no limit)$ 2
2008078427d, 183650190d, 1353939664d, glnvhzfylutbhugoszuhrqzkrakdqpcxdqskrzketdhd
1169030037d, 1333710445d, 250609978d, jviyhugodxdvywwuhfvsgvwnygaavbjgxtfennutnzon
3 records in total, 18 ms
```

[2] **TPC-H specification document**: http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-h_v2.17.3.pdf
[3] TPC-H specification p. 16; for a suggestion on the data type mapping, see *Appendix 1*
[4] 1024 Byte = 1 KiB, 1024 KiB = 1 MiB

Appendix 1 Data Type Mapping

We show in Table 1 the mapping of TPC-H data types to corresponding C types used in this exercise.

TPC-H data type	C type	Notes
identifier	uint32_t	
integer	int_32_t	
decimal	**float**	
fixed text, size *n*	char[n]	
variable text, size *n*	char[n]	Assume fixed text, size *n*
date	uint64_t	assume timestamp since Epoch as 64bit values

Table 1: Mapping and assumptions of TPC-H data types to C types for this exercise

Appendix 2 Measure Time Distances in C programs

The following code snipped may be used to measure time distances in C programs. It stores the starting time (`start`) before a code block is executed, and stores the ending time (`end`) after the execution finished. The time it takes to execute a code block is (`end – start`).

```
struct timeval timeval;
long start, end;

gettimeofday(&timeval, NULL);
start = (long)timeval.tv_sec * 1000 + (long)timeval.tv_usec / 1000;

/* code block to execute goes here */

gettimeofday(&timeval, NULL);
end = (long)timeval.tv_sec * 1000 + (long)timeval.tv_usec / 1000;
```

Appendix 3 Formatted String Output

The `printf(`*`format`*`, `*`args…`*`)` function prints formatted strings to standard output. The parameter *`format`* is the string that should be printed to standard output. This parameter may contain one or more format sequences that starts with `%` followed by a certain character sequence (e.g., `%s`) that are replaced by arguments of a particular type in *`args…`* (e.g., "`Hello World`"). In case the character `%` should be contained in format without replacing it by an argument, this character is escaped by `%%`.

Example

```
int32_t  int_val = 1;
uint32_t uint_val = -1;
char     string_val[256] = "...";
long     long_val = 23L;
float    float_val = 42.0f;

printf("%%, an integer: %d, an unsigned integer: %ud, a string: %s, a long: %ld, a float %f\n",
       int_val, uint_val, string_val, long_val, float_val);

// outputs: '%%, an integer: 1, an unsigned integer: -1, a string ..., a long 23, a float 42.0'
```

Appendix 4 Program Skeleton

The following program skeleton may be used to implement task 1. This code implements a hand-coded implementation of the TPC-H LineItem table as a row-oriented physical record storage and generates random data for 10,000 tuples.

```
#define NUM_TUPLES 10000

struct nsm_lineitem_tuple_t {
    uint32_t orderkey;
    uint32_t partkey;
    uint32_t suppkey;
    int32_t linenumber;
    float quantity;
    float extendedprice;
    float discount;
    float tax;
    char returnflag;
    char linestatus;
    uint64_t shipdate;
    uint64_t commitdate;
    uint64_t receipetdate;
    char shipinstr[25];
    char shipmode[10];
    char comment[44];
};

struct nsm_lineitem_table_t {
    struct nsm_lineitem_tuple_t tuples[NUM_TUPLES];
};

int main(void)
{
    struct nsm_lineitem_table_t nsm_table;

    /* fill with random data */
    for (unsigned i = 0; i < NUM_TUPLES; i++) {
        nsm_table.tuples[i].orderkey      = random() % UINT32_MAX;
        nsm_table.tuples[i].partkey       = random() % UINT32_MAX;
        nsm_table.tuples[i].suppkey       = random() % UINT32_MAX;
        nsm_table.tuples[i].linenumber    = random() % INT32_MAX;
        nsm_table.tuples[i].quantity      = random() % 1000;
        nsm_table.tuples[i].extendedprice = random() % 10000;
        nsm_table.tuples[i].discount      = random() % 50 / 100.0f;
        nsm_table.tuples[i].tax           = nsm_table.tuples[i].extendedprice * 0.17f;
        nsm_table.tuples[i].returnflag    = 'a' + (random() % 26);
        nsm_table.tuples[i].linestatus    = 'a' + (random() % 26);
        nsm_table.tuples[i].shipdate      = random() % UINT64_MAX;
        nsm_table.tuples[i].commitdate    = random() % UINT64_MAX;
        nsm_table.tuples[i].receipetdate  = random() % UINT64_MAX;
        for (int j = 0; j < 25; j++) {
            nsm_table.tuples[i].shipinstr[j] = 'a' + (random() % 26);
        }
        for (int j = 0; j < 10; j++) {
            nsm_table.tuples[i].shipmode[j]  = 'a' + (random() % 26);
        }
        for (int j = 0; j < 44; j++) {
            nsm_table.tuples[i].comment[j]   = 'a' + (random() % 26);
        }
    }

    /* TODO: add code here */

    return 0;
}
```