Chair of Applied Computer Science III
UNIVERSITY OF MANNHEIM

Prof. Dr. Guido Moerkotte
Email: moerkotte@uni-mannheim.de

Daniel Flachs
Email: daniel.flachs@uni-mannheim.de

Database Systems II
Spring Semester 2019

Exercise Sheet 5
Created March 22, 2019

Exercise 1

In this exercise, we review some bit manipulation techniques.

Exercise 1 a)

Perform the following bit computations by hand:

(i) `0110 + 0010`

(ii) `0011 * 0101`

(iii) `1101 >> 2`

Exercise 1 b)

Explain the two's complement. What is the sum of a positive number and its two's complement?

Exercise 1 c)

What does the following code do, given `n` is an integer?

```
((n & (n-1)) == 0)
```

Exercise 1 d)

This weeks exercise `zip` archive contains a file `bitvector/bitvector.cc`. Implement the `setBit` and the `hasZeroBit` member functions of the `Bitvector` class.

Exercise 1 e)

Take a look at the built-in functions that the GCC compiler has to offer. You'll find useful bit manipulation instructions among them.
`https://gcc.gnu.org/onlinedocs/gcc/x86-Built-in-Functions.html`
`https://gcc.gnu.org/onlinedocs/gcc/Other-Builtins.html`

Exercise 2

Let us consider a database with the following schema.

- `Customers: {[id:int, name:char(30), discount:double, country:int]}`
- `Countries: {[id:int, name:char(30), tax:double]}`
- `Products: {[id:int, name:char(30), price:double]}`
- `Orders: {[id:int, customer:int, product:int, quantity:int, date:int, totalPrice:double]}`

Exercise 2 a)

Recall the storage layout variant *row store* and *column store* from the script.

(i) Represent the database relations from the above schema in row store layout.

(ii) Represent the database relations from the above schema in column store layout.

You do not have to write C++ code. Pseudocode that shows the main difference with respect to data organization and data structures is sufficient.

Exercise 2 b)

Download this exercise's `zip` archive from the website. The folder `mmdb` contains code that you are asked to complete. The following files are included:

- In `common`, you find a data generator that creates data with a schema as described above, as well as the basic classes representing customers, countries, products and orders (`common/types.hh`).
- In `rowStore`, you find a class `RSDatabase` that implements a simple row store.
- Additionally, in `rowStore`, you find the file `rsMain.cc` that contains a `main` function and orchestrates the flow of the program for the row store.

Implement a column store for the above schema in a class `CSDatabase`. You may use the `RSDatabase` as an orientation. You can use the provided makefile to build the row store database. Warnings like `warning: suggest braces around initialization of subobject [-Wmissing-braces]` can be ignored.

If you would only like to implement the SQL queries in the next sub-task, the `zip` archive does also contain an implementation for `CSDatabase`.

Exercise 2 c)

Implement the following the SQL queries for both the row store and the column store. Variables preceded by an $ represent parameters, i.e. only this part of the query must be changeable, the rest can be hard-coded. Hint: Implement each query as a member function of the `RSDatabase` and `CSDatabase` class.

-
  ```
  select totalPrice from
  orders
  order by totalPrice desc
  fetch first 10 rows only;
  ```

-
  ```
  select date, sum(totalPrice)
  from orders
  where date >= $date
  group by date;
  ```

-
  ```
  select c.id, c.name, count(o.id)
  from customers c, orders o
  where c.id = o.customer
  group by c.id, c.name;
  ```

-
  ```
  update orders
  set totalPrice = $totalPrice
  where id = $orderId;
  ```