INF-2700 Mandatory Assignment Nr. 2

Deadline: Monday, 16 October 2023 23.59

DBMS Elements

In this assignment, you are going to implement in C some basic elements of a database management system (DBMS). By working on the specific tasks, you will learn a number of things. First, you will gain first-hand experience on how some DBMS elements work. In addition, the implementation will involve tremendous efforts on memory management and debugging. You will learn a great deal of these, too. For example, you are strongly suggested to use debugging and program analysis tools, such as `gdb` and `valgrind`. One important quality of code is readability and documentation. You will get some experience in code documentation, too. Finally, you will also gain some understanding of database performance.

You start with a base program that you can find in out git repository under

- `shared/assignments/assignments-2-3/db2700/`.

Assume your local INF-2700 files are organized under directory `inf2700` as below:

- `inf2700/`
  - `shared/`, for the shared git repository,
  - `xyz123/`, git root for your INF-2700 assignments (assuming `xyz123` is your UIT account).
    - `assignment-1/`, for the first assignment,
    - `assignments-2-3/`, the second (i.e. this) and the third assignments.

Under `inf2700/`, run:

- `cp -R shared/assignments/assignments-2-3 xyz123/`

  Now you can find the base program under `inf2700/assignments-2-3/db2700/`.

Please spend some time studying the source code first. Then, write your own program to finish the tasks described below. Your program should be well structured.

Task 1. Code understanding

The first thing to do is understanding how the base program works. You can do the following to improve your understanding of the program.

- Read the generated documentation. At `db2700/`, run `make doc` to generate the documentation of the API and data structures. Then start reading from `./doc/html/index.html` using your favorite web browser.
- Read the source code. Draw some diagrams while reading to clarify your understanding.
- Read the higher-layer program to understand how to use the lower-layer API. For example, the *front* layer uses the *schema* layer; the *schema* layer uses the *pager* layer.

- After running `make`, you get two executable files `run_test` and `run_front`. Try to run them in different ways to get a feeling how the programs work. Do whatever modifications and experiments you'd like to. Run them with option `-h` to get a help message.
- In order to keep your modifications under control, you should keep the base program in a `base` git branch. You then create new git branches and do your exploration there. This way, you are not afraid of making mistakes, because you can always roll back to a previous state or switch to a different branch.
  - Under you git root `inf2700/`, create a `.gitignore` file to make sure that you *do not* include any temporary table files and executables in git. These can be huge and waste a lot of git spaces.
- The base program has a number of `put_*_info()` procedures. Use them to observe intermediate system status.
- Attend the weekly colloquium sessions to show your understanding (teaching is one of the best ways of learning), listen to others' understanding and ask what you doubt about. You could also ask questions elsewhere, for example in Discord.
- Report bugs in the base program and hopefully also your debugging findings.

Task 2. Extending the types of queries

In the base program you have obtained, queries are restricted to equality search on integer attributes. That is, for query

select *attrs* **from** *table* **where** *attr op val*;

attr is restricted to an attribute of integer type, *op* is restricted to = for equality and *val* must be an integer value.

Extend the base program, such that queries on integer attributes are not restricted to equality search. More specifically, *op* can be one of <, <=, >, >= and !=.

You must test your program with a relatively large table. Therefore, you should also write some code that can generate a table of an arbitrary size.

Task 3. Binary search

The base program uses linear search to find records.

Extend the base program to use *binary search* on an integer field. Restrict to equality queries. Assume that there is no duplicate of rows with the searched attribute.

To test your program, you should also be able to generate a table of an arbitrary size, and the records in the table are ordered on a given integer field, without any duplicate. Do queries on that field that return all records with the given value.

For a relatively large table, report the performance of your program using linear search and binary search. The performance should *not* be measured with the actual time spent for the queries. Instead, use the profiling capability provided by the base program.

Task 4. Comparison with B^+-tree

This task is only on paper. No programming is required.

Now consider that the table is stored in a B^+-tree organised file.

Compare both the storage size of the table and the performance of queries between the solution you made in Task 3 and a B^+-tree organised file. If necessary, make your own assumption, such as what an index entry looks like.

Hand-in

Commit your final solution in the git `master` branch and push it to the INF-2700 git server `inf2700.cs.uit.no` by the deadline.

Do *not* include the data files for database tables and the executables. These should be able to be re-generated with your code. (That is, run `make cleanall` before you add files for git commits.)

In addition to the source code, you must hand in a report `report-assignment2.pdf` that includes

- a description of your design and implementation,
- instructions on how to run your program and experiments,
- performance of linear search and binary search,
- any special observations you have made, lessons you have learned and problems you have experienced,
- discussions for Task 4.

Place your report in the `assignments-2-3/docs/` directory.

Enjoy coding and good luck!