Databases, Assignment 3, 2022/23

Due: Dec 15, 23:59

Consider the following database
`Professor (id:int, name:char(50), address:char(50), age:int, department:float)`
`Course (cid:char(25), cname:char(50), credits:char(30), teacher:int)`

- `teacher` is a FK to `Professor`
- No attribute can be `null`.

Task
Create a java program that connects to your database (the one used for Assignment 2 from the sci-didattica.unitn.it) and performs the following operations in order:

1. Drops the above two tables from the database if they already exist.
2. Creates the two tables as described above.
3. Generates 1 million (random[1]) tuples, so that each tuple has a different value for the `department` attribute and inserts them into the table `Professor`. Make sure that the last inserted tuple, and only that one, has the value 1940 for the `Department` attribute.
4. Generates 1 more million (random[2]) tuples and inserts them in the table `Course`.
5. Retrieves from the database and prints to `stderr` all the `id` of the 1 million professors (one per line).
6. Updates all tuples that have value 1940 as `department` and makes them to have a `department` equal to 1973 – (your query should work even if there are many tuples that have value 1940 in the attribute `department`).
7. Selects from the table `Professor` and prints to the `stderr` the `id` and the `address` of the professor with `department` 1973 (each id and address pair appear on its own line, and the two values are separated by comma).
8. Creates a B+tree index on the attribute `department`.
9. Retrieves from the database and prints to the `stderr` the `id` of the 1 million professors (one per line).
10. Updates all the tuples that have value 1973 as `department` and makes them to have a `department` equal to 1974 -- (your query should work even if there are many tuples that have value 1973 in the attribute `department`).
11. Selects from the table `Professor` and prints to the `stderr` the `id` and the `address` of the professors with `department` 1974 (each id and address pair appear on its own line, and the two values are separated by comma).

For each of the above operations you need to report (print to the `stdout`) the time it took to execute it. To do so you may keep in a variable the time before starting the execution (in nanoseconds), then get the system time after the execution has been completed and the difference in nanoseconds is the approximate time it took for the step to be executed. Your standard output stream should be of the form:

```
Step 1 needs 10 ns
Step 2 needs 27 ns
Step 3 needs 77 ns
...
```

Notes & Delivery

- You need to deliver a single .java file named as A3_XXX.java where the XXX is your matricola. The file is your program that implements the above steps. You may assume JDBC to be available in the classpath (java). Do not use any other external library, besides the default Java libraries. The delivery is done via the Google form https://forms.gle/HcxfkWrxPMs8PoTr8

[1] Note that 1,2,3,4,5, … or any other auto increment function is not considered random. You can use any pseudo random generator the programming language offers or implement one of your own.

[2] Note that 1,2,3,4,5, … or any other auto increment function is not considered random. You can use any pseudo random generator the programming language offers or implement one of your own.

- You also need to deliver a second file, that is a text file called A3_XXX.txt, where XXX is your matricula. In that file, you should place the output of your program and some text in which you comment on all the times that are reported. For example, you may say : I see that the second time is larger than the first because the database needs to read more data, or needs to search a larger tree, The delivery of this second file is done via the google form: https://forms.gle/drdYikBfMoZKkwLM8
- You can update your solutions (the delivery), by updating a new file, up to a maximum of 10 times.