Database Systems - CSCI-GA.2433-001 - Fall 2019

Professor: Dennis Shasha

Homework 3 - Due: Monday, November 18, 2019 at 4:00 PM

1. This question is worth 60 points. Please find in-memory implementations of B-trees and hash structures on the internet or elsewhere. These can be from the same or different authors. (This will be useful for your final project) You will fill them each with data that we provide here: http://cs.nyu.edu/cs/faculty/shasha/papers/myindex Then make your program able to run from standard input a file of commands (in the format insert(key,value), delete(key). search(key)). There will be one program for Btree and one for hash. Each search should return the value associated with that key. We will test your program with up to 60 such operations.

   For example, our test file might have operations

   ```
   insert(3,98)
   delete(5)
   search(8)
   ...
   ```

   Your answer file will show the results of each search, i.e. the value corresponding to the key (or if the key is not present then simply "not present").

   An insert(k,v) should be interpreted as an "upsert" meaning that if there is already a record with key value k, then that record's value should be changed to v. When we execute your program (which you should submit in source form), it should show your timings for all operations together. Your code should specify where you got the implementations for Btree and Hash from. Also your program write the total time and the resulting table to standard out after all operations so we can check for correctness of the modifications. Please use Python 3 without any special libraries except for the ones for B-trees and hash structures. Java is also an option, but again no special libraries except for B-tree and hash structure libraries. For deletes, just mark the entry as deleted so you don't have to do a shift.

   Your program should store the data as an array of records. Your hash and B-tree structures, given a key, produce an index into that array. For example, your hash structure could be a Python 3 dictionary that indexes into the array (i.e. gives a key value and an array index).

2. This question is worth 20 points. Show the script in your database system of choice. Recall a schema similar to that from homework 1:

   Employee(ID, Name, Salary, Manager, Department)
   Course(EmpID, CourseID, Prof, Grade)

   Here is data to generate a large enough database so that the following queries require more than one second each on our test computer, but may take less time on yours.

   http://cs.nyu.edu/cs/faculty/shasha/papers/emp

   http://cs.nyu.edu/cs/faculty/shasha/papers/course

To load data, you can do something like this:

create table Employee(ID int, Name varchar(20), Salary int, Manager int, Department varchar(20)); create table Course(EmpID int, CourseID int, Prof varchar(20), Grade int);

– Then copy the data from my site to local files emp.txt and course.txt.

– Load data into tables, your file path may be different

```
LOAD DATA LOCAL INFILE 'emp.txt'
INTO TABLE Employee FIELDS TERMINATED BY '|'
LINES TERMINATED BY '\n'
(ID, Name, Salary, Manager, Department);

LOAD DATA LOCAL INFILE 'course.txt'
INTO TABLE Course FIELDS TERMINATED BY '|'
LINES TERMINATED BY '\n' (EmpID, CourseID, Prof, Grade);
```

Show the timings and the results.

(a) Find the number of employees who earn at least 50 more than their managers.

(b) For those departments in which more than one person takes courses, find the average salary by department.

(c) Find the average salary earned by people taking the course of prof1038.

Now show how to add indexes in such a way that each query requires less time. That is, show the create index statements to use and give new timing results.

3. This question is worth 20 points. Perform the following experiment using the emp file from above.

   (a) Create an empty emp table without indexes. Perform 100,000 inserts (taken in any order, but without duplicates) from the emp file into the employee table. Record the timing. These will be insert statements with explicit values, one row per insert statement. To do that, you will use a programming language like Python or Java and interface with the database for each insert.

   (b) Recreate an empty emp table but this time with a unique index on ID and an index on department. Perform 100,000 inserts from the emp file into the employee table Record the timings. If an insert with the same ID as an existing record comes in, then refuse that new insert.

   Discuss any conclusions you might draw.