CPSC 321: Homework 9 *Due: Thursday, December 3rd*

Reading. Continue with chapters 8 and 9 in ZyBooks and complete all of the corresponding reading "*activities*".

Technical Work: SQL Indexes. The goal of this part is to run some simple "experiments" to see how using indexes can alter the cost of database updates and queries in terms of execution time.

STEP 1. Define an Employee table with three attributes:

- an integer-valued `emp_id` attribute, which is the primary key
- an integer-valued `salary` attribute
- a string-valued `title` attribute

Use a script file (e.g., called `hw9-schema.sql`) that contains the create table statement, which will make some of the following steps easier.

STEP 2. Write a program to generate test data to insert into the Employee table. You can use whatever programming language you want to generate your test data. Your program must be able to generate an insert statement that inserts n rows and saves the insert statement within an SQL script file. You will use the program to generate three different files: one with 10,000 rows, one with 100,000 rows, and one with 1,000,000 rows. In my implementation, I named these files `hw9-data-10000.sql`, `hw9-data-100000.sql` and so on, and used a parameter within the program to select the corresponding value of n. The following is an example of what these files should look like.

```
INSERT INTO Employee VALUES
(1, 12740, "engineer"),
(2, 94704, "manager"),
(3, 81058, "salesperson"),
(4, 27955, "administrator"),
...
```

When generating the files, `salary` values for each row should be randomly selected from the range 12,000 to 150,000 (e.g., in Python using the `random.randint` function) and each row should have one of four possible titles ("administrator", "engineer", "manager", "salesperson"). The titles should be evenly distributed across rows, i.e., 25% of rows should have the title "administrator", 25% of rows should have the title "engineer", and so on.

STEP 3. Collect baseline time measurements for loading each data file (test case) and executing the SQL query "`SELECT * FROM Employee WHERE salary >= 29000 and salary <= 31000`". The result of this step will be six values, as shown in the example table below (note that your times will likely differ from these).

Rows	*Update (seconds)*	*Query (seconds)*
10000	0.2	0.01
100000	1.3	0.06
1000000	14.4	0.60

To generate the data from within mysql (for the 10,000 row case) do the following steps:

```
mysql> source hw9-schema.sql

mysql> source hw9-data-10000.sql

mysql> SELECT * FROM Employee WHERE salary >= 29000 and salary <= 31000
```

The result of the last two commands will report corresponding timing data.

STEP 4. Extend your script from STEP 1 to create an index on `salary`. Redo STEP 3 to get update and query times for the same test data sets but with `salary` indexed. Note that the update times should generally increase and the query times should generally decrease.

STEP 5. Extend your script from STEP 4 to also include an index on both `title` and `salary` (i.e., a composite index with `title` followed by `salary`). Define your own query having WHERE clause conditions over both `salary` and `title` that shows an improvement in query execution time over a non-indexed table. Note that your query needs to be well thought out in terms of determining appropriate conditions to take advantage of the indexes. Perform the query both on the test data sets without the index and with the index to create baseline measurements and indexed measurements for your example query. Finally, use MySQL's explain command over your queries (with and without indexes) and include the information generated by MySQL in your assignment write-up (see below).

STEP 6. Submit all of your source code and SQL files to your GitHub repository for the assignment. Make sure your code is well commented and properly formatted. Also include a write-up of your assignment as a single PDF file called `hw9.pdf` that contains: (1) your `hw9-schema.sql` file, (2) your test data generation program, (3) the MySQL runs showing the timing data (you do not need to include the output of the queries), (4) the result of running the explain command over your queries (both with and without indexes), and (5) an explanation of the results from your experiments. In particular, your explanation should have four tables similar to the table above showing update and query execution times for the baseline (non-indexed) and indexed cases. Thus, you should have two tables of measurement data for each query, where one table shows the baseline update and query times and the other shows the indexed update and query times. Be sure to clearly label each table. In addition, you should discuss the results within your analysis document in terms of the impact of using indexes for your experiments