M20Temp18: DATA SYSTEMS PROJECT PHASE 1

Soft Deadline : 11:55 PM , 25 September 2020

In this phase you are supposed to complete the following tasks and subparts.

Programming Language : C++

Task-1 : Indexing

1) **B+ Trees :** Implement B+ trees indexing for the tables in the database on an indexing column. The index structure you build should support the following query types.
   - **INSERT** r : Inserts the row r in the table and the index.
   - **FIND** x: Find the row/rows which have value x for the indexing column.
   - **DELETE** r : Delete the row r from index structure and the table

2) **Linear Hashing :** Implement linear hashing indexing for the tables in the database on an indexing column with uncontrolled splitting using overflow as the splitting condition.The index structure you build should support the following query types.
   - **INSERT** r : Inserts the row r in the table and the index .
   - **FIND** x: Find the row/rows which have value x for the indexing column.
   - **DELETE** r : Delete the row r from index structure and the table.

SYNTAX

INDEXING

The basic index statement has been implemented. You have to expand the syntax to include two options - FANOUT and BUCKETS

```
INDEX ON <column_name> FROM <table_name> USING
<indexing_strategy> [FANOUT <number_of_children_pointers>|
BUCKETS <bucket_count>]

Ex:
INDEX ON <column_name> FROM <table_name> USING BTREE FANOUT
<number_of_children_pointers>

INDEX ON <column_name> FROM <table_name> USING HASH BUCKETS
<bucket_count>]
```

- Here `<column_name>` is the column on which we build the index and `<indexing_strategy>` is the strategy for index i.e either **BTREE**(to indicate B+Tree indexing) or **HASH** (to indicate dynamic linear hashing) (or **NOTHING** if you want to delete the index). When the indexing strategy is BTREE, the FANOUT is provided as another argument. When the strategy is HASH, the number of initial buckets are provided as an argument

- For the purpose of this project you can assume we will only expect an index on one column of the table and won't expect multiple indices on different columns (although you may choose to do so if you want). You can choose to throw an error if the user of your system tries to index on another column before deleting the current index (using the **NOTHING)** option or you can choose to overwrite the existing index.

- The file given to you need not be ordered by the search key but you will have to implement a dense clustered index

- For the purpose of this project, you can assume that the index directory fits in the main memory

- In addition to implementing the index, note that depending on your implementation you will have to suitably modify other operators like project, rename and so on.

FIND

The find is implemented through the **SELECT** command. You have to optimize the current code for the selection operator to use indices on the select arguments if it exists.

For example, given a table `Table(A, B)` on which you perform the following indexing command

```
INDEX ON A FROM Table USING HASH BUCKETS 5
```

Then the following select statement will have to use the index during execution

```
A_IS_5 <- SELECT A == 5 FROM Table
```

INSERT

```
INSERT INTO <table_name> VALUES <value1>[,<value2>]*
```

Where `<table_name>` is the name of the table you are inserting into and `<value1>[,<value2>]*` is a comma-separated list of integers in the same order as that of the order of the columns in the table

DELETE

```
DELETE FROM <table_name> VALUES <value1>[,<value2>]*
```

Where `<table_name>` is the name of the table you are deleting from and `<value1>[,<value2>]*` is a comma-separated list of integers in the same order as that of the order of the columns in the table representing the WHOLE row to be deleted. If no row such row exists, deletion is not carried out.

Task-2 : Sorting (2 Phase Merge Sort)

Now that you have indexing implemented, you will have to implement the sorting function. You will be given a file of tuples which is larger than the main memory and you are required to sort the entries of the file.

Note: There can be duplicate entries in the file.

The following syntax has been implemented already in the system

```
<new_table_name> <- SORT <table_name> BY <column_name> IN
ASC | DESC
```

Here <table_name> represents the table in the database you want to sort, the`<table_name>mn_name> ` is the column on which you want to sort,the sorting order is either **ASC or DESC** based on whether the output should be ascending or descending.

First, we want you to overload the syntax with the following additional BUFFER option

```
<new_table_name> <- SORT <table_name> BY <column_name> IN
ASC | DESC BUFFER <buffer_size>
```

Where `<buffer_size>` is the number of blocks available in main memory that you can use to perform 2-Phase Merge Sort.

For the implementation we want you to make the following considerations.

- **[PART 1]** If the column you want to sort by is **INDEXED**, then you should use the index to sort the file (since it's a clustered index, the original table should already have been sorted by this column)

- **[PART 2]** If the column you want to sort by is **NOT INDEXED**, you have to use 2-Phase Merge Sort. The `BUFFER` option in the statement syntax indicates the number of blocks you can use to perform this 2-Phase Merge Sort

BONUS

For the project, you are allowed to assume that the index directory fits in the main memory. For the bonus, you will have to implement indexing directories larger than the main memory (the index directory need not fit in the main memory).

HINT: Multi-level indexing, sparse indices

SUBMISSION INSTRUCTIONS:

For this part of the project you will be graded on two factors: **correctness and efficiency**

We will run a series of commands and time the execution period your systems take.

- For correctness, we will export resultant tables (sorting and selection).

- For efficiency we will compare execution times. We will not check the efficiency of building the index itself (execution of the INDEX command), but we will check the efficiency of the SELECT, INSERT and DELETE commands. We will do the same for sorting (checking if sorting by indexed column vs non-indexed column, changing buffer sizes and so on)

From this phase onwards, you will only be provided with a soft deadline.

The intent of doing so is to give you time to work on the project, including any major upheaval you might want to do with the code base or to write one of your own. It is a given that the syntax must remain the same. We will release the next few phases shortly.

The hard deadline for all phases of the project (excluding phase 0) will be: 8th November 2020, 11:55PM

No submissions will be accepted after this hard deadline.

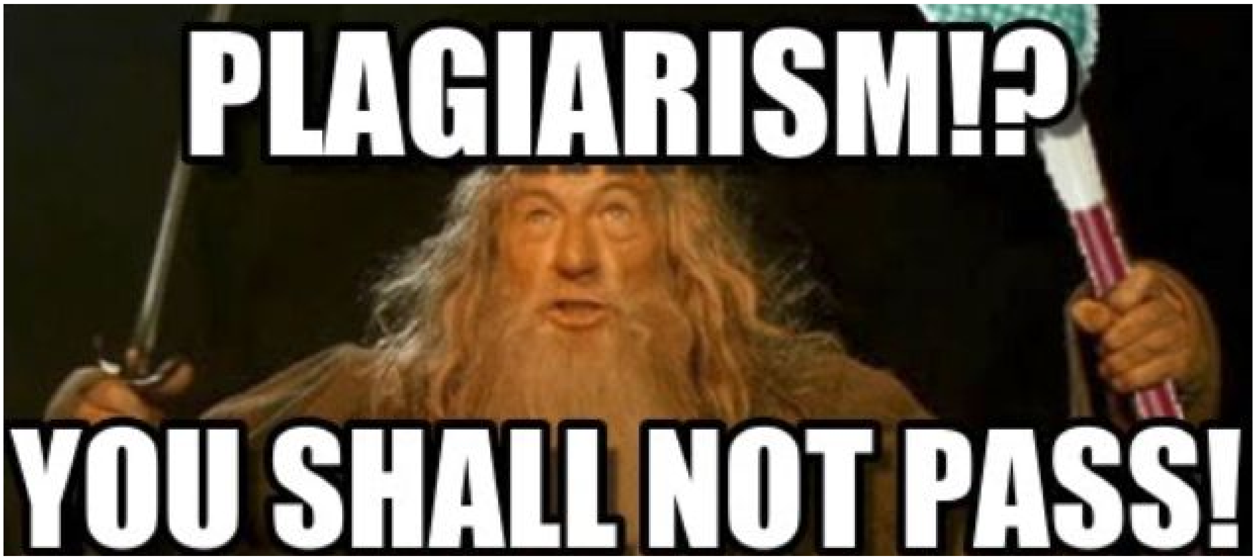
Note: Kindly follow the submission instructions before submitting your code as otherwise there will be a sizable penalty.

DOCUMENTATION

From this phase on you will also be graded for documentation of your project. To document your code, we expect you to use standard docstrings for functions and classes apart from normal single line comments.

You can use this link for reference

PLAGIARISM WILL LEAD TO POTENTIALLY GETTING AN F IN THE COURSE



References:

- http://delab.csd.auth.gr/papers/LinearHashing2017.pdf
- Btrees - https://www.cs.usfca.edu/~galles/visualization/BTree.html (note you have to implement B+Trees and not Btrees)
- https://www.baeldung.com/cs/b-trees-vs-btrees
- Abraham Silberschatz, Henry Korth, and S. Sudarshan. 2005. Database Systems Concepts (5th. ed.). McGraw-Hill, Inc., USA.