Data Systems Project Phase 2: Optimization

Hard Deadline: 11:55 PM, 8th November 2020

Total Marks - 75
Bonus - 25

In this phase you will have to implement the following advanced operators and optionally optimize them

BULK_INSERT (25 marks)

In the previous phase, you implemented an `INSERT` operator which inserts one per invocation. In this phase you have to insert multiple rows into the table using the `BULK_INSERT` operator.

Syntax

The general syntax of the `BULK_INSERT` is as follows

```
BULK_INSERT <csv_file_name> INTO <table_name>
```

If you have a table called `R` with columns `(A, B)` with the initial raw file (`R.csv`) containing the following

```
A, B
1, 2
2, 3
3, 4
```

You load the table with the following command

```
LOAD R
```

You have to insert the contents of the file(`T.csv`) present in the `data` folder into table `R`. The file `T.csv` will be of the same format as `R.csv` i.e. will have the same number of columns, with the first line being the column headers and all column headers being the same as that of table `R`. For example, if `T.csv` contents are as follows

```
A, B
4, 5
5, 6
6, 7
```

You have to insert all the rows from `T.csv` into `R` upon calling the bulk insert command

```
BULK_INSERT T INTO R
```

Now, table `R` contains the following rows

```
A, B
1, 2
2, 3
3, 4
4, 5
5, 6
6, 7
```

- You have to implement syntactic and semantic checks

While performing semantic checks, you also have to check to make sure the headers of the insertion file are the same as the headers of the table you're inserting into

- You have to make your operator index-compatible i.e. you should update your index structures when inserting these rows
- You will be graded on the correctness of your implementation

GROUP BY – AGGREGATES (25 marks)

Another advanced query you will have to implement is the `GROUP BY` query along with aggregate operators `MAX`, `MIN`, `SUM`, `AVG` where these aggregate operators take on their usual meaning. For those of you unfamiliar with the `GROUP BY` query, this query clusters the rows of a table based on the attribute value of the grouping attribute, and a query result is returned for each "group" or supposed element in the grouping attribute's domain

Syntax

The syntax of the `GROUP BY` statement is as follows

```
<new_table> <- GROUP BY <grouping_attribute> FROM <table_name> RETURN
   MAX|MIN|SUM|AVG(<attribute>)
```

For example, if you loaded a table `R` with the following contents

```
A, B, C
1, 2, 3
1, 4, 6
1, 6, 12
```

```
1, 8, 15
2, 2, 18
2, 4, 21
2, 6, 24
2, 8, 27
```

The expected output of executing `GROUP BY` command is detailed below

```
T1 <- GROUP BY A FROM R RETURN MAX(C)
```

New created table `T1` contains two columns `(A, MAXC)` with the following rows

The output column name will be the aggregate operator concatenated with column name it's operating on

```
MAX|MIN|SUM|AVG(X) -> MAX|MIN|SUM|AVGX
```

So, `MAX(C)-> MAXC`

```
A, MAXC
1, 15
2, 27
```

The following query produces `T2`

```
T2 <- GROUP BY C FROM R RETURN SUM(B)
```

Where `T2` has 2 columns `(C, SUMB)` with the following rows

```
C, SUMB
3, 2
6, 4
12, 6
15, 8
18, 2
21, 4
24, 6
27, 8
```

The following query produces `T3`

```
T3 <- GROUP BY A FROM R RETURN MIN(A)
```

Where `T3` has 2 columns `(A, MINA)` with the following rows

```
A, MINA
1, 1
2, 2
```

Again,

- You have to implement syntactic and semantic checks
- You will be graded on the correctness of your implementation

ALTER TABLE (25 marks)

The last advanced query you have to implement is the `ALTER TABLE` command which add or deletes a column of a table.

syntax

The syntax of the `ALTER TABLE` statement is as follows

```
ALTER TABLE <table_name> ADD|DELETE COLUMN <column_name>
```

Again for example, if you have table `R` loaded into the system as follows

```
A, B
1, 2
2, 4
3, 6
4, 8
```

The following command adds a column to the table `R`.

All new columns are initialized with value `0`

```
ALTER TABLE R ADD COLUMN C
```

The table `R` now contains the following data

```
A, B, C
1, 2, 0
2, 4, 0
3, 6, 0
4, 8, 0
```

You might wonder how adding a column of `0`s with no update command helps, you should remember the `INSERT` command works, so it's possible to insert a new record with a non-zero value for attribute `C`

The following command deletes column `A` from the table `R`

```
ALTER TABLE R DELETE COLUMN A
```

To produce the following changes in `R`

```
B, C
2, 0
4, 0
6, 0
8, 0
```

It's possible to delete an indexing attribute of a table

- You have to implement syntactic and semantic checks
- You have to make your operator index-compatible i.e. you should update your index structures when inserting these rows
- You will be graded on the correctness of your implementation

BONUS - Optimization (25 marks)

By just implementing the 3 advanced queries correctly, you will get 75 marks (25 marks each). This assignment is graded for 75 marks.

For this phase you are encouraged to attempt the bonus section to get an additional 25 marks. The focus of this section is to optimize the advanced queries you implemented in this phase i.e. `BULK_INSERT`, `GROUP BY and ALTER TABLE`.

What are you allowed to change to improve the performance of these queries? **Anything**

You could change

- the page layout
- the buffer manager strategy
- the algorithm
- implicitly implement an index
- store and update extra data
- or anything else

While optimization is encouraged, it is not encouraged at the cost of correctness, so all your other queries must still work correctly even if inefficiently. The syntax must remain the same, you may assume some default parameters within your code if necessary

To help you strategize, we will detail the grading scheme we will follow for this section

For each advanced query, we will grade teams by their average performance time. For each advanced query, the top 5 or so fastest systems will get 25 marks for their teams and the score will scale based on your system's performance (next 5 fastest teams get 20 marks and so on. The last few teams will get 1-2 marks)

Even though theoretically this means a team could possibly get 75 marks in the bonus if they're sufficiently fast in all 3 queries, the score will get cut off at 25 marks for the bonus. **So any team scoring higher than 25 marks in total in the bonus will get exactly 25 marks in the bonus**

So in theory you could choose to perfectly optimize **just 1 query** really well and you could get full marks in the bonus.

Submission Guidelines

- If you choose to attempt the bonus section (which is encouraged), you have to additionally submit a file called `strategy.md` in the `docs` folder detailing how you optimized your operators
- Ensure that the following parameters are set

```
BLOCK_SIZE = 8
BLOCK_COUNT = 10
PRINT_COUNT = 20
```

Since this is the last phase of the project, you have to merge all your code to your `master` branch including code for Phase 0. **Failing to do so will result in losing 40% of your project grade**. The final executable should be `server`. Ensure the correctness of the `EXPORT` function. Most of the grading process relies on this operation.

Any plagiarism will lead to getting an F in the course