Project Phase 0 - Code Familiarity

> **Deadline - 11:55 PM, 20th September 2021**

The objective of Phase 0 is to get you familiar with the boilerplate code. This phase consists of two parts: (1) Writing queries and (2) Extending SimpleRA to include matrices

GitHub Classroom and Project Setup

We will use GitHub classroom to manage the project this year. Please read all the instructions below before registering.

1. Please make sure you and your teammate are on a call together while doing this registration.
2. The registration of members of a team has to happen one after another. The first teammate to register has to click on this link
3. In the page that opens, the list of registered team numbers is displayed. If both team members have followed these instructions, your team should not exist in this page.
4. The first team member should then create the team by accepting the assignment and entering the team name as the team number provided in the teams list posted on Moodle (`courses.iiit.ac.in`)

> For example, if your team number if 20, you should just enter `20` as your team name and not `team-20`. The repository created will be named `project-20`

5. Now, the second team member can register by clicking on this link. In this page, the team number should now be displayed and the second team member should join the team.
6. We recommend that you perform the registration on call so your team does not get registered twice
7. Ensure you follow the instructions provided in the README of the boilerplate

After registration, you should be able to see a repository named `project-<team_number>` with the boilerplate code. All the work you do has to be pushed to this repository.

> Please ensure you complete this registration process before **11:55 PM, 5th September 2021 (Sunday)**
> If you encounter issues during the registration process, please send an email to datasystems_ta_m21@IIITAPhyd.onmicrosoft.com

---

Part 1 - Writing Queries

The first part of this phase is to get you familiar with the query language implemented. Please refer to the `overview.md` or `overview.html` document present in the `docs` folder of your repositories to get the list of implemented commands.

A basic version of the employee database with various tables has been provided to you in the `data` folder. You can find a detailed description of the dataset at this Link. Remember your queries will be evaluated on a superset of this database.

Q1: List all employees with salaries >= 30000
`output: Q1.csv - Q1(Ssn, Salary)`

Q2: List all pairs of employees and their supervisor both of whom have the same birth date
`output: Q2.csv - Q2(Ssn, Super_ssn, Bdate)`

Q3: List all projects which have both male and female employees working on them
`output: Q3.csv - Q3(Pno)`

Q4: List all projects (along with the employee and his supervisor) on which both an employee and his/her supervisor work
`output: Q4.csv - Q4(Ssn, Super_ssn, Pno)`

> **Note:** If a supervisor S working on project P has 2 employees A and B working under him on project P, there should be 2 entries.
>
> ```
> 1 A, S, P
> 2 B, S, P
> ```

Q5: List all projects belonging to department 1 or 5
`output: Q5.csv - Q5(Pnumber, Dnum)`

> The table you generate can have *duplicate rows*. The *order* of rows in the resultant tables does not matter

Submission Instructions

All your queries must be written in a file called `queries.txt` in the `data` folder of your repository. The contents of the `queries.txt` file should follow this format

```
<Q1 query>
...
<Q5 query>

EXPORT Q1
...
EXPORT Q5
QUIT
```

- Each query can consist of *multiple statements*
- *Every query should export a table* (Q1 - Q5). If the result of a query is an empty set, please still include the `EXPORT` command (it will print an error statement if the table doesn't exist, which is okay)

We will check the contents of the exported tables, so please ensure you follow the provided output format - table names, column headers and column orders

---

Part 2 - Extending SimpleRA to Include Matrices

The objective of the second part of the project is to get you familiar with the boilerplate code structure. In this part, you have to extend the boilerplate which currently only handles tables, to handle matrices as well.

Square $n \times n$, ($n \leq 10^5$) matrices will be provided as `.csv` files in the `data` folder. For instance, Given A

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

To load A into the system, a file called `A.csv` should be present in the `data` folder with the following contents

```
1,2
3,4
```

In matrices, there are no column names, so unlike tables, the first row of the `.csv` file will not contain the column names, it contains the first row of the matrix. As a part of this phase, you have to implement the following 4 commands

`LOAD MATRIX <matrix_name>`

Like the `LOAD` command for tables, the `LOAD MATRIX` command should load contents of the `.csv` and store it as blocks in the `temp` directory.

Therefore, to load matrix A into your system, you should be able to run

```
> LOAD MATRIX A
```

You are free to experiment with the page layout. It is important to note that you have to appropriately change the cursor and the buffer pool to handle matrices.

In the case of tables, we implicitly assume a block is at least as big as a table row, but this isn't the case for matrices. A whole row or column of the matrix need not fit into a block. The maximum allowed size of a block is `8KB`. The `BLOCK_SIZE` parameter in the boilerplate is in KB.

Sparse Matrices As a part of this task, we want you to treat sparse matrices as different from normal matrices to optimise storage and processing.

An $n \times n$ matrix is defined to be **sparse** iff at least 60% of the matrix consists of 0s

The following are examples of sparse matrices - $\begin{bmatrix}0\end{bmatrix}$, $\begin{bmatrix}0 & 0\\ 0 & 2\end{bmatrix}$

You have to store sparse matrices in a **compressed format**. You are free to choose whichever compression technique you want and consequently the page layout. The user of the system should be agnostic of how the matrix is stored in your system i.e. everything you can do with non-sparse matrices in your system, should be possible with sparse matrices

`PRINT MATRIX <matrix_name>`

Like the `PRINT` command for tables, you must implement a `PRINT MATRIX` command that prints the matrix on the terminal. If the matrix is big (i.e. $n > 20$), print only the first 20 rows.

For example, the following output is expected

```
> PRINT MATRIX A
1 2
3 4
```

`TRANSPOSE <matrix_name>`

You have to implement a `TRANSPOSE` command that transposes the matrix **IN PLACE**. Matrix transpose is an update operation, so the initial matrix has to be updated when transposed.

> Transposing a matrix **IN PLACE** means not using any additional disk blocks. You are allowed to use a limited amount of main memory (say 2 block). You have to transpose the matrix and write it back into the same blocks the matrix was stored in.

To transpose matrix A you would run the following command

```
> TRANSPOSE A
> PRINT MATRIX A
1 3
2 4
```

> The `TRANSPOSE` command should also work for sparse matrices. If a matrix is sparse, so is its transpose. Therefore, you have to transpose a sparse matrix into the same blocks the original sparse matrix was stored in. This is important to keep in mind when you decide on the compression format.

During evaluation we will monitor both the disk blocks (the `temp` directory) and the main memory usage.

`EXPORT MATRIX <matrix_name>`

Just like for tables, the `EXPORT` command should write the contents of the matrix named `<matrix_name>` into a file called `<matrix_name>.csv`

Therefore, if matrix A has been transposed using the `TRANSPOSE A` command and `EXPORT MATRIX A` is called, the contents of the `A.csv` file in the `data` folder should be updated like so

```
1,3
2,4
```

> For every command, you have to perform appropriate syntactic and semantic checks.

Submission Instructions

All your code should be pushed to your GitHub repository. At the deadline, your codebase will be automatically downloaded.

In addition to the code, you also have to submit a report titled `matrix.md` that should be present in the `docs` folder of your repository. The contents of your repository should contain the following information

- Page layout used to store matrices
- Compression technique used for sparse matrices and the corresponding page layout
- Compression ratio as a function of the percentage of sparseness of the matrix (If you cannot specify this, a reasonable explanation with examples will do)
- Describe your in-place transpose operation and how(if) it needs to be changed or altered for sparse matrices

For any doubts please use the doubts document exclusively. If you need to interact with a TA, please email us at - datasystems_ta_m21@IIITAPhyd.onmicrosoft.com and someone will respond with an appropriate time slot.

This course is intolerant of plagiarism. Any plagiarism will lead to an F in the course.