Assignment 3

CS 348 Fall 2014

Introduction

The goal of this assignment is to develop your programming skills when querying a database server through an external program. The assignment also highlights a drawback of RDBMs in that they are excellent for storing relational data but not necessarily optimal for specific applications where the data or operations on the data are not relational in nature.

You are **required** to complete Assignment 0 before starting this assignment. The schema and table definitions for this assignment have already been provided to you and if you have completed Assignment 0 you can begin writing code for this assignment immediately.

Background

A sparse matrix is one in which most of the entries are zero. The following example of a sparse matrix is taken from the Wikipedia entry for sparse matrix and it shows a matrix that contains only 9 nonzero elements of the 35, with 26 of those elements as zero.

$$\begin{pmatrix} 11 & 22 & 0 & 0 & 0 & 0 & 0 \\ 0 & 33 & 44 & 0 & 0 & 0 & 0 \\ 0 & 0 & 55 & 66 & 77 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 88 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 99 \end{pmatrix}$$

When storing a large number of matrices (imagine a few billion matrices) it is infeasible to create a table for each matrix because the database schema and catalogue will become too complex to manage. A neat way to store a large number of sparse matrices in an RDBMS is to use the following two relations that can accommodate any number of matrices of arbitrary dimensions:

`MATRIX`: (int `MATRIX_ID`, int `ROW_DIM`, int `COL_DIM`)

`MATRIX_DATA`: (int `MATRIX_ID`, int `ROW_NUM`, int `COL_NUM`, decimal(10,5) `VALUE`)

In this schema a tuple in the `MATRIX` relation such as (3,4,5) denotes the fact that the database contains a matrix with ID 3 that has dimensions 4x5. Similarly a tuple in the `MATRIX_DATA` relation such as (5,1,2,-5) represents the fact that in Matrix with ID 5 has a nonzero value at the first 1st row and 2nd column and it is -5. No zero values are ever stored in the database.

There are several important constraints/points to note:

a) For every unique Matrix ID encountered in the `MATRIX_DATA` relation there *must* exist a matrix definition in the `MATRIX` table. In other words there *should* be a foreign key relationship.
b) Since we are dealing with sparse matrices the `VALUE` attribute cannot be zero for any tuple `MATRIX_DATA` because we will be storing redundant information. This is simply done to save space in the database in cases where most of the matrix is full of zero entries.

c) Similarly there are many other constraints that are implicit (e.g., primary key: (row/matrix item definitions should not be repeated, row/column numbers cannot be negative nor exceed the dimensions of the matrices)

In this assignment you shall assume that no such constraint checking exists in the database and you have to implement them in your code when manipulating matrices. In short you will have to build a matrix storage/manipulation engine using an SQL as storage.

Example
Let us reconstruct a matrix to demonstrate how this schema is used. Say we execute the following query on the `MATRIX_DATA` table to retrieve all values for a single matrix:

```
SELECT * FROM MATRIX_DATA WHERE MATRIX_ID = 1
```

And we get the following result:

MATRIX_ID	ROW_NUM	COL_NUM	VALUE
1	2	7	5
1	5	2	-4
1	1	1	7

Before we can construct the matrix fully we need to know its dimensions so we also need to execute the following query:

```
SELECT * FROM MATRIX WHERE MATRIX_ID = 1
```

Let us assume that the result is as follows:

MATRIX_ID	ROW_DIM	COL_DIM
1	6	8

The complete 6x8 matrix can then be depicted as follows:

7	0	0	0	0	0	0	0
0	0	0	0	0	0	5	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	-4	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Deliverables

In this assignment you will build an application that handles sparse matrix storage and retrieval from a database server while providing support for basic matrix operations. You will also have to take the role of the programmer dealing with an inflexible DBA. You will not be able to modify the schema and must assume that the DBA has not implemented any constraints or integrity checking thereby forcing you to **do all error checking in your code**.

You are to write a **java program** that takes two parameters

1. A JDBC connection string
2. An input file name

Your program should connect to the database server as specified in the connection string and execute the commands given in the input file while producing the required output (System.Out). **Your queries should only address (select, insert, update, delete) the `MATRIX` and `MATRIX_DATA` tables and should not modify the database in any other way.** You must refer to these tables and all their attributes in CAPITAL LETTERS in your code/queries.

On every line in the input file is a new command for your program. Your program must produce one line of output (System.Out) for each command as specified below after executing the command. After each command the contents of the database should reflect what the commands have requested:

The commands in the input file (that you have to implement) are described as follows:

1. `GETV <MATRIX_ID> <ROW_NUM> <COLUMN_NUM>`

If the matrix exists and the row/column numbers are within range, print the value. Otherwise print `ERROR`

2. `SETV <MATRIX_ID> <ROW_NUM> <COLUMN_NUM> <VALUE>`

If possible, set the value of the specified matrix's row and column and print `DONE`, otherwise print `ERROR`.

3. `SETM <MATRIX_ID> <ROW_DIM> <COLUMN_DIM>`

This command can do several things. It (a) creates an all zero matrix specified by the matrix_id and of the given dimensions if the matrix_id provided does not already exist and (b) try to modify the dimensions of existing matrices by expanding or contracting them (i.e., padding or removing extra zeros). Creating a new matrix is relatively easy. So is expanding it, i.e., we simply increase the dimensions meaning we add extra rows/columns that are all zero valued. However when contracting it, you have to ensure that all existing values also fit in the new (smaller) sub-matrix.

For example the matrix $\begin{matrix}1 & 2\\ 3 & 4\end{matrix}$ is a 2x2 matrix can be expanded to 3x3 matrix $\begin{matrix}1 & 2 & 0\\ 3 & 4 & 0\\ 0 & 0 & 0\end{matrix}$

But $\begin{matrix}1 & 2 & 3\\ 4 & 5 & 6\\ 7 & 8 & 9\end{matrix}$ which is a 3x3 matrix cannot be contracted to $\begin{matrix}1 & 2\\ 4 & 5\end{matrix}$ without losing information

Another example: $\begin{matrix}1 & 2 & 0\\ 3 & 0 & 0\\ 0 & 1 & 0\end{matrix}$ is 3x3 and the most it can be contracted to is $\begin{matrix}1 & 2\\ 3 & 0\\ 0 & 1\end{matrix}$ which is a 3x2 matrix

To make things simpler, the expansions and contractions can only happen at the rightmost columns and bottommost rows. For example: $\begin{matrix}0 & 1 & 2\\ 0 & 3 & 0\\ 0 & 0 & 1\end{matrix}$ is 3x3 and it CANNOT be contracted to

$\begin{matrix}1 & 2\\ 3 & 0.\\ 0 & 1\end{matrix}$

Note that a matrix can be expanded in one direction while at the same time being contracted in another using this command. Your output should be `DONE` if the command succeeds or `ERROR` otherwise.

4. `DELETE <MATRIX_ID>`
5. `DELETE ALL`

These two commands are similar. Delete a specific matrix or delete all matrices. No error checking required if the matrix (or matries) is/are not existent. Your program should delete the specified matrix's definition and data values or all data from both tables in case of DELETE ALL. Your output must always be `DONE` for both these commands.

6. `ADD <MATRIX_1> <MATRIX_2> <MATRIX_3>`

If possible, add Matrix 2 and Matrix 3, then copy the result to Matrix 1 (remove all traces of the old Matrix 1's definition and data if it already existed), then print `DONE`. Otherwise print `ERROR`

7. `SUB <MATRIX_1> <MATRIX_2> <MATRIX_3>`

If possible, subtract Matrix 3 FROM Matrix 2 (i.e., M2-M3), then copy the result to Matrix 1 (remove all traces of the old Matrix 1's definition and data if it already existed), then print `DONE`. Otherwise print `ERROR`

8. `MULT <MATRIX_1> <MATRIX_2> <MATRIX_3>`

If possible, Perform matrix multiplication with Matrix 2 on left and Matrix 3 on right (M2xM3), then copy the result to Matrix 1 (remove all traces of the old Matrix 1's definition and data if it already existed), then print `DONE`. Otherwise print `ERROR`

9. `TRANSPOSE <MATRIX_1> <MATRIX_2>`

If possible, calculate the transpose of the Matrix 2, then copy the result to Matrix 1 (remove all traces of the old Matrix 1's definition and data if it already existed), then print `DONE`. Otherwise print `ERROR`

10. `SQL <querystring>`

You can assume that <querystring> will be a well formed SQL query and that it will only return 1 string/varchar. Print that one item.

What to submit

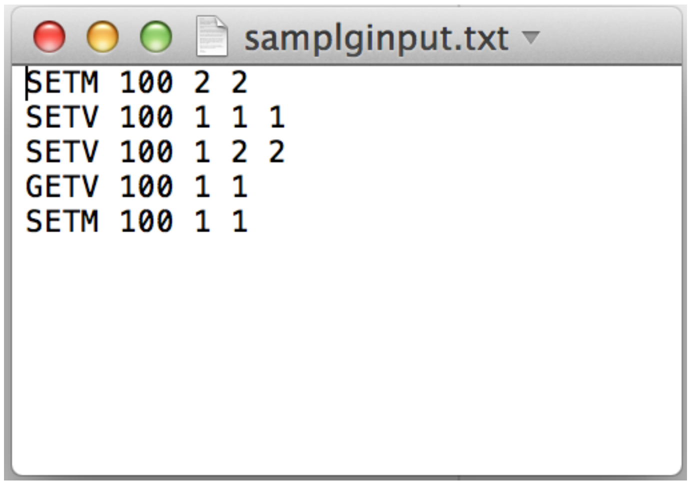
For this assignment you must submit **<u>only one file</u>** "A3.java" using the undergraduate CS assignment submission system. We will test only this file by compiling it into A3.class as follows:

```
javac A3.java
```

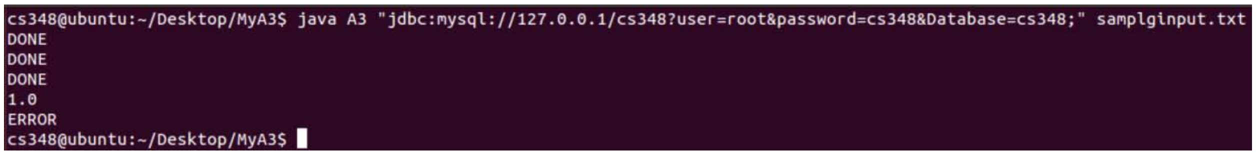
and then executing it by providing it with a connection string and input file name. An example with the MySQL driver accessing a database called cs348 is shown below. Note this example is taken from the environment created for you in Assignment 0. You can assume that the testing environment will be set up to work with the database server mentioned in the connection string and the appropriate JDBC driver will be present.

```
java A3 "jdbc:mysql://127.0.0.1/cs348?user=root&password=cs348&Database=cs348;" input.txt
```

Sample Execution



```
SETM 100 2 2
SETV 100 1 1 1
SETV 100 1 2 2
GETV 100 1 1
SETM 100 1 1
```



```
cs348@ubuntu:~/Desktop/MyA3$ java A3 "jdbc:mysql://127.0.0.1/cs348?user=root&password=cs348&Database=cs348;" samplginput.txt
DONE
DONE
DONE
1.0
ERROR
cs348@ubuntu:~/Desktop/MyA3$
```