CS 34800 - Project 3

Due Date: Tuesday Nov 20, 2018 - 11:59 PM on Blackboard
(10% penalty / late day. After five calendar days, project will not be accepted)

Before you start

- Please, **read and follow all the instructions**. Don't just skim through the handout.
- Grading is automated, so you have to follow the given input/output format **exactly**, with spaces, new lines, and case (capital vs. small letters) taken into consideration.
- Your code should catch exceptions and continue for any input, so that it can still produce some output for partial credit, even if it fails some of the test cases.

Project Description

In this project, you will implement a simple role-based access control scheme for a slightly modified version of the database you used in Projects 1 and 2. The project has three goals:

- Using JDBC to query/update your database from within a Java program
- Building a simple role-based access control scheme
- Building a simple data encryption mechanism to hide data from database users with insufficient access privileges

Preliminaries

Role-Based Access Control (RBAC)

Role-Based Access Control (RBAC) is a popular data protection mechanism, where the operations a user can perform on specific data are determined by the user's data access roles. In this scheme, data access rights are granted to groups having specific roles instead of individual users. Users can have multiple roles and are entitled to all privileges of their assigned roles. In this project, we use a simple RBAC mechanism, where regular (non-admin) users of the database are assigned roles by the administrator, who also assigns specific permissions regarding the use of the database tables to each role. For simplicity, the only operations regular users of the database can perform on the data are INSERT and SELECT. Owner of a data item can restrict access to the item by making it available in plaintext only to users with a specific role, and hiding some fields of the data from other users using encryption. A user can see all fields of a table if she has the privilege for the SELECT operation on that table, but may see encrypted versions of some of the attribute values based on her role permissions for that table.

Data Encryption/Decryption Using Autokey Cipher

Autokey Cipher **encryption** algorithm accepts a plaintext and a key to generate a ciphertext. Using the ciphertext and the key, the Autokey Cipher **decryption** algorithm can recover the original plaintext. To summarize, Autokey Cipher will have the following methods to implement its encryption and decryption algorithms:

- encrypt(String plaintext, String key) → String ciphertext
- decrypt(String ciphertext, String key) → String plaintext

To illustrate how Autokey Cipher works, we can view the alphabet algebraically. The letters A-Z are viewed as numbers from 0-25, and additions are performed modulo 26. For example, let the plaintext be "DATABASE" and the key be "KEY", then the encryption algorithm will work as:

- encrypt(“DATABASE”, “KEY”)
    - Generate string k` by prepending the key to plainText, limited by the length of plainText. For example, "DATABASE" has length of 8, so k` should be set to "KEYDATAB", i.e. the plaintext is appended to "KEY" till we reach length 8
    - Transform the plaintext and k` to numbers, where A = 0, B = 1 ... Z = 25
    - Add each corresponding two numbers in plaintext and k` modulo 26 to get ciphertext code, and then translate back to alphabet to get ciphertext

pt	D	A	T	A	B	A	S	E
	3	0	19	0	1	0	18	4
k`	K	E	Y	D	A	T	A	B
	10	4	24	3	0	19	0	1
ct	13	4	17	3	1	19	18	5
	N	E	R	D	B	T	S	F

- decrypt(“NERDBTSF”, “KEY”)
    - The decryption algorithm works similarly, except that it uses subtraction modulo 26 to reveal the plaintext

For this project, you can assume:

- Non-alphabetic characters in plaintext would be copied to ciphertext as is, including numbers, special characters and white spaces
    - “DATA @ BASE 1” → “NERD @ BTSF 1” with the key “KEY’
- ciphertext would be case-sensitive following the case of plaintext
    - “DataBase” → “NerdBtsf” with the key “KEY”
- Keys contain only alphabetic characters from A to Z, and are case-insensitive
    - “KEY”, “key” and “Key” all function similarly

For more details, see https://en.wikipedia.org/wiki/Autokey_cipher

Connecting to your Oracle database

For this project, you will use your Oracle account to host the database of the application. Your program should connect to this database using a database connection library (JDBC). To get started using JDBC, you can go through the tutorial at http://www.tutorialspoint.com/jdbc/jdbc-quick-guide.htm. This tutorial should provide the sufficient knowledge about JDBC for this project's purposes.

Note that when registering the database driver in your code, you should pass the argument "oracle.jdbc.OracleDriver" to the Class.forName() method, as you will be connecting to an Oracle database. Also, you will need to include the JDBC driver ojdbc8.jar (provided to you) in your classpath at runtime to be able to use the database connectivity function.

You should use the URL "jdbc:oracle:thin:@claros.cs.purdue.edu:1524:strep" with the getConnection method of the DriverManager class when connecting to the database and include your Oracle account username (without @csora) and password as the username and password parameter values.

Database Schema

The database in this project consists of the tables listed below. The scripts to create/drop the database are already provided to you.

Admin tables

These tables are for administrative tasks only and are not accessible by regular users: Users, Roles, UsersRoles, Privileges, and RolesPrivileges.

User tables

There are tables accessible by regular users. The users will need to have the appropriate permissions to insert data into or view data from these tables. Note that each of these tables has the fields EncryptedColumn and OwnerRole, the functions of which will be explained below. **These two columns should not be output when a SELECT command is issued by any user**.

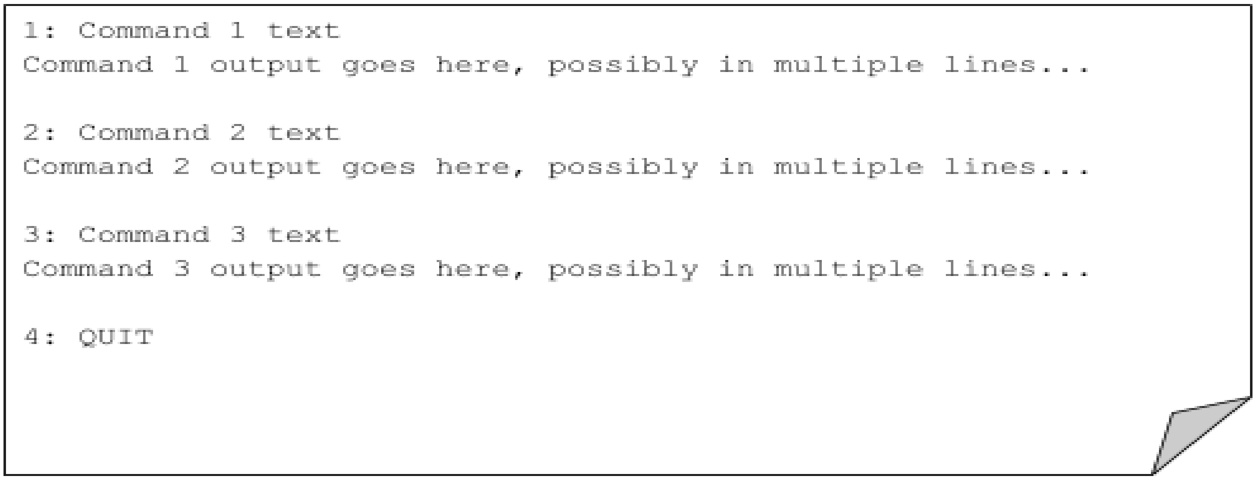
Command Set

Your program should accept input from a text file (with each command terminated with a newline character), process the input and produce as output the responses in another text file. Ultimately, your submission should be executed as follows:

```
java -cp .:ojdbc8.jar Project3 input.txt output.txt
```

Sample input and output files are also provided for you. input.txt is the file containing the commands and output.txt is a file you are going to create. output.txt should include the output for each command separated by newlines, and ends with a newline after the "QUIT" command.

For example, if we have three commands in input.txt before the "QUIT" command, output.txt should look like:



```
1: Command 1 text
Command 1 output goes here, possibly in multiple lines...

2: Command 2 text
Command 2 output goes here, possibly in multiple lines...

3: Command 3 text
Command 3 output goes here, possibly in multiple lines...

4: QUIT
```

Notice that "Command 1 text" is the actual first command text you read from input.txt. Following any other format will incur high penalty in grading.

The program should implement the command set listed below. Notice that the reserved words are in boldface and case-sensitive. Note that the commands will always be input in the correct format, i.e. you do not need to check for syntax errors.

- **`LOGIN`** `username password`
  Upon issuance of this command, you should check that the provided username and password pair matches the one in the Users table in the database. If not, you should print the error message “**Invalid login**”. If a match is found, you should update the current user to the one specified by username and print “**Login successful**”. Please follow the exact output statements; e.g. "Invalid login" with the same casing, as this will be counted in grading. Before you run your program for the first time, you should have a record in your Users table for the admin user (UserId=1, Username=’admin’, Password=’pass’). You should also have a record for the ADMIN role in the Roles table (RoleId=1, RoleName='ADMIN', EncryptionKey='AK'). You should have a record in your UsersRoles table, for which the UserId is 1 and the RoleId is 1 (i.e., user admin has role ADMIN). A script with this data initialization is also provided with this handout. When checking whether the current user is the admin, you can either check that the username is admin or that the user has the ADMIN role in the UsersRoles table (either approach will work here). For simplicity, we will only have one admin user in this project. Note that if the login command is issued before a user quits the program, you should just switch the current user to the user specified in the login command. You can assume that if the login is unsuccessful, the user will keep trying to login until it succeeds. The very first command issued to your program will always be the login command.

- **CREATE ROLE** `roleName encryptionKey`
  If the current user is the admin when this command is issued, you should insert a new record in the Roles table with the values (roleName, encryptionKey) and print “**Role created successfully**” (you should generate the role Id dynamically). You do not need to check whether this role already exists in the database. If the current user is not the admin, then print the error message “**Authorization failure**”.

- **CREATE USER** `username password`
  If the current user is the admin when this command is issued, you should insert a new record in the Users table with the values (username, password) and print “**User created successfully**” (you should generate the user id dynamically). You do not need to check whether this user already exists in the database. If the current user is not the admin, then print the error message “**Authorization failure**”.

- **GRANT ROLE** `username roleName`
  If the current user is the admin when this command is issued, you should insert a new record in the UsersRoles table with ids corresponding to (username, roleName) and print “**Role assigned successfully**”. You do not need to check whether this user exists in the Users table, roleName exists in the Roles table, or this pair exists in the UsersRoles table. If the current user is not the admin, print the error message "**Authorization failure**".

- **GRANT PRIVILEGE** `privName` **TO** `roleName` **ON** `tableName`
  You should populate the Privileges table with two rows (already included in the initialization script):
  - Insert Privilege: Id = 1, Name = INSERT
  - Select Privilege: Id = 2, Name = SELECT

  If the current user is the admin when this command is issued, you should insert a new record in the RolesPrivileges table with values corresponding to the parameters (roleName, privName, tableName) and print "**Privilege granted successfully**". You do not need to check whether these already exist in the database. If the current user is not the admin, print the error message “**Authorization failure**”.

- **REVOKE PRIVILEGE** `privName` **FROM** `roleName` **ON** `tableName`
  If the current user is the admin when this command is issued, you should delete the record in the RolesPrivileges table with values corresponding to the parameters (roleName, tableName, privName) and print "**Privilege revoked successfully**". You do not need to check whether these already exist in the database. If the current user is not the admin, print the error message “**Authorization failure**”.

- **INSERT INTO** tableName **VALUES** (valueList) **ENCRYPT** columnNo ownerRole
  Upon issuance of this command, you should first check if any of the roles of the current user has the "INSERT" privilege on the table tableName. If not, you should print the error message "**Authorization failure**". Otherwise, you should insert the values in valueList (which is a list of comma-separated strings enclosed in single quotes) into the table tableName. While inserting the tuple, you should set the OwnerRole attribute value to the id corresponding to ownerRole and set the value of the EncryptedColumn attribute to columnNo. You do not need to check whether this record already exists in the database. Before insertion, you should check the value of columnNo: If it is greater than 0, then you need to encrypt the corresponding attribute value with the encryption key of the owner role (which is found in the ROLES table). columnNo specifies the order of the column to be encrypted in the table and will never be negative or greater than number of columns in the table. For example, if columnNo is 3 when inserting into the table Companies, the value to be inserted for the column Address should be encrypted. Your output should be "**Row inserted successfully**" if the user is authorized.

- **SELECT** * **FROM** tableName
  Upon issuance of this command, you should first check if any of the roles of the current user has the "SELECT" privilege on the table tableName. If not, you should print the error message "**Authorization failure**". Otherwise, you should print the names of the attributes in this table (in upper case) on a line by itself, each separated by a comma and all records in the table tableName, one record per line, with each attribute value separated by a comma (attribute values in the same order as the attribute names listed on the first line). The value printed for each attribute will depend on the owner of the data. If any of the current user's roles is the owner role of a tuple, all attribute values for that tuple should be printed in plain text (i.e., encrypted attribute values should be decrypted before printing), otherwise attribute values should be printed as they are stored in the database. EncryptedColumn and OwnerRole are NOT printed to output, they are just to decide about encryption/decryption. You can assume that there will be at least one record in the table tableName when this command is issued. Notice that having the ADMIN role should not grant the admin user to see the encrypted fields of a table unless the admin has the role required to see these fields.

- **QUIT**
  Your program should terminate upon issuance of this command (you should ignore any command that appears after the QUIT command in the input file).

Submission Instructions

Please, follow these instructions strictly.
Grading is automated, and your grade would be significantly affected otherwise.

- The project should be implemented in Java and your main file should be named **Project3.java**.

- Before you submit your project, make sure to delete all data from your database tables, and drop all the tables, except for the data in the initialization script (To guarantee a completely clean database before testing your submission).

- If needed, create a README file containing identifying information, and include anything you might want us to know when grading your project.
  For example:

  CS348 -- Project 3
  Name: Name
  Login: author
  Email: author@cs.purdue.edu
  bla bla bla bla bla bla bla

- A sample run.sh script is given. This is used to clean and initialize the database, and compile and run your code. Make sure you edit the file to add your oracle username and password. As shown in the script file:
  Your submission should be compiled using the following command

  ```
  javac -cp .:ojdbc8.jar Project3.java
  ```

  and should be run using the following command

  ```
  java -cp .:ojdbc8.jar Project3 input.txt output.txt
  ```

- If you add other java files, you need to edit the run.sh file to compile these too.

- Please submit on Blackboard a ZIP file named "username_project3.zip" where username is your Purdue career account username (i.e. username@purdue.edu). The ZIP file should contain **ONLY** the following files with no subfolders:
  - **Project3.java** (Your java main file) and any other .java files you might be using
  - The modified **run.sh** file with your username and password
  - [Optional] **README**: your readme file, if needed
  - **No other files whatsoever**