Lab 9
Java Database Connectivity

CSE 4308

DATABASE MANAGEMENT SYSTEMS LAB

OCTOBER 29, 2023

1 Environment Setup

All the files mentioned below have been provided in Google Classroom.

1. Install Java Development Kit (JDK) using `jdk-19_windows-x64_bin.msi`.
2. If you want to use VSCode install the extension pack for java.
3. Create a new Java Project and add `ojdbc14.jar` or `ojdbc6.jar` file as an external JAR file.
   (a) For IntelliJ IDE, in the video, the whole process has been demonstrated.
   (b) For VSCode IDE, the process of adding an external JAR file to a project: from the Explorer Bar, go to `Java Projects` → <projectname> →`Referenced Libraries`. Then click on the plus sign and select the `ojdbc14.jar` or `ojdbc6.jar` and hit the selectjar Libraries button.
   (c) For Eclipse IDE, the process of adding an external JAR file to a project: from the Menu Bar, go to `Project` → `Properties` →. Then on the opened window, click on `Java Build Path` → `Classpath` → `Add External JARs`. This will open a File Explorer where you can navigate to the path where `ojdbc14.jar` or `ojdbc6.jar` is located to select and add it to your project.
4. Now add a new file named `jdbc_practice.java` file (given) and go through the code to get an idea about what is going on.

2 Scenario

Consider the (partial) schema of a Banking Management System shown in Figure 1:

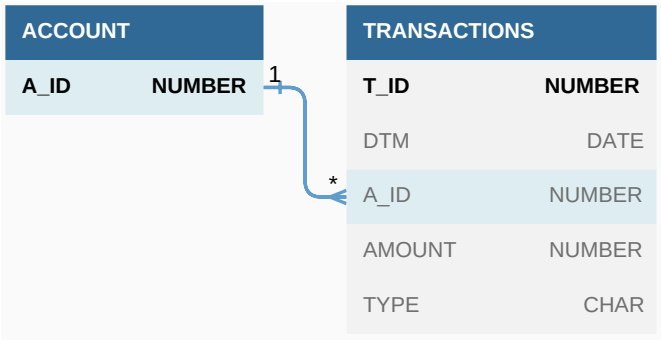

Figure 1: Schema for Lab Task

For simplicity, the `ACCOUNT` table contains only the account ID. The `TRANSACTIONS` table contains the information related to each transaction that occurs in the bank. The table stores the transaction ID, date of the transaction, account involved in the transaction, the amount transacted, and the type of the transaction. The `TYPE` column is set to 0 if the money is credited to the account, i.e., the money is added to the account, and 1 if the money is debited from the account, i.e., the money is subtracted from the account. There are 3 types of accounts in the bank:

1. **Commercially Important Person (CIP):** If the person has a balance of more than $1,000,000$ and all the transactions that they have made totals more than $5,000,000$.
2. **Very Important Person (VIP):** If the person has a balance of more than $500,000$ but less than $900,000$ and all the transactions that they have made totals more than $2,500,000$ but less than $4,500,000$.
3. **Ordinary Person (OP):** If the user has a balance less than $100,000$ and all the transactions they have made totals less than $1,000,000$.

3 Lab Task

Carry out the following instructions:

1. Download and execute `TableGenerator.cpp`. It will ask you to enter your student ID and create a file named `table.sql`. The SQL file contains the required SQL queries to create and populate the database. The values will be random except for the account numbers that start from 1 up to a random number.

2. Make sure that Oracle Database is installed on your PC. Login to your database (either using Command Prompt or SQL Command Line). Enter `"<directory>\table.sql"`. Here, `<directory>` should be replaced by the file path of `table.sql`.

3. Create a new Java project and add ojdbc6.jar (compatible with oracle11g and upper version) or ojdbc14.jar (can try this one with upper versions too) file as an external JAR file.

Now complete the following tasks:

3.1 Task 1 - Connection establish and Fetch data

NOTE: A solution template for task 1 has been added towards the end of this material to help with your coding.

- Write a JAVA code to:
  1. Establish connection to the database.
  2. Fetch required data from the `TRANSACTIONS` table.
  3. Count the total number of accounts, number of CIP, VIP, and Ordinary persons.
  4. Count the number of persons that were uncategorized.
- Add a comment above each variable declaration, explaining what it stores.
- Add comments above each function call (if any), explaining what it does and what parameters are passed to the function.

The output of your program should look like the following [1]:

```
Total Number of Accounts : 147
CIP count : 25
VIP count : 2
Ordinary count : 1
Uncategorized : 119
```

[1] Note that the numbers will not be the same for you. As your table entries will be generated randomly, everyone should have different answers. Some of you may face issues where your output does not generate any count for CIP or VIP. In such cases, check your generated table.sql file and see what the maximum amount of transaction is in your TRANSACTIONS table. If it is something around 140,000, it probably means your compiler cannot generate larger random numbers. If you check the cpp file, you will see that I have set the upper limit for the transaction amount to be generated to 1,000,000; so 140,000 should not be the maximum amount in your table. Anyway, if that's the case, I advise using any online compiler like www.onlinegdb.com to execute the cpp file and then use the generated table.sql file. For reference, check the RAND_MAX value generated by your compiler. It should be around 2147483647. If it's less than that, you will probably have trouble generating CIP and VIP counts.

3.2 Task 2 - Using prepared statements

`PreparedStatements` use "?" as values, specifying that the actual values will be provided later. This has a number of very important advantages. Each time the query is executed (with new values to replace the "?"s), the database system can reuse the previously compiled form of the query and apply the new values as parameters. The following code fragment shows how `PreparedStatements` can be used:

```
PreparedStatement pStmt = conn.prepareStatement(
    " insert into instructor values (? ,? ,? ,?)") ;
pStmt.setString (1 , " 88877 ") ;
pStmt.setString (2 , " Perry ") ;
pStmt.setString (3 , " Finance ") ;
pStmt.setInt (4 , 125000) ;
pStmt.executeUpdate () ;
pStmt.setString (1 , " 88878 ") ;
pStmt.executeUpdate () ;
```

Protection against SQL injection attacks. The `setString()` method automatically inserts any escape characters needed for syntactic correctness, and so, malicious users cannot manipulate SQL code to steal data or damage the database
Now your task is to:

- Use a prepared statement to insert two new records into the `TRANSACTIONS` table of the given schema. The records should have the following values [2]:

```
T_ID: 10001; DTM: February 12, 2022; A_ID: 2; Amount: 5000; Type: 1
T_ID: 10005; DTM: October 15, 2022; A_ID: 4; Amount: 10000; Type: 0
```

3.3 Task 3 - Using metadata features

Recall that when we submit a query using the `executeQuery()` method, the result of the query is contained in a ResultSet object. The interface `ResultSet` has a method, `getMetaData()`, that returns a `ResultSetMetaData` object that contains metadata about the result set.
In turn, `ResultSetMetaData`, has methods to find metadata information, such as the number of columns in the result, the name of a specified column, or the type of a specified column. The following Java code segment uses JDBC to print out the names and types of all columns of a result set:

```
ResultSetMetaData rsmd = rs.getMetaData() ;
for (int i = 1; i <= rsmd.getColumnCount() ; i ++) {
    System.out.println( rsmd.getColumnName(i)) ;
    System.out.println( rsmd.getColumnTypeName(i)) ;
}

// The getColumnCount () method returns the arity ( number of
   attributes ) of the result relation.
// For each attribute, we retrieve its name and data type
   using the methods getColumnName () and getColumnTypeName ()
    respectively.
```

[2] You should use the same prepared statement for inserting both the records. Do not create two separate `PreparedStatements`. In case you get a primary key violation error for these entries, check the T_ID value of the last transaction in your table.sql file, and use T_ID values higher than the one in your last transaction.

Now your task is to:

- Use these metadata features to print the number of columns, names of the columns, and the data types of the columns for the `ACCOUNT` and `TRANSACTIONS` entity in the given schema.

4 Solution Template

```
import java.sql.*;

public class jdbc {
    public static void main(String[] args) {

        String sqlQuery = "SELECT A_ID,AMOUNT,TYPE FROM
  TRANSACTIONS";

        int[] balance = new int[1000];
        int[] total = new int[1000];
        int account;
        int amount;
        String type;

        try {

            // 1) load the driver class
            Class.forName("oracle.jdbc.driver.OracleDriver");

            // 2) create the connection object
            Connection con = DriverManager.getConnection(
                    "jdbc:oracle:thin:@localhost:1521:xe", "
  DBMS", "dbms");

            System.out.println(" Connection to database
  successful ");

            // 3) Create the Statement object
            Statement statement = con.createStatement();

            // 4) Execute the query
            ResultSet result = statement.executeQuery(sqlQuery
  );
            while (result.next()) {
                account = result.getInt(" a_id "); // Could
  also be written as result . getInt (1) ;
                amount = result.getInt(" amount "); // Could
  also be written as result . getInt (2) ;
                type = result.getString(" type "); // Could
  also be written as result . getInt (3) ;
                System.out.println(account + " " + amount + "
  " + type);
                /* Additional Code */
            }
```

```
            /* Additional Code */

            // 5) Close the connection object
            con.close();
            statement.close();
            result.close();
        } catch (SQLException e) {
            System.out.println(" Error while connecting to
  database .Exception code : " + e);
        } catch (ClassNotFoundException e) {
            System.out.println(" Failed to register driver .
  Exception code : " + e);
        }
        System.out.println(" Thank You !");
    }
}
```