Tejal Gajare
A20287489

ITM 411 Mini Project 4: Read Me

1. Abstract

Aim: To develop a *Java SE NetBeans* project that uses a *Java Database Connectivity (JDBC)* connected to a *MySQL* database called *itm411DB* to provide the basic CRUD features

Objective: The objective of this mini project 4 was to help us understand the fundamental JAVA Connectivity to the database in order to perform the basic CRUD operations on the table present in a MySQL database

Basic Understanding:

JDBC Introduction:

The JDBC API is a Java API that can access any kind of tabular data, especially data stored in a Relational Database.

JDBC helps us to write Java applications that manage these three programming activities:

1. Connect to a data source, like a database
2. Send queries and update statements to the database
3. Retrieve and process the results received from the database in answer to the query

Relational Databases:

A database is a means of storing information in such a way that information can be retrieved from it. In simplest terms, a relational database is one that presents information in tables with rows and columns. A table is referred to as a relation in the sense that it is a collection of objects of the same type (rows). Data in a table can be related according to common keys or concepts, and the ability to retrieve related data from a table is the basis for the term relational database.

Establish a Connection:

First, we need to establish a connection with the data source we want to use. A data source can be a DBMS, a legacy file system, or some other source of data with a corresponding JDBC driver. Typically, a JDBC application connects to a target data source using one of two classes:

- DriverManager: This fully implemented class connects an application to a data source, which is specified by a database URL. When this class first attempts to establish a connection, it automatically loads any JDBC 4.0 drivers found within the class path.
- DataSource: This interface is preferred over DriverManager because it allows details about the underlying data source to be transparent to the application. A DataSource object's properties are set so that it represents a particular data source.

Tejal Gajare
A20287489

Establish a Database Connection URL:

A database connection URL is a string that the DBMS JDBC driver uses to connect to a database. It can contain information such as where to search for the database, the name of the database to connect to, and configuration properties. The exact syntax of a database connection URL is specified by the DBMS.

The following is the database connection URL syntax for Java DB:

```
jdbc:derby:[subsubprotocol:][databaseName][;attribute=value]
```

The following is the database connection URL syntax for MySQL Connector/J:

```
jdbc:mysql://[host][,failoverhost...]
             [:port]/[database]
             [?propertyName1][=propertyValue1]
             [&propertyName2][=propertyValue2]...
```

- *host*:*port* is the host name and port number of the computer hosting your database. If not specified, the default values of host and port are 127.0.0.1 and 3306, respectively.
- *database* is the name of the database to connect to. If not specified, a connection is made with no default database.
- *failover* is the name of a standby database (MySQL Connector/J supports failover).
- *propertyName=propertyValue* represents an optional, ampersand-separated list of properties. These attributes enable you to instruct MySQL Connector/J to perform various tasks.

ResultSet:

A ResultSet object is a table of data representing a database result set, which is usually generated by executing a statement that queries the database.

2. System Requirement

OS: Windows 7
JDK: 1.7.0
IDE: NetBeans IDE 7.2.1

3. Architecture or System Flow

3.1 Creation of Java Application:
Launch NetBeans and click on new Project to choose a new Java Application, name it as ‘MiniProject4’ and click on Finish

3.2 Creation of packages:
Along with the default package, miniproject4, create new packages ‘domain’ & ‘Utilities’ at the same level

Tejal Gajare
A20287489

3.3 Creation of classes/interfaces (Java files):

EmployeeRecord - This class is used to save the records fetched from the table Employees

UtilityFunctions - This class encapsulates the various methods in order to perform the tasks of creating, updating, retrieving and deleting of the records from the Employees Table of itm411db database created in MySql. It also has the displayMenu and getUserSelection methods to provide a menu driven program.

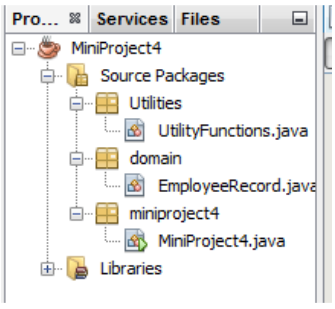
MiniProject4 - This is the main driver class which invokes the methods such as displayMenu and getUserSelection from the UtilityFunctions class. These functions are responsible to display the menu infinitely till the user quits the program.

3.4 Script file:
Created a 'script.bat' file located in the MiniProject4 folder. This file is responsible to convert the MiniProject4.jar file (This file is created after every successful build of the java code) into mp4out.txt file (located at the root level of the main project).

3.5 Javadoc API:
This document depicts in detail in HTML format, all the comments written in the code. It can be created by right clicking on the java project and choosing the 'Generate Javadocs' option. This file folder gets automatically created inside the main project folder's 'dist' folder. It can be moved to a suitable location if needed.

4. Screen Shots

4.1 Structure of the Java Project: The below screen capture shows the basic folder structure for the MiniProject4 JAVA application



4.2 Create record:

```
//This function is required to create a record in the employees table
public static List createRecord() {

    List<EmployeeRecord> recList = null;
    EmployeeRecord ER = new EmployeeRecord();   //create a EmployeeRecord objectin order to save the data retrieved fr
                                                //user or table temporarily
    try {

        recList = new ArrayList();
        Connection conn = getConnection();      //call to getConnection function

        //The following prepared statement is used for manipulating the database (eg: creation of records...)
        PreparedStatement prepareStatement1 = conn.prepareStatement("insert into employees values(?,?,?,?,?,?,?,?)");

        Scanner scan = new Scanner(System.in);
```

Tejal Gajare
A20287489

```
Statement stmt = conn.createStatement();
//Following statement is used to check if the empID exists in the database...
ResultSet rs = stmt.executeQuery("SELECT * FROM employees where employeeNumber=" + ER.getEmp_ID());
rs.last();
//If there is no record with the entered empID then the record is added to the table
if (rs.getRow() == 0) {

    prepareStatement1.setInt(1, ER.getEmp_ID());
    prepareStatement1.setString(2, ER.getLastName());
    prepareStatement1.setString(3, ER.getFirstName());
    prepareStatement1.setString(4, ER.getExtension());
    prepareStatement1.setString(5, ER.getEmail());
    prepareStatement1.setString(6, ER.getOfficeCode());
    prepareStatement1.setInt(7, ER.getReportsTo());
    prepareStatement1.setString(8, ER.getJobTitle());
    prepareStatement1.executeUpdate();
```

4.3 Retrieve by Emp ID

```
//This function is used to retrieve a record by EmpID...
public static boolean retrieveRecord_ID() {
    Connection conn = getConnection();
    List<EmployeeRecord> recList = null;
    EmployeeRecord ER = new EmployeeRecord();

    print("Enter the Employee ID for the record to be retrieved: ");
    Scanner scan = new Scanner(System.in);
    String ID = scan.nextLine();
ResultSet rs = stmt.executeQuery("SELECT * FROM employees where employeeNumber=" + ER.getEmp_ID());
rs.last();
if (rs.getRow() == 0) {
    return false;
}
rs.beforeFirst();
//save the record fetched from the table in the ER object through getter/setters...
while (rs.next()) {
    ER.setEmp_ID(rs.getInt("employeeNumber"));
    ER.setLastName(rs.getString("lastName"));
    ER.setFirstName(rs.getString("firstName"));
    ER.setExtension(rs.getString("extension"));
    ER.setEmail(rs.getString("email"));
    ER.setOfficeCode(rs.getString("OfficeCode"));
    ER.setReportsTo(rs.getInt("reportsTo"));
    ER.setJobTitle(rs.getString("jobTitle"));

    EmployeeRecord rec = new EmployeeRecord(ER.getEmp_ID(), ER.getLastName(), ER.getFirstName(), ER.getExte
            ER.getEmail(), ER.getOfficeCode(), ER.getReportsTo(), ER.getJobTitle());
    recList.add(rec);

    print(" \t \t EMPID \t \t LastName \t FirstName \t \t Extension \t \t Email \t \t \t  OfficeCode \t \t

    print("-------------------------------------------------------------------------------------------------
    for (EmployeeRecord r : recList) {

        print(r.toString());
```

Tejal Gajare
A20287489

4.4 Retrieve All Records:

```
//This function is used to retrieve all records from the employee table...
public static void retrieveRecords() {

    Connection conn = getConnection();
    List<EmployeeRecord> recList = null;
    EmployeeRecord ER = new EmployeeRecord();
    try {

        recList = new ArrayList();
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery("SELECT * FROM employees;");
        while (rs.next()) {
            ER.setEmp_ID(rs.getInt("employeeNumber"));
            ER.setLastName(rs.getString("lastName"));
            ER.setFirstName(rs.getString("firstName"));
            ER.setExtension(rs.getString("extension"));
            ER.setEmail(rs.getString("email"));
            ER.setOfficeCode(rs.getString("OfficeCode"));
            ER.setReportsTo(rs.getInt("reportsTo"));
            ER.setJobTitle(rs.getString("jobTitle"));
```

4.5 Update Record:

```
//This function is used to update a record looked up by the EmpID...
public static boolean updateRecord_ID() {

    List<EmployeeRecord> recList = null;
    EmployeeRecord ER = new EmployeeRecord();

    //Get the EmployeeID from the user...
    print("Enter the Employee ID for the record to be updated : ");
    Scanner scan = new Scanner(System.in);
    String ID = scan.nextLine();

    //To check if the entered EmpID is numeric else pop error message...
    if (ID.matches("[0-9]+")) {
        ER.setEmp_ID(Integer.parseInt(ID));
    } else {

        print("Enter an integer value in the 'Employee-ID' field");
        return false;
    }
try {

    recList = new ArrayList();
    Connection conn = getConnection();
    //The following prepared statement is used to set new values for the entered EmpID record...
    PreparedStatement prepareStatement2 = conn.prepareStatement("update employees set lastName=?,"
            + "firstName=?,extension=?,email=?,officeCode=?,reportsTo=?,jobTitle=? where employeeNumber=" + ER.getEmp_ID());
    Statement stmt = conn.createStatement();
    ResultSet rs = stmt.executeQuery("SELECT * FROM employees where employeeNumber=" + ER.getEmp_ID());

        prepareStatement2.setString(1, ER.getLastName());
        prepareStatement2.setString(2, ER.getFirstName());
        prepareStatement2.setString(3, ER.getExtension());
        prepareStatement2.setString(4, ER.getEmail());
        prepareStatement2.setString(5, ER.getOfficeCode());
        prepareStatement2.setInt(6, ER.getReportsTo());
        prepareStatement2.setString(7, ER.getJobTitle());
        prepareStatement2.executeUpdate();
        EmployeeRecord rec = new EmployeeRecord(ER.getEmp_ID(), ER.getLastName(), ER.getFirstName(), ER.getExtension(),
                ER.getEmail(), ER.getOfficeCode(), ER.getReportsTo(), ER.getJobTitle());
        recList.add(rec);
```

Tejal Gajare
A20287489

4.6 Delete Record:

```
//The following function is used to delete a record looked up by EmpID...
public static boolean deleteRecord_ID() {

    Connection conn = getConnection();

    EmployeeRecord ER = new EmployeeRecord();
    //Get the user's input...
    print("Enter the Employee ID for the record to be deleted: ");
    Scanner scan = new Scanner(System.in);
    String ID = scan.nextLine();
try {

    Statement stmt = conn.createStatement();
    ResultSet rs = stmt.executeQuery("SELECT * FROM employees where employeeNumber=" + ER.getEmp_ID());
    rs.last();
    //Check for the existence of the record in the table...
    if (rs.getRow() != 0) {
        Statement stmt1 = conn.createStatement();
        String sql = "DELETE FROM employees where employeeNumber=" + ER.getEmp_ID();
        stmt.executeUpdate(sql);

    } else {
        return false;
    }
```

4.7 Display Menu:

```
//This function is used to display the menu to the user...
public static void displayMenu() {
    try {
        //The following handling/logger variables are initiated...
        hand = new FileHandler("data/mp4log.txt");
        log = Logger.getLogger("");
        log.addHandler(hand);

        print("Please enter choice from the following options: \n ");
        print("a - Create an Employee Record.\n"
                + "b - Retrieve an Employee Record by ID.\n"
                + "c - Retrieve all Employee Records.\n"
                + "d - Update an Employee Record by ID.\n"
                + "e - Delete an Employee Record by ID.\n"
                + "q - quit the application.\n");

    } catch (IOException | SecurityException ex) {
        Logger.getLogger(UtilityFunctions.class.getName()).log(Level.SEVERE, null, ex);
    }
```

4.8 DB Connectivity:

```
//This function is used to establish a connection with the MySql database...
private static Connection getConnection() {

    Connection conn = null;
    try {

        conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/itm411db", "root", "tejalgajare");
    } catch (SQLException ex) {
        Logger.getLogger(UtilityFunctions.class.getName()).log(Level.SEVERE, null, ex);
    }
    return conn;
}
```

Tejal Gajare
A20287489

4.9 To Verify if the input entered for Emp ID and / ReportsTo fields are numeric only:

```
String ID = scan.nextLine();

if (ID.matches("[0-9]+")) {
    ER.setEmp_ID(Integer.parseInt(ID));
} else {

    print("Enter an integer value in the 'Employee-ID' field");
    return false;
}
```

4.10 To check if the record with the entered empID exists in the database: If yes then cannot create a record with the existing empID. If the entered empID does not exist then the corresponding record cannot be updated/retrieved or deleted.

```
//rs stores the returned record from the table after the query has been executed...
ResultSet rs = stmt.executeQuery("SELECT * FROM employees where employeeNumber=" + ER.getEmp_ID());
rs.last();
if (rs.getRow() == 0) {
    return false;
}
rs.beforeFirst();
//save the record fetched from the table in the ER object through getter/setters...
while (rs.next()) {
```

5. Conclusion

Thus, the code was implemented successfully. Please go through the mp4out.txt in order to view the Output of the code

6. Submission Package

It includes following folders:

1. MiniProject4:
This includes the main JAVA application

- It has a script.bat file which is responsible to create the mp4out.txt file at the project top level.
- It has a data folder which contains the log files from the handler

2. MiniProject4 Docs:
This includes the JavaAPI docs and the README file

7. Important Note

Please ignore the double print statements if you execute this code in IDE. The reason behind this being I have added log.info() along with the System.out.println() statement to display the output in the command prompt and mp4output.txt file respectively.

The output will also contain the system timestamp with the INFO label because of the log.info statement.