```java
package in.org.cris.mrsectt.dbConnection;

import java.sql.*;
import java.util.ArrayList;

import javax.naming.InitialContext;
import javax.sql.DataSource;

public class DBConnection {

   public DBConnection() {
      super();
   }

   //private static DataSource dataSource; // DataSource object that will
be used
   // used

   // for SQL Connection.

   // private Connection connection; // Connection object that will be used
for

   private Connection connection; // Connection object that will be used
for

   // sql operations.

   private Statement statement; // Statement object that will be used to

   // execute all sql statement.

//COMMENTING THE CODE WHICH WAS WRITTEN FOR JNDI SERVICE.
//FROM HERE----->
 protected synchronized static void getDataSource() throws Exception {
      //if (dataSource == null) {
         try {
            InitialContext ctx = new InitialContext();
             //dataSource = (DataSource) ctx.lookup("jndi/wpms")
            //-->// dataSource = (DataSource) ctx.lookup("jndi/mrsectt");
            //dataSource = (DataSource) ctx.lookup("jndi/MRSECTT_local");

            //dataSource = (DataSource) ctx.lookup("jndi/mrsectt");
         } catch (Exception e) {
            e.printStackTrace();
         }
      //}
   }

   public synchronized Connection getDBConnection() throws Exception {
      DataSource datasource = null;
      Connection con = null;
      try {
         String url = "jdbc:oracle:thin:@192.168.31.64:1521:XE"; // table
details
         String username = "SYSTEM"; // MySQL credentials
         String password = "ROOT";

         Class.forName("oracle.jdbc.OracleDriver"); // Driver name
         con = DriverManager.getConnection(url, username, password);
         System.out.println("Connection Established successfully");
```

```
         /*InitialContext ctx = new InitialContext();
         datasource = (DataSource) ctx.lookup("jndi/mrsectt");
         //dataSource = (DataSource) ctx.lookup("jndi/MRSECTT_local");
         //dataSource = (DataSource) ctx.lookup("jndi/mrsectt");
         con = datasource.getConnection();*/

      } catch (Exception e) {
         //System.out.println("exception in before returning --"
               //+ e.toString());
         e.printStackTrace();
      }
      return con;
   }

   public void openConnection() {
      try {
         //getDataSource();
         connection = getConnection();
         connection.setAutoCommit(false);
         statement = connection.createStatement();
      } catch (Exception e) {
         e.printStackTrace();
      }
   }


   public Statement getStatement() {
      Statement stmt = null;
      try {
         stmt = connection.createStatement(
               ResultSet.TYPE_SCROLL_INSENSITIVE,
               ResultSet.CONCUR_READ_ONLY);
      } catch (Exception e) {
         e.printStackTrace();
      }
      return stmt;
   }

   /**
    * This method closes the connection and statement objects. It is the
    * responsibility of the client to call this method at all times after
data
    * access.
    */
   public int closeConnection() {

      try {

         if (statement != null) {
            statement.close();
         }
      } catch (SQLException e) {
         return -1;
      } finally {
         if (connection != null) {
            try {
               connection.commit();
               connection.close();
               connection = null;
            } catch (SQLException e) {
```

```
                e.printStackTrace();
             }finally {
                try {
                   if (connection != null) {
                      connection.close();
                   }
                } catch (SQLException e) {
                   e.printStackTrace();
                }
                connection = null;
             }
          }
       }
       return 0;
    }

    /**
     * This method implements the delete logic. All subclasses must pass in
the
     * sql query for delete to this method and this method will perform the
     * necessary deletes.This method returns an integer which is either 0
for
     * successful delete or -1 for failures.
     */
    public int delete(String deleteSql) throws SQLException {
       return statement.executeUpdate(deleteSql);
    }

    /**
     * This method implements the insert logic. All subclasses must pass in
the
     * sql query for insert to this method and this method will perform the
     * necessary inserts. This method returns an integer which is either 0
for
     * successful insert or -1 for failures.
     */
    public int insert(String insertSql) throws SQLException {
       return statement.executeUpdate(insertSql);
    }

    /**
     * This method implements the update logic. All subclasses must pass in
the
     * sql query for update to this method and this method will perform the
     * necessary updates. This method returns an integer which is either 0
for
     * successful update or -1 for failures.
     */
    public int update(String updateSql) throws SQLException {
       return statement.executeUpdate(updateSql);
    }

    /**
     * This method implements the select logic. All subclasses must pass in
the
     * sql query for select to this method and this method will perform the
     * necessary selects. This method returns a ResultSet on sucessful
select.
     */
    public ResultSet select(String selectSql) throws SQLException {
       return statement.executeQuery(selectSql);
```

```
   }

   public ResultSet select(Statement stmt, String selectSql)
         throws SQLException {
      return stmt.executeQuery(selectSql);

   }

   public ResultSetMetaData getMetaData(String selectSql) throws
SQLException {

      return statement.executeQuery(selectSql).getMetaData();

   }

   /**
    * This method commits the changes to the database. It is the
responsibility
    * of the client to call this method at all times after data
manipulation.
    */
   public int commit() {
      try {
         connection.commit();
         return 0;
      } catch (SQLException sqle) {
         sqle.printStackTrace();
         return -1;
      }
   }

   /**
    * This method rolls back the changes to the database.
    */
   public int rollback() {
      try {
         connection.rollback();
         return 0;
      } catch (SQLException sqle) {
         sqle.printStackTrace();
         return -1;
      }
   }

   public PreparedStatement setPrepStatement(String strSQL) {
      PreparedStatement pstm = null;
      try {
         pstm = connection.prepareStatement(strSQL);
      } catch (Exception e) {

      }
      return pstm;
   }

   public Connection getConnection() throws ClassNotFoundException,
SQLException {
      String url = "jdbc:oracle:thin:@localhost:1521:xe"; // table details
      String username = "MRSECTT"; // MySQL credentials
      String password = "root";
```

```
      Class.forName("oracle.jdbc.OracleDriver"); // Driver name
      Connection name = DriverManager.getConnection(url, username,
password);
      System.out.println("Connection Established successfully");

      return name;
   }

   public void setConnection(Connection connection) {
      this.connection = connection;
   }

   /**
    * @author Rashmi(MIND)
    * @date 07 Jan 2010
    * @function executeBatchUpdate : for batch update
    * @parameter ArrayList
    * @return Returns integer update count.
    */
   public String executeBatchUpdate(ArrayList<String> arList) throws
SQLException {
      String updateCounts = "0";
      Statement stmt = null;
      try {
         stmt = connection.createStatement(
               ResultSet.TYPE_SCROLL_INSENSITIVE,
               ResultSet.CONCUR_UPDATABLE);
         connection.setAutoCommit(false);
         for (int i = 0; i < arList.size(); i++) {
            if (!(arList.get(i).equals(""))) {
               stmt.addBatch((String) arList.get(i));
            }
         }
         stmt.executeBatch();
         connection.commit();
         updateCounts = "1";
      } catch (SQLException e) {
         updateCounts = e.getMessage();
         e.printStackTrace();
         connection.rollback();
      } catch (Exception e) {
         updateCounts = e.getMessage();
         connection.rollback();
      } finally {
         connection.setAutoCommit(true);
         if (stmt != null) {
            stmt.close();
            stmt = null;
         }
      }
      return updateCounts;
   }
}
```