```java
package Conexao;

/**
 *
 * @author Cynthia Lopes
 * @author Edwar Saliba Júnior
 */
import java.io.FileNotFoundException;
import java.io.IOException;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.io.BufferedReader;
import java.io.FileReader;
import java.util.ArrayList;
import java.util.logging.Level;
import java.util.logging.Logger;

public final class Database {

    private Access informationDB;
    private Connection connectionDB;
    private Statement queryDB;
    private boolean enableMessages;
    private final String c_DATABASE_FILE = "AjaxEx08.txt";

    public Database() throws FileNotFoundException, ClassNotFoundException {

        FileReader file = new FileReader(c_DATABASE_FILE);
        BufferedReader reader = new BufferedReader(file);

        String dataBaseName;
        try {
            dataBaseName = reader.readLine();
            String password = reader.readLine();
            String host = reader.readLine();

            reader.close();
            file.close();

            this.informationDB = new Access(dataBaseName, password, host);
            this.connectionToDB();
            this.enableMessages = false;
        } catch (IOException ex) {
            Logger.getLogger(Database.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public Database(String dataBaseName, String password, String host) {
        try {
            this.informationDB = new Access(dataBaseName, password, host);
            this.connectionToDB();
            this.enableMessages = false;
        } catch (Exception ex) {
            Logger.getLogger(Database.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public Database(Access info) {
        this.informationDB = info;
        this.enableMessages = false;
    }

    public void connectionToDB() throws ClassNotFoundException {
        try {
            this.startDriver();

            connectionDB = DriverManager.getConnection(
                    this.informationDB.getURL(),
                    this.informationDB.getUsuario(),
                    this.informationDB.getSenha());

            if (this.enableMessages) {
                System.out.println("Connection with database '" +
                        this.informationDB.getNomeBD() + "' sucess completed.");
```

```
            }

            this.queryDB = this.connectionDB.createStatement(
                    ResultSet.TYPE_SCROLL_INSENSITIVE,
                    ResultSet.CONCUR_READ_ONLY);
        } catch (SQLException e) {
            System.out.println(e.toString());
        }
    }

    public void startDriver() throws ClassNotFoundException {
        Class.forName("org.postgresql.Driver");
    }

    public void insertValues(String tableName, String fieldsNames[],
            String fieldsValues[]) throws SQLException {
        String query = "INSERT INTO \"" + tableName + "\" (";
        query += returnFieldsNames(fieldsNames) + ")";
        query += " VALUES(";
        query += returnValues(fieldsValues, true) + ")";

        if (this.enableMessages) {
            System.out.println(query);
        }

        this.queryDB.execute(query);
        this.connectionDB.commit();
    }

    public void deleteValues(String tableName, String condition) throws SQLException {
        String query = "DELETE FROM \"" + tableName + "\"";

        if (!condition.equals("")) {
            query += " WHERE " + condition;
        }

        if (this.enableMessages) {
            System.out.println(query);
        }

        this.queryDB.execute(query);
        this.connectionDB.commit();
    }

    public void updateValues(String tableName, String fields[], String values[],
            String condition) throws SQLException {
        String query = "UPDATE \"" + tableName + "\" SET ";
        query += this.returnSetValues(fields, values);

        if (!condition.equals("")) {
            query += " WHERE " + condition;
        }

        if (this.enableMessages) {
            System.out.println(query);
        }

        this.queryDB.execute(query);
        this.connectionDB.commit();
    }

    public boolean existRow(String table, String condition) throws SQLException {
        boolean foundRow;

        String query = "SELECT 1 ";
        query += " FROM \"" + table + "\"";

        if (!condition.equals("")) {
            query += " WHERE " + condition;
        }

        ResultSet resultSet = this.queryDB.executeQuery(query);
        foundRow = resultSet.first();

        if (this.enableMessages) {
            System.out.println(query);
        }
```

```
        return (foundRow);
    }

    public void printSelection(String table, String fields[], String condition)
            throws SQLException {
        String query = "SELECT ";
        query += returnFieldsNames(fields);
        query += " FROM \"" + table + "\"";

        if (!condition.equals("")) {
            query += " WHERE " + condition;
        }

        ResultSet resultSet = this.queryDB.executeQuery(query);

        if (this.enableMessages) {
            System.out.println(query);
        }

        while (resultSet.next()) {
            String print = "";

            for (int i = 0; i < fields.length; i++) {
                print += "|" + resultSet.getString(fields[i]) + "|\t";
            }

            System.out.println(print);
        }
    }

    public ArrayList selection(String table, boolean putQuotationMarksOnTheFields,
            String fields[], String condition) {
        ArrayList resultsList = new ArrayList();

        try {
            String query = "SELECT ";
            if (putQuotationMarksOnTheFields) {
                query += returnFieldsNames(fields);
            } else {
                query += returnValues(fields, putQuotationMarksOnTheFields);
            }
            query += " FROM " + table;

            if (!condition.equals("")) {
                query += " WHERE " + condition;
            }

            ResultSet resultSet = this.queryDB.executeQuery(query);

            if (this.enableMessages) {
                System.out.println(query);
            }

            resultSet.beforeFirst();
            while (resultSet.next()) {
                String[] row = new String[resultSet.getMetaData().getColumnCount()];
                for (int i = 0; i < resultSet.getMetaData().getColumnCount(); i++) {
                    row[i] = resultSet.getString(i + 1);
                }
                resultsList.add(row);
            }
        } catch (SQLException ex) {
            System.out.println("Exceção SQL: " + ex);
        }
        return resultsList;
    }

    public ResultSet selection(String table, String fields[], boolean putQuotationMarksOnTheFields,
            String condition, boolean pointerToFirstRecord) throws SQLException {
        String query = "SELECT ";
        if (putQuotationMarksOnTheFields) {
            query += returnFieldsNames(fields);
        } else {
            query += returnValues(fields, putQuotationMarksOnTheFields);
        }
        query += " FROM \"" + table + "\"";
```

```
        if (!condition.equals("")) {
            query += " WHERE " + condition;
        }

        ResultSet resultSet = this.queryDB.executeQuery(query);
        if(pointerToFirstRecord){
            resultSet.next();
        }

        return (resultSet);
    }

    public ResultSet selection(String query) throws SQLException {
        ResultSet resultSet = this.queryDB.executeQuery(query);
        resultSet.next();

        return (resultSet);
    }

    public void printJoinSelection(String tables[], String fields[], String condition)
            throws SQLException {
        String query = "SELECT ";
        query += returnFieldsNames(fields);
        query += " FROM " + returnFieldsNames(tables);

        if (!condition.equals("")) {
            query += " WHERE " + condition;
        }

        ResultSet resultSet = this.queryDB.executeQuery(query);

        if (this.enableMessages) {
            System.out.println(query);
        }

        for (int i = 0; i < fields.length; i++) {
            int dotPosition = fields[i].indexOf('.');
            fields[i] = fields[i].substring(dotPosition + 1, fields[i].length());
        }

        while (resultSet.next()) {
            String print = "";

            for (int i = 0; i < fields.length; i++) {
                print += "|" + resultSet.getString(fields[i]) + "|\t";
            }

            System.out.println(print);
        }
    }

    public String returnValues(String values[], boolean putQuotationMarks) {
        String vals = "";

        if (putQuotationMarks) {
            for (int i = 0; i < values.length - 1; i++) {
                vals += "\'" + values[i] + "\', ";
            }

            vals += "\'" + values[values.length - 1] + "\'";
        } else {
            for (int i = 0; i < values.length - 1; i++) {
                vals += values[i] + ", ";
            }

            vals += values[values.length - 1];
        }

        return vals;
    }

    public String returnFieldsNames(String values[]) {
        String vals = "\"";

        for (int i = 0; i < values.length - 1; i++) {
            vals += values[i] + "\", \"";
```

```
        }

        vals += values[values.length - 1] + "\"";

        return vals;
    }

    public String returnSetValues(String fields[], String values[]) {

        String vals = "";

        for (int i = 0; i < values.length - 1; i++) {
            vals += "\"" + fields[i] + "\" = "
                    + (values[i].equals("") ? "\'\'" : "\'" + values[i] + "\'") + ", ";
        }

        vals += "\"" + fields[fields.length - 1] + "\" = "
                + (values[values.length - 1].equals("") ? "\'\'" : "\'"
                + values[values.length - 1] + "\'");

        return vals;
    }

    /**
     * Converte um vetor de inteiros para um formato aceitável por um campo do
     * tipo "array" do PostgreSQL.
     *
     * @param Vetor de Inteiros
     * @return "'{ val1, val2, ... }'"
     */
    public String convertToStringArray(int v[]) {

        String vetor = "";

        vetor += "{";

        for (int i = 0; i < v.length; i++) {
            if ((i < (v.length - 1)) && (!((i > 0) && (String.valueOf(v[i]).equals("0"))))) {
                vetor += String.valueOf(v[i]) + ",";
            } else {
                vetor += String.valueOf(v[i]);
                break;
            }
        }
        vetor += "}";

        return (vetor);
    }

    /**
     * Converte um campo do tipo "array" do PostgreSQL (String) num vetor de
     * inteiros.
     *
     * @param "String" - Ex.: {0,80,17,71,13,0}
     * @return "int []" - Ex.: [0,80,17,71,13,0]
     */
    public int[] convertToIntArray(String v) {
        v = v.trim();
        int vetor[] = new int[v.split(",", -1).length];

        int j = 0;
        String n = "";
        for (int i = 0; i < v.length(); i++) {
            if ((v.charAt(i) != '{') && (v.charAt(i) != '}')) {
                if ((v.charAt(i) != ',') && (v.charAt(i) != ' ')) {
                    n += v.substring(i, i + 1);
                } else {
                    vetor[j++] = Integer.parseInt(n);
                    n = "";
                }
            }
        }

        return (vetor);
    }

    public void closeDBConnection() throws SQLException {
```

```
        this.connectionDB.close();
    }

    public boolean isEnableMessages() {
        return enableMessages;
    }

    public void setEnableMessages(boolean enableMessages) {
        this.enableMessages = enableMessages;
    }

    @Override
    public void finalize() throws SQLException, Throwable {
        super.finalize();
        closeDBConnection();
    }

    private Object getFieldValue(String tableName, String fieldName,
            String condition) throws SQLException {

        Object value = null;
        String fields[] = {fieldName};

        ResultSet rs = selection(tableName, fields, true, condition, true);

        if (rs.first()) {
            value = rs.getObject(fieldName);
        }

        return (value);
    }

    public String getStringFieldValue(String tables, String fields,
            String condition) throws SQLException {
        String value = "";
        Object val;

        val = getFieldValue(tables, fields, condition);
        if (val != null) {
            value = val.toString();
        }

        return (value);
    }

    public int getIntFieldValue(String tables, String fields, String condition)
            throws SQLException {
        int value = 0;
        Object val;

        val = getFieldValue(tables, fields, condition);
        if (val != null) {
            value = Integer.valueOf(val.toString());
        }

        return (value);
    }

    public float getFloatFieldValue(String tables, String fields, String condition)
            throws SQLException {
        float value = 0;
        Object val;

        val = getFieldValue(tables, fields, condition);
        if (val != null) {
            value = Float.valueOf(val.toString());
        }

        return (value);
    }

    public boolean getBooleanFieldValue(String tables, String fields,
            String condition) throws SQLException {
        boolean value = false;
        Object val;

        val = getFieldValue(tables, fields, condition);
```

```
        if (val != null) {
            value = Boolean.valueOf(val.toString());
        }

        return (value);
    }

    public int getNumberOfRowsInTable(String table, boolean putQuotation)
            throws SQLException {
        ResultSet rs;

        if (putQuotation) {
            rs = selection("SELECT count(*) "
                    + "  FROM \"" + table + "\"");
        } else {
            rs = selection("SELECT count(*) "
                    + "  FROM " + table);
        }
        int value = rs.getInt(1);

        return value;
    }
}
```