Generic DAO

```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this
license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package dal;

import java.lang.reflect.Field;
import java.lang.reflect.InvocationTargetException;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
/**
 *
 * @author ADMIN
 */
public abstract class GenericDAO<T> extends DBContext {

    protected PreparedStatement statement;
    protected ResultSet resultSet;
    protected Map<String, Object> parameterMap;
    // Các constant đại diện cho giá trị true và false trong việc sử dụng OR và AND
    public static final boolean CONDITION_AND = true;
    public static final boolean CONDITION_OR = false;

    /**
     * Hàm này sử dụng để get dữ liệu từ database lên dựa trên tên bảng mà bạn
     * mong muốn.Hàm sẽ mặc định trả về một List có thể có giá trị hoặc List
     * rỗng
     *
     * @param clazz: tên bảng bạn muốn get dữ liệu về
     * @return list
     */
    protected List<T> queryGenericDAO(Class<T> clazz) {
        List<T> result = new ArrayList<>();
        try {
            // Lấy kết nối
            connection = getConnection();

            // Tạo câu lệnh SELECT
            StringBuilder sqlBuilder = new StringBuilder();
            sqlBuilder.append("SELECT * FROM ").append(clazz.getSimpleName());

            // Chuẩn bị câu lệnh
            statement = connection.prepareStatement(sqlBuilder.toString());
            // Thực thi truy vấn
            resultSet = statement.executeQuery();

            // Khai báo danh sách kết quả
```

```
            // Duyệt result set
            while (resultSet.next()) {
                // Gọi hàm mapRow để map đối tượng
                T obj = mapRow(resultSet, clazz);

                // Thêm vào danh sách kết quả
                result.add(obj);
            }

            return result;
        } catch (IllegalAccessException
                | IllegalArgumentException
                | InstantiationException
                | NoSuchMethodException
                | InvocationTargetException
                | SQLException e) {
            System.err.println("4USER: Bắn Exception ở hàm query: " + e.getMessage());
        } finally {
            try {
                // Đóng kết nối và các tài nguyên
                if (resultSet != null) {

                }
                if (statement != null) {
                    statement.close();
                }
                if (connection != null) {
                    connection.close();
                }
            } catch (Exception e) {
                System.err.println("4USER: Bắn Exception ở hàm query: " + e.getMessage());
            }
        }
        return result;
    }

    /**
     * Hàm này sử dụng để get dữ liệu từ database lên dựa trên tên bảng mà bạn
     * mong muốn Condition (optional) là ám chỉ những giá trị như and hoặc
     * or.Hãy sử dụng những biến sẵn có CONDITION_OR, CONDITION_AND ví dụ:
     * GenericDAO.CONDITION_OR hoặc GenericDAO.CONDITION_AND.Hàm sẽ mặc định trả
     * về một List có thể có giá trị hoặc List rỗng
     *
     * @param clazz: tên bảng bạn muốn get dữ liệu về
     * @param sql: câu lệnh SQL
     * @param parameterHashmap: hashmap chứa các parameter
     * @return list
     */
    protected List<T> queryGenericDAO(Class<T> clazz, String sql, Map<String, Object> paramet
erHashmap) {

        List<T> result = new ArrayList<>();
        try {
            // Lấy kết nối
            connection = getConnection();

            //List parameter
            List<Object> parameters = new ArrayList<>();
```

```
        // Thêm điều kiện
        if (parameterHashmap != null && !parameterHashmap.isEmpty()) {
            // code thêm điều kiện
            for (Map.Entry<String, Object> entry : parameterHashmap.entrySet()) {
                Object conditionValue = entry.getValue();

                parameters.add(conditionValue);
            }
            // Xóa phần AND hoặc OR cuối cùng khỏi câu truy vấn
        }

        // Chuẩn bị câu lệnh
        statement = connection.prepareStatement(sql);

        // Gán giá trị cho các tham số của câu truy vấn
        int index = 1;
        for (Object value : parameters) {
            statement.setObject(index, value);
            index++;
        }

        // Thực thi truy vấn
        resultSet = statement.executeQuery();

        // Khai báo danh sách kết quả
        // Duyệt result set
        while (resultSet.next()) {
            // Gọi hàm mapRow để map đối tượng
            T obj = mapRow(resultSet, clazz);

            // Thêm vào danh sách kết quả
            result.add(obj);
        }

        return result;
    } catch (IllegalAccessException
            | IllegalArgumentException
            | InstantiationException
            | NoSuchMethodException
            | InvocationTargetException
            | SQLException e) {
        System.err.println("4USER: Bắn Exception ở hàm query: " + e.getMessage());
    } finally {
        try {
            // Đóng kết nối và các tài nguyên
            if (resultSet != null) {

            }
            if (statement != null) {
                statement.close();
            }
            if (connection != null) {
                connection.close();
            }
        } catch (SQLException e) {
            System.err.println("4USER: Bắn Exception ở hàm query: " + e.getMessage());
        }
```

```
        }
        return result;
    }

    private static <T> T mapRow(ResultSet rs, Class<T> clazz) throws
            SQLException,
            NoSuchMethodException,
            InstantiationException,
            IllegalArgumentException,
            IllegalAccessException,
            InvocationTargetException {

        // Khởi tạo đối tượng
        T obj = clazz.getDeclaredConstructor().newInstance();

        // Lấy danh sách các field của lớp
        Field[] fields = clazz.getDeclaredFields();

        // Duyệt qua từng field
        for (Field field : fields) {

            // Set giá trị cho field
            Object value = getFieldValue(rs, field);
            field.setAccessible(true);
            field.set(obj, value);
        }

        return obj;
    }

    /**
     * Hàm lấy giá trị cho field từ result set
     *
     * @param rs
     * @param field
     * @return
     * @throws SQLException
     */
    private static Object getFieldValue(ResultSet rs, Field field) throws SQLException {

        Class<?> fieldType = field.getType();
        String fieldName = field.getName();

        // Kiểm tra kiểu dữ liệu và convert sang đúng kiểu
        if (fieldType == String.class) {
            return rs.getString(fieldName);
        } else if (fieldType == int.class || fieldType == Integer.class) {
            return rs.getInt(fieldName);
        } else if (fieldType == long.class || fieldType == Long.class) {
            return rs.getLong(fieldName);
        } else if (fieldType == double.class || fieldType == Double.class) {
            return rs.getDouble(fieldName);
        } else if (fieldType == boolean.class || fieldType == Boolean.class) {
            return rs.getBoolean(fieldName);
        } else if (fieldType == float.class || fieldType == Float.class) {
            return rs.getFloat(fieldName);
        } else {
            return rs.getObject(fieldName);
```

```
        }
    }

    /**
     * Hàm này sử dụng để update thông tin của một đối tượng trong Database.Hãy
     * nhớ rằng hàm này không update ID vì mặc định các bảng sẽ để ID tự động
     * tăng
     *
     * @param sql
     * @param parameterMap: hashmap chứa các parameter
     * @return true: update thành công | false: update thất bại
     */
    protected boolean updateGenericDAO(String sql, Map<String, Object> parameterMap) {

        List<Object> parameters = new ArrayList<>();

        for (Map.Entry<String, Object> entry : parameterMap.entrySet()) {
            Object conditionValue = entry.getValue();

            parameters.add(conditionValue);
        }

        try {
            connection = getConnection();
            connection.setAutoCommit(false);
            statement = connection.prepareStatement(sql);

            int index = 1;
            for (Object value : parameters) {
                statement.setObject(index, value);
                index++;
            }
            statement.executeUpdate();
            connection.commit();
            return true;
        } catch (SQLException e) {
            try {
                connection.rollback();
            } catch (SQLException ex) {
                System.err.println("4USER: Bắn Exception ở hàm update: " + ex.getMessage());
            }
            return false;
        } finally {
            try {
                if (connection != null) {
                    connection.close();
                }
                if (statement != null) {
                    statement.close();
                }
            } catch (SQLException e) {
                System.err.println("4USER: Bắn Exception ở hàm update: " + e.getMessage());
            }
        }
    }

    /**
     * Hàm này sử dụng để update thông tin của một đối tượng trong Database.Hãy
```

```
     * nhớ rằng hàm này không update ID vì mặc định các bảng sẽ để ID tự động
     * tăng
     *
     * @param sql
     * @param parameterMap: hashmap chứa các parameter
     * @return true: delete thành công | false: delete thất bại
     */
    protected boolean deleteGenericDAO(String sql, Map<String, Object> parameterMap) {
        List<Object> parameters = new ArrayList<>();

        for (Map.Entry<String, Object> entry : parameterMap.entrySet()) {
            Object conditionValue = entry.getValue();
            parameters.add(conditionValue);
        }

        try {
            connection = getConnection();
            connection.setAutoCommit(false);
            statement = connection.prepareStatement(sql);

            int index = 1;
            for (Object value : parameters) {
                statement.setObject(index, value);
                index++;
            }
            statement.executeUpdate();
            connection.commit();
            return true;
        } catch (SQLException e) {
            try {
                connection.rollback();
            } catch (SQLException ex) {
                System.err.println("4USER: Bắn Exception ở hàm delete: " + ex.getMessage());
            }
            return false;
        } finally {
            try {
                if (connection != null) {
                    connection.close();
                }
                if (statement != null) {
                    statement.close();
                }
            } catch (SQLException e) {
                System.err.println("4USER: Bắn Exception ở hàm update: " + e.getMessage());
            }
        }
    }

    /**
     * Hàm này sử dụng để insert một dữ liệu của một đối tượng vào một bảng
     * trong database
     *
     * @param object: đối tượng chứa các thông tin muốn insert
     * @return 0: insert thất bại: || !0 : insert thành công
     */
    protected int insertGenericDAO(T object) {
        Class<?> clazz = object.getClass();
```

```
        Field[] fields = clazz.getDeclaredFields();

        StringBuilder sqlBuilder = new StringBuilder();
        sqlBuilder.append("INSERT INTO ").append(clazz.getSimpleName()).append(" (");

        List<Object> parameters = new ArrayList<>();

        // Xây dựng danh sách các trường và giá trị tham số của câu truy vấn
        for (Field field : fields) {
            field.setAccessible(true);
            String fieldName = field.getName();
            Object fieldValue;
            try {
                fieldValue = field.get(object);
            } catch (IllegalAccessException e) {
                fieldValue = null;
            }

            if (fieldValue != null && !fieldName.equalsIgnoreCase("id")) {
                sqlBuilder.append(fieldName).append(", ");
                parameters.add(fieldValue);
            }
        }

        // Xóa dấu phẩy cuối cùng
        if (sqlBuilder.charAt(sqlBuilder.length() - 2) == ',') {
            sqlBuilder.delete(sqlBuilder.length() - 2, sqlBuilder.length());
        }

        sqlBuilder.append(") VALUES (");
        for (int i = 0; i < parameters.size(); i++) {
            sqlBuilder.append("?, ");
        }

        // Xóa dấu phẩy cuối cùng
        if (sqlBuilder.charAt(sqlBuilder.length() - 2) == ',') {
            sqlBuilder.delete(sqlBuilder.length() - 2, sqlBuilder.length());
        }

        sqlBuilder.append(")");
        connection = getConnection();
        int id = 0;
        try {
            // Bắt đầu giao dịch và chuẩn bị câu truy vấn
            connection.setAutoCommit(false);
            statement = connection.prepareStatement(sqlBuilder.toString(), Statement.RETURN_G
ENERATED_KEYS);

            int index = 1;
            for (Object value : parameters) {
                statement.setObject(index, value);
                index++;
            }

            // Thực thi câu truy vấn
            statement.executeUpdate();

            // Lấy khóa chính (ID) được tạo tự động
```

```
            resultSet = statement.getGeneratedKeys();
            if (resultSet.next()) {
                id = resultSet.getInt(1);
            }
            System.err.println("insertGenericDAO: " + sqlBuilder.toString());
            // Xác nhận giao dịch thành công
            connection.commit();
        } catch (SQLException e) {
            try {
                System.err.println("4USER: Bắn Exception ở hàm insert: " + e.getMessage());
                // Hoàn tác giao dịch nếu xảy ra lỗi
                connection.rollback();
            } catch (SQLException ex) {
                System.err.println("4USER: Bắn Exception ở hàm insert: " + ex.getMessage());
            }
        } finally {
            // Đảm bảo đóng kết nối và tài nguyên
            try {
                if (connection != null) {
                    connection.close();
                }
                if (statement != null) {
                    statement.close();
                }
                if (resultSet != null) {
                    resultSet.close();
                }
            } catch (SQLException e) {
                System.err.println("4USER: Bắn Exception ở hàm insert: " + e.getMessage());
            }
        }
        // Trả về ID được tạo tự động (nếu có)
        return id;
    }

    protected int insertGenericDAO(String sql, Map<String, Object> parameterMap) {
        List<Object> parameters = new ArrayList<>();

        for (Map.Entry<String, Object> entry : parameterMap.entrySet()) {
            Object conditionValue = entry.getValue();

            parameters.add(conditionValue);
        }

        connection = getConnection();
        int id = 0;
        try {
            // Bắt đầu giao dịch và chuẩn bị câu truy vấn
            connection.setAutoCommit(false);
            statement = connection.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS);

            int index = 1;
            for (Object value : parameters) {
                statement.setObject(index, value);
                index++;
            }

            // Thực thi câu truy vấn
```

```
            statement.executeUpdate();

            // Lấy khóa chính (ID) được tạo tự động
            resultSet = statement.getGeneratedKeys();
            if (resultSet.next()) {
                id = resultSet.getInt(1);
            }
            // Xác nhận giao dịch thành công
            connection.commit();
        } catch (SQLException e) {
            try {
                System.err.println("4USER: Bắn Exception ở hàm insert: " + e.getMessage());
                // Hoàn tác giao dịch nếu xảy ra lỗi
                connection.rollback();
            } catch (SQLException ex) {
                System.err.println("4USER: Bắn Exception ở hàm insert: " + ex.getMessage());
            }
        } finally {
            // Đảm bảo đóng kết nối và tài nguyên
            try {
                if (connection != null) {
                    connection.close();
                }
                if (statement != null) {
                    statement.close();
                }
                if (resultSet != null) {
                    resultSet.close();
                }
            } catch (SQLException e) {
                System.err.println("4USER: Bắn Exception ở hàm insert: " + e.getMessage());
            }
        }
        // Trả về ID được tạo tự động (nếu có)
        return id;
    }

    /**
     * Tìm số lượng record của 1 bảng nào đó
     *
     * @param clazz: bảng muốn tìm
     * @return số lượng record
     */
    protected int findTotalRecordGenericDAO(Class<T> clazz) {
        int total = 0;
        try {
            // Lấy kết nối
            connection = getConnection();

            // Tạo câu lệnh SELECT
            StringBuilder sqlBuilder = new StringBuilder();
            sqlBuilder.append("SELECT COUNT(*) FROM ").append(clazz.getSimpleName());
            //List parameter
            List<Object> parameters = new ArrayList<>();

            // Chuẩn bị câu lệnh
            statement = connection.prepareStatement(sqlBuilder.toString());
```

```
            // Gán giá trị cho các tham số của câu truy vấn
            int index = 1;
            for (Object value : parameters) {
                statement.setObject(index, value);
                index++;
            }

            // Thực thi truy vấn
            resultSet = statement.executeQuery();

            // Khai báo danh sách kết quả
            // Duyệt result set
            if (resultSet.next()) {
                total = resultSet.getInt(1);
            }

        } catch (IllegalArgumentException | SQLException e) {
            System.err.println("4USER: Bắn Exception ở hàm findTotalRecord: " + e.getMessage
());
        } finally {
            try {
                // Đóng kết nối và các tài nguyên
                if (resultSet != null) {

                }
                if (statement != null) {
                    statement.close();
                }
                if (connection != null) {
                    connection.close();
                }
            } catch (SQLException e) {
                System.err.println("4USER: Bắn Exception ở hàm findTotalRecord: " + e.getMess
age());
            }
        }
        return total;
    }

    /**
     * Tìm số lượng record của 1 bảng nào đó, điều kiện (optional)
     *
     * @param clazz: bảng muốn tìm
     * @param parameterMap: hashmap chứa các parameter
     * @return số lượng record
     */
    protected int findTotalRecordGenericDAO(Class<T> clazz, String sql, Map<String, Object> p
arameterMap) {
        int total = 0;
        try {
            // Lấy kết nối
            connection = getConnection();

            // Tạo câu lệnh SELECT
            StringBuilder sqlBuilder = new StringBuilder();
            sqlBuilder.append("SELECT COUNT(*) FROM ").append(clazz.getSimpleName());
            //List parameter
            List<Object> parameters = new ArrayList<>();
```

```
        // Thêm điều kiện
        if (parameterMap != null && !parameterMap.isEmpty()) {
            // code thêm điều kiện
            for (Map.Entry<String, Object> entry : parameterMap.entrySet()) {
                Object conditionValue = entry.getValue();

                parameters.add(conditionValue);
            }
        }

        // Chuẩn bị câu lệnh
        statement = connection.prepareStatement(sql);

        // Gán giá trị cho các tham số của câu truy vấn
        int index = 1;
        for (Object value : parameters) {
            statement.setObject(index, value);
            index++;
        }

        // Thực thi truy vấn
        resultSet = statement.executeQuery();

        // Khai báo danh sách kết quả
        // Duyệt result set
        if (resultSet.next()) {
            total = resultSet.getInt(1);
        }

    } catch (IllegalArgumentException | SQLException e) {
        System.err.println("4USER: Bắn Exception ở hàm findTotalRecord: " + e.getMessage
());
    } finally {
        try {
            // Đóng kết nối và các tài nguyên
            if (resultSet != null) {

            }
            if (statement != null) {
                statement.close();
            }
            if (connection != null) {
                connection.close();
            }
        } catch (SQLException e) {
            System.err.println("4USER: Bắn Exception ở hàm findTotalRecord: " + e.getMess
age());
        }
    }
    return total;
}

public abstract List<T> findAll();

public abstract int insert(T t);
```

```
}
```