```java
package fi.paatti.paattidatabaseutils.dbservice.querydelegates;


import com.vaadin.data.Container.Filter;
import com.vaadin.data.util.sqlcontainer.RowItem;
import com.vaadin.data.util.sqlcontainer.query.FreeformStatementDelegate;
import com.vaadin.data.util.sqlcontainer.query.OrderBy;
import com.vaadin.data.util.sqlcontainer.query.generator.StatementHelper;
import com.vaadin.data.util.sqlcontainer.query.generator.filter.QueryBuilder;
import java.io.Serializable;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;


/**
 * A base class for all other delegates. Has all the methods, fields etc. that
 * are common in all other delegates.
 * <p/>
 * <b>This delegate does not implement RowIdChange-events, though it has its own
 * implementation for informing about the acquired new row ID (see NewRowIdEvent).</b>
 * <p/>
 * @author Lauri Satokangas, lauri.n.satokangas@student.jyu.fi
 * @since 0.0.3
 */
public abstract class PaattiQueryDelegate implements FreeformStatementDelegate, Serializable {

  private final String tableName;
  private final String idColumn;
  private final String[] columnNames;
  private final String rowStatusColumn;
  private List<OrderBy> orderBys;
  private List<Filter> filters;
  private ArrayList<NewRowIDListener> listeners = new ArrayList<NewRowIDListener>();

  /**
   * The status for deleted row.
   */
  public static final String ROWSTATUS_DELETED = "DELETED";

  /**
   * The status for active row.
   */
```

```
  public static final String ROWSTATUS_ACTIVE = "ACTIVE";

  /**
   * Constructor.
   *
   * @param tableName The name of the database table.
   * @param idColumn The name of the id column in the database table.
   * @param columnNames The column names.
   */
  public PaattiQueryDelegate(String tableName, String idColumn, String[] columnNames) {
    this.tableName = tableName;
    this.idColumn = idColumn;
    this.columnNames = columnNames;
    this.rowStatusColumn = tableName + "_rowStatus";
  }

  /**
   * Fires an event after acquiring the new row ID from a succesful insert to
   * the database.
   *
   * @param newRowId The new row ID.
   */
  protected void fireNewRowIdEvent(Object newRowId) {
    for (NewRowIDListener delegateRowIdChangeListener : listeners) {
      delegateRowIdChangeListener.rowIdChanged(new NewRowIdEvent(newRowId));
    }
  }

  /**
   * Adds a NewRowIDListener to the list of listeners.
   *
   * @param listener The listener to be added.
   */
  public void addNewiRowIdListener(NewRowIDListener listener) {
    listeners.add(listener);
  }

  /**
   * Removes a NewRowIDListener from the list of listeners.
   *
   * @param listener The listener to be removed.
   */
  public void removeNewRowIdListener(NewRowIDListener listener) {
    listeners.remove(listener);
  }
```

```
  /**
   * Returns the row count from the delegate’s table.
   * <p/>
   * SELECT COUNT (*) FROM [table]
   * <p/>
   * If the filters are set they are used in the query.
   * <p/>
   * @return @throws UnsupportedOperationException.
   */
  public StatementHelper getCountStatement() {
    StatementHelper countHelper = new StatementHelper();
    StringBuilder querySb = new StringBuilder();

    querySb.append("SELECT COUNT(*) FROM ").append(getTableName());

    if (filters != null) {
      querySb.append(QueryBuilder.getWhereStringForFilters(filters, countHelper));
    }

    countHelper.setQueryString(querySb.toString());
    return countHelper;
  }

  /**
   * Creates an insert query from the given parameters.
   * <p/>
   * Example: INSERT INTO TASK (TASK_description, TASK_content, TASK_url,
   * TASK_sequence, TASK_EVENT_eventID, TASK_TASKTYPE_taskTypeID) VALUES (?,
   * ?, ?, ?, ?, ?)
   * <p/>
   * @param tableName The name of the table.
   * @param idColumn The name of the table id column.
   * @param allColumnNames The column names.
   * @return The INSERT query string.
   */
  public String getInsertQueryString(String tableName, String idColumn, String[] allColumnNames) {
    StringBuilder insertSb = new StringBuilder("INSERT INTO ");

    insertSb.append(tableName).append(" (");
    insertSb.append(getArrayAsStringWithCommasAndWithoutIdColumn(allColumnNames, idColumn));
    insertSb.append(" ) ");
    insertSb.append(getValuesString(allColumnNames.length - 1)); // dont add the id column
    return insertSb.toString();
  }

  /**
```

```
   * Uses the set tablename, id column and column names to create an insert
   * string.
   *
   * @see PaattiQueryDelegate#getInsertQueryString()
   * @return The INSERT query string.
   */
  public String getInsertQueryString() {
    return getInsertQueryString(getTableName(), getIdColumn(), getColumnNames());

  }

  /**
   * Create an update String.
   * <p/>
   * Example: UPDATE TASK SET TASK_description = ?, TASK_content = ?, TASK_url
   * = ?, TASK_sequence = ?, TASK_EVENT_eventID = ?, TASK_TASKTYPE_taskTypeID
   * = ? WHERE TASK_taskId = ?
   * <p/>
   * @param tableName The name of the table.
   * @param idColumn The name of the id column.
   * @param allColumnNames The array holding all the column names.
   * @return The string to be used in the update query.
   */
  public String getUpdateQueryString(String tableName, String idColumn, String[] allColumnNames) {
    StringBuilder updateSb = new StringBuilder("UPDATE ");

    updateSb.append(tableName).append(" SET ");
    updateSb.append(getArrayAsStringWithAddedString(allColumnNames, idColumn, " = ?, ", false));
    updateSb.deleteCharAt(updateSb.lastIndexOf(","));
    updateSb.append(" WHERE ").append(idColumn).append(" = ? ");
    return updateSb.toString();
  }

  /**
   * Uses the set tablename, id column and column names to create an update
   * string.
   *
   * @see PaattiQueryDelegate#getUpdateQueryString()
   * @return The combined update query string.
   */
  public String getUpdateQueryString() {
    return getUpdateQueryString(getTableName(), getIdColumn(), getColumnNames());
  }

  /**
   * Stores a row in the database.
```

```
   *
   * @param conn The database connection.
   * @param row The row where the values will be set.
   * @return The number of affected rows in the database.
   * @throws SQLException
   */
  public abstract int storeRow(Connection conn, RowItem row) throws SQLException;

  /**
   * Sets the values for the row.
   *
   * @param statement The prepared statement where the values will be set.
   * @param row The row where the values will be taken from.
   * @throws SQLException
   */
  protected abstract void setRowValues(PreparedStatement statement, RowItem row) throws SQLException;

  /**
   * Removes a row by setting its rowStatus to 'DELETED'.
   *
   * @param conn The database connection
   * @param row The row where the values will be set.
   * @return True if the count of affected rows is one, otherwise false.
   * @throws SQLException
   */
  public boolean removeRow(Connection conn, RowItem row) throws SQLException {

    StringBuilder removeQuerySb = new StringBuilder();

    removeQuerySb.append("UPDATE ").append(tableName).append(" SET ").append(rowStatusColumn).append(" = ?");
    removeQuerySb.append(" WHERE ").append(getIdColumn()).append(" = ");
    removeQuerySb.append("'").append(row.getItemProperty(getIdColumn()).getValue()).append("'");

    PreparedStatement removeStatement = conn.prepareStatement(removeQuerySb.toString());

    removeStatement.setString(1, ROWSTATUS_DELETED);

    int rowsChanged = removeStatement.executeUpdate();

    removeStatement.close();
    return rowsChanged == 1;
  }

  /**
   * Sets the delegate's container filters.
   *
```

```
   * @param filters The list of container filters.
   */
  public void setFilters(List<Filter> filters) {
    if (filters == null) {
      this.filters = null;
      return;
    }
    this.filters = Collections.unmodifiableList(filters);
  }

  /**
   * Sets the delegate's OrderBys.
   *
   * @param orderBys The list of sorting rules.
   */
  public void setOrderBy(List<OrderBy> orderBys) {
    if (orderBys == null) {
      this.orderBys = null;
      return;
    }
    this.orderBys = Collections.unmodifiableList(orderBys);
  }

  // ** HELPER-METHODS FOR CREATING STRINGS FOR THE SQLQUERIES **//
  /**
   * Creates a String with all the column names from the given array. Commas
   * are added between each column name.
   * <p/>
   * Example from TASK-table: " TASK_taskId, TASK_description, TASK_content,
   * TASK_url, TASK_sequence, TASK_EVENT_eventID, TASK_TASKTYPE_taskTypeID"
   * <p/>
   * @param allColumnNames The array of column names.
   * @return A string containing column names with a comma (and space) between
   * each name.
   */
  public String getArrayAsStringWithCommas(String[] allColumnNames) {
    return getArrayAsStringWithAddedString(allColumnNames, null, ", ", true);
  }

  /**
   * Creates a String with all the column names (except the id column) from
   * the given array. Commas are added between each column name.
   * <p/>
   * Example from TASK-table: " TASK_description, TASK_content, TASK_url,
   * TASK_sequence, TASK_EVENT_eventID, TASK_TASKTYPE_taskTypeID"
   * <p/>
```

```
   * @param allColumnNames The array of column names.
   * @param idColumn The name of the id column.
   * @return A string containing column names with a comma (and space) between
   * each name, not containing the id column.
   */
  public String getArrayAsStringWithCommasAndWithoutIdColumn(String[] allColumnNames, String idColumn) {
    return getArrayAsStringWithAddedString(allColumnNames, idColumn, ", ", true);
  }

  /**
   * Creates a string for inserting values.
   * <p/>
   * Example: " VALUES (?, ?, ?, ?)"
   * <p/>
   * @param numberOfValues The values count.
   * @return The VALUES-string
   */
  public String getValuesString(int numberOfValues) {
    StringBuilder valuesSb = new StringBuilder(" VALUES (");

    for (int i = 0; i < numberOfValues - 1; i++) {
      valuesSb.append("?, ");
    }

    valuesSb.append("?)");
    return valuesSb.toString();
  }

  /**
   * Creates a string from array adding the given string between arrays
   * fields.
   * <p/>
   * Example: getArrayAsStringWithAddedString([TASK_description, TASK_url], "
   * = ?," ) --> " TASK_description = ?, TASK_url = ?, "
   * <p/>
   * @param array The array.
   * @param fieldToBeLeftOut The field that will be left out. Set to null if
   * all the fields should be included.
   * @param stringToBeAdded The suffix for all the fields of the array.
   * @param removeLastStringToBeAdded Should the last added string be removed
   * or not.
   * @return The array as string with the string added between each value of
   * the array.
   */
  public String getArrayAsStringWithAddedString(String[] array, String fieldToBeLeftOut, String stringToBeAdded, boolean
removeLastStringToBeAdded) {
```

```
    StringBuilder querySb = new StringBuilder(" ");

    for (int i = 0; i < array.length; i++) {
      if (!array[i].equals(fieldToBeLeftOut)) {
        querySb.append(array[i]);
        querySb.append(stringToBeAdded);
      }
    }
    if (removeLastStringToBeAdded) {
      querySb.replace(querySb.length() - stringToBeAdded.length(), querySb.length(), "");
    }

    return querySb.toString();
  }

  /**
   * Creates a string from the list of filters, without the word 'WHERE'.
   *
   * @param sh StatementHelper
   * @param startWithAnd Whether the string should start with 'AND' or not.
   * @return The filter string.
   */
  public String getFilterStringWithoutWhere(StatementHelper sh, boolean startWithAnd) {
    if (filters == null || filters.isEmpty()) {
      return "";
    }
    StringBuilder filterSb = new StringBuilder();

    if (startWithAnd) {
      filterSb.append(" AND ");
    }

    filterSb.append(QueryBuilder.getJoinedFilterString(filters, "AND", sh));
    String toString = filterSb.toString();
    String replaceAll = toString.replaceAll("\"", "");

    return replaceAll;
  }

  // ** HELPER-METHODS FOR CREATING STRINGS FOR THE QUERIES **//

  /**
   * Returns the table name that is used as the delegate's "main" table.
   *
   * @return the name of the delegate's table.
```

```
   */
  public String getTableName() {
    return tableName;
  }

  /**
   * Returns the column name that is used as the main table's id column.
   *
   * @return the name of the idColumn.
   */
  public String getIdColumn() {
    return idColumn;
  }

  /**
   * Returns the column names that are used in this delegate.
   *
   * @return the columnNames array.
   */
  public String[] getColumnNames() {
    return columnNames;
  }

  /**
   * Returns the container filter list.
   *
   * @return The list of container filters.
   */
  public List<Filter> getFilters() {
    return filters;
  }

  /**
   * Returns the sorting rules (order by) list.
   *
   * @return The list of sorting rules.
   */
  public List<OrderBy> getOrderBys() {
    return orderBys;
  }

  /**
   * Inherited from FreeformStatementDelegate. Not in use.
   */
  public StatementHelper getContainsRowQueryStatement(Object... keys) throws UnsupportedOperationException {
    throw new UnsupportedOperationException("Not supported yet.");
```

```
  }

  /**
   * Inherited from FreeformStatementDelegate. Not in use.
   */
  public String getCountQuery() throws UnsupportedOperationException {
    throw new UnsupportedOperationException("Not supported yet.");
  }

  /**
   * Inherited from FreeformStatementDelegate. Not in use.
   */
  public String getContainsRowQueryString(Object... keys) throws UnsupportedOperationException {
    throw new UnsupportedOperationException("Not supported yet.");
  }

  /**
   * Inherited from FreeformStatementDelegate. Not in use.
   */
  public String getQueryString(int offset, int limit) throws UnsupportedOperationException {
    throw new UnsupportedOperationException("Not supported yet.");
  }
}
```