Tapestry5HibernateGridDatasource1

```
package de.hsofttec.core5.datasource;

import java.util.Iterator;
import java.util.regex.Pattern;

import org.apache.commons.lang.StringUtils;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.apache.tapestry.beaneditor.PropertyModel;
import org.apache.tapestry.grid.GridDataSource;
import org.apache.tapestry.ioc.internal.util.TapestryException;

import org.hibernate.QueryException;

import de.hsofttec.core5.dao.GenericDAO;

/**
 * grid datasource for hibernate queries.
 *
 * @author <a href="mailto:shomburg@hsofttec.com">shomburg</a>
 * @version $Id: HibernateDataSource.java 13 2007-09-27 16:25:32Z shomburg $
 */
public class HibernateDataSource implements GridDataSource
{
    private static Log _logger = LogFactory.getLog(HibernateDataSource.class);
    private GenericDAO _genericDAO;
    private String _hqlString;
    private String _hqlCountString;
    private int _rowCount = -1;
    private Iterator _pageRowList;

    public HibernateDataSource(GenericDAO genericDAO, String hqlString)
    {
        this(genericDAO, hqlString, null);
    }

    public HibernateDataSource(GenericDAO genericDAO, String hqlString, String hqlSpecialCountString)
    {
        _genericDAO = genericDAO;
        _hqlString = hqlString;
        _hqlCountString = hqlSpecialCountString;
    }

    /**
     * Returns the number of rows available in the data source.
     */
    public int getAvailableRows()
    {
        if (_rowCount < 0)
            _rowCount = getMaximumResultObjects();

        return _rowCount;
    }

    /**
     * Invoked to allow the source to prepare to present values. This gives the source a chance to
     * pre-fetch data (when appropriate) and informs the source of the desired sort order.
     *
     * @param startIndex the starting index to be retrieved
     * @param endIndex   the ending index to be retrieved
     * @param sortModel  the property model that defines what data will be used for sorting, or null if no
     *                   sorting is required (in which case, whatever natural order is provided by the
     *                   underlying data source will be used)
     * @param ascending  if true, then sort ascending, else decending
     */
    public void prepare(int startIndex, int endIndex, PropertyModel sortModel, boolean ascending)
```

```
    {
        if (_logger.isInfoEnabled())
            _logger.info(String.format("processing prepare(%d, %d) for '%s'",
                                       startIndex, endIndex, _genericDAO.getPersistentClass().getName()));

        _pageRowList = _genericDAO.findByQuery(_hqlString, startIndex, endIndex + 1).iterator();
    }

    /**
     * Returns the row value at the provided index. This method will be invoked in sequential order.
     */
    public Object getRowValue(int index)
    {
        Object entityObject = null;

        if (_pageRowList.hasNext())
            entityObject = _pageRowList.next();

        return entityObject;
    }

    /**
     * Returns the type of value in the rows, or null if not known.
     *
     * @return the row type, or null
     */
    public Class getRowType()
    {
        return _genericDAO.getPersistentClass();
    }

    /**
     * get the maximum count of rows to display.
     */
    private int getMaximumResultObjects()
    {
        String tempQuery1;
        String queryString = _hqlString;

        try
        {
            if (_logger.isInfoEnabled())
                _logger.info(String.format("processing getMaximumResultObjects() for '%s'",
                                           _genericDAO.getPersistentClass().getName()));

            if (_hqlCountString != null)
                tempQuery1 = _hqlCountString;
            else
                tempQuery1 = convertQueryString(queryString);

            Long result = (Long) _genericDAO.countByQuery(tempQuery1);
            if (result == null)
                return 0;

            return result.intValue();
        }
        catch (QueryException e)
        {
            throw new TapestryException("entity not \"mapped\" in hibernate.cfg.xml ?", this, e);
        }
    }

    /**
     * den SQL-Query so Aufbauen, das wir einen Count auf die Tabelle absetzen koennen.
     */
    private String convertQueryString(String originalQueryString)
    {
        String tempQueryString = StringUtils.substring(originalQueryString,
                                                       StringUtils.indexOf(StringUtils.
                                                                           upperCase(originalQueryString),
"FROM"));
```

```
        String convertedQueryString = "SELECT COUNT(*) ";

        // Split input with the pattern
        Pattern p = Pattern.compile("[\\s]+");
        String[] result = p.split(tempQueryString);

        for (String queryWord : result)
        {
            //
            // ist queryWord ein Query-Fragment, was in einem Count-Query nicht auftauchen darf ?
            //
            // if (queryWord.equalsIgnoreCase("LEFT") ||
                   queryWord.equalsIgnoreCase("JOIN") ||
                   queryWord.equalsIgnoreCase("FETCH"))
            if (queryWord.equalsIgnoreCase("FETCH"))
                continue;

            if (queryWord.equalsIgnoreCase("ORDER"))
                break;

            convertedQueryString += queryWord + " ";
        }

        if (_logger.isInfoEnabled())
            _logger.info("source: " + originalQueryString + System.getProperty("line.separator") +
                    "dest: " + convertedQueryString);

        return convertedQueryString;
    }
}
```