001/\*  
002 \* Licensed to the Apache Software Foundation (ASF) under one or more  
003 \* contributor license agreements. See the NOTICE file distributed with  
004 \* this work for additional information regarding copyright ownership.  
005 \* The ASF licenses this file to You under the Apache License, Version 2.0  
006 \* (the "License"); you may not use this file except in compliance with  
007 \* the License. You may obtain a copy of the License at  
008 \*  
009 \* http://www.apache.org/licenses/LICENSE-2.0  
010 \*  
011 \* Unless required by applicable law or agreed to in writing, software  
012 \* distributed under the License is distributed on an "AS IS" BASIS,  
013 \* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  
014 \* See the License for the specific language governing permissions and  
015 \* limitations under the License.  
016 \*/  
017  
018  
019package org.apache.commons.beanutils;  
020  
021  
022import java.sql.SQLException;  
023import java.util.Iterator;  
024import java.util.NoSuchElementException;  
025  
026  
027/\*\*  
028 \* <p>Implementation of <code>java.util.Iterator</code> returned by the  
029 \* <code>iterator()</code> method of {@link ResultSetDynaClass}. Each  
030 \* object returned by this iterator will be a {@link DynaBean} that  
031 \* represents a single row from the result set being wrapped.</p>  
032 \*  
033 \* @version $Id$  
034 \*/  
035  
036public class ResultSetIterator implements DynaBean, Iterator<DynaBean> {  
037  
038  
039 // ------------------------------------------------------------ Constructor  
040  
041  
042 /\*\*  
043 \* <p>Construct an <code>Iterator</code> for the result set being wrapped  
044 \* by the specified {@link ResultSetDynaClass}.</p>  
045 \*  
046 \* @param dynaClass The {@link ResultSetDynaClass} wrapping the  
047 \* result set we will iterate over  
048 \*/  
049 ResultSetIterator(final ResultSetDynaClass dynaClass) {  
050  
051 this.dynaClass = dynaClass;  
052  
053 }  
054  
055  
056 // ----------------------------------------------------- Instance Variables  
057  
058  
059  
060 /\*\*  
061 \* <p>Flag indicating whether the result set is currently positioned at a  
062 \* row for which we have not yet returned an element in the iteration.</p>  
063 \*/  
064 protected boolean current = false;  
065  
066  
067 /\*\*  
068 \* <p>The {@link ResultSetDynaClass} we are associated with.</p>  
069 \*/  
070 protected ResultSetDynaClass dynaClass = null;  
071  
072  
073 /\*\*  
074 \* <p>Flag indicating whether the result set has indicated that there are  
075 \* no further rows.</p>  
076 \*/  
077 protected boolean eof = false;  
078  
079  
080 // ------------------------------------------------------- DynaBean Methods  
081  
082  
083 /\*\*  
084 \* Does the specified mapped property contain a value for the specified  
085 \* key value?  
086 \*  
087 \* @param name Name of the property to check  
088 \* @param key Name of the key to check  
089 \* @return <code>true<code> if the mapped property contains a value for  
090 \* the specified key, otherwise <code>false</code>  
091 \*  
092 \* @throws IllegalArgumentException if there is no property  
093 \* of the specified name  
094 \*/  
095 public boolean contains(final String name, final String key) {  
096  
097 throw new UnsupportedOperationException  
098 ("FIXME - mapped properties not currently supported");  
099  
100 }  
101  
102  
103 /\*\*  
104 \* Return the value of a simple property with the specified name.  
105 \*  
106 \* @param name Name of the property whose value is to be retrieved  
107 \* @return The property's value  
108 \*  
109 \* @throws IllegalArgumentException if there is no property  
110 \* of the specified name  
111 \*/  
112 public Object get(final String name) {  
113  
114 if (dynaClass.getDynaProperty(name) == null) {  
115 throw new IllegalArgumentException(name);  
116 }  
117 try {  
118 return dynaClass.getObjectFromResultSet(name);  
119 } catch (final SQLException e) {  
120 throw new RuntimeException  
121 ("get(" + name + "): SQLException: " + e);  
122 }  
123  
124 }  
125  
126  
127 /\*\*  
128 \* Return the value of an indexed property with the specified name.  
129 \*  
130 \* @param name Name of the property whose value is to be retrieved  
131 \* @param index Index of the value to be retrieved  
132 \* @return The indexed property's value  
133 \*  
134 \* @throws IllegalArgumentException if there is no property  
135 \* of the specified name  
136 \* @throws IllegalArgumentException if the specified property  
137 \* exists, but is not indexed  
138 \* @throws IndexOutOfBoundsException if the specified index  
139 \* is outside the range of the underlying property  
140 \* @throws NullPointerException if no array or List has been  
141 \* initialized for this property  
142 \*/  
143 public Object get(final String name, final int index) {  
144  
145 throw new UnsupportedOperationException  
146 ("FIXME - indexed properties not currently supported");  
147  
148 }  
149  
150  
151 /\*\*  
152 \* Return the value of a mapped property with the specified name,  
153 \* or <code>null</code> if there is no value for the specified key.  
154 \*  
155 \* @param name Name of the property whose value is to be retrieved  
156 \* @param key Key of the value to be retrieved  
157 \* @return The mapped property's value  
158 \*  
159 \* @throws IllegalArgumentException if there is no property  
160 \* of the specified name  
161 \* @throws IllegalArgumentException if the specified property  
162 \* exists, but is not mapped  
163 \*/  
164 public Object get(final String name, final String key) {  
165  
166 throw new UnsupportedOperationException  
167 ("FIXME - mapped properties not currently supported");  
168  
169 }  
170  
171  
172 /\*\*  
173 \* Return the <code>DynaClass</code> instance that describes the set of  
174 \* properties available for this DynaBean.  
175 \*  
176 \* @return The associated DynaClass  
177 \*/  
178 public DynaClass getDynaClass() {  
179  
180 return (this.dynaClass);  
181  
182 }  
183  
184  
185 /\*\*  
186 \* Remove any existing value for the specified key on the  
187 \* specified mapped property.  
188 \*  
189 \* @param name Name of the property for which a value is to  
190 \* be removed  
191 \* @param key Key of the value to be removed  
192 \*  
193 \* @throws IllegalArgumentException if there is no property  
194 \* of the specified name  
195 \*/  
196 public void remove(final String name, final String key) {  
197  
198 throw new UnsupportedOperationException  
199 ("FIXME - mapped operations not currently supported");  
200  
201 }  
202  
203  
204 /\*\*  
205 \* Set the value of a simple property with the specified name.  
206 \*  
207 \* @param name Name of the property whose value is to be set  
208 \* @param value Value to which this property is to be set  
209 \*  
210 \* @throws ConversionException if the specified value cannot be  
211 \* converted to the type required for this property  
212 \* @throws IllegalArgumentException if there is no property  
213 \* of the specified name  
214 \* @throws NullPointerException if an attempt is made to set a  
215 \* primitive property to null  
216 \*/  
217 public void set(final String name, final Object value) {  
218  
219 if (dynaClass.getDynaProperty(name) == null) {  
220 throw new IllegalArgumentException(name);  
221 }  
222 try {  
223 dynaClass.getResultSet().updateObject(name, value);  
224 } catch (final SQLException e) {  
225 throw new RuntimeException  
226 ("set(" + name + "): SQLException: " + e);  
227 }  
228  
229 }  
230  
231  
232 /\*\*  
233 \* Set the value of an indexed property with the specified name.  
234 \*  
235 \* @param name Name of the property whose value is to be set  
236 \* @param index Index of the property to be set  
237 \* @param value Value to which this property is to be set  
238 \*  
239 \* @throws ConversionException if the specified value cannot be  
240 \* converted to the type required for this property  
241 \* @throws IllegalArgumentException if there is no property  
242 \* of the specified name  
243 \* @throws IllegalArgumentException if the specified property  
244 \* exists, but is not indexed  
245 \* @throws IndexOutOfBoundsException if the specified index  
246 \* is outside the range of the underlying property  
247 \*/  
248 public void set(final String name, final int index, final Object value) {  
249  
250 throw new UnsupportedOperationException  
251 ("FIXME - indexed properties not currently supported");  
252  
253 }  
254  
255  
256 /\*\*  
257 \* Set the value of a mapped property with the specified name.  
258 \*  
259 \* @param name Name of the property whose value is to be set  
260 \* @param key Key of the property to be set  
261 \* @param value Value to which this property is to be set  
262 \*  
263 \* @throws ConversionException if the specified value cannot be  
264 \* converted to the type required for this property  
265 \* @throws IllegalArgumentException if there is no property  
266 \* of the specified name  
267 \* @throws IllegalArgumentException if the specified property  
268 \* exists, but is not mapped  
269 \*/  
270 public void set(final String name, final String key, final Object value) {  
271  
272 throw new UnsupportedOperationException  
273 ("FIXME - mapped properties not currently supported");  
274  
275 }  
276  
277  
278 // ------------------------------------------------------- Iterator Methods  
279  
280  
281 /\*\*  
282 \* <p>Return <code>true</code> if the iteration has more elements.</p>  
283 \*  
284 \* @return <code>true</code> if the result set has another  
285 \* row, otherwise <code>false</code>  
286 \*/  
287 public boolean hasNext() {  
288  
289 try {  
290 advance();  
291 return (!eof);  
292 } catch (final SQLException e) {  
293 throw new RuntimeException("hasNext(): SQLException: " + e);  
294 }  
295  
296 }  
297  
298  
299 /\*\*  
300 \* <p>Return the next element in the iteration.</p>  
301 \*  
302 \* @return advance to the new row and return this  
303 \*/  
304 public DynaBean next() {  
305  
306 try {  
307 advance();  
308 if (eof) {  
309 throw new NoSuchElementException();  
310 }  
311 current = false;  
312 return (this);  
313 } catch (final SQLException e) {  
314 throw new RuntimeException("next(): SQLException: " + e);  
315 }  
316  
317 }  
318  
319  
320 /\*\*  
321 \* <p>Remove the current element from the iteration. This method is  
322 \* not supported.</p>  
323 \*/  
324 public void remove() {  
325  
326 throw new UnsupportedOperationException("remove()");  
327  
328 }  
329  
330  
331 // ------------------------------------------------------ Protected Methods  
332  
333  
334 /\*\*  
335 \* <p>Advance the result set to the next row, if there is not a current  
336 \* row (and if we are not already at eof).</p>  
337 \*  
338 \* @throws SQLException if the result set throws an exception  
339 \*/  
340 protected void advance() throws SQLException {  
341  
342 if (!current && !eof) {  
343 if (dynaClass.getResultSet().next()) {  
344 current = true;  
345 eof = false;  
346 } else {  
347 current = false;  
348 eof = true;  
349 }  
350 }  
351  
352 }  
353  
354  
355}