


Branch: trunk

Find file

Copy path

Kundera / src / kundera-mongo / src / test / java / com / impetus / client / crud / gfs / GridFSTest.java

 devender-yadav #665

92cf521 on May 7 2015

1 contributor

366 lines (313 sloc) 9.75 KB

Raw

Blame

History







```
1 /*****
2  * * Copyright 2015 Impetus Infotech.
3  * *
4  * * Licensed under the Apache License, Version 2.0 (the "License");
5  * * you may not use this file except in compliance with the License.
6  * * You may obtain a copy of the License at
7  * *
8  * *      http://www.apache.org/licenses/LICENSE-2.0
9  * *
10 * * Unless required by applicable law or agreed to in writing, software
11 * * distributed under the License is distributed on an "AS IS" BASIS,
12 * * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
13 * * See the License for the specific language governing permissions and
14 * * limitations under the License.
15 *****/
16 package com.impetus.client.crud.gfs;
17
18 import java.io.IOException;
19 import java.nio.file.Files;
20 import java.nio.file.Path;
21 import java.nio.file.Paths;
22 import java.util.List;
23
24 import javax.persistence.EntityManager;
25 import javax.persistence.EntityManagerFactory;
26 import javax.persistence.Persistence;
27 import javax.persistence.Query;
28
29 import org.junit.After;
30 import org.junit.AfterClass;
31 import org.junit.Assert;
32 import org.junit.Before;
33 import org.junit.BeforeClass;
34 import org.junit.Test;
35
36 import com.impetus.client.utils.MongoUtils;
37
38 /**
39  * The Class GridFSTest.
40  *
41  * @author Devender Yadav
42  */
43 public class GridFSTest
44 {
45
46     /** The Constant _PU. */
47     private static final String _PU = "gfs_pu";
48
49     /** The emf. */
50     private static EntityManagerFactory emf;
51
52     /** The em. */
53     private EntityManager em;
54
55     /** The profile pic1. */
56     private byte[] profilePic1;
57
58     /** The profile pic2. */
59     private byte[] profilePic2;
60
61     /** The profile pic3. */
```

```

62     private byte[] profilePic3;
63
64     /**
65      * Sets the up before class.
66      *
67      * @throws Exception
68      *         the exception
69      */
70     @BeforeClass
71     public static void SetUpBeforeClass() throws Exception
72     {
73         emf = Persistence.createEntityManagerFactory(_PU);
74     }
75
76     /**
77      * Sets the up.
78      *
79      * @throws Exception
80      *         the exception
81      */
82     @Before
83     public void setUp() throws Exception
84     {
85         em = emf.createEntityManager();
86     }
87
88     /**
89      * Tear down.
90      *
91      * @throws Exception
92      *         the exception
93      */
94     @After
95     public void tearDown() throws Exception
96     {
97         em.close();
98     }
99
100    /**
101     * Tear down after class.
102     *
103     * @throws Exception
104     *         the exception
105     */
106    @AfterClass
107    public static void tearDownAfterClass() throws Exception
108    {
109        MongoUtils.dropDatabase(emf, _PU);
110        emf.close();
111    }
112
113    /**
114     * Test CRUD GridFS.
115     *
116     * @throws Exception
117     *         the exception
118     */
119    @Test
120    public void testCRUDGridFS() throws Exception
121    {
122        testInsert();
123        testUpdateNonLobField();
124        testUpdateLobField();
125        testDelete();
126    }
127
128    /**
129     * Test query.
130     *
131     * @throws Exception
132     *         the exception
133     */
134    @Test
135    public void testQuery() throws Exception
136    {
137        profilePic1 = createBinaryData("src/test/resources/pic.jpg", 10);
138        GFSUser user1 = prepareUserObject(1, "Dev", profilePic1);
139
140        profilePic2 = createBinaryData("src/test/resources/pic.jpg", 15);
141        GFSUser user2 = prepareUserObject(2, "PG", profilePic2);
142
143        profilePic3 = createBinaryData("src/test/resources/pic.jpg", 20);

```

```

144     GFSUser user3 = prepareUserObject(3, "Amit", profilePic3);
145
146     em.persist(user1);
147     em.persist(user2);
148     em.persist(user3);
149
150     em.clear();
151
152     String query = "SELECT u FROM GFSUser u";
153     Query qry = em.createQuery(query);
154     List<GFSUser> userList = qry.getResultList();
155     Assert.assertEquals(3, userList.size());
156     assertUsers(userList, true, true, true);
157
158     query = "SELECT u FROM GFSUser u WHERE u.name = 'Dev'";
159     qry = em.createQuery(query);
160     userList = qry.getResultList();
161     GFSUser user = userList.get(0);
162     Assert.assertEquals("Dev", user.getName());
163     Assert.assertEquals(profilePic1.length, user.getProfilePic().length);
164
165     query = "SELECT u FROM GFSUser u WHERE u.name = 'Karthik'";
166     qry = em.createQuery(query);
167     userList = qry.getResultList();
168     Assert.assertEquals(true, userList.isEmpty());
169
170     query = "SELECT u FROM GFSUser u WHERE u.userId = 2";
171     qry = em.createQuery(query);
172     userList = qry.getResultList();
173     user = userList.get(0);
174     Assert.assertEquals("PG", user.getName());
175     Assert.assertEquals(profilePic2.length, user.getProfilePic().length);
176
177     query = "SELECT u FROM GFSUser u WHERE u.userId > 1";
178     qry = em.createQuery(query);
179     userList = qry.getResultList();
180     Assert.assertEquals(2, userList.size());
181     assertUsers(userList, false, true, true);
182
183     query = "SELECT u FROM GFSUser u WHERE u.userId = 2 and u.name = 'PG'";
184     qry = em.createQuery(query);
185     userList = qry.getResultList();
186     Assert.assertEquals(1, userList.size());
187     user = userList.get(0);
188     Assert.assertEquals("PG", user.getName());
189     Assert.assertEquals(profilePic2.length, user.getProfilePic().length);
190
191     query = "SELECT u FROM GFSUser u WHERE u.userId < 3 order by u.name DESC";
192     qry = em.createQuery(query);
193     userList = qry.getResultList();
194     Assert.assertEquals(2, userList.size());
195     assertUsers(userList, true, true, false);
196
197     // remove all users
198     GFSUser u1 = em.find(GFSUser.class, 1);
199     GFSUser u2 = em.find(GFSUser.class, 2);
200     GFSUser u3 = em.find(GFSUser.class, 3);
201     em.clear();
202     em.remove(u1);
203     em.remove(u2);
204     em.remove(u3);
205 }
206
207 /**
208  * Test insert.
209  */
210 private void testInsert()
211 {
212     byte[] profilePic = createBinaryData("src/test/resources/pic.jpg", 20);
213     GFSUser user = prepareUserObject(1, "Dev", profilePic);
214     em.persist(user);
215
216     em.clear();
217
218     GFSUser u = em.find(GFSUser.class, 1);
219     Assert.assertEquals("Dev", u.getName());
220     Assert.assertEquals(profilePic.length, u.getProfilePic().length);
221 }
222
223 /**
224  * Test update non lob field.

```

```

225     */
226 private void testUpdateNonLobField()
227 {
228     GFSUser user = em.find(GFSUser.class, 1);
229     user.setName("Devender");
230     em.merge(user);
231
232     em.clear();
233
234     GFSUser u1 = em.find(GFSUser.class, 1);
235
236     Assert.assertNotNull(u1);
237     Assert.assertEquals("Devender", u1.getName());
238 }
239
240 /**
241  * Test update lob field.
242  */
243 private void testUpdateLobField()
244 {
245     GFSUser user = em.find(GFSUser.class, 1);
246     byte[] profilePic = createBinaryData("src/test/resources/pic.jpg", 15);
247     user.setProfilePic(profilePic);
248
249     em.merge(user);
250
251     em.clear();
252
253     GFSUser u = em.find(GFSUser.class, 1);
254
255     Assert.assertNotNull(u);
256     Assert.assertEquals("Devender", u.getName());
257     Assert.assertEquals(profilePic.length, u.getProfilePic().length);
258 }
259
260 /**
261  * Test delete.
262  */
263 private void testDelete()
264 {
265     GFSUser user = em.find(GFSUser.class, 1);
266     em.remove(user);
267     em.clear();
268     GFSUser u = em.find(GFSUser.class, 1);
269     Assert.assertNull(u);
270 }
271
272 /**
273  * Prepare user object.
274  *
275  * @param userID
276  *         the user id
277  * @param name
278  *         the name
279  * @param profilePic
280  *         the profile pic
281  * @return the GFS user
282  */
283 private GFSUser prepareUserObject(int userID, String name, byte[] profilePic)
284 {
285     GFSUser user = new GFSUser();
286     user.setUserId(userID);
287     user.setName(name);
288     user.setProfilePic(profilePic);
289     return user;
290 }
291
292 /**
293  * Creates the binary data.
294  *
295  * @param locationPath
296  *         the location path
297  * @param multiplicationFactor
298  *         the multiplication factor
299  * @return the byte[]
300  */
301 private byte[] createBinaryData(String locationPath, int multiplicationFactor)
302 {
303     Path path = Paths.get(locationPath);
304     byte[] data = null;
305     try
306     {

```

```

307         data = Files.readAllBytes(path);
308     }
309     catch (IOException e)
310     {
311         Assert.fail();
312     }
313
314     byte[] multipliedData = new byte[data.length * multiplicationFactor];
315
316     for (int i = 0; i < multiplicationFactor; i++)
317     {
318         System.arraycopy(data, 0, multipliedData, i * data.length, data.length);
319     }
320
321     return multipliedData;
322 }
323
324 /**
325  * Assert users.
326  *
327  * @param userList
328  *         the user list
329  * @param foundUser1
330  *         the found user1
331  * @param foundUser2
332  *         the found user2
333  * @param foundUser3
334  *         the found user3
335  */
336 private void assertUsers(List<GFSUser> userList, boolean foundUser1, boolean foundUser2, boolean foundUser3)
337 {
338     for (GFSUser user : userList)
339     {
340         if (user.getUserId() == 1)
341         {
342             Assert.assertEquals("Dev", user.getName());
343             Assert.assertEquals(profilePic1.length, user.getProfilePic().length);
344         }
345
346         else if (user.getUserId() == 2)
347         {
348             Assert.assertEquals("PG", user.getName());
349             Assert.assertEquals(profilePic2.length, user.getProfilePic().length);
350         }
351
352         else if (user.getUserId() == 3)
353         {
354             Assert.assertEquals("Amit", user.getName());
355             Assert.assertEquals(profilePic3.length, user.getProfilePic().length);
356         }
357
358         else
359         {
360             Assert.fail();
361         }
362     }
363 }
364
365 }

```

