

Dismiss

Join GitHub today

GitHub is home to over 31 million developers working together to host and review code, manage projects, and build software together.

[Sign up](#)

Branch: master ▾ DB2 / src / test / java / index / TestLeafNode.java

[Find file](#) [Copy path](#) dreamlegends init add all files

2c520b8 26 days ago

1 contributor

377 lines (318 sloc) 13.2 KB

[Raw](#) [Blame](#) [History](#)   

```
1 package index;
2
3 import static org.junit.Assert.assertEquals;
4 import static org.junit.Assert.assertFalse;
5 import static org.junit.Assert.assertTrue;
6
7 import java.io.File;
8 import java.io.IOException;
9 import java.util.ArrayList;
10 import java.util.Arrays;
11 import java.util.Collections;
12 import java.util.Iterator;
13 import java.util.List;
14 import java.util.Optional;
15 import java.util.Random;
16
17 import org.junit.Rule;
18 import org.junit.Test;
19 import org.junit.rules.DisableOnDebug;
20 import org.junit.rules.TemporaryFolder;
21 import org.junit.rules.TestRule;
22 import org.junit.rules.Timeout;
23
24 import common.Pair;
25 import databox.DataBox;
26 import databox.IntDataBox;
27 import databox.Type;
28 import io.Page;
29 import io.PageAllocator;
30 import table.RecordId;
31
32 public class TestLeafNode {
33     public static final String testFile = "TestLeafNode";
34
35     @Rule
36     public TemporaryFolder tempFolder = new TemporaryFolder();
37
38     // 1 second max per method tested.
39     @Rule
40     public TestRule globalTimeout = new DisableOnDebug(Timeout.seconds(1));
41
42     private static DataBox d0 = new IntDataBox(0);
43     private static DataBox d1 = new IntDataBox(1);
44     private static DataBox d2 = new IntDataBox(2);
45     private static DataBox d3 = new IntDataBox(3);
```

```

private static DataBox d4 = new IntDataBox(4);

private static RecordId r0 = new RecordId(0, (short) 0);
private static RecordId r1 = new RecordId(1, (short) 1);
private static RecordId r2 = new RecordId(2, (short) 2);
private static RecordId r3 = new RecordId(3, (short) 3);
private static RecordId r4 = new RecordId(4, (short) 4);

// Helpers //////////////////////////////////////
private BPlusTreeMetadata getBPlusTreeMetadata(Type keySchema, int order)
    throws IOException {
    File file = tempFolder.newFile(testFile);
    String path = file.getAbsolutePath();
    PageAllocator allocator = new PageAllocator(path, false);
    return new BPlusTreeMetadata(allocator, keySchema, order);
}

private LeafNode getEmptyLeaf(BPlusTreeMetadata meta,
    Optional<Integer> rightSibling) {
    List<DataBox> keys = new ArrayList<>();
    List<RecordId> rids = new ArrayList<>();
    return new LeafNode(meta, keys, rids, rightSibling);
}

// Tests //////////////////////////////////////
@Test
public void testGet() throws IOException {
    BPlusTreeMetadata meta = getBPlusTreeMetadata(Type.intType(), 5);
    LeafNode leaf = getEmptyLeaf(meta, Optional.empty());
    for (int i = 0; i < 10; ++i) {
        assertEquals(leaf, leaf.get(new IntDataBox(i)));
    }
}

@Test
public void testGetLeftmostLeaf() throws IOException {
    BPlusTreeMetadata meta = getBPlusTreeMetadata(Type.intType(), 5);
    LeafNode leaf = getEmptyLeaf(meta, Optional.empty());
    assertEquals(leaf, leaf.getLeftmostLeaf());
}

@Test
public void testNoOverflowPuts() throws BPlusTreeException, IOException {
    int d = 5;
    BPlusTreeMetadata meta = getBPlusTreeMetadata(Type.intType(), d);
    LeafNode leaf = getEmptyLeaf(meta, Optional.empty());

    for (int i = 0; i < 2 * d; ++i) {
        DataBox key = new IntDataBox(i);
        RecordId rid = new RecordId(i, (short) i);
        assertEquals(Optional.empty(), leaf.put(key, rid));

        for (int j = 0; j <= i; ++j) {
            key = new IntDataBox(j);
            rid = new RecordId(j, (short) j);
            assertEquals(Optional.of(rid), leaf.getKey(key));
        }
    }
}

// HIDDEN
@Test
public void testNoOverflowOutOfOrderPuts()
    throws BPlusTreeException, IOException {
    int d = 2;
    BPlusTreeMetadata meta = getBPlusTreeMetadata(Type.intType(), d);
    LeafNode leaf = getEmptyLeaf(meta, Optional.empty());

    assertEquals(Optional.empty(), leaf.put(d3, r3));
}

```

```

116 assertEquals(Optional.empty(), leaf.put(d1, r1));
117 assertEquals(Optional.empty(), leaf.put(d2, r2));
118 assertEquals(Optional.empty(), leaf.put(d0, r0));
119
120 for (int i = 0; i < 2*d; ++i) {
121     IntDataBox key = new IntDataBox(i);
122     RecordId rid = new RecordId(i, (short) i);
123     assertEquals(Optional.of(rid), leaf.getKey(key));
124 }
125
126 @Test
127 public void testNoOverflowPutsFromDisk()
128     throws BPlusTreeException, IOException {
129     int d = 5;
130     BPlusTreeMetadata meta = getBPlusTreeMetadata(Type.intType(), d);
131     LeafNode leaf = getEmptyLeaf(meta, Optional.empty());
132
133     // Populate the leaf.
134     for (int i = 0; i < 2 * d; ++i) {
135         leaf.put(new IntDataBox(i), new RecordId(i, (short) i));
136     }
137
138     // Then read the leaf from disk.
139     int pageNum = leaf.getPage().getPageNum();
140     LeafNode fromDisk = LeafNode.fromBytes(meta, pageNum);
141
142     // Check to see that we can read from disk.
143     for (int i = 0; i < 2 * d; ++i) {
144         IntDataBox key = new IntDataBox(i);
145         RecordId rid = new RecordId(i, (short) i);
146         assertEquals(Optional.of(rid), fromDisk.getKey(key));
147     }
148 }
149
150 @Test(expected = BPlusTreeException.class)
151 public void testDuplicatePut() throws BPlusTreeException, IOException {
152     BPlusTreeMetadata meta = getBPlusTreeMetadata(Type.intType(), 4);
153     LeafNode leaf = getEmptyLeaf(meta, Optional.empty());
154
155     // The initial insert is fine.
156     leaf.put(new IntDataBox(0), new RecordId(0, (short) 0));
157
158     // The duplicate insert should raise an exception.
159     leaf.put(new IntDataBox(0), new RecordId(0, (short) 0));
160 }
161
162 // HIDDEN
163 @Test
164 public void testOverflowPuts() throws BPlusTreeException, IOException {
165     int d = 2;
166     BPlusTreeMetadata meta = getBPlusTreeMetadata(Type.intType(), d);
167     LeafNode left = getEmptyLeaf(meta, Optional.empty());
168
169     // Fill the left up completely.
170     //
171     // left
172     // +-----+-----+-----+-----+
173     // | 0:(0,0) | 1:(1,1) | 2:(2,2) | 3:(3,3) |
174     // +-----+-----+-----+-----+
175     for (int i = 0; i < 2 * d; ++i) {
176         DataBox key = new IntDataBox(i);
177         RecordId rid = new RecordId(i, (short) i);
178         assertEquals(Optional.empty(), left.put(key, rid));
179     }
180
181     // Overflow the left and split:
182     //
183     // left                right

```

```

184 // +-----+-----+-----+-----+
185 // | 0:(0,0) | 1:(1,1) | | 2:(2,2) | 3:(3,3) | 4:(4,4) |
186 // +-----+-----+-----+-----+
187 DataBox key = new IntDataBox(2*d);
188 RecordId rid = new RecordId(2*d, (short) (2*d));
189 Optional<Pair<DataBox, Integer>> o = left.put(key, rid);
190
191 assertTrue(o.isPresent());
192 Pair<DataBox, Integer> p = o.get();
193 DataBox splitKey = p.getFirst();
194 int rightPageNum = p.getSecond();
195
196 // Load the right child.
197 Page rightPage = meta.getAllocator().fetchPage(rightPageNum);
198 LeafNode right = LeafNode.fromBytes(meta, rightPageNum);
199
200 // Check everything.
201 assertEquals(new IntDataBox(2), splitKey);
202
203 assertEquals(Optional.of(right), left.getRightSibling());
204 assertEquals(Arrays.asList(d0, d1), left.getKeys());
205 assertEquals(Arrays.asList(r0, r1), left.getRids());
206
207 assertEquals(Optional.empty(), right.getRightSibling());
208 assertEquals(Arrays.asList(d2, d3, d4), right.getKeys());
209 assertEquals(Arrays.asList(r2, r3, r4), right.getRids());
210
211 // Make sure our left changes persisted on disk.
212 int leftPageNum = left.getPage().getPageNum();
213 LeafNode leftFromDisk = LeafNode.fromBytes(meta, leftPageNum);
214
215 assertEquals(Optional.of(right), leftFromDisk.getRightSibling());
216 assertEquals(Arrays.asList(d0, d1), leftFromDisk.getKeys());
217 assertEquals(Arrays.asList(r0, r1), leftFromDisk.getRids());
218 }
219
220 @Test
221 public void testSimpleRemoves() throws BPlusTreeException, IOException {
222     int d = 5;
223     BPlusTreeMetadata meta = getBPlusTreeMetadata(Type.intType(), d);
224     LeafNode leaf = getEmptyLeaf(meta, Optional.empty());
225
226     // Insert entries.
227     for (int i = 0; i < 2 * d; ++i) {
228         IntDataBox key = new IntDataBox(i);
229         RecordId rid = new RecordId(i, (short) i);
230         leaf.put(key, rid);
231         assertEquals(Optional.of(rid), leaf.getKey(key));
232     }
233
234     // Remove entries.
235     for (int i = 0; i < 2 * d; ++i) {
236         IntDataBox key = new IntDataBox(i);
237         leaf.remove(key);
238         assertEquals(Optional.empty(), leaf.getKey(key));
239     }
240 }
241
242 // HIDDEN
243 @Test
244 public void testOutOfOrderRemoves() throws BPlusTreeException, IOException {
245     int d = 5;
246     BPlusTreeMetadata meta = getBPlusTreeMetadata(Type.intType(), d);
247     LeafNode leaf = getEmptyLeaf(meta, Optional.empty());
248
249     List<DataBox> keys = new ArrayList<>();
250     List<RecordId> rids = new ArrayList<>();
251     for (int i = 0; i < 2 * d; ++i) {
252         keys.add(new IntDataBox(i));

```

```

117         rids.add(new RecordId(i, (short) i));
118     }
119
120     // Insert entries in random order.
121     Collections.shuffle(keys, new Random(42));
122     Collections.shuffle(rids, new Random(42));
123     for (int i = 0; i < 2 * d; ++i) {
124         assertEquals(Optional.empty(), leaf.put(keys.get(i), rids.get(i)));
125     }
126
127     // Remove entries in random order.
128     Collections.shuffle(keys, new Random(42));
129     Collections.shuffle(rids, new Random(42));
130     for (int i = 0; i < 2 * d; ++i) {
131         leaf.remove(keys.get(i));
132         assertEquals(Optional.empty(), leaf.getKey(keys.get(i)));
133     }
134 }
135
136 // HIDDEN
137 @Test
138 public void testAbsentRemoves() throws BPlusTreeException, IOException {
139     int d = 5;
140     BPlusTreeMetadata meta = getBPlusTreeMetadata(Type.intType(), d);
141     LeafNode leaf = getEmptyLeaf(meta, Optional.empty());
142
143     // Removing absent keys is ok; it doesn't throw an exception.
144     for (int i = 0; i < 2 * d; ++i) {
145         IntDataBox key = new IntDataBox(i);
146         leaf.remove(key);
147         assertEquals(Optional.empty(), leaf.getKey(key));
148     }
149 }
150
151 @Test
152 public void testScanAll() throws BPlusTreeException, IOException {
153     int d = 5;
154     BPlusTreeMetadata meta = getBPlusTreeMetadata(Type.intType(), d);
155     LeafNode leaf = getEmptyLeaf(meta, Optional.empty());
156
157     // Insert tuples in reverse order to make sure that scanAll is returning
158     // things in sorted order.
159     for (int i = 2 * d - 1; i >= 0; --i) {
160         leaf.put(new IntDataBox(i), new RecordId(i, (short) i));
161     }
162
163     Iterator<RecordId> iter = leaf.scanAll();
164     for (int i = 0; i < 2 * d; ++i) {
165         assertTrue(iter.hasNext());
166         assertEquals(new RecordId(i, (short) i), iter.next());
167     }
168     assertFalse(iter.hasNext());
169 }
170
171 @Test
172 public void testScanGreaterEqual() throws BPlusTreeException, IOException {
173     int d = 5;
174     BPlusTreeMetadata meta = getBPlusTreeMetadata(Type.intType(), d);
175     LeafNode leaf = getEmptyLeaf(meta, Optional.empty());
176
177     // Insert tuples in reverse order to make sure that scanAll is returning
178     // things in sorted order.
179     for (int i = 2 * d - 1; i >= 0; --i) {
180         leaf.put(new IntDataBox(i), new RecordId(i, (short) i));
181     }
182
183     Iterator<RecordId> iter = leaf.scanGreaterEqual(new IntDataBox(5));
184     for (int i = 5; i < 2 * d; ++i) {
185         assertTrue(iter.hasNext());

```

```

122     assertEquals(new RecordId(i, (short) i), iter.next());
123 }
124 assertEquals(iter.hasNext());
125 }
126
127 @Test
128 public void testMaxOrder() {
129     // Note that this white box test depend critically on the implementation
130     // of toBytes and includes a lot of magic numbers that won't make sense
131     // unless you read toBytes.
132     assertEquals(4, Type.intType().getSizeInBytes());
133     assertEquals(6, RecordId.getSizeInBytes());
134     for (int d = 0; d < 10; ++d) {
135         int dd = d + 1;
136         for (int i = 9 + (2*d) * (4+6); i < 9 + (2*dd) * (4+6); ++i) {
137             assertEquals(d, LeafNode.maxOrder(i, Type.intType()));
138         }
139     }
140 }
141
142 @Test
143 public void testToSexp() throws BPlusTreeException, IOException {
144     int d = 2;
145     BPlusTreeMetadata meta = getBPlusTreeMetadata(Type.intType(), d);
146     LeafNode leaf = getEmptyLeaf(meta, Optional.empty());
147
148     assertEquals("()", leaf.toSexp());
149     leaf.put(new IntDataBox(4), new RecordId(4, (short) 4));
150     assertEquals("((4 (4 4)))", leaf.toSexp());
151     leaf.put(new IntDataBox(1), new RecordId(1, (short) 1));
152     assertEquals("((1 (1 1)) (4 (4 4)))", leaf.toSexp());
153     leaf.put(new IntDataBox(2), new RecordId(2, (short) 2));
154     assertEquals("((1 (1 1)) (2 (2 2)) (4 (4 4)))", leaf.toSexp());
155     leaf.put(new IntDataBox(3), new RecordId(3, (short) 3));
156     assertEquals("((1 (1 1)) (2 (2 2)) (3 (3 3)) (4 (4 4)))", leaf.toSexp());
157 }
158
159 @Test
160 public void testToAndFromBytes() throws BPlusTreeException, IOException {
161     int d = 5;
162     BPlusTreeMetadata meta = getBPlusTreeMetadata(Type.intType(), d);
163
164     List<DataBox> keys = new ArrayList<>();
165     List<RecordId> rids = new ArrayList<>();
166     LeafNode leaf = new LeafNode(meta, keys, rids, Optional.of(42));
167     int pageNum = leaf.getPage().getPageNum();
168
169     assertEquals(leaf, LeafNode.fromBytes(meta, pageNum));
170
171     for (int i = 0; i < 2 * d; ++i) {
172         leaf.put(new IntDataBox(i), new RecordId(i, (short) i));
173         assertEquals(leaf, LeafNode.fromBytes(meta, pageNum));
174     }
175 }
176 }

```