dreamlegends / DB2

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

```java
package index;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;

import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.Iterator;
import java.util.List;
import java.util.Optional;
import java.util.Random;

import org.junit.Rule;
import org.junit.Test;
import org.junit.rules.DisableOnDebug;
import org.junit.rules.TemporaryFolder;
import org.junit.rules.TestRule;
import org.junit.rules.Timeout;

import common.Pair;
import databox.DataBox;
import databox.IntDataBox;
import databox.Type;
import io.Page;
import io.PageAllocator;
import table.RecordId;

public class TestLeafNode {
    public static final String testFile = "TestLeafNode";

    @Rule
    public TemporaryFolder tempFolder = new TemporaryFolder();

    // 1 second max per method tested.
    @Rule
    public TestRule globalTimeout = new DisableOnDebug(Timeout.seconds(1));

    private static DataBox d0 = new IntDataBox(0);
    private static DataBox d1 = new IntDataBox(1);
    private static DataBox d2 = new IntDataBox(2);
    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));
```

```
    assertEquals(Optional.empty(), leaf.put(d1, r1));
    assertEquals(Optional.empty(), leaf.put(d2, r2));
    assertEquals(Optional.empty(), leaf.put(d0, r0));

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

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

    // Populate the leaf.
    for (int i = 0; i < 2 * d; ++i) {
      leaf.put(new IntDataBox(i), new RecordId(i, (short) i));
    }

    // Then read the leaf from disk.
    int pageNum = leaf.getPage().getPageNum();
    LeafNode fromDisk = LeafNode.fromBytes(meta, pageNum);

    // Check to see that we can read from disk.
    for (int i = 0; i < 2 * d; ++i) {
      IntDataBox key = new IntDataBox(i);
      RecordId rid = new RecordId(i, (short) i);
      assertEquals(Optional.of(rid), fromDisk.getKey(key));
    }
  }

  @Test(expected = BPlusTreeException.class)
  public void testDuplicatePut() throws BPlusTreeException, IOException {
    BPlusTreeMetadata meta = getBPlusTreeMetadata(Type.intType(), 4);
    LeafNode leaf = getEmptyLeaf(meta, Optional.empty());

    // The initial insert is fine.
    leaf.put(new IntDataBox(0), new RecordId(0, (short) 0));

    // The duplicate insert should raise an exception.
    leaf.put(new IntDataBox(0), new RecordId(0, (short) 0));
  }

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

    // Fill the left up completely.
    //
    //   left
    //   +---------+---------+---------+---------+
    //   | 0:(0,0) | 1:(1,1) | 2:(2,2) | 3:(3,3) |
    //   +---------+---------+---------+---------+
    for (int i = 0; i < 2 * d; ++i) {
      DataBox key = new IntDataBox(i);
      RecordId rid = new RecordId(i, (short) i);
      assertEquals(Optional.empty(), left.put(key, rid));
    }

    // Overflow the left and split:
    //
    //   left                  right
    //   +---------+---------+ +---------+---------+---------+
    //   | 0:(0,0) | 1:(1,1) | | 2:(2,2) | 3:(3,3) | 4:(4,4) |
    //   +---------+---------+ +---------+---------+---------+
    DataBox key = new IntDataBox(2*d);
    RecordId rid = new RecordId(2*d, (short) (2*d));
    Optional<Pair<DataBox, Integer>> o = left.put(key, rid);

    assertTrue(o.isPresent());
    Pair<DataBox, Integer> p = o.get();
    DataBox splitKey = p.getFirst();
    int rightPageNum = p.getSecond();

    // Load the right child.
    Page rightPage = meta.getAllocator().fetchPage(rightPageNum);
    LeafNode right = LeafNode.fromBytes(meta, rightPageNum);

    // Check everything.
    assertEquals(new IntDataBox(2), splitKey);

    assertEquals(Optional.of(right), left.getRightSibling());
    assertEquals(Arrays.asList(d0, d1), left.getKeys());
    assertEquals(Arrays.asList(r0, r1), left.getRids());

    assertEquals(Optional.empty(), right.getRightSibling());
    assertEquals(Arrays.asList(d2, d3, d4), right.getKeys());
    assertEquals(Arrays.asList(r2, r3, r4), right.getRids());

    // Make sure our left changes persisted on disk.
    int leftPageNum = left.getPage().getPageNum();
    LeafNode leftFromDisk = LeafNode.fromBytes(meta, leftPageNum);

    assertEquals(Optional.of(right), leftFromDisk.getRightSibling());
    assertEquals(Arrays.asList(d0, d1), leftFromDisk.getKeys());
    assertEquals(Arrays.asList(r0, r1), leftFromDisk.getRids());
  }

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

    // Insert entries.
    for (int i = 0; i < 2 * d; ++i) {
      IntDataBox key = new IntDataBox(i);
      RecordId rid = new RecordId(i, (short) i);
      leaf.put(key, rid);
      assertEquals(Optional.of(rid), leaf.getKey(key));
    }

    // Remove entries.
    for (int i = 0; i < 2 * d; ++i) {
      IntDataBox key = new IntDataBox(i);
      leaf.remove(key);
      assertEquals(Optional.empty(), leaf.getKey(key));
    }
  }

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

    List<DataBox> keys = new ArrayList<>();
    List<RecordId> rids = new ArrayList<>();
    for (int i = 0; i < 2 * d; ++i) {
      keys.add(new IntDataBox(i));
```

```java
      rids.add(new RecordId(i, (short) i));
    }

    // Insert entries in random order.
    Collections.shuffle(keys, new Random(42));
    Collections.shuffle(rids, new Random(42));
    for (int i = 0; i < 2 * d; ++i) {
      assertEquals(Optional.empty(), leaf.put(keys.get(i), rids.get(i)));
    }

    // Remove entries in random order.
    Collections.shuffle(keys, new Random(42));
    Collections.shuffle(rids, new Random(42));
    for (int i = 0; i < 2 * d; ++i) {
      leaf.remove(keys.get(i));
      assertEquals(Optional.empty(), leaf.getKey(keys.get(i)));
    }
  }

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

    // Removing absent keys is ok; it doesn't throw an exception.
    for (int i = 0; i < 2 * d; ++i) {
      IntDataBox key = new IntDataBox(i);
      leaf.remove(key);
      assertEquals(Optional.empty(), leaf.getKey(key));
    }
  }

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

    // Insert tuples in reverse order to make sure that scanAll is returning
    // things in sorted order.
    for (int i = 2 * d - 1; i >= 0; --i) {
      leaf.put(new IntDataBox(i), new RecordId(i, (short) i));
    }

    Iterator<RecordId> iter = leaf.scanAll();
    for (int i = 0; i < 2 * d; ++i) {
      assertTrue(iter.hasNext());
      assertEquals(new RecordId(i, (short) i), iter.next());
    }
    assertFalse(iter.hasNext());
  }

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

    // Insert tuples in reverse order to make sure that scanAll is returning
    // things in sorted order.
    for (int i = 2 * d - 1; i >= 0; --i) {
      leaf.put(new IntDataBox(i), new RecordId(i, (short) i));
    }

    Iterator<RecordId> iter = leaf.scanGreaterEqual(new IntDataBox(5));
    for (int i = 5; i < 2 * d; ++i) {
      assertTrue(iter.hasNext());
```

```java
            assertEquals(new RecordId(i, (short) i), iter.next());
        }
        assertFalse(iter.hasNext());
    }

    @Test
    public void testMaxOrder() {
        // Note that this white box test depend critically on the implementation
        // of toBytes and includes a lot of magic numbers that won't make sense
        // unless you read toBytes.
        assertEquals(4, Type.intType().getSizeInBytes());
        assertEquals(6, RecordId.getSizeInBytes());
        for (int d = 0; d < 10; ++d) {
            int dd = d + 1;
            for (int i = 9 + (2*d) * (4+6); i < 9 + (2*dd) * (4+6); ++i) {
                assertEquals(d, LeafNode.maxOrder(i, Type.intType()));
            }
        }
    }

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

        assertEquals("()", leaf.toSexp());
        leaf.put(new IntDataBox(4), new RecordId(4, (short) 4));
        assertEquals("((4 (4 4)))", leaf.toSexp());
        leaf.put(new IntDataBox(1), new RecordId(1, (short) 1));
        assertEquals("((1 (1 1)) (4 (4 4)))", leaf.toSexp());
        leaf.put(new IntDataBox(2), new RecordId(2, (short) 2));
        assertEquals("((1 (1 1)) (2 (2 2)) (4 (4 4)))", leaf.toSexp());
        leaf.put(new IntDataBox(3), new RecordId(3, (short) 3));
        assertEquals("((1 (1 1)) (2 (2 2)) (3 (3 3)) (4 (4 4)))", leaf.toSexp());
    }

    @Test
    public void testToAndFromBytes() throws BPlusTreeException, IOException {
        int d = 5;
        BPlusTreeMetadata meta = getBPlusTreeMetadata(Type.intType(), d);

        List<DataBox> keys = new ArrayList<>();
        List<RecordId> rids = new ArrayList<>();
        LeafNode leaf = new LeafNode(meta, keys, rids, Optional.of(42));
        int pageNum = leaf.getPage().getPageNum();

        assertEquals(leaf, LeafNode.fromBytes(meta, pageNum));

        for (int i = 0; i < 2 * d; ++i) {
            leaf.put(new IntDataBox(i), new RecordId(i, (short) i));
            assertEquals(leaf, LeafNode.fromBytes(meta, pageNum));
        }
    }
}
```