dreamlegends / DB2

Branch: master DB2 / src / test / java / index / TestInnerNode.java

Find file Copy path

dreamlegends init add all files

2c520b8 26 days ago

1 contributor

475 lines (409 sloc) 17.2 KB

Raw Blame History

```java
package index;

import static org.junit.Assert.assertEquals;
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.List;
import java.util.Optional;

import org.junit.Before;
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.PageAllocator;
import table.RecordId;

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

    // The page allocator associated with the TestInnerNode file. See
    // resetMembers for initialization.
    PageAllocator allocator;

    @Rule
    public TemporaryFolder tempFolder = new TemporaryFolder();

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

    // inner, leaf0, leaf1, and leaf2 collectively form the following B+ tree:
    //
    //                               inner
    //                               +----+----+----+----+
//                                | 10 | 20 |    |    |
//                                +----+----+----+----+
//                               /     |     \
//                          ____/      |      \____
//                         /           |           \
//   +----+----+----+----+  +----+----+----+----+  +----+----+----+----+
//   |  1 |  2 |  3 |    |  | 11 | 12 | 13 |    |  | 21 | 22 | 23 |    |
//   +----+----+----+----+  +----+----+----+----+  +----+----+----+----+
//   leaf0                  leaf1                  leaf2
//
// innerKeys, innerChildren, keys0, rids0, keys1, rids1, keys2, and rids2
// hold *copies* of the contents of the nodes. To test out a certain method
// of a tree---for example, put---we can issue a put against the tree,
// update one of innerKeys, innerChildren, keys{0,1,2}, or rids{0,1,2}, and
// then check that the contents of the tree match our expectations. For
// example:
//
//   IntDataBox key = new IntDataBox(4);
//   RecordId rid = new RecordId(4, (short) 4);
//   inner.put(key, rid);
//
//   // (4, (4, 4)) is added to leaf 0, so we update keys0 and rids0 and
//   // check that it matches the contents of leaf0.
//   keys0.add(key);
//   rids0.add(rid);
//   assertEquals(keys0, getLeaf(leaf0).getKeys());
//   assertEquals(rids0, getLeaf(leaf0).getRids());
//
//   // Leaf 1 should be unchanged which we can check:
//   assertEquals(keys1, getLeaf(leaf1).getKeys());
//   assertEquals(rids1, getLeaf(leaf1).getRids());
//
//   // Writing all these assertEquals is boilerplate, so we can abstract
//   // it in checkTreeMatchesExpectations().
//   checkTreeMatchesExpectations();
//
// Note that we cannot simply store the LeafNodes as members because when
// we call something like inner.put(k), the inner node constructs a new
// LeafNode from the serialization and forwards the put to that. It would
// not affect our the in-memory values of our members. Also note that all
// of these members are initialized by resetMembers before every test case
// is run.
private List<DataBox> innerKeys;
private List<Integer> innerChildren;
private InnerNode inner;
private List<DataBox> keys0;
private List<RecordId> rids0;
private int leaf0;
private List<DataBox> keys1;
private List<RecordId> rids1;
private int leaf1;
private List<DataBox> keys2;
private List<RecordId> rids2;
private int leaf2;

// See comment above.
@Before
public void resetMembers() throws IOException {
  File file = tempFolder.newFile(testFile);
  this.allocator = new PageAllocator(file.getAbsolutePath(), false);

  BPlusTreeMetadata meta = getBPlusTreeMetadata(Type.intType(), 2);

  // Leaf 2
  List<DataBox> keys2 = new ArrayList<>();
  keys2.add(new IntDataBox(21));
  keys2.add(new IntDataBox(22));
  keys2.add(new IntDataBox(23));
  List<RecordId> rids2 = new ArrayList<>();
```

```java
    rids2.add(new RecordId(21, (short) 21));
    rids2.add(new RecordId(22, (short) 22));
    rids2.add(new RecordId(23, (short) 23));
    Optional<Integer> sibling2 = Optional.empty();
    LeafNode leaf2 = new LeafNode(meta, keys2, rids2, sibling2);

    this.keys2 = new ArrayList<>(keys2);
    this.rids2 = new ArrayList<>(rids2);
    this.leaf2 = leaf2.getPage().getPageNum();

    // Leaf 1
    keys1 = new ArrayList<>();
    keys1.add(new IntDataBox(11));
    keys1.add(new IntDataBox(12));
    keys1.add(new IntDataBox(13));
    rids1 = new ArrayList<>();
    rids1.add(new RecordId(11, (short) 11));
    rids1.add(new RecordId(12, (short) 12));
    rids1.add(new RecordId(13, (short) 13));
    Optional<Integer> sibling1 = Optional.of(leaf2.getPage().getPageNum());
    LeafNode leaf1 = new LeafNode(meta, keys1, rids1, sibling1);

    this.keys1 = new ArrayList<>(keys1);
    this.rids1 = new ArrayList<>(rids1);
    this.leaf1 = leaf1.getPage().getPageNum();

    // Leaf 0
    List<DataBox> keys0 = new ArrayList<>();
    keys0.add(new IntDataBox(1));
    keys0.add(new IntDataBox(2));
    keys0.add(new IntDataBox(3));
    List<RecordId> rids0 = new ArrayList<>();
    rids0.add(new RecordId(1, (short) 1));
    rids0.add(new RecordId(2, (short) 2));
    rids0.add(new RecordId(3, (short) 3));
    Optional<Integer> sibling0 = Optional.of(leaf1.getPage().getPageNum());
    LeafNode leaf0 = new LeafNode(meta, keys0, rids0, sibling0);
    this.keys0 = new ArrayList<>(keys0);
    this.rids0 = new ArrayList<>(rids0);
    this.leaf0 = leaf0.getPage().getPageNum();

    // Inner node
    List<DataBox> innerKeys = new ArrayList<>();
    innerKeys.add(new IntDataBox(10));
    innerKeys.add(new IntDataBox(20));

    List<Integer> innerChildren = new ArrayList<>();
    innerChildren.add(this.leaf0);
    innerChildren.add(this.leaf1);
    innerChildren.add(this.leaf2);

    this.innerKeys = new ArrayList<>(innerKeys);
    this.innerChildren = new ArrayList<>(innerChildren);
    this.inner = new InnerNode(meta, innerKeys, innerChildren);
  }

  // See comment above.
  private LeafNode getLeaf(int pageNum) throws IOException {
    BPlusTreeMetadata meta = getBPlusTreeMetadata(Type.intType(), 2);
    return LeafNode.fromBytes(meta, pageNum);
  }

  // See comment above.
  private void checkTreeMatchesExpectations() throws IOException {
    LeafNode leaf0 = getLeaf(this.leaf0);
    LeafNode leaf1 = getLeaf(this.leaf1);
    LeafNode leaf2 = getLeaf(this.leaf2);

    assertEquals(keys0, leaf0.getKeys());
```

```java
        assertEquals(rids0, leaf0.getRids());
        assertEquals(keys1, leaf1.getKeys());
        assertEquals(rids1, leaf1.getRids());
        assertEquals(keys2, leaf2.getKeys());
        assertEquals(rids2, leaf2.getRids());
        assertEquals(innerKeys, inner.getKeys());
        assertEquals(innerChildren, inner.getChildren());
    }

    private BPlusTreeMetadata getBPlusTreeMetadata(Type keySchema, int order)
            throws IOException {
        return new BPlusTreeMetadata(allocator, keySchema, order);
    }

    // Tests //////////////////////////////////////////////////////////////////
    @Test
    public void testGet() throws IOException {
        LeafNode leaf0 = getLeaf(this.leaf0);
        for (int i = 0; i < 10; ++i) {
            assertEquals(leaf0, inner.get(new IntDataBox(i)));
        }

        LeafNode leaf1 = getLeaf(this.leaf1);
        for (int i = 10; i < 20; ++i) {
            assertEquals(leaf1, inner.get(new IntDataBox(i)));
        }

        LeafNode leaf2 = getLeaf(this.leaf2);
        for (int i = 20; i < 30; ++i) {
            assertEquals(leaf2, inner.get(new IntDataBox(i)));
        }
    }

    @Test
    public void testGetLeftmostLeaf() throws IOException {
        assertEquals(getLeaf(leaf0), inner.getLeftmostLeaf());
    }

    @Test
    public void testNoOverflowPuts() throws BPlusTreeException, IOException {
        IntDataBox key = null;
        RecordId rid = null;

        // Add to leaf 0.
        key = new IntDataBox(0);
        rid = new RecordId(0, (short) 0);
        assertEquals(Optional.empty(), inner.put(key, rid));
        keys0.add(0, key);
        rids0.add(0, rid);
        checkTreeMatchesExpectations();

        // Add to leaf 1.
        key = new IntDataBox(14);
        rid = new RecordId(14, (short) 14);
        assertEquals(Optional.empty(), inner.put(key, rid));
        keys1.add(3, key);
        rids1.add(3, rid);
        checkTreeMatchesExpectations();

        // Add to leaf 2.
        key = new IntDataBox(20);
        rid = new RecordId(20, (short) 20);
        assertEquals(Optional.empty(), inner.put(key, rid));
        keys2.add(0, key);
        rids2.add(0, rid);
        checkTreeMatchesExpectations();
    }

    // HIDDEN
```

```java
@Test
public void testOverflowPuts() throws BPlusTreeException, IOException {
  // Overflow the first leaf. The tree look like this:
  //
  //   (3, 10, 20)
  //   (1, 2) (3, 4, 5) (11, 12, 13) (21, 22, 23)
  //          a
  inner.put(new IntDataBox(4), new RecordId(4, (short) 4));
  inner.put(new IntDataBox(5), new RecordId(5, (short) 5));

  int leafa =
    getLeaf(this.leaf0).getRightSibling().get().getPage().getPageNum();
  innerKeys.add(0, new IntDataBox(3));
  innerChildren.add(1, leafa);
  assertEquals(innerKeys, inner.getKeys());
  assertEquals(innerChildren, inner.getChildren());

  // Overflow the second leaf.
  //
  //   (3, 5, 10, 20)
  //   (1, 2) (3, 4) (5, 6, 7) (11, 12, 13) (21, 22, 23)
  //          a      b
  inner.put(new IntDataBox(6), new RecordId(6, (short) 6));
  inner.put(new IntDataBox(7), new RecordId(7, (short) 7));

  int leafb = getLeaf(leafa).getRightSibling().get().getPage().getPageNum();
  innerKeys.add(1, new IntDataBox(5));
  innerChildren.add(2, leafb);
  assertEquals(innerKeys, inner.getKeys());
  assertEquals(innerChildren, inner.getChildren());

  // Again! This one overflows the index too.
  //   (7)
  //   (3, 5) (10, 20)
  //   (1, 2) (3, 4) (5, 6) (7, 8, 9) (11, 12, 13) (21, 22, 23)
  //          a      b      c
  inner.put(new IntDataBox(8), new RecordId(8, (short) 8));
  Optional<Pair<DataBox, Integer>> o =
    inner.put(new IntDataBox(9), new RecordId(9, (short) 9));

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

  assertEquals(new IntDataBox(7), splitKey);

  innerKeys = innerKeys.subList(0, 2);
  innerChildren = innerChildren.subList(0, 3);
  assertEquals(innerKeys, inner.getKeys());
  assertEquals(innerChildren, inner.getChildren());

  BPlusTreeMetadata meta = getBPlusTreeMetadata(Type.intType(), 2);
  InnerNode newInner = InnerNode.fromBytes(meta, newInnerPageNum);

  List<DataBox> newInnerKeys = new ArrayList<>();
  newInnerKeys.add(new IntDataBox(10));
  newInnerKeys.add(new IntDataBox(20));

  List<Integer> newInnerChildren = new ArrayList<>();
  int leafc = getLeaf(leafb).getRightSibling().get().getPage().getPageNum();
  newInnerChildren.add(leafc);
  newInnerChildren.add(this.leaf1);
  newInnerChildren.add(this.leaf2);

  assertEquals(newInnerKeys, newInner.getKeys());
  assertEquals(newInnerChildren, newInner.getChildren());

  // Make sure we can read inner from disk.
```

```java
    int innerPageNum = inner.getPage().getPageNum();
    InnerNode innerFromDisk = InnerNode.fromBytes(meta, innerPageNum);
    assertEquals(innerKeys, innerFromDisk.getKeys());
    assertEquals(innerChildren, innerFromDisk.getChildren());
  }

  @Test
  public void testRemove() throws IOException {
    // Remove from leaf 0.
    inner.remove(new IntDataBox(1));
    keys0.remove(0);
    rids0.remove(0);
    checkTreeMatchesExpectations();

    inner.remove(new IntDataBox(3));
    keys0.remove(1);
    rids0.remove(1);
    checkTreeMatchesExpectations();

    inner.remove(new IntDataBox(2));
    keys0.remove(0);
    rids0.remove(0);
    checkTreeMatchesExpectations();

    // Remove from leaf 1.
    inner.remove(new IntDataBox(11));
    keys1.remove(0);
    rids1.remove(0);
    checkTreeMatchesExpectations();

    inner.remove(new IntDataBox(13));
    keys1.remove(1);
    rids1.remove(1);
    checkTreeMatchesExpectations();

    inner.remove(new IntDataBox(12));
    keys1.remove(0);
    rids1.remove(0);
    checkTreeMatchesExpectations();

    // Remove from leaf 2.
    inner.remove(new IntDataBox(23));
    keys2.remove(2);
    rids2.remove(2);
    checkTreeMatchesExpectations();

    inner.remove(new IntDataBox(22));
    keys2.remove(1);
    rids2.remove(1);
    checkTreeMatchesExpectations();

    inner.remove(new IntDataBox(21));
    keys2.remove(0);
    rids2.remove(0);
    checkTreeMatchesExpectations();
  }

  @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 = 5 + (2*d*4) + ((2*d+1)*4); i < 5 + (2*dd*4) + ((2*dd+1)*4); ++i) {
        assertEquals(d, InnerNode.maxOrder(i, Type.intType()));
      }
    }
  }

  @Test
  public void testnumLessThanEqual() {
    List<Integer> empty = Arrays.asList();
    assertEquals(0, InnerNode.numLessThanEqual(0, empty));

    List<Integer> contiguous = Arrays.asList(1, 2, 3, 4, 5);
    assertEquals(0, InnerNode.numLessThanEqual(0, contiguous));
    assertEquals(1, InnerNode.numLessThanEqual(1, contiguous));
    assertEquals(2, InnerNode.numLessThanEqual(2, contiguous));
    assertEquals(3, InnerNode.numLessThanEqual(3, contiguous));
    assertEquals(4, InnerNode.numLessThanEqual(4, contiguous));
    assertEquals(5, InnerNode.numLessThanEqual(5, contiguous));
    assertEquals(5, InnerNode.numLessThanEqual(6, contiguous));
    assertEquals(5, InnerNode.numLessThanEqual(7, contiguous));

    List<Integer> sparseWithDuplicates = Arrays.asList(1, 3, 3, 3, 5);
    assertEquals(0, InnerNode.numLessThanEqual(0, sparseWithDuplicates));
    assertEquals(1, InnerNode.numLessThanEqual(1, sparseWithDuplicates));
    assertEquals(1, InnerNode.numLessThanEqual(2, sparseWithDuplicates));
    assertEquals(4, InnerNode.numLessThanEqual(3, sparseWithDuplicates));
    assertEquals(4, InnerNode.numLessThanEqual(4, sparseWithDuplicates));
    assertEquals(5, InnerNode.numLessThanEqual(5, sparseWithDuplicates));
    assertEquals(5, InnerNode.numLessThanEqual(6, sparseWithDuplicates));
    assertEquals(5, InnerNode.numLessThanEqual(7, sparseWithDuplicates));
  }

  @Test
  public void testnumLessThan() {
    List<Integer> empty = Arrays.asList();
    assertEquals(0, InnerNode.numLessThanEqual(0, empty));

    List<Integer> contiguous = Arrays.asList(1, 2, 3, 4, 5);
    assertEquals(0, InnerNode.numLessThan(0, contiguous));
    assertEquals(0, InnerNode.numLessThan(1, contiguous));
    assertEquals(1, InnerNode.numLessThan(2, contiguous));
    assertEquals(2, InnerNode.numLessThan(3, contiguous));
    assertEquals(3, InnerNode.numLessThan(4, contiguous));
    assertEquals(4, InnerNode.numLessThan(5, contiguous));
    assertEquals(5, InnerNode.numLessThan(6, contiguous));
    assertEquals(5, InnerNode.numLessThan(7, contiguous));

    List<Integer> sparseWithDuplicates = Arrays.asList(1, 3, 3, 3, 5);
    assertEquals(0, InnerNode.numLessThan(0, sparseWithDuplicates));
    assertEquals(0, InnerNode.numLessThan(1, sparseWithDuplicates));
    assertEquals(1, InnerNode.numLessThan(2, sparseWithDuplicates));
    assertEquals(1, InnerNode.numLessThan(3, sparseWithDuplicates));
    assertEquals(4, InnerNode.numLessThan(4, sparseWithDuplicates));
    assertEquals(4, InnerNode.numLessThan(5, sparseWithDuplicates));
    assertEquals(5, InnerNode.numLessThan(6, sparseWithDuplicates));
    assertEquals(5, InnerNode.numLessThan(7, sparseWithDuplicates));
  }

  @Test
  public void testToSexp() {
    String leaf0 = "((1 (1 1)) (2 (2 2)) (3 (3 3)))";
    String leaf1 = "((11 (11 11)) (12 (12 12)) (13 (13 13)))";
    String leaf2 = "((21 (21 21)) (22 (22 22)) (23 (23 23)))";
    String expected = String.format("(%s 10 %s 20 %s)", leaf0, leaf1, leaf2);
    assertEquals(expected, inner.toSexp());
  }

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

```
      List<DataBox> keys = new ArrayList<>();
      List<Integer> children = new ArrayList<>();
      children.add(42);

      for (int i = 0; i < 2 * d; ++i) {
        keys.add(new IntDataBox(i));
        children.add(i);

        InnerNode inner = new InnerNode(meta, keys, children);
        int pageNum = inner.getPage().getPageNum();
        InnerNode parsed = InnerNode.fromBytes(meta, pageNum);
        assertEquals(inner, parsed);
      }
    }
}
```