Branch: master ▾ DB2 / src / main / java / index / BPlusTree.java

Find file | Copy path

dreamlegends init add all files 2c520b8 7 days ago

1 contributor

395 lines (358 sloc) 14.3 KB

Raw | Blame | History

```
package index;

import java.nio.ByteBuffer;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.NoSuchElementException;
import java.util.Optional;

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

/**
 * A persistent B+ tree.
 * <p>
 * // Create an order 2, integer-valued B+ tree that is persisted in tree.txt.
 * BPlusTree tree = new BPlusTree("tree.txt", Type.intType(), 2);
 * <p>
 * // Insert some values into the tree.
 * tree.put(new IntDataBox(0), new RecordId(0, (short) 0));
 * tree.put(new IntDataBox(1), new RecordId(1, (short) 1));
 * tree.put(new IntDataBox(2), new RecordId(2, (short) 2));
 * <p>
 * // Get some values out of the tree.
 * tree.get(new IntDataBox(0)); // Optional.of(RecordId(0, 0))
 * tree.get(new IntDataBox(1)); // Optional.of(RecordId(1, 1))
 * tree.get(new IntDataBox(2)); // Optional.of(RecordId(2, 2))
 * tree.get(new IntDataBox(3)); // Optional.empty();
 * <p>
 * // Iterate over the record ids in the tree.
 * tree.scanEqual(new IntDataBox(2));        // [(2, 2)]
 * tree.scanAll();                           // [(0, 0), (1, 1), (2, 2)]
 * tree.scanGreaterEqual(new IntDataBox(1)); // [(1, 1), (2, 2)]
 * <p>
 * // Remove some elements from the tree.
 * tree.get(new IntDataBox(0)); // Optional.of(RecordId(0, 0))
 * tree.remove(new IntDataBox(0));
 * tree.get(new IntDataBox(0)); // Optional.empty()
 * <p>
 * // Load the tree from disk.
```

```
 * BPlusTree fromDisk = new BPlusTree("tree.txt");
 * <p>
 * // All the values are still there.
 * fromDisk.get(new IntDataBox(0)); // Optional.empty()
 * fromDisk.get(new IntDataBox(1)); // Optional.of(RecordId(1, 1))
 * fromDisk.get(new IntDataBox(2)); // Optional.of(RecordId(2, 2))
 */
public class BPlusTree {
    public static final String FILENAME_PREFIX = "db";
    public static final String FILENAME_EXTENSION = ".index";

    private BPlusTreeMetadata metadata;
    private Page headerPage;
    private BPlusNode root;

    // Constructors ///////////////////////////////////////////////////////////

    /**
     * Construct a new B+ tree which is serialized into the file `filename`,
     * stores keys of type `keySchema`, and has order `order`. For example,
     * `new BPlusTree("tree.txt", Type.intType(), 2)` constructs a B+ tree that
     * is serialized to "tree.txt", that maps integers to record ids, and that
     * has order 2.
     * <p>
     * If the specified order is so large that a single node cannot fit on a
     * single page, then a BPlusTree exception is thrown. If you want to have
     * maximally full B+ tree nodes, then use the BPlusTree.maxOrder function
     * to get the appropriate order.
     * <p>
     * We reserve the first page (i.e. page number 0) of the file for a header
     * page which contains:
     * <p>
     * - the key schema of the tree,
     * - the order of the tree, and
     * - the page number of the root of the tree.
     * <p>
     * All other pages are serializations of inner and leaf nodes. See
     * writeHeader for details.
     */
    public BPlusTree(String filename, Type keySchema, int order)
            throws BPlusTreeException {
        // Sanity checks.
        if (order < 0) {
            String msg = String.format(
                    "You cannot construct a B+ tree with negative order %d.",
                    order);
            throw new BPlusTreeException(msg);
        }

        int maxOrder = BPlusTree.maxOrder(Page.pageSize, keySchema);
        if (order > maxOrder) {
            String msg = String.format(
                    "You cannot construct a B+ tree with order %d greater than the " +
                            "max order %d.",
                    order, maxOrder);
            throw new BPlusTreeException(msg);
        }

        // Initialize the page allocator.
        PageAllocator allocator = new PageAllocator(filename, true /* wipe */);
        this.metadata = new BPlusTreeMetadata(allocator, keySchema, order);

        // Allocate the header page.
        int headerPageNum = allocator.allocPage();
        assert (headerPageNum == 0);
        this.headerPage = allocator.fetchPage(headerPageNum);

        // Construct the root.
        List<DataBox> keys = new ArrayList<>();
```

```
        List<RecordId> rids = new ArrayList<>();
        Optional<Integer> rightSibling = Optional.empty();
        this.root = new LeafNode(this.metadata, keys, rids, rightSibling);

        // Initialize the header page.
        writeHeader(headerPage.getByteBuffer());
    }

    /**
     * Read a B+ tree that was previously serialized to filename.
     */
    public BPlusTree(String filename) {
        // Initialize the page allocator and fetch the header page.
        PageAllocator allocator = new PageAllocator(filename, false /* wipe */);
        Page headerPage = allocator.fetchPage(0);
        ByteBuffer buf = headerPage.getByteBuffer();

        // Read the contents of the header page. See writeHeader for information
        // on exactly what is written to the header page.
        Type keySchema = Type.fromBytes(buf);
        int order = buf.getInt();
        int rootPageNum = buf.getInt();

        // Initialize members.
        this.metadata = new BPlusTreeMetadata(allocator, keySchema, order);
        this.headerPage = allocator.fetchPage(0);
        this.root = BPlusNode.fromBytes(this.metadata, rootPageNum);
    }

    // Core API ////////////////////////////////////////////////////////////

    /**
     * Returns the value associated with `key`.
     * <p>
     * // Create a B+ tree and insert a single value into it.
     * BPlusTree tree = new BPlusTree("t.txt", Type.intType(), 4);
     * DataBox key = new IntDataBox(42);
     * RecordId rid = new RecordId(0, (short) 0);
     * tree.put(key, rid);
     * <p>
     * // Get the value we put and also try to get a value we never put.
     * tree.get(key);                 // Optional.of(rid)
     * tree.get(new IntDataBox(100)); // Optional.empty()
     */
    public Optional<RecordId> get(DataBox key) {
        typecheck(key);
        throw new UnsupportedOperationException("TODO: implement");
    }

    /**
     * scanEqual(k) is equivalent to get(k) except that it returns an iterator
     * instead of an Optional. That is, if get(k) returns Optional.empty(),
     * then scanEqual(k) returns an empty iterator. If get(k) returns
     * Optional.of(rid) for some rid, then scanEqual(k) returns an iterator
     * over rid.
     */
    public Iterator<RecordId> scanEqual(DataBox key) {
        typecheck(key);
        Optional<RecordId> rid = get(key);
        if (rid.isPresent()) {
            ArrayList<RecordId> l = new ArrayList<>();
            l.add(rid.get());
            return l.iterator();
        } else {
            return new ArrayList<RecordId>().iterator();
        }
    }

    /**
```

```
     * Returns an iterator over all the RecordIds stored in the B+ tree in
     * ascending order of their corresponding keys.
     * <p>
     * // Create a B+ tree and insert some values into it.
     * BPlusTree tree = new BPlusTree("t.txt", Type.intType(), 4);
     * tree.put(new IntDataBox(2), new RecordId(2, (short) 2));
     * tree.put(new IntDataBox(5), new RecordId(5, (short) 5));
     * tree.put(new IntDataBox(4), new RecordId(4, (short) 4));
     * tree.put(new IntDataBox(1), new RecordId(1, (short) 1));
     * tree.put(new IntDataBox(3), new RecordId(3, (short) 3));
     * <p>
     * Iterator<RecordId> iter = tree.scanAll();
     * iter.next(); // RecordId(1, 1)
     * iter.next(); // RecordId(2, 2)
     * iter.next(); // RecordId(3, 3)
     * iter.next(); // RecordId(4, 4)
     * iter.next(); // RecordId(5, 5)
     * iter.next(); // NoSuchElementException
     * <p>
     * Note that you CAN NOT materialize all record ids in memory and then
     * return an iterator over them. Your iterator must lazily scan over the
     * leaves of the B+ tree. Solutions that materialize all record ids in
     * memory will receive 0 points.
     */
    public Iterator<RecordId> scanAll() {
        throw new UnsupportedOperationException("TODO: implement");
    }

    /**
     * Returns an iterator over all the RecordIds stored in the B+ tree that
     * are greater than or equal to `key`. RecordIds are returned in ascending
     * of their corresponding keys.
     * <p>
     * // Create a B+ tree and insert some values into it.
     * BPlusTree tree = new BPlusTree("t.txt", Type.intType(), 4);
     * tree.put(new IntDataBox(2), new RecordId(2, (short) 2));
     * tree.put(new IntDataBox(5), new RecordId(5, (short) 5));
     * tree.put(new IntDataBox(4), new RecordId(4, (short) 4));
     * tree.put(new IntDataBox(1), new RecordId(1, (short) 1));
     * tree.put(new IntDataBox(3), new RecordId(3, (short) 3));
     * <p>
     * Iterator<RecordId> iter = tree.scanGreaterEqual(new IntDataBox(3));
     * iter.next(); // RecordId(3, 3)
     * iter.next(); // RecordId(4, 4)
     * iter.next(); // RecordId(5, 5)
     * iter.next(); // NoSuchElementException
     * <p>
     * Note that you CAN NOT materialize all record ids in memory and then
     * return an iterator over them. Your iterator must lazily scan over the
     * leaves of the B+ tree. Solutions that materialize all record ids in
     * memory will receive 0 points.
     */
    public Iterator<RecordId> scanGreaterEqual(DataBox key) {
        throw new UnsupportedOperationException("TODO: implement");
    }

    /**
     * Inserts a (key, rid) pair into a B+ tree. If the key already exists in
     * the B+ tree, then the pair is not inserted and an exception is raised.
     * <p>
     * BPlusTree tree = new BPlusTree("t.txt", Type.intType(), 4);
     * DataBox key = new IntDataBox(42);
     * RecordId rid = new RecordId(42, (short) 42);
     * tree.put(key, rid); // Sucess :)
     * tree.put(key, rid); // BPlusTreeException :(
     */
    public void put(DataBox key, RecordId rid) throws BPlusTreeException {
        typecheck(key);
        throw new UnsupportedOperationException("TODO: implement");
```

```
    }

    /**
     * Deletes a (key, rid) pair from a B+ tree.
     * <p>
     * BPlusTree tree = new BPlusTree("t.txt", Type.intType(), 4);
     * DataBox key = new IntDataBox(42);
     * RecordId rid = new RecordId(42, (short) 42);
     * <p>
     * tree.put(key, rid);
     * tree.get(key); // Optional.of(rid)
     * tree.remove(key);
     * tree.get(key); // Optional.empty()
     */
    public void remove(DataBox key) {
        typecheck(key);
        throw new UnsupportedOperationException("TODO: implement");
    }

    // Helpers /////////////////////////////////////////////////////////////////

    /**
     * Returns a sexp representation of this tree. See BPlusNode.toSexp for
     * more information.
     */
    public String toSexp() {
        return root.toSexp();
    }

    /**
     * Debugging large B+ trees is hard. To make it a bit easier, we can print
     * out a B+ tree as a DOT file which we can then convert into a nice
     * picture of the B+ tree. tree.toDot() returns the contents of DOT file
     * which illustrates the B+ tree. The details of the file itself is not at
     * all important, just know that if you call tree.toDot() and save the
     * output to a file called tree.dot, then you can run this command
     * <p>
     * dot -T pdf tree.dot -o tree.pdf
     * <p>
     * to create a PDF of the tree.
     */
    public String toDot() {
        List<String> strings = new ArrayList<>();
        strings.add("digraph g {");
        strings.add("  node [shape=record, height=0.1];");
        strings.add(root.toDot());
        strings.add("}");
        return String.join("\n", strings);
    }

    /**
     * Returns the largest number d such that the serialization of a LeafNode
     * with 2d entries and an InnerNode with 2d keys will fit on a single page
     * of size `pageSizeInBytes`.
     */
    public static int maxOrder(int pageSizeInBytes, Type keySchema) {
        int leafOrder = LeafNode.maxOrder(pageSizeInBytes, keySchema);
        int innerOrder = InnerNode.maxOrder(pageSizeInBytes, keySchema);
        return Math.min(leafOrder, innerOrder);
    }

    /**
     * Returns the number of pages used to serialize the tree.
     */
    public int getNumPages() {
        return metadata.getAllocator().getNumPages();
    }
```

```
    /**
     * Serializes the header page to buf.
     */
    private void writeHeader(ByteBuffer buf) {
        buf.put(metadata.getKeySchema().toBytes());
        buf.putInt(metadata.getOrder());
        buf.putInt(root.getPage().getPageNum());
    }

    private void typecheck(DataBox key) {
        Type t = metadata.getKeySchema();
        if (!key.type().equals(t)) {
            String msg = String.format("DataBox %s is not of type %s", key, t);
            throw new IllegalArgumentException(msg);
        }
    }

    // Iterator ////////////////////////////////////////////////////////////////
    private class BPlusTreeIterator implements Iterator<RecordId> {
        // A BPlusTreeIterator iterates over the entries of a B+ tree leaf by
        // leaf. We maintain the following invariants:
        //
        //   - leaf is null if and only if iter is null
        //   - iter is not null if and only if iter.hasNext()
        private LeafNode leaf;
        private Iterator<RecordId> iter;

        public BPlusTreeIterator(LeafNode leaf, Iterator<RecordId> iter) {
            assert (leaf != null);
            assert (iter != null);
            this.leaf = leaf;
            this.iter = iter;

            if (!this.iter.hasNext()) {
                advance();
            }
        }

        private void advance() {
            Optional<LeafNode> sibling = leaf.getRightSibling();
            if (sibling.isPresent()) {
                this.leaf = sibling.get();
                this.iter = this.leaf.scanAll();
                if (!this.iter.hasNext()) {
                    advance();
                }
            } else {
                this.leaf = null;
                this.iter = null;
            }
        }

        @Override
        public boolean hasNext() {
            return iter != null;
        }

        @Override
        public RecordId next() {
            if (!hasNext()) {
                throw new NoSuchElementException();
            }
            assert (leaf != null);
            assert (iter != null);
            assert (iter.hasNext());

            RecordId rid = iter.next();
            if (!iter.hasNext()) {
                advance();
```

```
            }
            return rid;
        }
    }
}
```