package com.twitter.search.core.earlybird.index.inverted;

import java.io.IOException;

import org.apache.lucene.util.Bits;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.search.common.util.io.flushable.DataDeserializer;

import com.twitter.search.common.util.io.flushable.DataSerializer;

import com.twitter.search.common.util.io.flushable.FlushInfo;

import com.twitter.search.common.util.io.flushable.Flushable;

import com.twitter.search.core.earlybird.index.DocIDToTweetIDMapper;

import it.unimi.dsi.fastutil.ints.Int2IntOpenHashMap;

public abstract class DeletedDocs implements Flushable {

private static final Logger LOG = LoggerFactory.getLogger(DeletedDocs.class);

/\*\*

\* Deletes the given document.

\*/

public abstract boolean deleteDoc(int docID);

/\*\*

\* Returns a point-in-time view of the deleted docs. Calling {@link #deleteDoc(int)} afterwards

\* will not alter this View.

\*/

public abstract View getView();

/\*\*

\* Number of deletions.

\*/

public abstract int numDeletions();

/\*\*

\* Returns a DeletedDocs instance that has the same deleted tweet IDs, but mapped to the doc IDs

\* in the optimizedTweetIdMapper.

\*

\* @param originalTweetIdMapper The original DocIDToTweetIDMapper instance that was used to add

\* doc IDs to this DeletedDocs instance.

\* @param optimizedTweetIdMapper The new DocIDToTweetIDMapper instance.

\* @return An DeletedDocs instance that has the same tweets deleted, but mapped to the doc IDs in

\* optimizedTweetIdMapper.

\*/

public abstract DeletedDocs optimize(

DocIDToTweetIDMapper originalTweetIdMapper,

DocIDToTweetIDMapper optimizedTweetIdMapper) throws IOException;

public abstract class View {

/\*\*

\* Returns true, if the given document was deleted.

\*/

public abstract boolean isDeleted(int docID);

/\*\*

\* Returns true, if there are any deleted documents in this View.

\*/

public abstract boolean hasDeletions();

/\*\*

\* Returns {@link Bits} where all deleted documents have their bit set to 0, and

\* all non-deleted documents have their bits set to 1.

\*/

public abstract Bits getLiveDocs();

}

public static class Default extends DeletedDocs {

private static final int KEY\_NOT\_FOUND = -1;

private final int size;

private final Int2IntOpenHashMap deletes;

// Each delete is marked with a unique, consecutively-increasing sequence ID.

private int sequenceID = 0;

public Default(int size) {

this.size = size;

deletes = new Int2IntOpenHashMap(size);

deletes.defaultReturnValue(KEY\_NOT\_FOUND);

}

/\*\*

\* Returns false, if this call was a noop, i.e. if the document was already deleted.

\*/

@Override

public boolean deleteDoc(int docID) {

if (deletes.putIfAbsent(docID, sequenceID) == KEY\_NOT\_FOUND) {

sequenceID++;

return true;

}

return false;

}

private boolean isDeleted(int internalID, int readerSequenceID) {

int deletedSequenceId = deletes.get(internalID);

return (deletedSequenceId >= 0) && (deletedSequenceId < readerSequenceID);

}

private boolean hasDeletions(int readerSequenceID) {

return readerSequenceID > 0;

}

@Override

public int numDeletions() {

return sequenceID;

}

@Override

public View getView() {

return new View() {

private final int readerSequenceID = sequenceID;

// liveDocs bitset contains inverted (decreasing) docids.

public final Bits liveDocs = !hasDeletions() ? null : new Bits() {

@Override

public final boolean get(int docID) {

return !isDeleted(docID);

}

@Override

public final int length() {

return size;

}

};

@Override

public Bits getLiveDocs() {

return liveDocs;

}

// Operates on internal (increasing) docids.

@Override

public final boolean isDeleted(int internalID) {

return DeletedDocs.Default.this.isDeleted(internalID, readerSequenceID);

}

@Override

public final boolean hasDeletions() {

return DeletedDocs.Default.this.hasDeletions(readerSequenceID);

}

};

}

@Override

public DeletedDocs optimize(DocIDToTweetIDMapper originalTweetIdMapper,

DocIDToTweetIDMapper optimizedTweetIdMapper) throws IOException {

DeletedDocs optimizedDeletedDocs = new Default(size);

for (int deletedDocID : deletes.keySet()) {

long tweetID = originalTweetIdMapper.getTweetID(deletedDocID);

int optimizedDeletedDocID = optimizedTweetIdMapper.getDocID(tweetID);

optimizedDeletedDocs.deleteDoc(optimizedDeletedDocID);

}

return optimizedDeletedDocs;

}

@SuppressWarnings("unchecked")

@Override

public Default.FlushHandler getFlushHandler() {

return new Default.FlushHandler(this, size);

}

public static final class FlushHandler extends Flushable.Handler<Default> {

private final int size;

public FlushHandler(Default objectToFlush, int size) {

super(objectToFlush);

this.size = size;

}

public FlushHandler(int size) {

this.size = size;

}

@Override

protected void doFlush(FlushInfo flushInfo, DataSerializer out) throws IOException {

long startTime = getClock().nowMillis();

Int2IntOpenHashMap deletes = getObjectToFlush().deletes;

out.writeIntArray(deletes.keySet().toIntArray());

getFlushTimerStats().timerIncrement(getClock().nowMillis() - startTime);

}

@Override

protected Default doLoad(FlushInfo flushInfo, DataDeserializer in) throws IOException {

Default deletedDocs = new Default(size);

long startTime = getClock().nowMillis();

int[] deletedDocIDs = in.readIntArray();

for (int docID : deletedDocIDs) {

deletedDocs.deleteDoc(docID);

}

getLoadTimerStats().timerIncrement(getClock().nowMillis() - startTime);

return deletedDocs;

}

}

}

public static final DeletedDocs NO\_DELETES = new DeletedDocs() {

@Override

public <T extends Flushable> Handler<T> getFlushHandler() {

return null;

}

@Override

public boolean deleteDoc(int docID) {

return false;

}

@Override

public DeletedDocs optimize(DocIDToTweetIDMapper originalTweetIdMapper,

DocIDToTweetIDMapper optimizedTweetIdMapper) {

return this;

}

@Override

public int numDeletions() {

return 0;

}

@Override

public View getView() {

return new View() {

@Override

public boolean isDeleted(int docID) {

return false;

}

@Override

public boolean hasDeletions() {

return false;

}

@Override

public Bits getLiveDocs() {

return null;

}

};

}

};

}