package com.twitter.search.earlybird.index;

import java.io.IOException;

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

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

public abstract class TweetIDMapper implements DocIDToTweetIDMapper, Flushable {

private long minTweetID;

private long maxTweetID;

private int minDocID;

private int maxDocID;

private int numDocs;

protected TweetIDMapper() {

this(Long.MAX\_VALUE, Long.MIN\_VALUE, Integer.MAX\_VALUE, Integer.MIN\_VALUE, 0);

}

protected TweetIDMapper(

long minTweetID, long maxTweetID, int minDocID, int maxDocID, int numDocs) {

this.minTweetID = minTweetID;

this.maxTweetID = maxTweetID;

this.minDocID = minDocID;

this.maxDocID = maxDocID;

this.numDocs = numDocs;

}

// Realtime updates minTweetID and maxTweetID in addMapping.

// Archives updates minTweetID and maxTweetID in prepareToRead.

protected void setMinTweetID(long minTweetID) {

this.minTweetID = minTweetID;

}

protected void setMaxTweetID(long maxTweetID) {

this.maxTweetID = maxTweetID;

}

protected void setMinDocID(int minDocID) {

this.minDocID = minDocID;

}

protected void setMaxDocID(int maxDocID) {

this.maxDocID = maxDocID;

}

protected void setNumDocs(int numDocs) {

this.numDocs = numDocs;

}

public long getMinTweetID() {

return this.minTweetID;

}

public long getMaxTweetID() {

return this.maxTweetID;

}

public int getMinDocID() {

return minDocID;

}

public int getMaxDocID() {

return maxDocID;

}

@Override

public int getNumDocs() {

return numDocs;

}

/\*\*

\* Given a tweetId, find the corresponding doc ID to start, or end, a search.

\*

\* In the ordered, dense doc ID mappers, this returns either the doc ID assigned to the tweet ID,

\* or doc ID of the next lowest tweet ID, if the tweet is not in the index. In this case

\* findMaxDocID is ignored.

\*

\* In {@link OutOfOrderRealtimeTweetIDMapper}, doc IDs are not ordered within a millisecond, so we

\* want to search the entire millisecond bucket for a filter. To accomplish this,

\* if findMaxDocId is true we return the largest possible doc ID for that millisecond.

\* If findMaxDocId is false, we return the smallest possible doc ID for that millisecond.

\*

\* The returned doc ID will be between smallestDocID and largestDocID (inclusive).

\* The returned doc ID may not be in the index.

\*/

public int findDocIdBound(long tweetID,

boolean findMaxDocID,

int smallestDocID,

int largestDocID) throws IOException {

if (tweetID > maxTweetID) {

return smallestDocID;

}

if (tweetID < minTweetID) {

return largestDocID;

}

int internalID = findDocIDBoundInternal(tweetID, findMaxDocID);

return Math.max(smallestDocID, Math.min(largestDocID, internalID));

}

@Override

public final int getNextDocID(int docID) {

if (numDocs <= 0) {

return ID\_NOT\_FOUND;

}

if (docID < minDocID) {

return minDocID;

}

if (docID >= maxDocID) {

return ID\_NOT\_FOUND;

}

return getNextDocIDInternal(docID);

}

@Override

public final int getPreviousDocID(int docID) {

if (numDocs <= 0) {

return ID\_NOT\_FOUND;

}

if (docID <= minDocID) {

return ID\_NOT\_FOUND;

}

if (docID > maxDocID) {

return maxDocID;

}

return getPreviousDocIDInternal(docID);

}

@Override

public int addMapping(final long tweetID) {

int docId = addMappingInternal(tweetID);

if (docId != ID\_NOT\_FOUND) {

++numDocs;

if (tweetID > maxTweetID) {

maxTweetID = tweetID;

}

if (tweetID < minTweetID) {

minTweetID = tweetID;

}

if (docId > maxDocID) {

maxDocID = docId;

}

if (docId < minDocID) {

minDocID = docId;

}

}

return docId;

}

/\*\*

\* Returns the smallest valid doc ID in this mapper that's strictly higher than the given doc ID.

\* If no such doc ID exists, ID\_NOT\_FOUND must be returned.

\*

\* The given docID is guaranteed to be in the range [minDocID, maxDocID).

\*

\* @param docID The current doc ID.

\* @return The smallest valid doc ID in this mapper that's strictly higher than the given doc ID,

\* or a negative number, if no such doc ID exists.

\*/

protected abstract int getNextDocIDInternal(int docID);

/\*\*

\* Returns the smallest valid doc ID in this mapper that's strictly higher than the given doc ID.

\* If no such doc ID exists, ID\_NOT\_FOUND must be returned.

\*

\* The given docID is guaranteed to be in the range (minDocID, maxDocID].

\*

\* @param docID The current doc ID.

\* @return The smallest valid doc ID in this mapper that's strictly higher than the given doc ID,

\* or a negative number, if no such doc ID exists.

\*/

protected abstract int getPreviousDocIDInternal(int docID);

protected abstract int addMappingInternal(final long tweetID);

/\*\*

\* See {@link TweetIDMapper#findDocIdBound}.

\*/

protected abstract int findDocIDBoundInternal(long tweetID,

boolean findMaxDocID) throws IOException;

}