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

import java.io.IOException;

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

/\*\*

\* Maps timestamps to the doc IDs assigned to the documents that are indexed (tweets, users, etc.).

\*/

public interface TimeMapper extends Flushable {

// Unless specified, all time fields are seconds-since-epoch.

int ILLEGAL\_TIME = Integer.MIN\_VALUE;

/\*\*

\* Returns the time of the newest tweet in the index.

\*

\* @return The time of the newest tweet in the index.

\*/

int getLastTime();

/\*\*

\* Returns the time of the oldest tweet in the index.

\*

\* @return The time of the oldest tweet in the index.

\*/

int getFirstTime();

/\*\*

\* Returns the timestamp of the document mapped to the given doc ID, or ILLEGAL\_TIME if this

\* mapper doesn't know about this doc ID.

\*

\* @param docID The document's internal ID.

\* @return The timestamp of the document mapped to the given doc ID.

\*/

int getTime(int docID);

/\*\*

\* Returns the doc ID of the first indexed document with a timestamp equal to or greater than the

\* given timestamp.

\*

\* If timeSeconds is larger than the max timestamp in this mapper, smallestDocID is returned.

\* If timeSeconds is smaller than the min timestamp in the mapper, the largest docID is returned.

\*

\* Note that when tweets are indexed out of order, this method might return the doc ID of a tweet

\* with a timestamp greater than timeSeconds, even if there's a tweet with a timestamp of

\* timeSeconds. So the callers of this method can use the returned doc ID as a starting point for

\* iteration purposes, but should have a check that the traversed doc IDs have a timestamp in the

\* desired range. See SinceUntilFilter.getDocIdSet() for an example.

\*

\* Example:

\* DocIds: 6, 5, 4, 3, 2, 1, 0

\* Times: 1, 5, 3, 4, 4, 3, 6

\* With that data:

\* findFirstDocId(1, 0) should return 6.

\* findFirstDocId(3, 0) should return 5.

\* findFirstDocId(4, 0) should return 5.

\* findFirstDocId(5, 0) should return 5.

\* findFirstDocId(6, 0) should return 0.

\*

\* @param timeSeconds The boundary timestamp, in seconds.

\* @param smallestDocID The doc ID to return if the given time boundary is larger than the max

\* timestamp in this mapper.

\*/

int findFirstDocId(int timeSeconds, int smallestDocID) throws IOException;

/\*\*

\* Optimizes this time mapper.

\*

\* At segment optimization time, the doc IDs assigned to the documents in that segment might

\* change (they might be mapped to a more compact space for performance reasons, for example).

\* When that happens, we need to remap accordingly the doc IDs stored in the time mapper for that

\* segment too. It would also be a good time to optimize the data stored in the time mapper.

\*

\* @param originalDocIdMapper The doc ID mapper used by this segment before it was optimized.

\* @param optimizedDocIdMapper The doc ID mapper used by this segment after it was optimized.

\* @return An optimized TimeMapper with the same tweet IDs.

\*/

TimeMapper optimize(DocIDToTweetIDMapper originalDocIdMapper,

DocIDToTweetIDMapper optimizedDocIdMapper) throws IOException;

}