package com.twitter.search.earlybird.search.queries;

import java.io.IOException;

import org.apache.lucene.index.LeafReader;

import org.apache.lucene.index.LeafReaderContext;

import org.apache.lucene.search.BooleanClause;

import org.apache.lucene.search.BooleanQuery;

import org.apache.lucene.search.DocIdSetIterator;

import org.apache.lucene.search.IndexSearcher;

import org.apache.lucene.search.Query;

import org.apache.lucene.search.ScoreMode;

import org.apache.lucene.search.Weight;

import com.twitter.search.common.query.DefaultFilterWeight;

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

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

import com.twitter.search.core.earlybird.index.util.AllDocsIterator;

import com.twitter.search.core.earlybird.index.util.RangeFilterDISI;

// Filters tweets according to since time and until time (in seconds).

// Note that since time is inclusive, and until time is exclusive.

public final class SinceUntilFilter extends Query {

public static final int NO\_FILTER = -1;

// These are both in seconds since the epoch.

private final int minTimeInclusive;

private final int maxTimeExclusive;

public static Query getSinceQuery(int sinceTimeSeconds) {

return new BooleanQuery.Builder()

.add(new SinceUntilFilter(sinceTimeSeconds, NO\_FILTER), BooleanClause.Occur.FILTER)

.build();

}

public static Query getUntilQuery(int untilTimeSeconds) {

return new BooleanQuery.Builder()

.add(new SinceUntilFilter(NO\_FILTER, untilTimeSeconds), BooleanClause.Occur.FILTER)

.build();

}

public static Query getSinceUntilQuery(int sinceTimeSeconds, int untilTimeSeconds) {

return new BooleanQuery.Builder()

.add(new SinceUntilFilter(sinceTimeSeconds, untilTimeSeconds), BooleanClause.Occur.FILTER)

.build();

}

private SinceUntilFilter(int sinceTime, int untilTime) {

this.minTimeInclusive = sinceTime != NO\_FILTER ? sinceTime : 0;

this.maxTimeExclusive = untilTime != NO\_FILTER ? untilTime : Integer.MAX\_VALUE;

}

@Override

public int hashCode() {

return (int) (minTimeInclusive \* 17 + maxTimeExclusive);

}

@Override

public boolean equals(Object obj) {

if (!(obj instanceof SinceUntilFilter)) {

return false;

}

SinceUntilFilter filter = SinceUntilFilter.class.cast(obj);

return (minTimeInclusive == filter.minTimeInclusive)

&& (maxTimeExclusive == filter.maxTimeExclusive);

}

@Override

public String toString(String field) {

if (minTimeInclusive > 0 && maxTimeExclusive != Integer.MAX\_VALUE) {

return "SinceFilter:" + this.minTimeInclusive + ",UntilFilter:" + maxTimeExclusive;

} else if (minTimeInclusive > 0) {

return "SinceFilter:" + this.minTimeInclusive;

} else {

return "UntilFilter:" + this.maxTimeExclusive;

}

}

@Override

public Weight createWeight(IndexSearcher searcher, ScoreMode scoreMode, float boost)

throws IOException {

return new DefaultFilterWeight(this) {

@Override

protected DocIdSetIterator getDocIdSetIterator(LeafReaderContext context) throws IOException {

LeafReader indexReader = context.reader();

if (!(indexReader instanceof EarlybirdIndexSegmentAtomicReader)) {

return new AllDocsIterator(indexReader);

}

EarlybirdIndexSegmentAtomicReader reader = (EarlybirdIndexSegmentAtomicReader) indexReader;

TimeMapper timeMapper = reader.getSegmentData().getTimeMapper();

int smallestDocID = timeMapper.findFirstDocId(maxTimeExclusive, reader.getSmallestDocID());

int largestDoc = timeMapper.findFirstDocId(minTimeInclusive, reader.getSmallestDocID());

int smallestDoc = smallestDocID > 0 ? smallestDocID - 1 : 0;

return new SinceUntilDocIdSetIterator(

reader,

timeMapper,

smallestDoc,

largestDoc,

minTimeInclusive,

maxTimeExclusive);

}

};

}

// Returns true if this TimeMapper is at least partially covered by these time filters.

public static boolean sinceUntilTimesInRange(

TimeMapper timeMapper, int sinceTime, int untilTime) {

return (sinceTime == NO\_FILTER || sinceTime <= timeMapper.getLastTime())

&& (untilTime == NO\_FILTER || untilTime >= timeMapper.getFirstTime());

}

private static final class SinceUntilDocIdSetIterator extends RangeFilterDISI {

private final TimeMapper timeMapper;

private final int minTimeInclusive;

private final int maxTimeExclusive;

public SinceUntilDocIdSetIterator(EarlybirdIndexSegmentAtomicReader reader,

TimeMapper timeMapper,

int smallestDocID,

int largestDocID,

int minTimeInclusive,

int maxExclusive) throws IOException {

super(reader, smallestDocID, largestDocID);

this.timeMapper = timeMapper;

this.minTimeInclusive = minTimeInclusive;

this.maxTimeExclusive = maxExclusive;

}

@Override

protected boolean shouldReturnDoc() {

final int docTime = timeMapper.getTime(docID());

return docTime >= minTimeInclusive && docTime < maxTimeExclusive;

}

}

}