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

import java.io.IOException;

import java.util.Arrays;

import java.util.Collection;

import com.google.common.base.Preconditions;

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.common.search.IntArrayDocIdSetIterator;

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

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

import com.twitter.search.earlybird.index.TweetIDMapper;

public final class RequiredStatusIDsFilter extends Query {

private final Collection<Long> statusIDs;

public static Query getRequiredStatusIDsQuery(Collection<Long> statusIDs) {

return new BooleanQuery.Builder()

.add(new RequiredStatusIDsFilter(statusIDs), BooleanClause.Occur.FILTER)

.build();

}

private RequiredStatusIDsFilter(Collection<Long> statusIDs) {

this.statusIDs = Preconditions.checkNotNull(statusIDs);

}

@Override

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

return new DefaultFilterWeight(this) {

@Override

protected DocIdSetIterator getDocIdSetIterator(LeafReaderContext context) throws IOException {

LeafReader leafReader = context.reader();

if (!(leafReader instanceof EarlybirdIndexSegmentAtomicReader)) {

return DocIdSetIterator.empty();

}

EarlybirdIndexSegmentAtomicReader reader = (EarlybirdIndexSegmentAtomicReader) leafReader;

TweetIDMapper idMapper = (TweetIDMapper) reader.getSegmentData().getDocIDToTweetIDMapper();

int docIdsSize = 0;

int[] docIds = new int[statusIDs.size()];

for (long statusID : statusIDs) {

int docId = idMapper.getDocID(statusID);

if (docId >= 0) {

docIds[docIdsSize++] = docId;

}

}

Arrays.sort(docIds, 0, docIdsSize);

DocIdSetIterator statusesDISI =

new IntArrayDocIdSetIterator(Arrays.copyOf(docIds, docIdsSize));

DocIdSetIterator allDocsDISI = new AllDocsIterator(reader);

// We only want to return IDs for fully indexed documents. So we need to make sure that

// every doc ID we return exists in allDocsDISI. However, allDocsDISI has all documents in

// this segment, so driving by allDocsDISI would be very slow. So we want to drive by

// statusesDISI, and use allDocsDISI as a post-filter. What this comes down to is that we do

// not want to call allDocsDISI.nextDoc(); we only want to call allDocsDISI.advance(), and

// only on the doc IDs returned by statusesDISI.

return new DocIdSetIterator() {

@Override

public int docID() {

return statusesDISI.docID();

}

@Override

public int nextDoc() throws IOException {

statusesDISI.nextDoc();

return advanceToNextFullyIndexedDoc();

}

@Override

public int advance(int target) throws IOException {

statusesDISI.advance(target);

return advanceToNextFullyIndexedDoc();

}

private int advanceToNextFullyIndexedDoc() throws IOException {

while (docID() != DocIdSetIterator.NO\_MORE\_DOCS) {

// Check if the current doc is fully indexed.

// If it is, then we can return it. If it's not, then we need to keep searching.

int allDocsDocId = allDocsDISI.advance(docID());

if (allDocsDocId == docID()) {

break;

}

statusesDISI.advance(allDocsDocId);

}

return docID();

}

@Override

public long cost() {

return statusesDISI.cost();

}

};

}

};

}

@Override

public int hashCode() {

return statusIDs.hashCode();

}

@Override

public boolean equals(Object obj) {

if (!(obj instanceof RequiredStatusIDsFilter)) {

return false;

}

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

return statusIDs.equals(filter.statusIDs);

}

@Override

public final String toString(String field) {

return String.format("RequiredStatusIDs[%s]", statusIDs);

}

}