package com.twitter.search.earlybird.util;

import java.io.IOException;

import java.io.PrintWriter;

import java.io.UnsupportedEncodingException;

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.List;

import java.util.Locale;

import java.util.Set;

import java.util.TreeSet;

import com.google.common.collect.ImmutableSet;

import com.google.common.collect.Lists;

import org.apache.lucene.index.IndexOptions;

import org.apache.lucene.index.NumericDocValues;

import org.apache.lucene.index.PostingsEnum;

import org.apache.lucene.index.Terms;

import org.apache.lucene.index.TermsEnum;

import org.apache.lucene.search.DocIdSetIterator;

import org.apache.lucene.util.BytesRef;

import com.twitter.search.common.constants.thriftjava.ThriftLanguage;

import com.twitter.search.common.schema.base.Schema;

import com.twitter.search.common.schema.earlybird.EarlybirdFieldConstants.EarlybirdFieldConstant;

import com.twitter.search.common.schema.thriftjava.ThriftCSFType;

import com.twitter.search.common.util.analysis.IntTermAttributeImpl;

import com.twitter.search.common.util.analysis.LongTermAttributeImpl;

import com.twitter.search.common.util.analysis.SortableLongTermAttributeImpl;

import com.twitter.search.common.util.spatial.GeoUtil;

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

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

import com.twitter.search.core.earlybird.index.inverted.MPHTermDictionary;

import com.twitter.search.core.earlybird.index.inverted.RealtimeIndexTerms;

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

import geo.google.datamodel.GeoCoordinate;

public class IndexViewer {

/\*\*

\* Fields whose terms are indexed using

\* {@link com.twitter.search.common.util.analysis.IntTermAttribute}

\*/

private static final Set<String> INT\_TERM\_ATTRIBUTE\_FIELDS = ImmutableSet.of(

EarlybirdFieldConstant.CREATED\_AT\_FIELD.getFieldName(),

EarlybirdFieldConstant.LINK\_CATEGORY\_FIELD.getFieldName(),

EarlybirdFieldConstant

.NORMALIZED\_FAVORITE\_COUNT\_GREATER\_THAN\_OR\_EQUAL\_TO\_FIELD.getFieldName(),

EarlybirdFieldConstant

.NORMALIZED\_REPLY\_COUNT\_GREATER\_THAN\_OR\_EQUAL\_TO\_FIELD.getFieldName(),

EarlybirdFieldConstant

.NORMALIZED\_RETWEET\_COUNT\_GREATER\_THAN\_OR\_EQUAL\_TO\_FIELD.getFieldName(),

EarlybirdFieldConstant.COMPOSER\_SOURCE.getFieldName());

/\*\*

\* Fields whose terms are indexed using

\* {@link com.twitter.search.common.util.analysis.LongTermAttribute}

\*/

private static final Set<String> LONG\_TERM\_ATTRIBUTE\_FIELDS = ImmutableSet.of(

EarlybirdFieldConstant.CONVERSATION\_ID\_FIELD.getFieldName(),

EarlybirdFieldConstant.LIKED\_BY\_USER\_ID\_FIELD.getFieldName(),

EarlybirdFieldConstant.QUOTED\_TWEET\_ID\_FIELD.getFieldName(),

EarlybirdFieldConstant.QUOTED\_USER\_ID\_FIELD.getFieldName(),

EarlybirdFieldConstant.REPLIED\_TO\_BY\_USER\_ID.getFieldName(),

EarlybirdFieldConstant.RETWEETED\_BY\_USER\_ID.getFieldName(),

EarlybirdFieldConstant.DIRECTED\_AT\_USER\_ID\_FIELD.getFieldName(),

EarlybirdFieldConstant.FROM\_USER\_ID\_FIELD.getFieldName(),

EarlybirdFieldConstant.IN\_REPLY\_TO\_TWEET\_ID\_FIELD.getFieldName(),

EarlybirdFieldConstant.IN\_REPLY\_TO\_USER\_ID\_FIELD.getFieldName(),

EarlybirdFieldConstant.RETWEET\_SOURCE\_TWEET\_ID\_FIELD.getFieldName(),

EarlybirdFieldConstant.RETWEET\_SOURCE\_USER\_ID\_FIELD.getFieldName());

/\*\*

\* Fields whose terms index using SORTED

\* {@link com.twitter.search.common.util.analysis.LongTermAttribute}

\*/

private static final Set<String> SORTED\_LONG\_TERM\_ATTRIBUTE\_FIELDS =

ImmutableSet.of(EarlybirdFieldConstant.ID\_FIELD.getFieldName());

private final EarlybirdSingleSegmentSearcher searcher;

private final EarlybirdIndexSegmentAtomicReader twitterReader;

public long getTimeSliceId() {

return searcher.getTimeSliceID();

}

public static class Options {

private boolean dumpHexTerms = false;

private String charset;

private double[] histogramBuckets;

private boolean termLengthHistogram;

public Options setDumpHexTerms(boolean dumpHexTermsParam) {

this.dumpHexTerms = dumpHexTermsParam;

return this;

}

public Options setCharset(String charsetParam) {

this.charset = charsetParam;

return this;

}

public Options setHistogramBuckets(double[] histogramBucketsParam) {

this.histogramBuckets = histogramBucketsParam;

return this;

}

public Options setTermLengthHistogram(boolean termLengthHistogramParam) {

this.termLengthHistogram = termLengthHistogramParam;

return this;

}

}

/\*\*

\* Data Transfer Object for Terms, encapsulates the "json" serialization

\* while maintaining streaming mode

\*/

private static class TermDto {

private final String field;

private final String term;

private final String docFreq;

private final String percent;

private final PostingsEnum docsEnum;

private final TermsEnum termsEnum;

private final Integer maxDocs;

public TermDto(String field, String term, String docFreq, String percent,

PostingsEnum docsEnum, TermsEnum termsEnum, Integer maxDocs) {

this.field = field;

this.term = term;

this.docFreq = docFreq;

this.percent = percent;

this.docsEnum = docsEnum;

this.termsEnum = termsEnum;

this.maxDocs = maxDocs;

}

public void write(ViewerWriter writer,

EarlybirdIndexSegmentAtomicReader twitterReader) throws IOException {

writer.beginObject();

writer.name("field").value(field);

writer.name("term").value(term);

writer.name("docFreq").value(docFreq);

writer.name("percent").value(percent);

if (docsEnum != null) {

appendFrequencyAndPositions(writer, field, docsEnum, twitterReader);

}

if (maxDocs != null) {

appendDocs(writer, termsEnum, maxDocs, twitterReader);

}

writer.endObject();

}

}

/\*\*

\* Data Transfer Object for Terms, encapsulates the "json" serialization

\* while maintaining streaming mode

\*/

private static class StatsDto {

private final String field;

private final String numTerms;

private final String terms;

public StatsDto(String field, String numTerms, String terms) {

this.field = field;

this.numTerms = numTerms;

this.terms = terms;

}

public void write(ViewerWriter writer) throws IOException {

writer.beginObject();

writer.name("field").value(field);

writer.name("numTerms").value(numTerms);

writer.name("terms").value(terms);

writer.endObject();

}

}

public IndexViewer(EarlybirdSingleSegmentSearcher searcher) {

this.searcher = searcher;

this.twitterReader = searcher.getTwitterIndexReader();

}

private boolean shouldSeekExact(Terms terms, TermsEnum termsEnum) {

return terms instanceof RealtimeIndexTerms

|| termsEnum instanceof MPHTermDictionary.MPHTermsEnum;

}

/\*\*

\* Dumps all terms for a given tweet id.

\* @param writer writer being used

\* @param tweetId the tweet id to use

\*/

public void dumpTweetDataByTweetId(ViewerWriter writer, long tweetId, Options options)

throws IOException {

int docId = twitterReader.getSegmentData().getDocIDToTweetIDMapper().getDocID(tweetId);

dumpTweetDataByDocId(writer, docId, options);

}

/\*\*

\* Dumps all terms for a given doc id.

\* @param writer writer being used

\* @param docId the document id to use.

\*/

public void dumpTweetDataByDocId(ViewerWriter writer, int docId, Options options)

throws IOException {

writer.beginObject();

printHeader(writer);

long tweetID = twitterReader.getSegmentData().getDocIDToTweetIDMapper().getTweetID(docId);

if (docId < twitterReader.maxDoc() && tweetID >= 0) {

writer.name("docId").value(Integer.toString(docId));

writer.name("tweetId").value(Long.toString(tweetID));

dumpIndexedFields(writer, docId, options);

dumpCsfFields(writer, docId);

}

writer.endObject();

}

/\*\*

\* Dumps all tweet IDs in the current segment to the given file.

\*/

public void dumpTweetIds(ViewerWriter writer, String logFile, PrintWriter logWriter)

throws IOException {

writeTweetIdsToLogFile(logWriter);

writer.beginObject();

writer.name(Long.toString(searcher.getTimeSliceID())).value(logFile);

writer.endObject();

}

private void writeTweetIdsToLogFile(PrintWriter logWriter) {

DocIDToTweetIDMapper mapper = twitterReader.getSegmentData().getDocIDToTweetIDMapper();

int docId = Integer.MIN\_VALUE;

while ((docId = mapper.getNextDocID(docId)) != DocIDToTweetIDMapper.ID\_NOT\_FOUND) {

long tweetId = mapper.getTweetID(docId);

// Ensure tweet ID is valid and non-deleted

if ((tweetId > 0) && !twitterReader.getDeletesView().isDeleted(docId)) {

logWriter.println(tweetId);

}

}

}

private void dumpIndexedFields(ViewerWriter writer, int docId,

Options options) throws IOException {

writer.name("indexedFields");

writer.beginArray();

writer.newline();

for (String field : sortedFields()) {

dumpTweetData(writer, field, docId, options);

}

writer.endArray();

writer.newline();

}

private void dumpCsfFields(ViewerWriter writer, int docId) throws IOException {

writer.name("csfFields");

writer.beginArray();

writer.newline();

dumpCSFData(writer, docId);

writer.endArray();

}

/\*\*

\* Dumps all CSF values for a given doc id.

\* @param writer writer being used

\* @param docId the document id to use.

\*/

private void dumpCSFData(ViewerWriter writer, int docId) throws IOException {

Schema tweetSchema = twitterReader.getSchema();

// Sort the FieldInfo objects to generate fixed order to make testing easier

List<Schema.FieldInfo> sortedFieldInfos = new ArrayList<>(tweetSchema.getFieldInfos());

sortedFieldInfos.sort(Comparator.comparing(Schema.FieldInfo::getFieldId));

for (Schema.FieldInfo fieldInfo: sortedFieldInfos) {

String csfFieldInfoName = fieldInfo.getName();

ThriftCSFType csfType = tweetSchema.getCSFFieldType(csfFieldInfoName);

NumericDocValues csfDocValues = twitterReader.getNumericDocValues(csfFieldInfoName);

// If twitterReader.getNumericDocValues(value.getName()) == null,

// means no NumericDocValue was indexed for the field so ignore

if (csfType != null && csfDocValues != null && csfDocValues.advanceExact(docId)) {

long csfValue = csfDocValues.longValue();

writer.beginObject();

writer.name("field").value(formatField(csfFieldInfoName));

writer.name("value");

if (csfFieldInfoName.equals(EarlybirdFieldConstant.LAT\_LON\_CSF\_FIELD.getFieldName())) {

writer.value(latlongDecode(csfValue));

} else if (csfFieldInfoName.equals(EarlybirdFieldConstant.LANGUAGE.getFieldName())) {

writer.value(languageDecode(csfValue));

} else if (csfFieldInfoName.equals(EarlybirdFieldConstant.CARD\_LANG\_CSF.getFieldName())) {

writer.value(languageDecode(csfValue));

} else {

writer.value(Long.toString(csfValue));

}

writer.endObject();

writer.newline();

}

}

}

/\*\*

\* Decipher long value gotten, put into format (lat, lon)

\* Decode the stored long value by creating a geocode

\*/

private String latlongDecode(long csfValue) {

StringBuilder sb = new StringBuilder();

GeoCoordinate geoCoordinate = new GeoCoordinate();

if (GeoUtil.decodeLatLonFromInt64(csfValue, geoCoordinate)) {

sb.append(geoCoordinate.getLatitude()).append(", ").append(geoCoordinate.getLongitude());

} else {

sb.append(csfValue).append(" (Value Unset or Invalid Coordinate)");

}

return sb.toString();

}

/\*\*

\* Decipher long value gotten into string of tweet's language

\*/

private String languageDecode(long csfValue) {

StringBuilder sb = new StringBuilder();

ThriftLanguage languageType = ThriftLanguage.findByValue((int) csfValue);

sb.append(csfValue).append(" (").append(languageType).append(")");

return sb.toString();

}

private void dumpTweetData(ViewerWriter writer,

String field,

int docId,

Options options) throws IOException {

Terms terms = twitterReader.terms(field);

if (terms != null) {

TermsEnum termsEnum = terms.iterator();

if (shouldSeekExact(terms, termsEnum)) {

long numTerms = terms.size();

for (int i = 0; i < numTerms; i++) {

termsEnum.seekExact(i);

dumpTweetDataTerm(writer, field, termsEnum, docId, options);

}

} else {

while (termsEnum.next() != null) {

dumpTweetDataTerm(writer, field, termsEnum, docId, options);

}

}

}

}

private void dumpTweetDataTerm(ViewerWriter writer, String field, TermsEnum termsEnum,

int docId, Options options) throws IOException {

PostingsEnum docsAndPositionsEnum = termsEnum.postings(null, PostingsEnum.ALL);

if (docsAndPositionsEnum != null && docsAndPositionsEnum.advance(docId) == docId) {

printTerm(writer, field, termsEnum, docsAndPositionsEnum, null, options);

}

}

/\*\*

\* Prints the histogram for the currently viewed index.

\* @param writer current viewerWriter

\* @param field if null, will use all fields

\* @param options options for dumping out text

\*/

public void dumpHistogram(ViewerWriter writer, String field, Options options) throws IOException {

writer.beginObject();

printHeader(writer);

writer.name("histogram");

writer.beginArray();

writer.newline();

if (field == null) {

for (String field2 : sortedFields()) {

dumpFieldHistogram(writer, field2, options);

}

} else {

dumpFieldHistogram(writer, field, options);

}

writer.endArray();

writer.endObject();

}

private void dumpFieldHistogram(ViewerWriter writer, String field, Options options)

throws IOException {

Histogram histo = new Histogram(options.histogramBuckets);

Terms terms = twitterReader.terms(field);

if (terms != null) {

TermsEnum termsEnum = terms.iterator();

if (shouldSeekExact(terms, termsEnum)) {

long numTerms = terms.size();

for (int i = 0; i < numTerms; i++) {

termsEnum.seekExact(i);

countHistogram(options, histo, termsEnum);

}

} else {

while (termsEnum.next() != null) {

countHistogram(options, histo, termsEnum);

}

}

printHistogram(writer, field, options, histo);

}

}

private void printHistogram(ViewerWriter writer, String field, Options options,

Histogram histo) throws IOException {

String bucket = options.termLengthHistogram ? "termLength" : "df";

for (Histogram.Entry histEntry : histo.entries()) {

String format =

String.format(Locale.US,

"field: %s %sBucket: %11s count: %10d "

+ "percent: %6.2f%% cumulative: %6.2f%% totalCount: %10d"

+ " sum: %15d percent: %6.2f%% cumulative: %6.2f%% totalSum: %15d",

formatField(field),

bucket,

histEntry.getBucketName(),

histEntry.getCount(),

histEntry.getCountPercent() \* 100.0,

histEntry.getCountCumulative() \* 100.0,

histo.getTotalCount(),

histEntry.getSum(),

histEntry.getSumPercent() \* 100.0,

histEntry.getSumCumulative() \* 100.0,

histo.getTotalSum()

);

writer.value(format);

writer.newline();

}

}

private void countHistogram(Options options, Histogram histo, TermsEnum termsEnum)

throws IOException {

if (options.termLengthHistogram) {

final BytesRef bytesRef = termsEnum.term();

histo.addItem(bytesRef.length);

} else {

histo.addItem(termsEnum.docFreq());

}

}

/\*\*

\* Prints terms and optionally documents for the currently viewed index.

\* @param writer writer being used

\* @param field if null, will use all fields

\* @param term if null will use all terms

\* @param maxTerms will print at most this many terms per field. If null will print 0 terms.

\* @param maxDocs will print at most this many documents, If null, will not print docs.

\* @param options options for dumping out text

\*/

public void dumpData(ViewerWriter writer, String field, String term, Integer maxTerms,

Integer maxDocs, Options options, boolean shouldSeekToTerm) throws IOException {

writer.beginObject();

printHeader(writer);

writer.name("terms");

writer.beginArray();

writer.newline();

dumpDataInternal(writer, field, term, maxTerms, maxDocs, options, shouldSeekToTerm);

writer.endArray();

writer.endObject();

}

private void dumpDataInternal(ViewerWriter writer, String field, String term, Integer maxTerms,

Integer maxDocs, Options options, boolean shouldSeekToTerm) throws IOException {

if (field == null) {

dumpDataForAllFields(writer, term, maxTerms, maxDocs, options);

return;

}

if (term == null) {

dumpDataForAllTerms(writer, field, maxTerms, maxDocs, options);

return;

}

Terms terms = twitterReader.terms(field);

if (terms != null) {

TermsEnum termsEnum = terms.iterator();

TermsEnum.SeekStatus status = termsEnum.seekCeil(new BytesRef(term));

if (status == TermsEnum.SeekStatus.FOUND) {

printTerm(writer, field, termsEnum, null, maxDocs, options);

}

if (shouldSeekToTerm) {

dumpTermsAfterSeek(writer, field, terms, maxTerms, maxDocs, options, termsEnum, status);

}

}

}

/\*\*

\* if term (cursor) is found for an indexed segment - dump the next termsLeft words

\* starting from the current position in the enum. For an indexed segment,

\* seekCeil will place the enum at the word or the next "ceiling" term. For

\* a realtime index, if the word is not found we do not paginate anything

\* We also only paginate if the TermsEnum is not at the end.

\*/

private void dumpTermsAfterSeek(ViewerWriter writer, String field, Terms terms, Integer maxTerms,

Integer maxDocs, Options options, TermsEnum termsEnum, TermsEnum.SeekStatus status)

throws IOException {

if (status != TermsEnum.SeekStatus.END) {

// for realtime, to not repeat the found word

if (shouldSeekExact(terms, termsEnum)) {

termsEnum.next();

}

if (status != TermsEnum.SeekStatus.FOUND) {

// if not found, print out curr term before calling next()

printTerm(writer, field, termsEnum, null, maxDocs, options);

}

for (int termsLeft = maxTerms - 1; termsLeft > 0 && termsEnum.next() != null; termsLeft--) {

printTerm(writer, field, termsEnum, null, maxDocs, options);

}

}

}

private void dumpDataForAllFields(ViewerWriter writer, String term, Integer maxTerms,

Integer maxDocs, Options options) throws IOException {

for (String field : sortedFields()) {

dumpDataInternal(writer, field, term, maxTerms, maxDocs, options, false);

}

}

private List<String> sortedFields() {

// Tweet facets are added to a special $facets field, which is not part of the schema.

// We include it here, because seeing the facets for a tweet is generally useful.

List<String> fields = Lists.newArrayList("$facets");

for (Schema.FieldInfo fieldInfo : twitterReader.getSchema().getFieldInfos()) {

if (fieldInfo.getFieldType().indexOptions() != IndexOptions.NONE) {

fields.add(fieldInfo.getName());

}

}

Collections.sort(fields);

return fields;

}

private void dumpDataForAllTerms(ViewerWriter writer,

String field,

Integer maxTerms,

Integer maxDocs,

Options options) throws IOException {

Terms terms = twitterReader.terms(field);

if (terms != null) {

TermsEnum termsEnum = terms.iterator();

if (shouldSeekExact(terms, termsEnum)) {

long numTerms = terms.size();

long termToDump = maxTerms == null ? 0 : Math.min(numTerms, maxTerms);

for (int i = 0; i < termToDump; i++) {

termsEnum.seekExact(i);

printTerm(writer, field, termsEnum, null, maxDocs, options);

}

} else {

int max = maxTerms == null ? 0 : maxTerms;

while (max > 0 && termsEnum.next() != null) {

printTerm(writer, field, termsEnum, null, maxDocs, options);

max--;

}

}

}

}

private String termToString(String field, BytesRef bytesTerm, Options options)

throws UnsupportedEncodingException {

if (INT\_TERM\_ATTRIBUTE\_FIELDS.contains(field)) {

return Integer.toString(IntTermAttributeImpl.copyBytesRefToInt(bytesTerm));

} else if (LONG\_TERM\_ATTRIBUTE\_FIELDS.contains(field)) {

return Long.toString(LongTermAttributeImpl.copyBytesRefToLong(bytesTerm));

} else if (SORTED\_LONG\_TERM\_ATTRIBUTE\_FIELDS.contains(field)) {

return Long.toString(SortableLongTermAttributeImpl.copyBytesRefToLong(bytesTerm));

} else {

if (options != null && options.charset != null && !options.charset.isEmpty()) {

return new String(bytesTerm.bytes, bytesTerm.offset, bytesTerm.length, options.charset);

} else {

return bytesTerm.utf8ToString();

}

}

}

private void printTerm(ViewerWriter writer, String field, TermsEnum termsEnum,

PostingsEnum docsEnum, Integer maxDocs, Options options)

throws IOException {

final BytesRef bytesRef = termsEnum.term();

StringBuilder termToString = new StringBuilder();

termToString.append(termToString(field, bytesRef, options));

if (options != null && options.dumpHexTerms) {

termToString.append(" ").append(bytesRef.toString());

}

final int df = termsEnum.docFreq();

double dfPercent = ((double) df / this.twitterReader.numDocs()) \* 100.0;

TermDto termDto = new TermDto(field, termToString.toString(), Integer.toString(df),

String.format(Locale.US, "%.2f%%", dfPercent),

docsEnum, termsEnum, maxDocs);

termDto.write(writer, twitterReader);

writer.newline();

}

private static void appendFrequencyAndPositions(ViewerWriter writer, String field,

PostingsEnum docsEnum, EarlybirdIndexSegmentAtomicReader twitterReader) throws IOException {

final int frequency = docsEnum.freq();

writer.name("freq").value(Integer.toString(frequency));

Schema schema = twitterReader.getSchema();

Schema.FieldInfo fieldInfo = schema.getFieldInfo(field);

if (fieldInfo != null

&& (fieldInfo.getFieldType().indexOptions() == IndexOptions.DOCS\_AND\_FREQS\_AND\_POSITIONS

|| fieldInfo.getFieldType().indexOptions()

== IndexOptions.DOCS\_AND\_FREQS\_AND\_POSITIONS\_AND\_OFFSETS)) {

appendPositions(writer, docsEnum);

}

}

private static void appendPositions(ViewerWriter writer, PostingsEnum docsAndPositionsEnum)

throws IOException {

writer.name("positions");

writer.beginArray();

final int frequency = docsAndPositionsEnum.freq();

for (int i = 0; i < frequency; i++) {

int position = docsAndPositionsEnum.nextPosition();

writer.value(Integer.toString(position));

}

writer.endArray();

}

private static void appendDocs(ViewerWriter writer, TermsEnum termsEnum, int maxDocs,

EarlybirdIndexSegmentAtomicReader twitterReader)

throws IOException {

writer.name("docIds");

writer.beginArray();

PostingsEnum docs = termsEnum.postings(null, 0);

int docsReturned = 0;

int docId;

boolean endedEarly = false;

DocIDToTweetIDMapper mapper = twitterReader.getSegmentData().getDocIDToTweetIDMapper();

while ((docId = docs.nextDoc()) != DocIdSetIterator.NO\_MORE\_DOCS) {

if (docsReturned < maxDocs) {

docsReturned++;

long tweetID = mapper.getTweetID(docId);

writer.beginObject();

writer.name("docId").value(Long.toString(docId));

writer.name("tweetId").value(Long.toString(tweetID));

writer.endObject();

} else {

endedEarly = true;

break;

}

}

if (endedEarly) {

writer.beginObject();

writer.name("status").value("ended early");

writer.endObject();

}

writer.endArray();

}

/\*\*

\* Prints generic stats for all fields in the currently viewed index.

\*/

public void dumpStats(ViewerWriter writer) throws IOException {

writer.beginObject();

printHeader(writer);

// stats section

writer.name("stats");

writer.beginArray();

writer.newline();

for (String field : sortedFields()) {

Terms terms = twitterReader.terms(field);

if (terms != null) {

printStats(writer, field, terms);

}

}

writer.endArray();

writer.endObject();

}

private void printStats(ViewerWriter writer, String field, Terms terms) throws IOException {

StatsDto statsDto = new StatsDto(

field, String.valueOf(terms.size()), terms.getClass().getCanonicalName());

statsDto.write(writer);

writer.newline();

}

private void printHeader(ViewerWriter writer) throws IOException {

writer.name("timeSliceId").value(Long.toString(this.searcher.getTimeSliceID()));

writer.name("maxDocNumber").value(Integer.toString(this.twitterReader.maxDoc()));

writer.newline();

}

private static String formatField(String field) {

return String.format("%20s", field);

}

/\*\*

\* Dumps out the schema of the current segment.

\* @param writer to be used for printing

\*/

public void dumpSchema(ViewerWriter writer) throws IOException {

writer.beginObject();

printHeader(writer);

writer.name("schemaFields");

writer.beginArray();

writer.newline();

Schema schema = this.twitterReader.getSchema();

// The fields in the schema are not sorted. Sort them so that the output is deterministic

Set<String> fieldNameSet = new TreeSet<>();

for (Schema.FieldInfo fieldInfo: schema.getFieldInfos()) {

fieldNameSet.add(fieldInfo.getName());

}

for (String fieldName : fieldNameSet) {

writer.value(fieldName);

writer.newline();

}

writer.endArray();

writer.endObject();

}

/\*\*

\* Dumps out the indexed fields inside the current segment.

\* Mainly used to help the front end populate the fields.

\* @param writer writer to be used for printing

\*/

public void dumpFields(ViewerWriter writer) throws IOException {

writer.beginObject();

printHeader(writer);

writer.name("fields");

writer.beginArray();

writer.newline();

for (String field : sortedFields()) {

writer.value(field);

writer.newline();

}

writer.endArray();

writer.endObject();

}

/\*\*

\* Dumps out the mapping of the tweet/tweetId to

\* a docId as well as segment/timeslide pair.

\* @param writer writer to be used for writing

\* @param tweetId tweetId that is input by user

\*/

public void dumpTweetIdToDocIdMapping(ViewerWriter writer, long tweetId) throws IOException {

writer.beginObject();

printHeader(writer);

writer.name("tweetId").value(Long.toString(tweetId));

int docId = twitterReader.getSegmentData().getDocIDToTweetIDMapper().getDocID(tweetId);

writer.name("docId").value(Integer.toString(docId));

writer.endObject();

writer.newline();

}

/\*\*

\* Dumps out the mapping of the docId to

\* tweetId and timeslice/segmentId pairs.

\* @param writer writer to be used for writing

\* @param docid docId that is input by user

\*/

public void dumpDocIdToTweetIdMapping(ViewerWriter writer, int docid) throws IOException {

writer.beginObject();

printHeader(writer);

long tweetId = twitterReader.getSegmentData().getDocIDToTweetIDMapper().getTweetID(docid);

writer.name("tweetId");

if (tweetId >= 0) {

writer.value(Long.toString(tweetId));

} else {

writer.value("Does not exist in segment");

}

writer.name("docid").value(Integer.toString(docid));

writer.endObject();

}

/\*\*

\* Print a response indicating that the given tweet id is not found in the index.

\*

\* Note that this method does not actually need the underlying index, and hence is setup as

\* a util function.

\*/

public static void writeTweetDoesNotExistResponse(ViewerWriter writer, long tweetId)

throws IOException {

writer.beginObject();

writer.name("tweetId");

writer.value(Long.toString(tweetId));

writer.name("docId");

writer.value("does not exist on this earlybird.");

writer.endObject();

}

}