package com.twitter.search.common.schema;

import java.io.IOException;

import java.io.StringReader;

import java.util.Collections;

import java.util.List;

import java.util.Set;

import com.google.common.annotations.VisibleForTesting;

import com.google.common.base.Preconditions;

import com.google.common.collect.ImmutableList;

import com.google.common.collect.Sets;

import org.apache.lucene.analysis.Analyzer;

import org.apache.lucene.analysis.TokenStream;

import org.apache.lucene.analysis.tokenattributes.CharTermAttribute;

import org.apache.lucene.analysis.tokenattributes.TermToBytesRefAttribute;

import org.apache.lucene.document.Document;

import org.apache.lucene.document.Field;

import org.apache.lucene.facet.sortedset.SortedSetDocValuesFacetField;

import org.apache.lucene.util.BytesRef;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import com.twitter.common.text.token.TwitterTokenStream;

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

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

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

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

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

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

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

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

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

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

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

import com.twitter.search.common.util.text.HighFrequencyTermPairs;

import com.twitter.search.common.util.text.OmitNormTextField;

import com.twitter.search.common.util.text.SingleTokenStream;

/\*\*

\* A document factory that converts {@link ThriftDocument} into Lucene {@link Document}s

\* using the provided {@link com.twitter.search.common.schema.base.Schema}.

\*/

public class SchemaDocumentFactory {

private static final Logger LOG = LoggerFactory.getLogger(SchemaDocumentFactory.class);

private final Schema schema;

private final ImmutableList<TokenStreamRewriter> tokenStreamRewriters;

/\*\*

\* Creates a SchemaDocumentFactory with a schema and the tokenStreamRewriters.

\*

\* @param tokenStreamRewriters a list of token stream rewriters, which will be applied in order.

\*/

public SchemaDocumentFactory(

Schema schema,

List<TokenStreamRewriter> tokenStreamRewriters) {

this.schema = schema;

this.tokenStreamRewriters = ImmutableList.copyOf(tokenStreamRewriters);

}

/\*\*

\* Creates a SchemaDocumentFactory with no tokenStreamRewriters.

\*/

public SchemaDocumentFactory(Schema schema) {

this(schema, Collections.EMPTY\_LIST);

}

public final Document newDocument(ThriftDocument document) throws IOException {

return innerNewDocument(document);

}

/\*\*

\* Create a Lucene document from the ThriftDocument.

\*/

@VisibleForTesting

public Document innerNewDocument(ThriftDocument document) throws IOException {

Document luceneDocument = new Document();

Set<String> hfTerms = Sets.newHashSet();

Set<String> hfPhrases = Sets.newHashSet();

Analyzer defaultAnalyzer = schema.getDefaultAnalyzer(document.getDefaultAnalyzerOverride());

for (ThriftField field : document.getFields()) {

boolean successful = false;

try {

addLuceneFields(field, defaultAnalyzer, luceneDocument, hfTerms, hfPhrases);

successful = true;

} finally {

if (!successful) {

LOG.warn("Unexpected exception while trying to add field. Field ID: "

+ field.getFieldConfigId() + " Field Name: "

+ schema.getFieldName(field.getFieldConfigId()));

}

}

}

for (String token : hfTerms) {

for (String token2 : hfTerms) {

if (token.compareTo(token2) < 0) {

luceneDocument.add(new Field(ImmutableSchema.HF\_TERM\_PAIRS\_FIELD,

HighFrequencyTermPairs.createPair(token, token2),

OmitNormTextField.TYPE\_NOT\_STORED));

}

}

}

for (String phrase : hfPhrases) {

// Tokens in the phrase set are not terms and have already been processed with

// HighFrequencyTermPairs.createPhrasePair.

luceneDocument.add(new Field(ImmutableSchema.HF\_PHRASE\_PAIRS\_FIELD, phrase,

OmitNormTextField.TYPE\_NOT\_STORED));

}

return schema.getFacetsConfig().build(luceneDocument);

}

private void addLuceneFields(ThriftField field, Analyzer analyzer, Document doc,

Set<String> hfTerms, Set<String> hfPhrases) throws IOException {

Schema.FieldInfo fieldInfo =

schema.getFieldInfo(field.getFieldConfigId(), field.getFieldConfigOverride());

if (fieldInfo == null) {

// field not defined in schema - skip it

return;

}

ThriftFieldData fieldData = field.getFieldData();

if (fieldInfo.getFieldType().getCsfType() != null) {

addCSFField(doc, fieldInfo, fieldData);

return;

}

// Checking which data type is set is not sufficient here. We also need to check schema to

// see what the type the field is configured to be. See SEARCH-5173 for more details.

// The problem is that Pig, while converting Tuples to Thrift, sets all primitive type

// fields to 0. (i.e. the isSet calls will return true).

IndexedNumericFieldSettings numericSettings =

fieldInfo.getFieldType().getNumericFieldSettings();

if (fieldData.isSetTokenStreamValue()) {

addTokenField(doc, hfTerms, hfPhrases, fieldInfo, fieldData);

} else if (fieldData.isSetStringValue()) {

addStringField(analyzer, doc, hfTerms, hfPhrases, fieldInfo, fieldData);

} else if (fieldData.isSetBytesValue()) {

addBytesField(doc, fieldInfo, fieldData);

} else if (fieldData.isSetGeoCoordinate()) {

addGeoField(doc, fieldInfo, fieldData);

} else if (numericSettings != null) {

// handle numeric fields.

switch (numericSettings.getNumericType()) {

case INT:

Preconditions.checkState(fieldData.isSetIntValue(),

"Int field does not have int value set. Field name: %s", fieldInfo.getName());

addIntField(doc, fieldInfo, fieldData);

break;

case LONG:

Preconditions.checkState(fieldData.isSetLongValue(),

"Long field does not have long value set. Field name: %s", fieldInfo.getName());

addLongField(doc, fieldInfo, fieldData);

break;

case FLOAT:

Preconditions.checkState(fieldData.isSetFloatValue(),

"Float field does not have float value set. Field name: %s ", fieldInfo.getName());

addFloatField();

break;

case DOUBLE:

Preconditions.checkState(fieldData.isSetDoubleValue(),

"Double field does not have double value set. Field name: %s", fieldInfo.getName());

addDoubleFIeld();

break;

default:

throw new UnsupportedOperationException("Earlybird does not know how to handle field "

+ field.getFieldConfigId() + " " + field);

}

} else {

throw new UnsupportedOperationException("Earlybird does not know how to handle field "

+ field.getFieldConfigId() + " " + field);

}

}

private void addCSFField(Document doc, Schema.FieldInfo fieldInfo, ThriftFieldData fieldData) {

if (fieldInfo.getFieldType().getCsfFixedLengthNumValuesPerDoc() > 1) {

// As an optimization, TBinaryProtocol stores a byte array field as a part of a larger byte

// array field. Must call fieldData.getBytesValue(). fieldData.bytesValue.array() will

// return extraneous data. See: SEARCH-3996

doc.add(new Field(fieldInfo.getName(), fieldData.getBytesValue(), fieldInfo.getFieldType()));

} else {

doc.add(new CSFField(fieldInfo.getName(), fieldInfo.getFieldType(), fieldData));

}

}

private void addTokenField(

Document doc,

Set<String> hfTerms,

Set<String> hfPhrases,

Schema.FieldInfo fieldInfo,

ThriftFieldData fieldData) throws IOException {

TwitterTokenStream twitterTokenStream

= fieldInfo.getFieldType().getTokenStreamSerializer().deserialize(

fieldData.getTokenStreamValue(), fieldData.getStringValue());

try {

for (TokenStreamRewriter rewriter : tokenStreamRewriters) {

twitterTokenStream = rewriter.rewrite(fieldInfo, twitterTokenStream);

}

expandStream(doc, fieldInfo, twitterTokenStream, hfTerms, hfPhrases);

doc.add(new Field(fieldInfo.getName(), twitterTokenStream, fieldInfo.getFieldType()));

} finally {

twitterTokenStream.close();

}

}

private void addStringField(Analyzer analyzer, Document doc, Set<String> hfTerms,

Set<String> hfPhrases, Schema.FieldInfo fieldInfo,

ThriftFieldData fieldData) {

doc.add(new Field(fieldInfo.getName(), fieldData.getStringValue(), fieldInfo.getFieldType()));

if (fieldInfo.getFieldType().tokenized()) {

try {

TokenStream tokenStream = analyzer.tokenStream(fieldInfo.getName(),

new StringReader(fieldData.getStringValue()));

try {

expandStream(

doc,

fieldInfo,

tokenStream,

hfTerms,

hfPhrases);

} finally {

tokenStream.close();

}

} catch (IOException e) {

LOG.error("IOException expanding token stream", e);

}

} else {

addFacetField(doc, fieldInfo, fieldData.getStringValue());

}

}

private void addBytesField(Document doc, Schema.FieldInfo fieldInfo, ThriftFieldData fieldData) {

doc.add(new Field(fieldInfo.getName(), fieldData.getBytesValue(), fieldInfo.getFieldType()));

}

private void addIntField(Document doc, Schema.FieldInfo fieldInfo,

ThriftFieldData fieldData) {

int value = fieldData.getIntValue();

addFacetField(doc, fieldInfo, String.valueOf(value));

if (fieldInfo.getFieldType().getNumericFieldSettings() == null) {

// No NumericFieldSettings. Even though the data is numeric, this field is not

// really a numerical field. Just add as a string.

doc.add(new Field(fieldInfo.getName(), String.valueOf(value), fieldInfo.getFieldType()));

} else if (fieldInfo.getFieldType().getNumericFieldSettings().isUseTwitterFormat()) {

addIntTermAttributeField(value, fieldInfo, doc);

} else {

// Use lucene style numerical fields

doc.add(NumericField.newIntField(fieldInfo.getName(), value));

}

}

private void addIntTermAttributeField(int value,

Schema.FieldInfo fieldInfo,

Document doc) {

SingleTokenStream singleToken = new SingleTokenStream();

IntTermAttribute termAtt = singleToken.addAttribute(IntTermAttribute.class);

termAtt.setTerm(value);

doc.add(new Field(fieldInfo.getName(), singleToken, fieldInfo.getFieldType()));

}

private void addLongField(Document doc, Schema.FieldInfo fieldInfo,

ThriftFieldData fieldData) {

long value = fieldData.getLongValue();

addFacetField(doc, fieldInfo, String.valueOf(value));

if (fieldInfo.getFieldType().getNumericFieldSettings() == null) {

// No NumericFieldSettings. Even though the data is numeric, this field is not

// really a numerical field. Just add as a string.

doc.add(new Field(fieldInfo.getName(), String.valueOf(value), fieldInfo.getFieldType()));

} else if (fieldInfo.getFieldType().getNumericFieldSettings().isUseTwitterFormat()) {

// Twitter style numerical field: use LongTermAttribute

addLongTermAttributeField(value, fieldInfo, doc);

} else {

// Use lucene style numerical fields

doc.add(NumericField.newLongField(fieldInfo.getName(), value));

}

}

private void addLongTermAttributeField(long value,

Schema.FieldInfo fieldInfo,

Document doc) {

SingleTokenStream singleToken = new SingleTokenStream();

boolean useSortableEncoding =

fieldInfo.getFieldType().getNumericFieldSettings().isUseSortableEncoding();

if (useSortableEncoding) {

SortableLongTermAttribute termAtt = singleToken.addAttribute(SortableLongTermAttribute.class);

termAtt.setTerm(value);

} else {

LongTermAttribute termAtt = singleToken.addAttribute(LongTermAttribute.class);

termAtt.setTerm(value);

}

doc.add(new Field(fieldInfo.getName(), singleToken, fieldInfo.getFieldType()));

}

private void addFloatField() {

throw new UnsupportedOperationException("Earlybird does not support float values yet.");

}

private void addDoubleFIeld() {

throw new UnsupportedOperationException("Earlybird does not support double values yet.");

}

private void addGeoField(Document doc, Schema.FieldInfo fieldInfo, ThriftFieldData fieldData) {

ThriftGeoCoordinate coord = fieldData.getGeoCoordinate();

if (GeoUtil.validateGeoCoordinates(coord.getLat(), coord.getLon())) {

GeoUtil.fillGeoFields(doc, fieldInfo.getName(),

coord.getLat(), coord.getLon(), coord.getAccuracy());

}

}

private void addFacetField(Document doc, Schema.FieldInfo fieldInfo, String value) {

Preconditions.checkArgument(doc != null);

Preconditions.checkArgument(fieldInfo != null);

Preconditions.checkArgument(value != null);

if (fieldInfo.getFieldType().getFacetName() != null) {

doc.add(new SortedSetDocValuesFacetField(fieldInfo.getFieldType().getFacetName(), value));

}

}

private String getTerm(TermToBytesRefAttribute attr) {

if (attr instanceof CharTermAttribute) {

return ((CharTermAttribute) attr).toString();

} else if (attr instanceof IntTermAttribute) {

return String.valueOf(((IntTermAttribute) attr).getTerm());

} else if (attr instanceof LongTermAttribute) {

return String.valueOf(((LongTermAttribute) attr).getTerm());

} else {

return attr.getBytesRef().utf8ToString();

}

}

/\*\*

\* Expand the TwitterTokenStream and populate high-frequency terms, phrases and/or facet category paths.

\*/

private void expandStream(

Document doc,

Schema.FieldInfo fieldInfo,

TokenStream stream,

Set<String> hfTerms,

Set<String> hfPhrases) throws IOException {

// Checkstyle does not allow assignment to parameters.

Set<String> facetHfTerms = hfTerms;

Set<String> facetHfPhrases = hfPhrases;

if (!(HighFrequencyTermPairs.INDEX\_HF\_TERM\_PAIRS

&& fieldInfo.getFieldType().isIndexHFTermPairs())) {

// high-frequency terms and phrases are not needed

if (fieldInfo.getFieldType().getFacetName() == null) {

// Facets are not needed either, simply return, would do nothing otherwise

return;

}

facetHfTerms = null;

facetHfPhrases = null;

}

final TermToBytesRefAttribute attr = stream.getAttribute(TermToBytesRefAttribute.class);

stream.reset();

String lastHFTerm = null;

while (stream.incrementToken()) {

String term = getTerm(attr);

if (fieldInfo.getFieldType().getFacetName() != null) {

addFacetField(doc, fieldInfo, term);

}

if (HighFrequencyTermPairs.HF\_TERM\_SET.contains(term)) {

if (facetHfTerms != null) {

facetHfTerms.add(term);

}

if (lastHFTerm != null) {

if (facetHfPhrases != null) {

facetHfPhrases.add(HighFrequencyTermPairs.createPhrasePair(lastHFTerm, term));

}

}

lastHFTerm = term;

} else {

lastHFTerm = null;

}

}

}

public static final class CSFField extends Field {

/\*\*

\* Create a CSFField with the given fieldType, containing the given field data.

\*/

public CSFField(String name, EarlybirdFieldType fieldType, ThriftFieldData data) {

super(name, fieldType);

if (fieldType.isCsfVariableLength()) {

fieldsData = new BytesRef(data.getBytesValue());

} else {

switch (fieldType.getCsfType()) {

case BYTE:

fieldsData = Long.valueOf(data.getByteValue());

break;

case INT:

fieldsData = Long.valueOf(data.getIntValue());

break;

case LONG:

fieldsData = Long.valueOf(data.getLongValue());

break;

case FLOAT:

fieldsData = Long.valueOf(Float.floatToRawIntBits((float) data.getFloatValue()));

break;

case DOUBLE:

fieldsData = Long.valueOf(Double.doubleToRawLongBits(data.getDoubleValue()));

break;

default:

throw new IllegalArgumentException("Unknown csf type: " + fieldType.getCsfType());

}

}

}

}

public interface TokenStreamRewriter {

/\*\*

\* Rewrite the token stream.

\*/

TwitterTokenStream rewrite(Schema.FieldInfo fieldInfo, TwitterTokenStream stream);

}

}